Shadow SC DOM

A Formal Model of the Safely Composable Document Object Model *with Shadow Roots*

Achim D. Brucker* Michael Herzberg†

September 1, 2025

*Department of Computer Science, University of Exeter, Exeter, UK
a.brucker@exeter.ac.uk

† Department of Computer Science, The University of Sheffield, Sheffield, UK
msherzberg1@sheffield.ac.uk

Abstract

In this AFP entry, we extend our formalization of the safely composable DOM (Core_SC_DOM) with *Shadow Roots*. Shadow roots are a recent proposal of the web community to support a component-based development approach for client-side web applications.

Shadow roots are a significant extension to the DOM standard and, as web standards are condemned to be backward compatible, such extensions often result in complex specification that may contain unwanted subtleties that can be detected by a formalization.

Our Isabelle/HOL formalization is, in the sense of object-orientation, an extension of our formalization of the core DOM and enjoys the same basic properties, i.e., it is 1. *extensible*, i.e., can be extended without the need of re-proving already proven properties and 2. *executable*, i.e., we can generate executable code from our specification. We exploit the executability to show that our formalization complies to the official standard of the W3C, respectively, the WHATWG.

Keywords: Document Object Model, DOM, Shadow Root, Web Component, Formal Semantics, Isabelle/HOL

Contents

1 Introduction **7**

2 The Shadow DOM **9**
2.1 The Shadow DOM Data Model (ShadowRootClass) 9
2.2 Shadow Root Monad (ShadowRootMonad) 16
2.3 The Shadow DOM (Shadow_DOM) 33

3 Test Suite **239**
3.1 Shadow DOM Base Tests (Shadow_DOM_BaseTest) 239
3.2 Testing slots (slots) 246
3.3 Testing slots_fallback (slots_fallback) 261
3.4 Testing Document_adoptNode (Shadow_DOM_Document_adoptNode) 269
3.5 Testing Document_getElementById (Shadow_DOM_Document_getElementById) 271
3.6 Testing Node_insertBefore (Shadow_DOM_Node_insertBefore) 275
3.7 Testing Node_removeChild (Shadow_DOM_Node_removeChild) 276
3.8 Shadow DOM Tests (Shadow_DOM_Tests) 278

1 Introduction

In a world in which more and more applications are offered as services on the internet, web browsers start to take on a similarly central role in our daily IT infrastructure as operating systems. Thus, web browsers should be developed as rigidly and formally as operating systems. While formal methods are a well-established technique in the development of operating systems (see, e. g., Klein [15] for an overview of formal verification of operating systems), there are few proposals for improving the development of web browsers using formal approaches [1, 12, 14, 16].

In [5], we formalized the core of the safely composable Document Object Model (DOM) in Isabelle/HOL. The DOM [17, 18] is *the* central data structure of all modern web browsers. In this work, we extend the formalization presented in [3] with support for *shadow trees*. Shadow trees are a recent addition to the DOM standard [18] that promise support for web components. As we will see, this promise is not fully achieved and, for example, the DOM standard itself does not formally define what a component should be. In this work, we focus on a standard compliant representation of the DOM with shadow trees. As [5], our formalization has the following properties:

- It provides a *consistency guarantee.* Since all definitions in our formal semantics are conservative and all rules are derived, the logical consistency of the DOM node-tree is reduced to the consistency of HOL.
- It serves as a *technical basis for a proof system.* Based on the derived rules and specific setup of proof tactics over node-trees, our formalization provides a generic proof environment for the verification of programs manipulating node-trees.
- It is *executable*, which allows to validate its compliance to the standard by evaluating the compliance test suite on the formal model and
- It is *extensible* in the sense of [2, 10], i. e., properties proven over the core DOM do not need to be re-proven for object-oriented extensions such as the HTML document model.

In this AFP entry, we limit ourselves to the faithful formalization of the DOM. As the DOM standard does not formally define web components, we address the question of formally defining web components and discussing their safety properties elsewhere [6, 8].

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle (for a more abstract presentation and more explanations, please see [13]). The structure follows the theory dependencies (see Figure 1.1): first, we formalize the DOM with Shadow Roots (chapter 2) and then formalize we the relevant compliance test cases in chapter 3.

Important Note: This document describes the formalization of the *Safely Composable Document Object Model with Shadow Roots* (SC DOM with Shadow Roots), which deviated in one important aspect from the official DOM standard: in the SC DOM, the shadow root is a sub-class of the document class (instead of a base class). This modification results in a stronger notion of web components that provide improved safety properties for the composition of web components. While the SC DOM still passes the compliance test suite as provided by the authors of the DOM standard, its data model is different. We refer readers interested in a formalisation of the standard compliant DOM to the AFP entries “Core_DOM” [3] and “Shadow_DOM” [7].

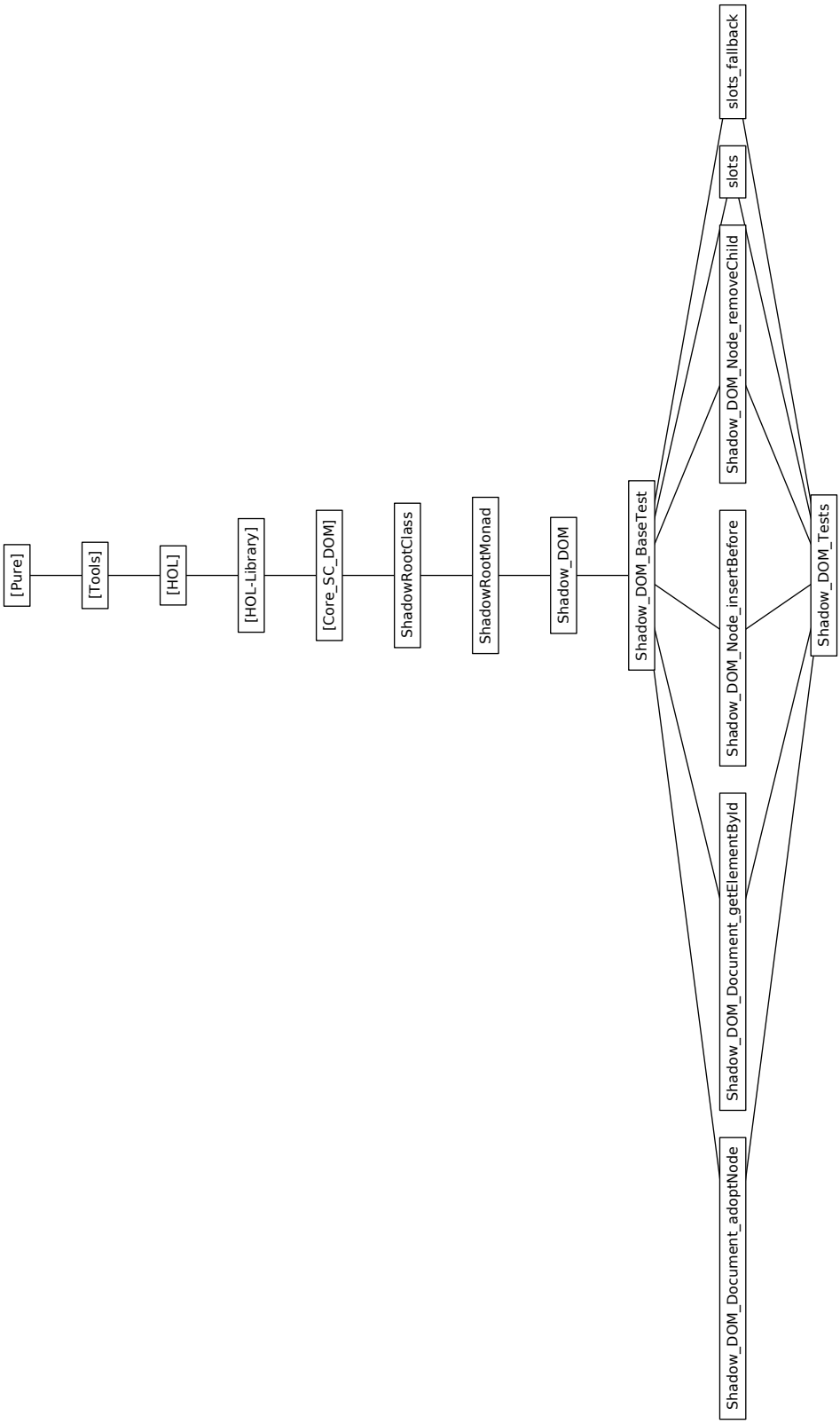

Figure 1.1: The Dependency Graph of the Isabelle Theories.

2 The Shadow DOM

In this chapter, we introduce the formalization of the core DOM *with Shadow Roots*, i.e., the most important algorithms for querying or modifying the Shadow DOM, as defined in the standard.

2.1 The Shadow DOM Data Model (ShadowRootClass)

```
theory ShadowRootClass
  imports
    Core_SC_DOM.ShadowRootPointer
    Core_SC_DOM.DocumentClass
begin
```

2.1.1 ShadowRoot

```
datatype shadow_root_mode = Open | Closed
record ('node_ptr, 'element_ptr, 'character_data_ptr) RShadowRoot =
  "('node_ptr, 'element_ptr, 'character_data_ptr) RDocument" +
  nothing :: unit
  mode :: shadow_root_mode
  child_nodes :: "('node_ptr, 'element_ptr, 'character_data_ptr) node_ptr list"
type_synonym ('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot) ShadowRoot
  = "('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot option) RShadowRoot_scheme"
register_default_tvars "('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot) ShadowRoot"
type_synonym ('node_ptr, 'element_ptr, 'character_data_ptr, 'Document, 'ShadowRoot) Document
  = "('node_ptr, 'element_ptr, 'character_data_ptr, ('node_ptr, 'element_ptr, 'character_data_ptr,
'ShadowRoot option) RShadowRoot_ext + 'Document) Document"
register_default_tvars "('node_ptr, 'element_ptr, 'character_data_ptr, 'Document, 'ShadowRoot) Document"
type_synonym ('node_ptr, 'element_ptr, 'character_data_ptr, 'shadow_root_ptr, 'Object, 'Node,
    'Element, 'CharacterData, 'Document,
    'ShadowRoot) Object
  = "('node_ptr, 'element_ptr, 'character_data_ptr, 'shadow_root_ptr, 'Object, 'Node, 'Element,
'CharacterData, ('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot option)
      RShadowRoot_ext + 'Document) Object"
register_default_tvars "('node_ptr, 'element_ptr, 'character_data_ptr, 'shadow_root_ptr, 'Object,
'Node, 'Element, 'CharacterData,
    'Document, 'ShadowRoot) Object"

type_synonym ('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr,
    'shadow_root_ptr, 'Object, 'Node,
    'Element, 'CharacterData, 'Document, 'ShadowRoot) heap
  = "('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr, 'shadow_root_ptr,
'Object, 'Node, 'Element, 'CharacterData, ('node_ptr, 'element_ptr,
      'character_data_ptr, 'ShadowRoot option) RShadowRoot_ext + 'Document) heap"
register_default_tvars "('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr,
'shadow_root_ptr, 'Object,
    'Node, 'Element, 'CharacterData, 'Document, 'ShadowRoot) heap"
type_synonym heap_final = "(unit, unit, unit, unit, unit, unit, unit, unit, unit, unit, unit, unit) heap"


definition shadow_root_ptr_kinds :: "(_) heap ⇒ (_) shadow_root_ptr fset"
  where
    "shadow_root_ptr_kinds heap =
the |`| (cast_document_ptr2shadow_root_ptr |`| (ffilter is_shadow_root_ptr_kind (document_ptr_kinds heap)))"
```

```
lemma shadow_root_ptr_kinds_simp [simp]:
  "shadow_root_ptr_kinds (Heap (fmupd (cast shadow_root_ptr) shadow_root (the_heap h))) =
{|shadow_root_ptr|} |∪| shadow_root_ptr_kinds h"
  by (auto simp add: shadow_root_ptr_kinds_def)

definition shadow_root_ptrs :: "(_) heap ⇒ (_) shadow_root_ptr fset"
  where
    "shadow_root_ptrs heap = ffilter is_shadow_root_ptr (shadow_root_ptr_kinds heap)"

definition cast_Document2ShadowRoot :: "(_) Document ⇒ (_) ShadowRoot option"
  where
    "cast_Document2ShadowRoot document = (case RDocument.more document of Some (Inl shadow_root) ⇒
Some (RDocument.extend (RDocument.truncate document) shadow_root) | _ ⇒ None)"
adhoc_overloading cast ⇌ cast_Document2ShadowRoot

abbreviation cast_Object2ShadowRoot :: "(_) Object ⇒ (_) ShadowRoot option"
  where
    "cast_Object2ShadowRoot obj ≡ (case cast_Object2Document obj of
Some document ⇒ cast_Document2ShadowRoot document | None ⇒ None)"
adhoc_overloading cast ⇌ cast_Object2ShadowRoot

definition cast_ShadowRoot2Document :: "(_) ShadowRoot ⇒ (_) Document"
  where
    "cast_ShadowRoot2Document shadow_root = RDocument.extend (RDocument.truncate shadow_root)
(Some (Inl (RDocument.more shadow_root)))"
adhoc_overloading cast ⇌ cast_ShadowRoot2Document

abbreviation cast_ShadowRoot2Object :: "(_) ShadowRoot ⇒ (_) Object"
  where
    "cast_ShadowRoot2Object ptr ≡ cast_Document2Object (cast_ShadowRoot2Document ptr)"
adhoc_overloading cast ⇌ cast_ShadowRoot2Object

consts is_shadow_root_kind :: 'a
definition is_shadow_root_kind_Document :: "(_) Document ⇒ bool"
  where
    "is_shadow_root_kind_Document ptr ⟷ cast_Document2ShadowRoot ptr ≠ None"

adhoc_overloading is_shadow_root_kind ⇌ is_shadow_root_kind_Document
lemmas is_shadow_root_kind_def = is_shadow_root_kind_Document_def

abbreviation is_shadow_root_kind_Object :: "(_) Object ⇒ bool"
  where
    "is_shadow_root_kind_Object ptr ≡ cast_Object2ShadowRoot ptr ≠ None"
adhoc_overloading is_shadow_root_kind ⇌ is_shadow_root_kind_Object

definition get_ShadowRoot :: "(_) shadow_root_ptr ⇒ (_) heap ⇒ (_) ShadowRoot option"
  where
    "get_ShadowRoot shadow_root_ptr h = Option.bind (get_Document (cast shadow_root_ptr) h) cast"
adhoc_overloading get ⇌ get_ShadowRoot

locale l_type_wf_def_ShadowRoot
begin
definition a_type_wf :: "(_) heap ⇒ bool"
  where
    "a_type_wf h = (DocumentClass.type_wf h ∧ (∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h)
                                                .get_ShadowRoot shadow_root_ptr h ≠ None))"
end
global_interpretation l_type_wf_def_ShadowRoot defines type_wf = a_type_wf .
lemmas type_wf_defs = a_type_wf_def

locale l_type_wf_ShadowRoot = l_type_wf type_wf for type_wf :: "((_) heap ⇒ bool)" +
  assumes type_wf_ShadowRoot: "type_wf h ⟹ ShadowRootClass.type_wf h"
```

```
sublocale l_type_wf_ShadowRoot ⊆ l_type_wf_Document
  apply(unfold_locales)
  using DocumentClass.a_type_wf_def
  by (meson ShadowRootClass.a_type_wf_def l_type_wf_ShadowRoot_axioms l_type_wf_ShadowRoot_def)

locale l_get_ShadowRoot_lemmas = l_type_wf_ShadowRoot
begin
sublocale l_get_Document_lemmas by unfold_locales

lemma get_ShadowRoot_type_wf:
  assumes "type_wf h"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟷ get_ShadowRoot shadow_root_ptr h ≠ None"
proof
  assume "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  then
  show "get shadow_root_ptr h ≠ None"
    using l_type_wf_ShadowRoot_axioms[unfolded l_type_wf_ShadowRoot_def type_wf_defs] assms
    by meson
next
  assume "get shadow_root_ptr h ≠ None"
  then
  show "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
    apply(auto simp add: get_ShadowRoot_def get_Document_def get_Object_def shadow_root_ptr_kinds_def
        document_ptr_kinds_def object_ptr_kinds_def
        split: Option.bind_splits)[1]
    by (metis (no_types, lifting) IntI document_ptr_casts_commute2 document_ptr_document_ptr_cast
        fmdomI image_iff is_shadow_root_ptr_kind_cast mem_Collect_eq option.sel
        shadow_root_ptr_casts_commute2)
qed
end

global_interpretation l_get_ShadowRoot_lemmas type_wf
  by unfold_locales

definition put_ShadowRoot :: "(_) shadow_root_ptr ⇒ (_) ShadowRoot ⇒ (_) heap ⇒ (_) heap"
  where
    "put_ShadowRoot shadow_root_ptr shadow_root = put_Document (cast shadow_root_ptr) (cast shadow_root)"
adhoc_overloading put ⇌ put_ShadowRoot

lemma put_ShadowRoot_ptr_in_heap:
  assumes "put_ShadowRoot shadow_root_ptr shadow_root h = h'"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h'"
  using assms
  unfolding put_ShadowRoot_def shadow_root_ptr_kinds_def
  by (metis ffmember_filter fimage_eqI is_shadow_root_ptr_kind_cast option.sel
      put_Document_ptr_in_heap shadow_root_ptr_casts_commute2)

lemma put_ShadowRoot_put_ptrs:
  assumes "put_ShadowRoot shadow_root_ptr shadow_root h = h'"
  shows "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast shadow_root_ptr|}"
  using assms
  by (simp add: put_ShadowRoot_def put_Document_put_ptrs)



lemma cast_ShadowRoot2Document_inject [simp]:
  "cast_ShadowRoot2Document x = cast_ShadowRoot2Document y ⟷ x = y"
  apply(auto simp add: cast_ShadowRoot2Document_def RObject.extend_def RDocument.extend_def
      RDocument.truncate_def)[1]
  by (metis RDocument.select_convs(5) RShadowRoot.surjective old.unit.exhaust)

lemma cast_Document2ShadowRoot_none [simp]:
  "cast_Document2ShadowRoot document = None ⟷ ¬ (∃shadow_root. cast_ShadowRoot2Document shadow_root =
```

```
document)"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RObject.extend_def
      RDocument.extend_def RDocument.truncate_def
      split: sum.splits option.splits)[1]
  by (metis (mono_tags, lifting) RDocument.select_convs(2) RDocument.select_convs(3)
      RDocument.select_convs(4) RDocument.select_convs(5) RDocument.surjective old.unit.exhaust)


lemma cast_Document2ShadowRoot_some [simp]:
  "cast_Document2ShadowRoot document = Some shadow_root ⟷ cast_ShadowRoot2Document shadow_root = document"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RObject.extend_def
      RDocument.extend_def RDocument.truncate_def
      split: sum.splits option.splits)[1]
  by (metis RDocument.select_convs(5) RShadowRoot.surjective old.unit.exhaust)


lemma cast_Document2ShadowRoot_inv [simp]:
  "cast_Document2ShadowRoot (cast_ShadowRoot2Document shadow_root) = Some shadow_root"
  by simp

lemma is_shadow_root_kind_doctype [simp]:
  "is_shadow_root_kind x ⟷ is_shadow_root_kind (doctype_update (λ_. v) x)"
  apply(auto simp add: is_shadow_root_kind_def cast_ShadowRoot2Document_def RDocument.extend_def
      RDocument.truncate_def split: option.splits)[1]
   apply (metis RDocument.ext_inject RDocument.select_convs(3) RDocument.surjective RObject.ext_inject)
  by (smt RDocument.select_convs(2) RDocument.select_convs(3) RDocument.select_convs(4)
      RDocument.select_convs(5) RDocument.surjective RDocument.update_convs(2) old.unit.exhaust)

lemma is_shadow_root_kind_document_element [simp]:
  "is_shadow_root_kind x ⟷ is_shadow_root_kind (document_element_update (λ_. v) x)"
  apply(auto simp add: is_shadow_root_kind_def cast_ShadowRoot2Document_def RDocument.extend_def
      RDocument.truncate_def  split: option.splits)[1]
   apply (metis RDocument.ext_inject RDocument.select_convs(3) RDocument.surjective RObject.ext_inject)
  by (metis (no_types, lifting) RDocument.ext_inject RDocument.surjective RDocument.update_convs(3)
      RObject.select_convs(1) RObject.select_convs(2))

lemma is_shadow_root_kind_disconnected_nodes [simp]:
  "is_shadow_root_kind x ⟷ is_shadow_root_kind (disconnected_nodes_update (λ_. v) x)"
  apply(auto simp add: is_shadow_root_kind_def cast_ShadowRoot2Document_def RDocument.extend_def
      RDocument.truncate_def split: option.splits)[1]
   apply (metis RDocument.ext_inject RDocument.select_convs(3) RDocument.surjective RObject.ext_inject)
  by (metis (no_types, lifting) RDocument.ext_inject RDocument.surjective RDocument.update_convs(4)
      RObject.select_convs(1) RObject.select_convs(2))

lemma shadow_root_ptr_kinds_commutes [simp]:
  "cast shadow_root_ptr |∈| document_ptr_kinds h ⟷ shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  apply(auto simp add: document_ptr_kinds_def shadow_root_ptr_kinds_def)[1]
  by (metis Int_iff imageI is_shadow_root_ptr_kind_cast mem_Collect_eq option.sel
      shadow_root_ptr_casts_commute2)

lemma get_shadow_root_ptr_simp1 [simp]:
  "get_ShadowRoot shadow_root_ptr (put_ShadowRoot shadow_root_ptr shadow_root h) = Some shadow_root"
  by(auto simp add: get_ShadowRoot_def put_ShadowRoot_def)
lemma get_shadow_root_ptr_simp2 [simp]:
  "shadow_root_ptr ≠ shadow_root_ptr'
   ⟹ get_ShadowRoot shadow_root_ptr (put_ShadowRoot shadow_root_ptr' shadow_root h) =
get_ShadowRoot shadow_root_ptr h"
  by(auto simp add: get_ShadowRoot_def put_ShadowRoot_def)

lemma get_shadow_root_ptr_simp3 [simp]:
  "get_Element element_ptr (put_ShadowRoot shadow_root_ptr f h) = get_Element element_ptr h"
  by(auto simp add: get_Element_def get_Node_def put_ShadowRoot_def put_Document_def)
lemma get_shadow_root_ptr_simp4 [simp]:
```

```
  "get_ShadowRoot shadow_root_ptr (put_Element element_ptr f h) = get_ShadowRoot shadow_root_ptr h"
  by(auto simp add: get_ShadowRoot_def get_Document_def put_Element_def put_Node_def)
lemma get_shadow_root_ptr_simp5 [simp]:
  "get_CharacterData character_data_ptr (put_ShadowRoot shadow_root_ptr f h) = get_CharacterData character_data_ptr
h"
  by(auto simp add: get_CharacterData_def get_Node_def put_ShadowRoot_def put_Document_def)
lemma get_shadow_root_ptr_simp6 [simp]:
  "get_ShadowRoot shadow_root_ptr (put_CharacterData character_data_ptr f h) = get_ShadowRoot shadow_root_ptr
h"
  by(auto simp add: get_ShadowRoot_def get_Document_def put_CharacterData_def put_Node_def)

lemma get_shadow_root_put_document [simp]:
  "cast shadow_root_ptr ≠ document_ptr
   ⟹ get_ShadowRoot shadow_root_ptr (put_Document document_ptr document h) = get_ShadowRoot shadow_root_ptr
h"
  by(auto simp add: get_ShadowRoot_def put_ShadowRoot_def)
lemma get_document_put_shadow_root [simp]:
  "document_ptr ≠ cast shadow_root_ptr
   ⟹ get_Document document_ptr (put_ShadowRoot shadow_root_ptr shadow_root h) = get_Document document_ptr
h"
  by(auto simp add: put_ShadowRoot_def)

lemma new_Element_get_ShadowRoot [simp]:
  assumes "new_Element h = (new_element_ptr, h')"
  shows "get_ShadowRoot ptr h = get_ShadowRoot ptr h'"
  using assms
  by(auto simp add: new_Element_def Let_def)

lemma new_CharacterData_get_ShadowRoot [simp]:
  assumes "new_CharacterData h = (new_character_data_ptr, h')"
  shows "get_ShadowRoot ptr h = get_ShadowRoot ptr h'"
  using assms
  by(auto simp add: new_CharacterData_def Let_def)

lemma new_Document_get_ShadowRoot [simp]:
  assumes "new_Document h = (new_document_ptr, h')"
  assumes "cast ptr ≠ new_document_ptr"
  shows "get_ShadowRoot ptr h = get_ShadowRoot ptr h'"
  using assms
  by(auto simp add: new_Document_def Let_def)


abbreviation "create_shadow_root_obj mode_arg child_nodes_arg
  ≡ (| RObject.nothing = (), RDocument.nothing = (), RDocument.doctype = ''html'',
RDocument.document_element = None, RDocument.disconnected_nodes = [], RShadowRoot.nothing = (),
mode = mode_arg, RShadowRoot.child_nodes = child_nodes_arg, ... = None |)"

definition new_ShadowRoot :: "(_)heap ⇒ ((_) shadow_root_ptr × (_) heap)"
  where
    "new_ShadowRoot h = (let new_shadow_root_ptr = shadow_root_ptr.Ref
(Suc (fMax (shadow_root_ptr.the_ref |`| (shadow_root_ptrs h)))) in
      (new_shadow_root_ptr, put new_shadow_root_ptr (create_shadow_root_obj Open []) h))"

lemma new_ShadowRoot_ptr_in_heap:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "new_shadow_root_ptr |∈| shadow_root_ptr_kinds h'"
  using assms
  unfolding new_ShadowRoot_def Let_def
  using put_ShadowRoot_ptr_in_heap by blast

lemma new_shadow_root_ptr_new: "shadow_root_ptr.Ref
(Suc (fMax (finsert 0 (shadow_root_ptr.the_ref |`| shadow_root_ptrs h)))) |∉| shadow_root_ptrs h"
  by (metis Suc_n_not_le_n shadow_root_ptr.sel(1) fMax_ge fimage_finsert finsertI1 finsertI2
```

```
    set_finsert)

lemma new_ShadowRoot_ptr_not_in_heap:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "new_shadow_root_ptr |∉| shadow_root_ptr_kinds h"
  using assms
  apply(auto simp add: Let_def new_ShadowRoot_def shadow_root_ptr_kinds_def)[1]
  by (metis Suc_n_not_le_n fMax_ge ffmember_filter fimageI is_shadow_root_ptr_ref
      shadow_root_ptr.disc(1) shadow_root_ptr.exhaust shadow_root_ptr.is_Ref_def shadow_root_ptr.sel(1)
      shadow_root_ptr.simps(4) shadow_root_ptr_casts_commute3 shadow_root_ptr_kinds_commutes
      shadow_root_ptrs_def)

lemma new_ShadowRoot_new_ptr:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
  using assms
  by (metis Pair_inject new_ShadowRoot_def put_ShadowRoot_put_ptrs)

lemma new_ShadowRoot_is_shadow_root_ptr:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "is_shadow_root_ptr new_shadow_root_ptr"
  using assms
  by(auto simp add: new_ShadowRoot_def Let_def)


lemma new_ShadowRoot_get_Object [simp]:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  assumes "ptr ≠ cast new_shadow_root_ptr"
  shows "get_Object ptr h = get_Object ptr h'"
  using assms
  by(auto simp add: new_ShadowRoot_def Let_def put_ShadowRoot_def put_Document_def)

lemma new_ShadowRoot_get_Node [simp]:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "get_Node ptr h = get_Node ptr h'"
  using assms
  apply(simp add: new_ShadowRoot_def Let_def put_ShadowRoot_def put_Document_def)
  by(auto simp add: get_Node_def)

lemma new_ShadowRoot_get_Element [simp]:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "get_Element ptr h = get_Element ptr h'"
  using assms
  by(auto simp add: new_ShadowRoot_def Let_def)

lemma new_ShadowRoot_get_CharacterData [simp]:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  shows "get_CharacterData ptr h = get_CharacterData ptr h'"
  using assms
  by(auto simp add: new_ShadowRoot_def Let_def)

lemma new_ShadowRoot_get_Document [simp]:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  assumes "ptr ≠ cast new_shadow_root_ptr"
  shows "get_Document ptr h = get_Document ptr h'"
  using assms
  apply(simp add: new_ShadowRoot_def Let_def put_ShadowRoot_def put_Document_def)
  by(auto simp add: get_Document_def)

lemma new_ShadowRoot_get_ShadowRoot [simp]:
  assumes "new_ShadowRoot h = (new_shadow_root_ptr, h')"
  assumes "ptr ≠ new_shadow_root_ptr"
  shows "get_ShadowRoot ptr h = get_ShadowRoot ptr h'"
```

```
  using assms
  by(auto simp add: new_ShadowRoot_def Let_def)


locale l_known_ptr_ShadowRoot
begin
definition a_known_ptr :: "(_) object_ptr ⇒ bool"
  where
    "a_known_ptr ptr = (known_ptr ptr ∨ is_shadow_root_ptr ptr)"

lemma known_ptr_not_shadow_root_ptr: "a_known_ptr ptr ⟹ ¬is_shadow_root_ptr ptr ⟹ known_ptr ptr"
  by(simp add: a_known_ptr_def)
lemma known_ptr_new_shadow_root_ptr: "a_known_ptr ptr ⟹ ¬known_ptr ptr ⟹ is_shadow_root_ptr ptr"
  using l_known_ptr_ShadowRoot.known_ptr_not_shadow_root_ptr by blast

end
global_interpretation l_known_ptr_ShadowRoot defines known_ptr = a_known_ptr .
lemmas known_ptr_defs = a_known_ptr_def

locale l_known_ptrs_ShadowRoot = l_known_ptr known_ptr for known_ptr :: "(_) object_ptr ⇒ bool"
begin
definition a_known_ptrs :: "(_) heap ⇒ bool"
  where
    "a_known_ptrs h = (∀ptr ∈ fset (object_ptr_kinds h). known_ptr ptr)"

lemma known_ptrs_known_ptr: "a_known_ptrs h ⟹ ptr |∈| object_ptr_kinds h ⟹ known_ptr ptr"
  by (simp add: a_known_ptrs_def)

lemma known_ptrs_preserved:
  "object_ptr_kinds h = object_ptr_kinds h' ⟹ a_known_ptrs h = a_known_ptrs h'"
  by(auto simp add: a_known_ptrs_def)
lemma known_ptrs_subset:
  "object_ptr_kinds h' |⊆| object_ptr_kinds h ⟹ a_known_ptrs h ⟹ a_known_ptrs h'"
  by(simp add: a_known_ptrs_def less_eq_fset.rep_eq subsetD)
lemma known_ptrs_new_ptr:
  "object_ptr_kinds h' = object_ptr_kinds h |∪| {|new_ptr|} ⟹ known_ptr new_ptr ⟹
a_known_ptrs h ⟹ a_known_ptrs h'"
  by(simp add: a_known_ptrs_def)
end
global_interpretation l_known_ptrs_ShadowRoot known_ptr defines known_ptrs = a_known_ptrs .
lemmas known_ptrs_defs = a_known_ptrs_def

lemma known_ptrs_is_l_known_ptrs  [instances]: "l_known_ptrs known_ptr known_ptrs"
  using known_ptrs_known_ptr known_ptrs_preserved l_known_ptrs_def known_ptrs_subset known_ptrs_new_ptr
  by blast

lemma known_ptrs_implies: "DocumentClass.known_ptrs h ⟹ ShadowRootClass.known_ptrs h"
  by(auto simp add: DocumentClass.known_ptrs_defs DocumentClass.known_ptr_defs
      ShadowRootClass.known_ptrs_defs ShadowRootClass.known_ptr_defs)

definition delete_ShadowRoot :: "(_) shadow_root_ptr ⇒ (_) heap ⇒ (_) heap option" where
  "delete_ShadowRoot shadow_root_ptr = delete_Object (cast shadow_root_ptr)"

lemma delete_ShadowRoot_pointer_removed:
  assumes "delete_ShadowRoot ptr h = Some h'"
  shows "ptr |∉| shadow_root_ptr_kinds h'"
  using assms
  by(auto simp add: delete_Object_pointer_removed delete_ShadowRoot_def shadow_root_ptr_kinds_def
      document_ptr_kinds_def split: if_splits)

lemma delete_ShadowRoot_pointer_ptr_in_heap:
  assumes "delete_ShadowRoot ptr h = Some h'"
  shows "ptr |∈| shadow_root_ptr_kinds h"
```

```
  using assms
  apply(auto simp add: delete_Object_pointer_ptr_in_heap delete_ShadowRoot_def split: if_splits)[1]
  using delete_Object_pointer_ptr_in_heap
  by fastforce

lemma delete_ShadowRoot_ok:
  assumes "ptr |∈| shadow_root_ptr_kinds h"
  shows "delete_ShadowRoot ptr h ≠ None"
  using assms
  by (simp add: delete_Object_ok delete_ShadowRoot_def)

lemma shadow_root_delete_get_1 [simp]:
  "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹ get_ShadowRoot shadow_root_ptr h' = None"
  by(auto simp add: delete_ShadowRoot_def delete_Object_def get_ShadowRoot_def get_Document_def get_Object_def
      split: if_splits)
lemma shadow_root_delete_get_2 [simp]:
  "shadow_root_ptr ≠ shadow_root_ptr' ⟹ delete_ShadowRoot shadow_root_ptr' h = Some h' ⟹
get_ShadowRoot shadow_root_ptr h' = get_ShadowRoot shadow_root_ptr h"
  by(auto simp add: delete_ShadowRoot_def delete_Object_def get_ShadowRoot_def get_Document_def get_Object_def
      split: if_splits)


lemma shadow_root_delete_get_3 [simp]:
  "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹ object_ptr ≠ cast shadow_root_ptr ⟹
get_Object object_ptr h' = get_Object object_ptr h"
  by(auto simp add: delete_ShadowRoot_def delete_Object_def get_ShadowRoot_def get_Object_def split: if_splits)
lemma shadow_root_delete_get_4 [simp]: "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹
get_Node node_ptr h' = get_Node node_ptr h"
  by(auto simp add: get_Node_def)
lemma shadow_root_delete_get_5 [simp]: "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹
get_Element element_ptr h' = get_Element element_ptr h"
  by(simp add: get_Element_def)
lemma shadow_root_delete_get_6 [simp]: "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹
get_CharacterData character_data_ptr h' = get_CharacterData character_data_ptr h"
  by(simp add: get_CharacterData_def)
lemma shadow_root_delete_get_7 [simp]:
  "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹ document_ptr  ≠ cast shadow_root_ptr ⟹
get_Document document_ptr h' = get_Document document_ptr h"
  by(simp add: get_Document_def)
lemma shadow_root_delete_get_8 [simp]:
  "delete_ShadowRoot shadow_root_ptr h = Some h' ⟹ shadow_root_ptr' ≠ shadow_root_ptr ⟹
get_ShadowRoot shadow_root_ptr' h' = get_ShadowRoot shadow_root_ptr' h"
  by(auto simp add: delete_ShadowRoot_def delete_Object_def get_ShadowRoot_def get_Object_def split: if_splits)
end
```

2.2 Shadow Root Monad (ShadowRootMonad)

```
theory ShadowRootMonad
  imports
    "Core_SC_DOM.DocumentMonad"
    "../classes/ShadowRootClass"
begin


type_synonym ('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr,
    'shadow_root_ptr, 'Object, 'Node, 'Element, 'CharacterData, 'Document, 'ShadowRoot, 'result) dom_prog
  = "((_) heap, exception, 'result) prog"
register_default_tvars "('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr,
'shadow_root_ptr, 'Object, 'Node, 'Element, 'CharacterData, 'Document, 'ShadowRoot, 'result) dom_prog"
```

```
global_interpretation l_ptr_kinds_M shadow_root_ptr_kinds defines shadow_root_ptr_kinds_M = a_ptr_kinds_M
.
lemmas shadow_root_ptr_kinds_M_defs = a_ptr_kinds_M_def

lemma shadow_root_ptr_kinds_M_eq:
  assumes "|h ⊢ object_ptr_kinds_M|_r = |h' ⊢ object_ptr_kinds_M|_r"
  shows "|h ⊢ shadow_root_ptr_kinds_M|_r = |h' ⊢ shadow_root_ptr_kinds_M|_r"
  using assms
  by(auto simp add: shadow_root_ptr_kinds_M_defs document_ptr_kinds_def object_ptr_kinds_M_defs
      shadow_root_ptr_kinds_def)



global_interpretation l_dummy defines get_M_ShadowRoot = "l_get_M.a_get_M get_ShadowRoot" .
lemma get_M_is_l_get_M: "l_get_M get_ShadowRoot type_wf shadow_root_ptr_kinds"
proof -
  have "∀ptr h. (∃y. get ptr h = Some y) ⟶ ptr |∈| shadow_root_ptr_kinds h"
  apply(auto simp add: get_ShadowRoot_def shadow_root_ptr_kinds_def bind_eq_Some_conv image_iff Bex_def)
  by (metis (no_types, opaque_lifting) DocumentMonad.l_get_M_axioms is_shadow_root_ptr_kind_cast l_get_M_def
      option.sel option.simps(3) shadow_root_ptr_casts_commute2)
  thus ?thesis
    by (simp add: get_ShadowRoot_type_wf l_get_M_def)
qed

lemmas get_M_defs = get_M_ShadowRoot_def[unfolded l_get_M.a_get_M_def[OF get_M_is_l_get_M]]

adhoc_overloading get_M ⇌ get_M_ShadowRoot

locale l_get_M_ShadowRoot_lemmas = l_type_wf_ShadowRoot
begin
sublocale l_get_M_CharacterData_lemmas by unfold_locales

interpretation l_get_M get_ShadowRoot type_wf shadow_root_ptr_kinds
  apply(unfold_locales)
   apply (simp add: get_ShadowRoot_type_wf local.type_wf_ShadowRoot)
  by (meson ShadowRootMonad.get_M_is_l_get_M l_get_M_def)
lemmas get_M_ShadowRoot_ok = get_M_ok[folded get_M_ShadowRoot_def]
lemmas get_M_ShadowRoot_ptr_in_heap = get_M_ptr_in_heap[folded get_M_ShadowRoot_def]
end

global_interpretation l_get_M_ShadowRoot_lemmas type_wf by unfold_locales


global_interpretation l_put_M type_wf shadow_root_ptr_kinds get_ShadowRoot put_ShadowRoot rewrites
  "a_get_M = get_M_ShadowRoot" defines put_M_ShadowRoot = a_put_M
   apply (simp add: get_M_is_l_get_M l_put_M_def)
  by (simp add: get_M_ShadowRoot_def)

lemmas put_M_defs = a_put_M_def
adhoc_overloading put_M ⇌ put_M_ShadowRoot


locale l_put_M_ShadowRoot_lemmas = l_type_wf_ShadowRoot
begin
sublocale l_put_M_CharacterData_lemmas by unfold_locales

interpretation l_put_M type_wf shadow_root_ptr_kinds get_ShadowRoot put_ShadowRoot
  apply(unfold_locales)
   apply (simp add: get_ShadowRoot_type_wf local.type_wf_ShadowRoot)
  by (meson ShadowRootMonad.get_M_is_l_get_M l_get_M_def)
lemmas put_M_ShadowRoot_ok = put_M_ok[folded put_M_ShadowRoot_def]
end
```

```
global_interpretation l_put_M_ShadowRoot_lemmas type_wf by unfold_locales


lemma shadow_root_put_get [simp]: "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
  ⟹ (⋀x. getter (setter (λ_. v) x) = v)
  ⟹ h' ⊢ get_M_ShadowRoot shadow_root_ptr getter →_r v"
  by(auto simp add: put_M_defs get_M_defs split: option.splits)
lemma get_M_Mshadow_root_preserved1 [simp]:
  "shadow_root_ptr ≠ shadow_root_ptr'
    ⟹ h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
    ⟹ preserved (get_M_ShadowRoot shadow_root_ptr' getter) h h'"
  by(auto simp add: put_M_defs get_M_defs preserved_def split: option.splits dest: get_heap_E)
lemma shadow_root_put_get_preserved [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
   ⟹ (⋀x. getter (setter (λ_. v) x) = getter x)
   ⟹ preserved (get_M_ShadowRoot shadow_root_ptr' getter) h h'"
  apply(cases "shadow_root_ptr = shadow_root_ptr'")
  by(auto simp add: put_M_defs get_M_defs preserved_def split: option.splits dest: get_heap_E)

lemma get_M_Mshadow_root_preserved2 [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h' ⟹ preserved (get_M_Node node_ptr getter) h h'"
  by(auto simp add: put_M_defs get_M_defs NodeMonad.get_M_defs
      put_ShadowRoot_def put_Document_def get_Node_def preserved_def split: option.splits dest: get_heap_E)

lemma get_M_Mshadow_root_preserved3 [simp]:
  "cast shadow_root_ptr ≠ document_ptr
   ⟹ h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
   ⟹ preserved (get_M_Document document_ptr getter) h h'"
  by(auto simp add: put_M_defs get_M_defs put_ShadowRoot_def put_Document_def DocumentMonad.get_M_defs
      preserved_def split: option.splits dest: get_heap_E)

lemma get_M_Mshadow_root_preserved4  [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
   ⟹ (⋀x. getter (cast (setter (λ_. v) x)) = getter (cast x))
   ⟹ preserved (get_M_Document document_ptr getter) h h'"
  apply(cases "cast shadow_root_ptr ≠ document_ptr")[1]
  by(auto simp add: put_M_defs get_M_defs get_ShadowRoot_def put_ShadowRoot_def get_Document_def put_Document_def
      DocumentMonad.get_M_defs preserved_def
      split: option.splits bind_splits dest: get_heap_E)

lemma get_M_Mshadow_root_preserved3a [simp]:
  "cast shadow_root_ptr ≠ object_ptr
   ⟹ h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
   ⟹ preserved (get_M_Object object_ptr getter) h h'"
  by(auto simp add: put_M_defs get_M_defs put_ShadowRoot_def put_Document_def ObjectMonad.get_M_defs
      preserved_def split: option.splits dest: get_heap_E)

lemma get_M_Mshadow_root_preserved4a  [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
   ⟹ (⋀x. getter (cast (setter (λ_. v) x)) = getter (cast x))
   ⟹ preserved (get_M_Object object_ptr getter) h h'"
  apply(cases "cast shadow_root_ptr ≠ object_ptr")[1]
  by(auto simp add: put_M_defs get_M_defs get_ShadowRoot_def put_ShadowRoot_def get_Document_def put_Document_def
      ObjectMonad.get_M_defs preserved_def
      split: option.splits bind_splits dest: get_heap_E)

lemma get_M_Mshadow_root_preserved5 [simp]:
  "cast shadow_root_ptr ≠ object_ptr
  ⟹ h ⊢ put_M_Object object_ptr setter v →_h h'
  ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter) h h'"
  by(auto simp add: ObjectMonad.put_M_defs get_M_defs get_ShadowRoot_def ObjectMonad.get_M_defs
      preserved_def split: option.splits dest: get_heap_E)
```

```
lemma get_M_Mshadow_root_preserved6 [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h' ⟹ preserved (get_M_Element element_ptr getter)
h h'"
  by(auto simp add: put_M_defs ElementMonad.get_M_defs preserved_def
      split: option.splits dest: get_heap_E)
lemma get_M_Mshadow_root_preserved7 [simp]:
  "h ⊢ put_M_Element element_ptr setter v →_h h' ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter)
h h'"
  by(auto simp add: ElementMonad.put_M_defs get_M_defs preserved_def
      split: option.splits dest: get_heap_E)
lemma get_M_Mshadow_root_preserved8 [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
    ⟹ preserved (get_M_CharacterData character_data_ptr getter) h h'"
  by(auto simp add: put_M_defs CharacterDataMonad.get_M_defs preserved_def
      split: option.splits dest: get_heap_E)
lemma get_M_Mshadow_root_preserved9 [simp]:
  "h ⊢ put_M_CharacterData character_data_ptr setter v →_h h'
    ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter) h h'"
  by(auto simp add: CharacterDataMonad.put_M_defs get_M_defs preserved_def
      split: option.splits dest: get_heap_E)

lemma get_M_shadow_root_put_M_document_different_pointers [simp]:
  "cast shadow_root_ptr ≠ document_ptr
    ⟹ h ⊢ put_M_Document document_ptr setter v →_h h'
    ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter) h h'"
  by(auto simp add: DocumentMonad.put_M_defs get_M_defs DocumentMonad.get_M_defs preserved_def
      split: option.splits dest: get_heap_E)
lemma get_M_shadow_root_put_M_document [simp]:
  "h ⊢ put_M_Document document_ptr setter v →_h h'
   ⟹ (⋀x. is_shadow_root_kind x ⟷ is_shadow_root_kind (setter (λ_. v) x))
   ⟹ (⋀x. getter (the (cast (((setter (λ_. v) (cast x)))))) = getter ((x)))
   ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter) h h'"
  apply(cases "cast shadow_root_ptr ≠ document_ptr ")
   apply(auto simp add: preserved_def is_shadow_root_kind_def DocumentMonad.put_M_defs
      get_ShadowRoot_def get_M_defs DocumentMonad.get_M_defs split: option.splits)[1]
  apply(auto simp add: preserved_def is_shadow_root_kind_def DocumentMonad.put_M_defs
      get_ShadowRoot_def get_M_defs DocumentMonad.get_M_defs split: option.splits)[1]
   apply(metis cast_Document2ShadowRoot_inv option.sel)
  apply(metis cast_Document2ShadowRoot_inv option.sel)
  done

lemma get_M_document_put_M_shadow_root_different_pointers [simp]:
  "document_ptr ≠ cast shadow_root_ptr
    ⟹ h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
    ⟹ preserved (get_M_Document document_ptr getter) h h'"
  by(auto simp add: put_M_defs get_M_defs DocumentMonad.get_M_defs preserved_def
      split: option.splits dest: get_heap_E)
lemma get_M_document_put_M_shadow_root [simp]:
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
   ⟹ (⋀x. is_shadow_root_kind x ⟹  getter ((cast (((setter (λ_. v) (the (cast x))))))) = getter ((x)))
   ⟹ preserved (get_M_Document document_ptr getter) h h'"
  apply(cases "document_ptr ≠ cast shadow_root_ptr ")
   apply(auto simp add: preserved_def is_document_kind_def put_Document_def put_ShadowRoot_def put_M_defs
      get_Document_def get_ShadowRoot_def DocumentMonad.get_M_defs ShadowRootMonad.get_M_defs
      split: option.splits Option.bind_splits)[1]
  apply(auto simp add: preserved_def is_document_kind_def put_Document_def put_ShadowRoot_def put_M_defs
      get_Document_def get_ShadowRoot_def DocumentMonad.get_M_defs ShadowRootMonad.get_M_defs
      split: option.splits Option.bind_splits)[1]
  using is_shadow_root_kind_Document_def apply force
  by (metis cast_Document2ShadowRoot_inv is_shadow_root_kind_Document_def option.distinct(1) option.sel)

lemma cast_shadow_root_child_nodes_document_disconnected_nodes [simp]:
  "RShadowRoot.child_nodes (the (cast (cast x(|disconnected_nodes := y|)))) = RShadowRoot.child_nodes x"
```

```
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RDocument.extend_def RDocument.trun
      split: option.splits)[1]
  by (metis RDocument.ext_inject RDocument.surjective RObject.ext_inject RShadowRoot.ext_inject
      RShadowRoot.surjective)
lemma cast_shadow_root_child_nodes_document_doctype [simp]:
  "RShadowRoot.child_nodes (the (cast (cast x(|doctype := y|)))) = RShadowRoot.child_nodes x"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RDocument.extend_def RDocument.trun
      split: option.splits)[1]
  by (metis RDocument.ext_inject RDocument.surjective RObject.ext_inject RShadowRoot.ext_inject RShadowRoot.surjec
lemma cast_shadow_root_child_nodes_document_document_element [simp]:
  "RShadowRoot.child_nodes (the (cast (cast x(|document_element := y|)))) = RShadowRoot.child_nodes x"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RDocument.extend_def RDocument.trun
      split: option.splits)[1]
  by (metis RDocument.ext_inject RDocument.surjective RObject.ext_inject RShadowRoot.ext_inject
      RShadowRoot.surjective)

lemma cast_shadow_root_mode_document_disconnected_nodes [simp]:
  "RShadowRoot.mode (the (cast (cast x(|disconnected_nodes := y|)))) = RShadowRoot.mode x"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RDocument.extend_def
      RDocument.truncate_def split: option.splits)[1]
  by (metis RDocument.ext_inject RDocument.surjective RObject.ext_inject RShadowRoot.ext_inject
      RShadowRoot.surjective)
lemma cast_shadow_root_mode_document_doctype [simp]:
  "RShadowRoot.mode (the (cast (cast x(|doctype := y|)))) = RShadowRoot.mode x"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RDocument.extend_def RDocument.trun
      split: option.splits)[1]
  by (metis RDocument.ext_inject RDocument.surjective RObject.ext_inject RShadowRoot.ext_inject RShadowRoot.surjec
lemma cast_shadow_root_mode_document_document_element [simp]:
  "RShadowRoot.mode (the (cast (cast x(|document_element := y|)))) = RShadowRoot.mode x"
  apply(auto simp add: cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def RDocument.extend_def RDocument.trun
      split: option.splits)[1]
  by (metis RDocument.ext_inject RDocument.surjective RObject.ext_inject RShadowRoot.ext_inject RShadowRoot.surjec

lemma cast_document_disconnected_nodes_shadow_root_child_nodes [simp]:
  "is_shadow_root_kind x ⟹
disconnected_nodes (cast (the (cast x)(|RShadowRoot.child_nodes := arg|))) = disconnected_nodes x"
  by(auto simp add: is_shadow_root_kind_def cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def
      RDocument.extend_def RDocument.truncate_def split: option.splits sum.splits)
lemma cast_document_doctype_shadow_root_child_nodes [simp]:
  "is_shadow_root_kind x ⟹ doctype (cast (the (cast x)(|RShadowRoot.child_nodes := arg|))) = doctype x"
  by(auto simp add: is_shadow_root_kind_def cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def
      RDocument.extend_def RDocument.truncate_def split: option.splits sum.splits)
lemma cast_document_document_element_shadow_root_child_nodes [simp]:
  "is_shadow_root_kind x ⟹
document_element (cast (the (cast x)(|RShadowRoot.child_nodes := arg|))) = document_element x"
  by(auto simp add: is_shadow_root_kind_def cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def
      RDocument.extend_def RDocument.truncate_def split: option.splits sum.splits)
lemma cast_document_disconnected_nodes_shadow_root_mode [simp]:
  "is_shadow_root_kind x ⟹
disconnected_nodes (cast (the (cast x)(|RShadowRoot.mode := arg|))) = disconnected_nodes x"
  by(auto simp add: is_shadow_root_kind_def cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def
      RDocument.extend_def RDocument.truncate_def split: option.splits sum.splits)
lemma cast_document_doctype_shadow_root_mode [simp]:
  "is_shadow_root_kind x ⟹
doctype (cast (the (cast x)(|RShadowRoot.mode := arg|))) = doctype x"
  by(auto simp add: is_shadow_root_kind_def cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def
      RDocument.extend_def RDocument.truncate_def split: option.splits sum.splits)
lemma cast_document_document_element_shadow_root_mode [simp]:
  "is_shadow_root_kind x ⟹
document_element (cast (the (cast x)(|RShadowRoot.mode := arg|))) = document_element x"
  by(auto simp add: is_shadow_root_kind_def cast_Document2ShadowRoot_def cast_ShadowRoot2Document_def
      RDocument.extend_def RDocument.truncate_def split: option.splits sum.splits)
```

```
lemma new_element_get_M_ShadowRoot:
  "h ⊢ new_element →_h h' ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
  by(auto simp add: new_element_def get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_character_data_get_M_ShadowRoot:
  "h ⊢ new_character_data →_h h' ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
  by(auto simp add: new_character_data_def get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_document_get_M_ShadowRoot:
  "h ⊢ new_document →_r new_document_ptr ⟹ h ⊢ new_document →_h h'
    ⟹ cast ptr ≠ new_document_ptr ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
  by(auto simp add: new_document_def get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)


definition delete_ShadowRoot_M :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog" where
  "delete_ShadowRoot_M shadow_root_ptr = do {
    h ← get_heap;
    (case delete_ShadowRoot shadow_root_ptr h of
      Some h ⇒ return_heap h |
      None ⇒ error HierarchyRequestError)
  }"
adhoc_overloading delete_M ⇌ delete_ShadowRoot_M

lemma delete_ShadowRoot_M_ok [simp]:
  assumes "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  shows "h ⊢ ok (delete_ShadowRoot_M shadow_root_ptr)"
  using assms
  by(auto simp add: delete_ShadowRoot_M_def delete_ShadowRoot_def delete_Object_def split: prod.splits)

lemma delete_ShadowRoot_M_ptr_in_heap:
  assumes "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  using assms
  by(auto simp add: delete_ShadowRoot_M_def delete_ShadowRoot_def delete_Object_def split: if_splits)

lemma delete_ShadowRoot_M_ptr_not_in_heap:
  assumes "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'"
  shows "shadow_root_ptr |∉| shadow_root_ptr_kinds h'"
  using assms
  by(auto simp add: delete_ShadowRoot_M_def delete_ShadowRoot_def delete_Object_def shadow_root_ptr_kinds_def
      document_ptr_kinds_def object_ptr_kinds_def split: if_splits)

lemma delete_shadow_root_pointers:
  assumes "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h' |∪| {|cast shadow_root_ptr|}"
  using assms
  by(auto simp add: delete_ShadowRoot_M_def delete_ShadowRoot_def delete_Object_def shadow_root_ptr_kinds_def
      document_ptr_kinds_def object_ptr_kinds_def split: if_splits)

lemma delete_shadow_root_get_M_Object:
  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹ ptr ≠ cast shadow_root_ptr ⟹
preserved (get_M_Object ptr getter) h h'"
  by(auto simp add: delete_ShadowRoot_M_def delete_Object_def ObjectMonad.get_M_defs preserved_def
      split: prod.splits option.splits if_splits elim!: bind_returns_heap_E)
lemma delete_shadow_root_get_M_Node:
  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹ preserved (get_M_Node ptr getter) h h'"
  by(auto simp add: delete_ShadowRoot_M_def delete_Object_def NodeMonad.get_M_defs ObjectMonad.get_M_defs
      preserved_def
      split: prod.splits option.splits if_splits elim!: bind_returns_heap_E)
```

```
lemma delete_shadow_root_get_M_Element:
  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹ preserved (get_M_Element ptr getter) h h'"
  by(auto simp add: delete_ShadowRoot_M_def delete_Object_def ElementMonad.get_M_defs NodeMonad.get_M_defs
      ObjectMonad.get_M_defs preserved_def
      split: prod.splits option.splits if_splits elim!: bind_returns_heap_E)
lemma delete_shadow_root_get_M_CharacterData:
  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹ preserved (get_M_CharacterData ptr getter) h h'"
  by(auto simp add: delete_ShadowRoot_M_def delete_Object_def CharacterDataMonad.get_M_defs NodeMonad.get_M_defs
      ObjectMonad.get_M_defs preserved_def
      split: prod.splits option.splits if_splits elim!: bind_returns_heap_E)
lemma delete_shadow_root_get_M_Document:
  "cast shadow_root_ptr ≠ ptr ⟹ h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹ preserved (get_M_Document
ptr getter) h h'"
  by(auto simp add: delete_ShadowRoot_M_def delete_Object_def DocumentMonad.get_M_defs ObjectMonad.get_M_defs
      preserved_def
      split: prod.splits option.splits if_splits elim!: bind_returns_heap_E)
lemma delete_shadow_root_get_M_ShadowRoot:
  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹ shadow_root_ptr ≠ shadow_root_ptr' ⟹ preserved
(get_M_ShadowRoot shadow_root_ptr' getter) h h'"
  by(auto simp add: delete_ShadowRoot_M_def delete_Object_def get_M_defs ObjectMonad.get_M_defs preserved_def
      split: prod.splits option.splits if_splits elim!: bind_returns_heap_E)
```

2.2.1 new_M

```
definition new_ShadowRoot_M :: "(_, (_) shadow_root_ptr) dom_prog"
  where
    "new_ShadowRoot_M = do {
      h ← get_heap;
      (new_ptr, h') ← return (new_ShadowRoot h);
      return_heap h';
      return new_ptr
    }"

lemma new_ShadowRoot_M_ok [simp]:
  "h ⊢ ok new_ShadowRoot_M"
  by(auto simp add: new_ShadowRoot_M_def split: prod.splits)

lemma new_ShadowRoot_M_ptr_in_heap:
  assumes "h ⊢ new_ShadowRoot_M →_h h'"
    and "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "new_shadow_root_ptr |∈| shadow_root_ptr_kinds h'"
  using assms
  unfolding new_ShadowRoot_M_def
  by(auto simp add: new_ShadowRoot_M_def new_ShadowRoot_def Let_def put_ShadowRoot_ptr_in_heap is_OK_returns_result_I
      elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_ShadowRoot_M_ptr_not_in_heap:
  assumes "h ⊢ new_ShadowRoot_M →_h h'"
    and "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "new_shadow_root_ptr |∉| shadow_root_ptr_kinds h"
  using assms new_ShadowRoot_ptr_not_in_heap
  by(auto simp add: new_ShadowRoot_M_def split: prod.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_ShadowRoot_M_new_ptr:
  assumes "h ⊢ new_ShadowRoot_M →_h h'"
    and "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
  using assms new_ShadowRoot_new_ptr
  by(auto simp add: new_ShadowRoot_M_def split: prod.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_ShadowRoot_M_is_shadow_root_ptr:
  assumes "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "is_shadow_root_ptr new_shadow_root_ptr"
```

```
  using assms new_ShadowRoot_is_shadow_root_ptr
  by(auto simp add: new_ShadowRoot_M_def elim!: bind_returns_result_E split: prod.splits)

lemma new_shadow_root_mode:
  assumes "h ⊢ new_ShadowRoot_M →_h h'"
  assumes "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "h' ⊢ get_M new_shadow_root_ptr mode →_r Open"
  using assms
  by(auto simp add: get_M_defs new_ShadowRoot_M_def new_ShadowRoot_def Let_def
      split: option.splits prod.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_shadow_root_children:
  assumes "h ⊢ new_ShadowRoot_M →_h h'"
  assumes "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "h' ⊢ get_M new_shadow_root_ptr child_nodes →_r []"
  using assms
  by(auto simp add: get_M_defs new_ShadowRoot_M_def new_ShadowRoot_def Let_def
      split: option.splits prod.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_shadow_root_disconnected_nodes:
  assumes "h ⊢ new_ShadowRoot_M →_h h'"
  assumes "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "h' ⊢ get_M (cast_shadow_root_ptr2document_ptr new_shadow_root_ptr) disconnected_nodes →_r []"
  using assms
  by(auto simp add: DocumentMonad.get_M_defs put_M_defs put_ShadowRoot_def new_ShadowRoot_M_def new_ShadowRoot_def
Let_def
      cast_shadow_root_ptr2document_ptr_def cast_ShadowRoot2Document_def RDocument.extend_def RDocument.truncate_def
      split: option.splits prod.splits elim!: bind_returns_result_E bind_returns_heap_E)

lemma new_shadow_root_get_M_Object:
  "h ⊢ new_ShadowRoot_M →_h h' ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
    ⟹ ptr ≠ cast new_shadow_root_ptr ⟹ preserved (get_M_Object ptr getter) h h'"
  by(auto simp add: new_ShadowRoot_M_def ObjectMonad.get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)
lemma new_shadow_root_get_M_Node:
  "h ⊢ new_ShadowRoot_M →_h h' ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
    ⟹ preserved (get_M_Node ptr getter) h h'"
  by(auto simp add: new_ShadowRoot_M_def NodeMonad.get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)
lemma new_shadow_root_get_M_Element:
  "h ⊢ new_ShadowRoot_M →_h h' ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
    ⟹ preserved (get_M_Element ptr getter) h h'"
  by(auto simp add: new_ShadowRoot_M_def ElementMonad.get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)
lemma new_shadow_root_get_M_CharacterData:
  "h ⊢ new_ShadowRoot_M →_h h' ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
    ⟹ preserved (get_M_CharacterData ptr getter) h h'"
  by(auto simp add: new_ShadowRoot_M_def CharacterDataMonad.get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)
lemma new_shadow_root_get_M_Document:
  "h ⊢ new_ShadowRoot_M →_h h'
    ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ ptr ≠ cast new_shadow_root_ptr
    ⟹ preserved (get_M_Document ptr getter) h h'"
  by(auto simp add: new_ShadowRoot_M_def DocumentMonad.get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)
lemma new_shadow_root_get_M_ShadowRoot:
  "h ⊢ new_ShadowRoot_M →_h h'
    ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ ptr ≠ new_shadow_root_ptr
    ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
  by(auto simp add: new_ShadowRoot_M_def get_M_defs preserved_def
      split: prod.splits option.splits elim!: bind_returns_result_E bind_returns_heap_E)
```

2.2.2 modified heaps

```
lemma shadow_root_get_put_1 [simp]: "get_ShadowRoot shadow_root_ptr (put_Object ptr obj h) =
(if ptr = cast shadow_root_ptr then cast obj else get shadow_root_ptr h)"
  by(auto simp add: get_ShadowRoot_def split: option.splits Option.bind_splits)

lemma shadow_root_ptr_kinds_new [simp]: "shadow_root_ptr_kinds (put_Object ptr obj h) =
shadow_root_ptr_kinds h |∪| (if is_shadow_root_ptr_kind ptr then {|the (cast ptr)|} else {||})"
  by(auto simp add: shadow_root_ptr_kinds_def is_document_ptr_kind_def split: option.splits)

lemma type_wf_put_I:
  assumes "type_wf h"
  assumes "DocumentClass.type_wf (put_Object ptr obj h)"
  assumes "is_shadow_root_ptr_kind ptr ⟹ is_shadow_root_kind obj"
  shows "type_wf (put_Object ptr obj h)"
  using assms
  by(auto simp add: type_wf_defs is_shadow_root_kind_def split: option.splits)

lemma type_wf_put_ptr_not_in_heap_E:
  assumes "type_wf (put_Object ptr obj h)"
  assumes "ptr |∉| object_ptr_kinds h"
  shows "type_wf h"
  using assms
  by (metis (no_types, opaque_lifting) DocumentMonad.type_wf_put_ptr_not_in_heap_E ObjectClass.get_Object_type_wf
      ObjectMonad.type_wf_put_ptr_not_in_heap_E ShadowRootClass.type_wf_Object ShadowRootClass.type_wf_defs
      document_ptr_kinds_commutes get_ShadowRoot_def get_document_ptr_simp get_object_ptr_simp2
      shadow_root_ptr_kinds_commutes)


lemma type_wf_put_ptr_in_heap_E:
  assumes "type_wf (put_Object ptr obj h)"
  assumes "ptr |∈| object_ptr_kinds h"
  assumes "DocumentClass.type_wf h"
  assumes "is_shadow_root_ptr_kind ptr ⟹ is_shadow_root_kind (the (get ptr h))"
  shows "type_wf h"
  using assms
  apply(auto simp add: type_wf_defs elim!: DocumentMonad.type_wf_put_ptr_in_heap_E
      split: option.splits if_splits)[1]
  by (metis (no_types, opaque_lifting) DocumentClass.l_get_Object_lemmas_axioms assms(2) bind.bind_lunit
      cast_Document2ShadowRoot_inv cast_Object2Document_inv get_Document_def get_ShadowRoot_def l_get_Object_lemmas.get
option.collapse)
```

2.2.3 type_wf

```
lemma new_element_type_wf_preserved [simp]:
  assumes "h ⊢ new_element →_h h'"
  shows "type_wf h = type_wf h'"
proof -
  obtain new_element_ptr where "h ⊢ new_element →_r new_element_ptr"
    using assms
    by (meson is_OK_returns_heap_I is_OK_returns_result_E)
  with assms have "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_element_ptr|}"
    using new_element_new_ptr by auto
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)

  with assms show ?thesis
    by(auto simp add: ElementMonad.new_element_def type_wf_defs Let_def elim!: bind_returns_heap_E
        split: prod.splits)
qed

lemma put_M_Element_tag_name_type_wf_preserved [simp]:
  assumes "h ⊢ put_M element_ptr tag_name_update v →_h h'"
```

```
  shows "type_wf h = type_wf h'"
proof -
  have "object_ptr_kinds h = object_ptr_kinds h'"
    using writes_singleton assms object_ptr_kinds_preserved unfolding all_args_def by fastforce
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  with assms show ?thesis
    by(auto simp add: ElementMonad.put_M_defs type_wf_defs)
qed
lemma put_M_Element_child_nodes_type_wf_preserved [simp]:
  assumes "h ⊢ put_M element_ptr RElement.child_nodes_update v →_h h'"
  shows "type_wf h = type_wf h'"
proof -
  have "object_ptr_kinds h = object_ptr_kinds h'"
    using writes_singleton assms object_ptr_kinds_preserved unfolding all_args_def by fastforce
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  with assms show ?thesis
    by(auto simp add: ElementMonad.put_M_defs type_wf_defs)
qed
lemma put_M_Element_attrs_type_wf_preserved [simp]:
  assumes "h ⊢ put_M element_ptr attrs_update v →_h h'"
  shows "type_wf h = type_wf h'"
proof -
  have "object_ptr_kinds h = object_ptr_kinds h'"
    using writes_singleton assms object_ptr_kinds_preserved unfolding all_args_def by fastforce
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  with assms show ?thesis
    by(auto simp add: ElementMonad.put_M_defs type_wf_defs)
qed
lemma put_M_Element_shadow_root_opt_type_wf_preserved [simp]:
  assumes "h ⊢ put_M element_ptr shadow_root_opt_update v →_h h'"
  shows "type_wf h = type_wf h'"
proof -
  have "object_ptr_kinds h = object_ptr_kinds h'"
    using writes_singleton assms object_ptr_kinds_preserved unfolding all_args_def by fastforce
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  with assms show ?thesis
    by(auto simp add: ElementMonad.put_M_defs type_wf_defs)
qed

lemma new_character_data_type_wf_preserved [simp]:
  assumes "h ⊢ new_character_data →_h h'"
  shows "type_wf h = type_wf h'"
proof -
  obtain new_character_data_ptr where "h ⊢ new_character_data →_r new_character_data_ptr"
    using assms
    by (meson is_OK_returns_heap_I is_OK_returns_result_E)
  with assms have "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    using new_character_data_new_ptr by auto
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  with assms show ?thesis
    by(auto simp add: CharacterDataMonad.new_character_data_def type_wf_defs Let_def
        elim!: bind_returns_heap_E split: prod.splits)
qed
lemma put_M_CharacterData_val_type_wf_preserved [simp]:
  assumes "h ⊢ put_M character_data_ptr val_update v →_h h'"
  shows "type_wf h = type_wf h'"
proof -
  have "object_ptr_kinds h = object_ptr_kinds h'"
```

```
    using writes_singleton assms object_ptr_kinds_preserved unfolding all_args_def by fastforce
  then have "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  with assms show ?thesis
    by(auto simp add: CharacterDataMonad.put_M_defs type_wf_defs)
qed

lemma new_document_type_wf_preserved [simp]:
  "h ⊢ new_document →h h' ⟹ type_wf h = type_wf h'"
  apply(auto simp add: new_document_def new_Document_def Let_def put_Document_def
      type_wf_Document
      type_wf_CharacterData  type_wf_Element
      type_wf_Node type_wf_Object
      is_node_ptr_kind_none
      elim!: bind_returns_heap_E type_wf_put_ptr_not_in_heap_E
      intro!: type_wf_put_I DocumentMonad.type_wf_put_I ElementMonad.type_wf_put_I CharacterDataMonad.type_wf_put_I
      NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I
      split: if_splits)[1]
     apply(auto simp add: type_wf_defs ElementClass.type_wf_defs CharacterDataClass.type_wf_defs
      NodeClass.type_wf_defs ObjectClass.type_wf_defs is_document_kind_def
      split: option.splits)[1]
    apply (metis fMax_finsert fimage_is_fempty new_Document_def new_Document_ptr_not_in_heap)
   apply(auto simp add: type_wf_defs ElementClass.type_wf_defs CharacterDataClass.type_wf_defs
      NodeClass.type_wf_defs ObjectClass.type_wf_defs is_document_kind_def
      split: option.splits)[1]
  apply(metis Suc_n_not_le_n document_ptr.sel(1) document_ptrs_def fMax_ge ffmember_filter fimage_eqI is_document_
  done

lemma put_M_Document_doctype_type_wf_preserved [simp]:
  "h ⊢ put_M_Document document_ptr doctype_update v →h h' ⟹ type_wf h = type_wf h'"
  apply(auto simp add: DocumentMonad.put_M_defs put_Document_def dest!: get_heap_E
      elim!: bind_returns_heap_E2
      intro!: type_wf_put_I DocumentMonad.type_wf_put_I CharacterDataMonad.type_wf_put_I
      ElementMonad.type_wf_put_I NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I)[1]
           apply(auto simp add: is_document_kind_def type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
          apply(auto simp add: is_document_kind_def type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_d
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
         apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_d
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
        apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_de
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
       apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_def
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
      apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
     apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
    apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
proof -
  fix x
  assume 0: "h ⊢ get_M_Document document_ptr id →r x"
    and 1: "h' = put (cast document_ptr) (cast_Document2Object (x(|doctype := v|))) h"
    and 2: "ShadowRootClass.type_wf h"
```

```
    and 3: "is_shadow_root_ptr_kind document_ptr"
  obtain shadow_root_ptr where shadow_root_ptr: "document_ptr = cast shadow_root_ptr" and
    "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
    by (metis "0" "3" DocumentMonad.get_M_ptr_in_heap is_OK_returns_result_I
        is_shadow_root_ptr_kind_obtains shadow_root_ptr_kinds_commutes)

  then have "is_shadow_root_kind x"
    using 0 2
    apply(auto simp add: is_document_kind_def type_wf_defs is_shadow_root_kind_def get_ShadowRoot_def
        split: option.splits Option.bind_splits)[1]
    by (metis (no_types, lifting) DocumentMonad.get_M_defs get_heap_returns_result
        id_apply option.simps(5) return_returns_result)


  then show "∃y. cast y = x(|doctype := v|)"
    using cast_Document2ShadowRoot_none is_shadow_root_kind_doctype is_shadow_root_kind_Document_def by blast
next
  fix x
  assume 0: "h ⊢ get_M_Document document_ptr id →_r x"
    and 1: "h' = put (cast document_ptr) (cast_Document2Object (x(|doctype := v|))) h"
    and 2: "ShadowRootClass.type_wf (put (cast document_ptr) (cast_Document2Object (x(|doctype := v|))) h)"
  have 3: "⋀document_ptr'. document_ptr' ≠ document_ptr ⟹ get_Object (cast document_ptr') h = get_Object
(cast document_ptr') h'"
    by (simp add: "1")
  have "document_ptr |∈| document_ptr_kinds h"
    by (meson "0" DocumentMonad.get_M_ptr_in_heap is_OK_returns_result_I)
  show "ShadowRootClass.type_wf h"
  proof (cases "is_shadow_root_ptr_kind document_ptr")
    case True
    then obtain shadow_root_ptr where shadow_root_ptr: "document_ptr = cast shadow_root_ptr"
      using is_shadow_root_ptr_kind_obtains by blast
    then
    have "is_shadow_root_kind (x(|doctype := v|))"
      using 2 True
      by(simp add: type_wf_defs is_shadow_root_kind_Document_def split: if_splits option.splits)
    then
    have "is_shadow_root_kind x"
      using is_shadow_root_kind_doctype by blast
    then
    have "is_shadow_root_kind (the (get_Object (cast document_ptr) h))"
      using 0
      by(auto simp add: DocumentMonad.a_get_M_def get_Document_def get_Object_def is_shadow_root_kind_def
          split: option.splits Option.bind_splits)
    show ?thesis
      using 0 2 <is_shadow_root_kind x> shadow_root_ptr
      by(auto simp add: DocumentMonad.a_get_M_def get_ShadowRoot_def is_shadow_root_kind_def
          is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
          NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
          CharacterDataClass.type_wf_defs split: option.splits if_splits)
  next
    case False
    then show ?thesis
      using 0 1 2
      by(auto simp add: DocumentMonad.a_get_M_def get_ShadowRoot_def is_shadow_root_kind_def
          is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
          NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
          CharacterDataClass.type_wf_defs split: option.splits if_splits)
  qed
qed

lemma put_M_Document_document_element_type_wf_preserved [simp]:
  assumes "h ⊢ put_M_Document document_ptr document_element_update v →_h h'"
  shows "type_wf h = type_wf h'"
```

```
  using assms
  apply(auto simp add: DocumentMonad.put_M_defs put_Document_def dest!: get_heap_E
      elim!: bind_returns_heap_E2
      intro!: type_wf_put_I DocumentMonad.type_wf_put_I CharacterDataMonad.type_wf_put_I
      ElementMonad.type_wf_put_I NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I)[1]
           apply(auto simp add: is_document_kind_def type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
          apply(auto simp add: is_document_kind_def type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_d
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
         apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_d
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
        apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_de
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
       apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_def
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
      apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
     apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
    apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
proof -
  fix x
  assume 0: "h ⊢ get_M_Document document_ptr id →_r x"
    and 1: "h' = put (cast document_ptr) (cast_Document2Object (x(|document_element := v|))) h"
    and 2: "ShadowRootClass.type_wf h"
    and 3: "is_shadow_root_ptr_kind document_ptr"
  obtain shadow_root_ptr where shadow_root_ptr: "document_ptr = cast shadow_root_ptr" and
    "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
    by (metis "0" "3" DocumentMonad.get_M_ptr_in_heap is_OK_returns_result_I
        is_shadow_root_ptr_kind_obtains shadow_root_ptr_kinds_commutes)

  then have "is_shadow_root_kind x"
    using 0 2
    apply(auto simp add: is_document_kind_def type_wf_defs is_shadow_root_kind_def get_ShadowRoot_def
        split: option.splits Option.bind_splits)[1]
    by (metis (no_types, lifting) DocumentMonad.get_M_defs get_heap_returns_result id_def
        option.simps(5) return_returns_result)

  then show "∃y. cast y = x(|document_element := v|)"
    using cast_Document2ShadowRoot_none is_shadow_root_kind_document_element is_shadow_root_kind_Document_def
    by blast
next
  fix x
  assume 0: "h ⊢ get_M_Document document_ptr id →_r x"
    and 1: "h' = put (cast document_ptr) (cast_Document2Object (x(|document_element := v|))) h"
    and 2: "ShadowRootClass.type_wf (put (cast document_ptr) (cast_Document2Object (x(|document_element :=
v|))) h)"
  have 3: "⋀document_ptr'. document_ptr' ≠ document_ptr ⟹ get_Object (cast document_ptr') h = get_Object
(cast document_ptr') h'"
    by (simp add: "1")
  have "document_ptr |∈| document_ptr_kinds h"
    by (meson "0" DocumentMonad.get_M_ptr_in_heap is_OK_returns_result_I)
  show "ShadowRootClass.type_wf h"
  proof (cases "is_shadow_root_ptr_kind document_ptr")
```

```
    case True
    then obtain shadow_root_ptr where shadow_root_ptr: "document_ptr = cast shadow_root_ptr"
      using is_shadow_root_ptr_kind_obtains by blast
    then
    have "is_shadow_root_kind (x(|document_element := v|))"
      using 2 True
      by(simp add: type_wf_defs is_shadow_root_kind_Document_def split: if_splits option.splits)
    then
    have "is_shadow_root_kind x"
      using is_shadow_root_kind_document_element by blast
    then
    have "is_shadow_root_kind (the (get_Object (cast document_ptr) h))"
      using 0
      by(auto simp add: DocumentMonad.a_get_M_def get_Document_def get_Object_def is_shadow_root_kind_def
          split: option.splits Option.bind_splits)
    show ?thesis
      using 0 2 ‹is_shadow_root_kind x› shadow_root_ptr
      by(auto simp add: DocumentMonad.a_get_M_def get_ShadowRoot_def is_shadow_root_kind_def
          is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
          NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
          CharacterDataClass.type_wf_defs split: option.splits if_splits)
  next
    case False
    then show ?thesis
      using 0 1 2
      by(auto simp add: DocumentMonad.a_get_M_def get_ShadowRoot_def is_shadow_root_kind_def
          is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
          NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
          CharacterDataClass.type_wf_defs split: option.splits if_splits)
  qed
qed

lemma put_M_Document_disconnected_nodes_type_wf_preserved [simp]:
  assumes "h ⊢ put_M_Document document_ptr disconnected_nodes_update v →_h h'"
  shows "type_wf h = type_wf h'"

  using assms
  apply(auto simp add: DocumentMonad.put_M_defs put_Document_def dest!: get_heap_E
      elim!: bind_returns_heap_E2
      intro!: type_wf_put_I DocumentMonad.type_wf_put_I CharacterDataMonad.type_wf_put_I
      ElementMonad.type_wf_put_I NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I)[1]
           apply(auto simp add: is_document_kind_def type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
          apply(auto simp add: is_document_kind_def type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_d
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
         apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_d
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
        apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_de
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
       apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_def
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
      apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
     apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
    apply(auto simp add: is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
```

```
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
      CharacterDataClass.type_wf_defs split: option.splits)[1]
proof -
  fix x
  assume 0: "h ⊢ get_M_Document document_ptr id →_r x"
    and 1: "h' = put (cast document_ptr) (cast_Document2Object (x(|disconnected_nodes := v|))) h"
    and 2: "ShadowRootClass.type_wf h"
    and 3: "is_shadow_root_ptr_kind document_ptr"
  obtain shadow_root_ptr where shadow_root_ptr: "document_ptr = cast shadow_root_ptr" and
    "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
    by (metis "0" "3" DocumentMonad.get_M_ptr_in_heap is_OK_returns_result_I is_shadow_root_ptr_kind_obtains
        shadow_root_ptr_kinds_commutes)

  then have "is_shadow_root_kind x"
    using 0 2
    apply(auto simp add: is_document_kind_def type_wf_defs is_shadow_root_kind_def get_ShadowRoot_def
        split: option.splits Option.bind_splits)[1]
    by (metis (no_types, lifting) DocumentMonad.get_M_defs get_heap_returns_result
        id_def option.simps(5) return_returns_result)

  then show "∃y. cast y = x(|disconnected_nodes := v|)"
    using cast_Document2ShadowRoot_none is_shadow_root_kind_disconnected_nodes is_shadow_root_kind_Document_def
    by blast
next
  fix x
  assume 0: "h ⊢ get_M_Document document_ptr id →_r x"
    and 1: "h' = put (cast document_ptr) (cast_Document2Object (x(|disconnected_nodes := v|))) h"
    and 2: "ShadowRootClass.type_wf (put (cast document_ptr) (cast_Document2Object (x(|disconnected_nodes
:= v|))) h)"
  have 3: "⋀document_ptr'. document_ptr' ≠ document_ptr ⟹ get_Object (cast document_ptr') h = get_Object
(cast document_ptr') h'"
    by (simp add: "1")
  have "document_ptr |∈| document_ptr_kinds h"
    by (meson "0" DocumentMonad.get_M_ptr_in_heap is_OK_returns_result_I)
  show "ShadowRootClass.type_wf h"
  proof (cases "is_shadow_root_ptr_kind document_ptr")
    case True
    then obtain shadow_root_ptr where shadow_root_ptr: "document_ptr = cast shadow_root_ptr"
      using is_shadow_root_ptr_kind_obtains by blast
    then
    have "is_shadow_root_kind (x(|disconnected_nodes := v|))"
      using 2 True
      by(simp add: type_wf_defs is_shadow_root_kind_Document_def split: if_splits option.splits)
    then
    have "is_shadow_root_kind x"
      using is_shadow_root_kind_disconnected_nodes by blast
    then
    have "is_shadow_root_kind (the (get_Object (cast document_ptr) h))"
      using 0
      by(auto simp add: DocumentMonad.a_get_M_def get_Document_def get_Object_def is_shadow_root_kind_def
          split: option.splits Option.bind_splits)
    show ?thesis
      using 0 2 <is_shadow_root_kind x> shadow_root_ptr
      by(auto simp add: DocumentMonad.a_get_M_def get_ShadowRoot_def is_shadow_root_kind_def
          is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
          NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
          CharacterDataClass.type_wf_defs split: option.splits if_splits)
  next
    case False
    then show ?thesis
      using 0 1 2
      by(auto simp add: DocumentMonad.a_get_M_def get_ShadowRoot_def is_shadow_root_kind_def
          is_document_kind_def type_wf_defs  DocumentClass.type_wf_defs ElementClass.type_wf_defs
```

```
        NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs
        CharacterDataClass.type_wf_defs split: option.splits if_splits)
  qed
qed

lemma put_M_ShadowRoot_mode_type_wf_preserved [simp]:
  "h ⊢ put_M shadow_root_ptr mode_update v →_h h' ⟹ type_wf h = type_wf h'"
  by(auto simp add: get_M_defs get_ShadowRoot_def DocumentMonad.get_M_defs put_M_defs put_ShadowRoot_def
      put_Document_def dest!: get_heap_E  elim!: bind_returns_heap_E2 intro!: type_wf_put_I DocumentMonad.type_wf_p
      CharacterDataMonad.type_wf_put_I ElementMonad.type_wf_put_I NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I
      simp add: is_shadow_root_kind_def is_document_kind_def type_wf_defs  ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs CharacterDataClass.type_wf_defs
DocumentClass.type_wf_defs split: option.splits)[1]

lemma put_M_ShadowRoot_child_nodes_type_wf_preserved [simp]:
  "h ⊢ put_M shadow_root_ptr RShadowRoot.child_nodes_update v →_h h' ⟹ type_wf h = type_wf h'"
  by(auto simp add: get_M_defs get_ShadowRoot_def DocumentMonad.get_M_defs put_M_defs put_ShadowRoot_def
      put_Document_def dest!: get_heap_E  elim!: bind_returns_heap_E2 intro!: type_wf_put_I
      DocumentMonad.type_wf_put_I CharacterDataMonad.type_wf_put_I ElementMonad.type_wf_put_I
      NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I
      simp add: is_shadow_root_kind_def is_document_kind_def type_wf_defs  ElementClass.type_wf_defs
      NodeClass.type_wf_defs ElementMonad.get_M_defs ObjectClass.type_wf_defs CharacterDataClass.type_wf_defs
      DocumentClass.type_wf_defs split: option.splits)[1]

lemma shadow_root_ptr_kinds_small:
  assumes "⋀object_ptr. preserved (get_M_Object object_ptr RObject.nothing) h h'"
  shows "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
  by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def preserved_def
      object_ptr_kinds_preserved_small[OF assms])

lemma shadow_root_ptr_kinds_preserved:
  assumes "writes SW setter h h'"
  assumes "h ⊢ setter →_h h'"
  assumes "⋀h h'. ∀w ∈ SW. h ⊢ w →_h h' ⟶ (∀object_ptr. preserved (get_M_Object object_ptr RObject.nothing)
h h')"
  shows "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
  using writes_small_big[OF assms]
  apply(simp add: reflp_def transp_def preserved_def shadow_root_ptr_kinds_def document_ptr_kinds_def)
  by (metis assms object_ptr_kinds_preserved)


lemma new_shadow_root_known_ptr:
  assumes "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr"
  shows "known_ptr (cast new_shadow_root_ptr)"
  using assms
  apply(auto simp add: new_ShadowRoot_M_def new_ShadowRoot_def Let_def a_known_ptr_def
      elim!: bind_returns_result_E2 split: prod.splits)[1]
  using assms new_ShadowRoot_M_is_shadow_root_ptr by blast


lemma new_shadow_root_type_wf_preserved [simp]: "h ⊢ new_ShadowRoot_M →_h h' ⟹ type_wf h = type_wf h'"
  apply(auto simp add: new_ShadowRoot_M_def new_ShadowRoot_def Let_def put_ShadowRoot_def put_Document_def
      ShadowRootClass.type_wf_Document ShadowRootClass.type_wf_CharacterData  ShadowRootClass.type_wf_Element
      ShadowRootClass.type_wf_Node ShadowRootClass.type_wf_Object
      is_node_ptr_kind_none new_ShadowRoot_ptr_not_in_heap
      elim!: bind_returns_heap_E type_wf_put_ptr_not_in_heap_E
      intro!: type_wf_put_I DocumentMonad.type_wf_put_I ElementMonad.type_wf_put_I CharacterDataMonad.type_wf_put_I
      NodeMonad.type_wf_put_I ObjectMonad.type_wf_put_I
      split: if_splits)[1]
  by(auto simp add: type_wf_defs DocumentClass.type_wf_defs ElementClass.type_wf_defs CharacterDataClass.type_wf_d
      NodeClass.type_wf_defs ObjectClass.type_wf_defs is_shadow_root_kind_def is_document_kind_def
      split: option.splits)[1]
```

```
locale l_new_shadow_root = l_type_wf +
  assumes new_shadow_root_types_preserved: "h ⊢ new_ShadowRoot_M →_h h' ⟹ type_wf h = type_wf h'"

lemma new_shadow_root_is_l_new_shadow_root  [instances]: "l_new_shadow_root type_wf"
  using l_new_shadow_root.intro new_shadow_root_type_wf_preserved
  by blast

lemma type_wf_preserved_small:
  assumes "⋀object_ptr. preserved (get_M_Object object_ptr RObject.nothing) h h'"
  assumes "⋀node_ptr. preserved (get_M_Node node_ptr RNode.nothing) h h'"
  assumes "⋀element_ptr. preserved (get_M_Element element_ptr RElement.nothing) h h'"
  assumes "⋀character_data_ptr. preserved (get_M_CharacterData character_data_ptr RCharacterData.nothing)
h h'"
  assumes "⋀document_ptr. preserved (get_M_Document document_ptr RDocument.nothing) h h'"
  assumes "⋀shadow_root_ptr. preserved (get_M_ShadowRoot shadow_root_ptr RShadowRoot.nothing) h h'"
  shows "type_wf h = type_wf h'"
  using type_wf_preserved_small[OF assms(1) assms(2) assms(3) assms(4) assms(5)]
    allI[OF assms(6), of id, simplified] shadow_root_ptr_kinds_small[OF assms(1)]
  apply(auto simp add: type_wf_defs )[1]
   apply(auto simp add: preserved_def get_M_defs shadow_root_ptr_kinds_small[OF assms(1)]
      split: option.splits)[1]
   apply(force)
  apply(auto simp add: preserved_def get_M_defs shadow_root_ptr_kinds_small[OF assms(1)]
      split: option.splits)[1]
  apply(force)
  done

lemma type_wf_preserved:
  assumes "writes SW setter h h'"
  assumes "h ⊢ setter →_h h'"
  assumes "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ ∀object_ptr. preserved (get_M_Object object_ptr RObject.nothing)
h h'"
  assumes "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ ∀node_ptr. preserved (get_M_Node node_ptr RNode.nothing)
h h'"
  assumes "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ ∀element_ptr. preserved (get_M_Element element_ptr RElement.noth
h h'"
  assumes "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ ∀character_data_ptr. preserved (get_M_CharacterData character_da
RCharacterData.nothing) h h'"
  assumes "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ ∀document_ptr. preserved (get_M_Document document_ptr
RDocument.nothing) h h'"
  assumes "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ ∀shadow_root_ptr. preserved (get_M_ShadowRoot shadow_root_ptr
RShadowRoot.nothing) h h'"
  shows "type_wf h = type_wf h'"
proof -
  have "⋀h h' w. w ∈ SW ⟹ h ⊢ w →_h h' ⟹ type_wf h = type_wf h'"
    using assms type_wf_preserved_small by fast
  with assms(1) assms(2) show ?thesis
    apply(rule writes_small_big)
    by(auto simp add: reflp_def transp_def)
qed

lemma type_wf_drop: "type_wf h ⟹ type_wf (Heap (fmdrop ptr (the_heap h)))"
  apply(auto simp add: type_wf_defs)[1]
  using type_wf_drop
   apply blast
  by (metis (no_types, opaque_lifting) DocumentClass.type_wf_Object DocumentMonad.type_wf_drop
      ObjectClass.get_Object_type_wf document_ptr_kinds_commutes fmlookup_drop get_Document_def
      get_Object_def get_ShadowRoot_def heap.sel shadow_root_ptr_kinds_commutes)

lemma delete_shadow_root_type_wf_preserved [simp]:
  assumes "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'"
  assumes "type_wf h"
```

```
  shows "type_wf h'"
  using assms
  using type_wf_drop
  by(auto simp add: delete_ShadowRoot_M_def delete_ShadowRoot_def delete_Object_def split: if_splits)



lemma new_element_is_l_new_element [instances]:
  "l_new_element type_wf"
  using l_new_element.intro new_element_type_wf_preserved
  by blast

lemma new_character_data_is_l_new_character_data [instances]:
  "l_new_character_data type_wf"
  using l_new_character_data.intro new_character_data_type_wf_preserved
  by blast

lemma new_document_is_l_new_document [instances]:
  "l_new_document type_wf"
  using l_new_document.intro new_document_type_wf_preserved
  by blast
end
```

2.3 The Shadow DOM (Shadow_DOM)

```
theory Shadow_DOM
  imports
    "monads/ShadowRootMonad"
    Core_SC_DOM.Core_DOM
begin



abbreviation "safe_shadow_root_element_types ≡ {''article'', ''aside'', ''blockquote'', ''body'',
  ''div'', ''footer'', ''h1'', ''h2'', ''h3'', ''h4'', ''h5'', ''h6'', ''header'', ''main'',
  ''nav'', ''p'', ''section'', ''span''}"
```

2.3.1 Function Definitions

get_child_nodes

```
locale l_get_child_nodes_Shadow_DOM_defs =
  CD: l_get_child_nodes_Core_DOM_defs
begin
definition get_child_nodes_shadow_root_ptr :: "(_) shadow_root_ptr ⇒ unit
    ⇒ (_, (_) node_ptr list) dom_prog" where
  "get_child_nodes_shadow_root_ptr shadow_root_ptr _ = get_M shadow_root_ptr RShadowRoot.child_nodes"

definition a_get_child_nodes_tups :: "(((_) object_ptr ⇒ bool) × ((_) object_ptr ⇒ unit
    ⇒ (_, (_) node_ptr list) dom_prog)) list" where
  "a_get_child_nodes_tups ≡ [(is_shadow_root_ptr_object_ptr, get_child_nodes_shadow_root_ptr ∘ the ∘ cast)]"

definition a_get_child_nodes :: "(_) object_ptr ⇒ (_, (_) node_ptr list) dom_prog" where
  "a_get_child_nodes ptr = invoke (CD.a_get_child_nodes_tups @ a_get_child_nodes_tups) ptr ()"

definition a_get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" where
  "a_get_child_nodes_locs ptr ≡
    (if is_shadow_root_ptr_kind ptr
    then {preserved (get_M (the (cast ptr)) RShadowRoot.child_nodes)} else {}) ∪
    CD.a_get_child_nodes_locs ptr"

definition first_child :: "(_) object_ptr ⇒ (_, (_) node_ptr option) dom_prog"
  where
```

```
    "first_child ptr = do {
      children ← a_get_child_nodes ptr;
      return (case children of [] ⇒ None | child#_ ⇒ Some child)}"
end

global_interpretation l_get_child_nodes_Shadow_DOM_defs defines
  get_child_nodes = l_get_child_nodes_Shadow_DOM_defs.a_get_child_nodes and
  get_child_nodes_locs = l_get_child_nodes_Shadow_DOM_defs.a_get_child_nodes_locs
  .

locale l_get_child_nodes_Shadow_DOM =
  l_type_wf type_wf +
  l_known_ptr known_ptr +
  l_get_child_nodes_Shadow_DOM_defs +
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  CD: l_get_child_nodes_Core_DOM type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes_Core_DOM
  get_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes get_child_nodes_impl: "get_child_nodes = a_get_child_nodes"
  assumes get_child_nodes_locs_impl: "get_child_nodes_locs = a_get_child_nodes_locs"
begin
lemmas get_child_nodes_def = get_child_nodes_impl[unfolded a_get_child_nodes_def get_child_nodes_def]
lemmas get_child_nodes_locs_def = get_child_nodes_locs_impl[unfolded a_get_child_nodes_locs_def
    get_child_nodes_locs_def, folded CD.get_child_nodes_locs_impl]

lemma get_child_nodes_ok:
  assumes "known_ptr ptr"
  assumes "type_wf h"
  assumes "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_child_nodes ptr)"
  using assms[unfolded known_ptr_impl type_wf_impl]
  apply(auto simp add: get_child_nodes_def)[1]
  apply(split CD.get_child_nodes_splits, rule conjI)+
  using ShadowRootClass.type_wf_Document CD.get_child_nodes_ok CD.known_ptr_impl CD.type_wf_impl
   apply blast
  apply(auto simp add: CD.known_ptr_impl a_get_child_nodes_tups_def get_child_nodes_shadow_root_ptr_def
      get_M_ShadowRoot_ok
      dest!: known_ptr_new_shadow_root_ptr intro!: bind_is_OK_I2)[1]
  by(auto dest: get_M_ShadowRoot_ok split: option.splits)

lemma get_child_nodes_ptr_in_heap:
  assumes "h ⊢ get_child_nodes ptr →r children"
  shows "ptr |∈| object_ptr_kinds h"
  using assms
  by(auto simp add: get_child_nodes_def invoke_ptr_in_heap dest: is_OK_returns_result_I)

lemma get_child_nodes_pure [simp]:
  "pure (get_child_nodes ptr) h"
  apply (auto simp add: get_child_nodes_def a_get_child_nodes_tups_def)[1]
  apply(split CD.get_child_nodes_splits, rule conjI)+
   apply(simp)
  apply(split invoke_splits, rule conjI)+
   apply(simp)
  by(auto simp add: get_child_nodes_shadow_root_ptr_def intro!: bind_pure_I)
```

```
lemma get_child_nodes_reads: "reads (get_child_nodes_locs ptr) (get_child_nodes ptr) h h'"
  apply (simp add: get_child_nodes_def a_get_child_nodes_tups_def get_child_nodes_locs_def
      CD.get_child_nodes_locs_def)
  apply(split CD.get_child_nodes_splits, rule conjI)+
   apply(auto intro!: reads_subset[OF CD.get_child_nodes_reads[unfolded  CD.get_child_nodes_locs_def]]
      split: if_splits)[1]
  apply(split invoke_splits, rule conjI)+
   apply(auto)[1]
  apply(auto simp add: get_child_nodes_shadow_root_ptr_def
      intro: reads_subset[OF reads_singleton] reads_subset[OF check_in_heap_reads]
      intro!: reads_bind_pure reads_subset[OF return_reads] split: option.splits)[1]
  done
end

interpretation i_get_child_nodes?: l_get_child_nodes_Shadow_DOM type_wf known_ptr DocumentClass.type_wf
  DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
  by(simp add: l_get_child_nodes_Shadow_DOM_def l_get_child_nodes_Shadow_DOM_axioms_def instances)
declare l_get_child_nodes_Shadow_DOM_axioms [instances]

lemma get_child_nodes_is_l_get_child_nodes [instances]: "l_get_child_nodes type_wf known_ptr
    get_child_nodes get_child_nodes_locs"
  apply(auto simp add: l_get_child_nodes_def instances)[1]
  using get_child_nodes_reads get_child_nodes_ok get_child_nodes_ptr_in_heap get_child_nodes_pure
  by blast+
```

new_document

```
locale l_new_document_get_child_nodes_Shadow_DOM =
  CD: l_new_document_get_child_nodes_Core_DOM type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes_Core_DOM
  get_child_nodes_locs_Core_DOM
  + l_get_child_nodes_Shadow_DOM type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes
  get_child_nodes_locs get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and get_child_nodes_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_child_nodes_new_document:
  "ptr' ≠ cast new_document_ptr ⟹ h ⊢ new_document →r new_document_ptr
     ⟹ h ⊢ new_document →h h' ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
  apply(auto simp add: get_child_nodes_locs_def)[1]
  using CD.get_child_nodes_new_document
   apply (metis document_ptr_casts_commute3 empty_iff is_document_ptr_kind_none
      new_document_get_M_ShadowRoot option.case_eq_if shadow_root_ptr_casts_commute3 singletonD)
  by (simp add: CD.get_child_nodes_new_document)

lemma new_document_no_child_nodes:
  "h ⊢ new_document →r new_document_ptr ⟹ h ⊢ new_document →h h'
     ⟹ h' ⊢ get_child_nodes (cast new_document_ptr) →r []"
  apply(auto simp add: get_child_nodes_def)[1]
  apply(split CD.get_child_nodes_splits, rule conjI)+
  using CD.new_document_no_child_nodes apply auto[1]
  by(auto simp add: DocumentClass.a_known_ptr_def CD.known_ptr_impl known_ptr_def
      dest!: new_document_is_document_ptr)
end
interpretation i_new_document_get_child_nodes?:
  l_new_document_get_child_nodes_Shadow_DOM type_wf known_ptr get_child_nodes get_child_nodes_locs
  DocumentClass.type_wf DocumentClass.known_ptr Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
```

```
  by(unfold_locales)
declare l_new_document_get_child_nodes_Core_DOM_axioms[instances]

lemma new_document_get_child_nodes_is_l_new_document_get_child_nodes [instances]:
  "l_new_document_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"
  using new_document_is_l_new_document get_child_nodes_is_l_get_child_nodes
  apply(simp add: l_new_document_get_child_nodes_def l_new_document_get_child_nodes_axioms_def)
  using get_child_nodes_new_document new_document_no_child_nodes
  by fast
```

new_shadow_root

```
locale l_new_shadow_root_get_child_nodes_Shadow_DOM =
  l_get_child_nodes_Shadow_DOM type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes
  get_child_nodes_locs get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and get_child_nodes_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_child_nodes_new_shadow_root:
  "ptr' ≠ cast new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
     ⟹ h ⊢ new_ShadowRoot_M →_h h' ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
  apply(auto simp add: get_child_nodes_locs_def)[1]
   apply (metis document_ptr_casts_commute3 insert_absorb insert_not_empty is_document_ptr_kind_none
      new_shadow_root_get_M_ShadowRoot option.case_eq_if shadow_root_ptr_casts_commute3 singletonD)
  apply(auto simp add: CD.get_child_nodes_locs_def)[1]
  using new_shadow_root_get_M_Object apply blast
   apply (smt insertCI new_shadow_root_get_M_Element singleton_iff)
  apply (metis document_ptr_casts_commute3 empty_iff new_shadow_root_get_M_Document singletonD)
  done

lemma new_shadow_root_no_child_nodes:
  "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
     ⟹ h' ⊢ get_child_nodes (cast new_shadow_root_ptr) →_r []"
  apply(auto simp add: get_child_nodes_def )[1]
  apply(split CD.get_child_nodes_splits, rule conjI)+
   apply(auto simp add: CD.get_child_nodes_def CD.a_get_child_nodes_tups_def)[1]
   apply(split invoke_splits, rule conjI)+
  using NodeClass.a_known_ptr_def known_ptr_not_character_data_ptr known_ptr_not_document_ptr
    known_ptr_not_element_ptr local.CD.known_ptr_impl apply blast
     apply(auto simp add: is_document_ptr_def cast_shadow_root_ptr2document_ptr_def
      split: option.splits document_ptr.splits)[1]
    apply(auto simp add: is_character_data_ptr_def cast_shadow_root_ptr2document_ptr_def
      split: option.splits document_ptr.splits)[1]
   apply(auto simp add: is_element_ptr_def cast_shadow_root_ptr2document_ptr_def
      split: option.splits document_ptr.splits)[1]
  apply(auto simp add: a_get_child_nodes_tups_def)[1]
  apply(split invoke_splits, rule conjI)+
   apply(auto simp add: is_shadow_root_ptr_def split: shadow_root_ptr.splits
      dest!: new_ShadowRoot_M_is_shadow_root_ptr)[1]
  apply(auto intro!: bind_pure_returns_result_I)[1]
   apply(drule(1) new_ShadowRoot_M_ptr_in_heap)
   apply(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)[1]
   apply (metis check_in_heap_ptr_in_heap is_OK_returns_result_E old.unit.exhaust)
  using  new_shadow_root_children
  by (simp add: new_shadow_root_children get_child_nodes_shadow_root_ptr_def)
end
interpretation i_new_shadow_root_get_child_nodes?:
  l_new_shadow_root_get_child_nodes_Shadow_DOM type_wf known_ptr get_child_nodes get_child_nodes_locs
  DocumentClass.type_wf DocumentClass.known_ptr Core_DOM_Functions.get_child_nodes
```

```
  Core_DOM_Functions.get_child_nodes_locs
  by(unfold_locales)
declare l_new_shadow_root_get_child_nodes_Shadow_DOM_def[instances]

locale l_new_shadow_root_get_child_nodes = l_get_child_nodes +
  assumes get_child_nodes_new_shadow_root:
    "ptr' ≠ cast new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
     ⟹ h ⊢ new_ShadowRoot_M →_h h' ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
  assumes new_shadow_root_no_child_nodes:
    "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
     ⟹ h' ⊢ get_child_nodes (cast new_shadow_root_ptr) →_r []"

lemma new_shadow_root_get_child_nodes_is_l_new_shadow_root_get_child_nodes [instances]:
  "l_new_shadow_root_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"
  apply(simp add: l_new_shadow_root_get_child_nodes_def l_new_shadow_root_get_child_nodes_axioms_def instances)
  using get_child_nodes_new_shadow_root new_shadow_root_no_child_nodes
  by fast
```

new_element

```
locale l_new_element_get_child_nodes_Shadow_DOM =
  l_get_child_nodes_Shadow_DOM +
  l_new_element_get_child_nodes_Core_DOM type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes_Core_DOM get_child_n
begin
lemma get_child_nodes_new_element:
  "ptr' ≠ cast new_element_ptr ⟹ h ⊢ new_element →_r new_element_ptr ⟹ h ⊢ new_element →_h h'
     ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
  by (auto simp add: get_child_nodes_locs_def CD.get_child_nodes_locs_def new_element_get_M_Object new_element_get_M
      new_element_get_M_Document new_element_get_M_ShadowRoot split: prod.splits if_splits option.splits
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)

lemma new_element_no_child_nodes:
  "h ⊢ new_element →_r new_element_ptr ⟹ h ⊢ new_element →_h h'
   ⟹ h' ⊢ get_child_nodes (cast new_element_ptr) →_r []"
  apply(auto simp add: get_child_nodes_def a_get_child_nodes_tups_def
      split: prod.splits elim!: bind_returns_result_E bind_returns_heap_E)[1]
  apply(split CD.get_child_nodes_splits, rule conjI)+
  using local.new_element_no_child_nodes apply auto[1]
  apply(auto simp add: invoke_def)[1]
  using case_optionE apply fastforce
  apply(auto simp add: new_element_ptr_in_heap get_child_nodes_element_ptr_def check_in_heap_def
      new_element_child_nodes intro!: bind_pure_returns_result_I
      intro: new_element_is_element_ptr elim!: new_element_ptr_in_heap)[1]
proof -
  assume " h ⊢ new_element →_r new_element_ptr"
  assume "h ⊢ new_element →_h h'"
  assume "¬ is_shadow_root_ptr_object_ptr (cast_element_ptr2object_ptr new_element_ptr)"
  assume "¬ known_ptr_Core_DOM (cast_element_ptr2object_ptr new_element_ptr)"
  moreover
  have "known_ptr (cast new_element_ptr)"
    using new_element_is_element_ptr ‹h ⊢ new_element →_r new_element_ptr›
    by(auto simp add: known_ptr_impl ShadowRootClass.a_known_ptr_def DocumentClass.a_known_ptr_def
        CharacterDataClass.a_known_ptr_def ElementClass.a_known_ptr_def)
  ultimately show "False"
    by(simp add: known_ptr_impl CD.known_ptr_impl ShadowRootClass.a_known_ptr_def is_document_ptr_kind_none)
qed
end

interpretation i_new_element_get_child_nodes?:
  l_new_element_get_child_nodes_Shadow_DOM type_wf known_ptr DocumentClass.type_wf
  DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
  by(unfold_locales)
declare l_new_element_get_child_nodes_Shadow_DOM_axioms[instances]
```

```
lemma new_element_get_child_nodes_is_l_new_element_get_child_nodes [instances]:
  "l_new_element_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"
  using new_element_is_l_new_element get_child_nodes_is_l_get_child_nodes
  apply(auto simp add: l_new_element_get_child_nodes_def l_new_element_get_child_nodes_axioms_def)[1]
  using get_child_nodes_new_element new_element_no_child_nodes
  by fast+
```

delete_shadow_root

```
locale l_delete_shadow_root_get_child_nodes_Shadow_DOM =
  l_get_child_nodes_Shadow_DOM
begin
lemma get_child_nodes_delete_shadow_root:
  "ptr' ≠ cast shadow_root_ptr ⟹ h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹
r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
  by(auto simp add: get_child_nodes_locs_def CD.get_child_nodes_locs_def delete_shadow_root_get_M_Object
      delete_shadow_root_get_M_ShadowRoot delete_shadow_root_get_M_Document delete_shadow_root_get_M_Element
      split: if_splits  intro: is_shadow_root_ptr_kind_obtains
      intro: is_shadow_root_ptr_kind_obtains delete_shadow_root_get_M_ShadowRoot delete_shadow_root_get_M_Document
      simp add: shadow_root_ptr_casts_commute3 delete_shadow_root_get_M_Document
      intro!: delete_shadow_root_get_M_Document dest: document_ptr_casts_commute3
      split: option.splits)
end

locale l_delete_shadow_root_get_child_nodes = l_get_child_nodes_defs +
  assumes get_child_nodes_delete_shadow_root:
    "ptr' ≠ cast shadow_root_ptr ⟹ h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h' ⟹
r ∈ get_child_nodes_locs ptr' ⟹ r h h'"

interpretation l_delete_shadow_root_get_child_nodes_Shadow_DOM type_wf known_ptr DocumentClass.type_wf
  DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
  by(auto simp add: l_delete_shadow_root_get_child_nodes_Shadow_DOM_def instances)

lemma l_delete_shadow_root_get_child_nodes_get_child_nodes_locs [instances]: "l_delete_shadow_root_get_child_node
get_child_nodes_locs"
  apply(auto simp add: l_delete_shadow_root_get_child_nodes_def)[1]
  using get_child_nodes_delete_shadow_root apply fast
  done
```

set_child_nodes

```
locale l_set_child_nodes_Shadow_DOM_defs =
  CD: l_set_child_nodes_Core_DOM_defs
begin
definition set_child_nodes_shadow_root_ptr :: "(_) shadow_root_ptr ⇒ (_) node_ptr list
    ⇒ (_, unit) dom_prog" where
  "set_child_nodes_shadow_root_ptr shadow_root_ptr = put_M shadow_root_ptr RShadowRoot.child_nodes_update"

definition a_set_child_nodes_tups :: "(((_) object_ptr ⇒ bool) × ((_) object_ptr ⇒ (_) node_ptr list
    ⇒ (_, unit) dom_prog)) list" where
  "a_set_child_nodes_tups ≡ [(is_shadow_root_ptr_object_ptr, set_child_nodes_shadow_root_ptr ∘ the ∘ cast)]"

definition a_set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ (_, unit) dom_prog"
  where
    "a_set_child_nodes ptr children = invoke (CD.a_set_child_nodes_tups @ a_set_child_nodes_tups) ptr children"

definition a_set_child_nodes_locs :: "(_) object_ptr ⇒ (_, unit) dom_prog set"
  where
    "a_set_child_nodes_locs ptr ≡
      (if is_shadow_root_ptr_kind ptr then all_args (put_M (the (cast ptr)) RShadowRoot.child_nodes_update)
else {}) ∪
      CD.a_set_child_nodes_locs ptr"
```

```
end

global_interpretation l_set_child_nodes_Shadow_DOM_defs defines
  set_child_nodes = l_set_child_nodes_Shadow_DOM_defs.a_set_child_nodes and
  set_child_nodes_locs = l_set_child_nodes_Shadow_DOM_defs.a_set_child_nodes_locs
  .

locale l_set_child_nodes_Shadow_DOM =
  l_type_wf type_wf +
  l_known_ptr known_ptr +
  l_set_child_nodes_Shadow_DOM_defs +
  l_set_child_nodes_defs set_child_nodes set_child_nodes_locs +
  CD: l_set_child_nodes_Core_DOM type_wf_Core_DOM known_ptr_Core_DOM set_child_nodes_Core_DOM set_child_nodes_loc
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ (_, unit) dom_prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ (_, unit) dom_prog set"
    and set_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ (_, unit) dom_prog"
    and set_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ (_, unit) dom_prog set" +
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes set_child_nodes_impl: "set_child_nodes = a_set_child_nodes"
  assumes set_child_nodes_locs_impl: "set_child_nodes_locs = a_set_child_nodes_locs"
begin
lemmas set_child_nodes_def = set_child_nodes_impl[unfolded a_set_child_nodes_def set_child_nodes_def]
lemmas set_child_nodes_locs_def =set_child_nodes_locs_impl[unfolded a_set_child_nodes_locs_def
    set_child_nodes_locs_def, folded CD.set_child_nodes_locs_impl]

lemma set_child_nodes_writes: "writes (set_child_nodes_locs ptr) (set_child_nodes ptr children) h h'"
  apply (simp add: set_child_nodes_def a_set_child_nodes_tups_def set_child_nodes_locs_def)
  apply(split CD.set_child_nodes_splits, rule conjI)+
   apply (simp add: CD.set_child_nodes_writes writes_union_right_I)
  apply(split invoke_splits, rule conjI)+
   apply(auto simp add: a_set_child_nodes_def)[1]
  apply(auto simp add: set_child_nodes_shadow_root_ptr_def intro!: writes_bind_pure
      intro: writes_union_right_I writes_union_left_I split: list.splits)[1]
  by (metis is_shadow_root_ptr_implies_kind option.case_eq_if)

lemma set_child_nodes_pointers_preserved:
  assumes "w ∈ set_child_nodes_locs object_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  using assms(1) object_ptr_kinds_preserved[OF writes_singleton2 assms(2)]
  by(auto simp add: all_args_def set_child_nodes_locs_def CD.set_child_nodes_locs_def split: if_splits)

lemma set_child_nodes_types_preserved:
  assumes "w ∈ set_child_nodes_locs object_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  using assms(1) type_wf_preserved[OF writes_singleton2 assms(2)] type_wf_impl
  by(auto simp add: all_args_def a_set_child_nodes_tups_def set_child_nodes_locs_def CD.set_child_nodes_locs_def
      split: if_splits option.splits)
end

interpretation
  i_set_child_nodes?: l_set_child_nodes_Shadow_DOM type_wf known_ptr DocumentClass.type_wf
  DocumentClass.known_ptr set_child_nodes set_child_nodes_locs Core_DOM_Functions.set_child_nodes
  Core_DOM_Functions.set_child_nodes_locs
  apply(unfold_locales)
  by (auto simp add: set_child_nodes_def set_child_nodes_locs_def)
declare l_set_child_nodes_Shadow_DOM_axioms[instances]
```

```
lemma set_child_nodes_is_l_set_child_nodes [instances]: "l_set_child_nodes type_wf
set_child_nodes set_child_nodes_locs"
  apply(auto simp add: l_set_child_nodes_def instances)[1]
  using  set_child_nodes_writes apply fast
  using set_child_nodes_pointers_preserved apply(fast, fast)
  using set_child_nodes_types_preserved apply(fast, fast)
  done
```

get_child_nodes

```
locale l_set_child_nodes_get_child_nodes_Shadow_DOM =
  l_get_child_nodes_Shadow_DOM
  type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes get_child_nodes_locs
  get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  + l_set_child_nodes_Shadow_DOM
  type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM set_child_nodes set_child_nodes_locs
  set_child_nodes_Core_DOM set_child_nodes_locs_Core_DOM
  + CD: l_set_child_nodes_get_child_nodes_Core_DOM
  type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  set_child_nodes_Core_DOM set_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
begin

lemma set_child_nodes_get_child_nodes:
  assumes "known_ptr ptr"
  assumes "type_wf h"
  assumes "h ⊢ set_child_nodes ptr children →_h h'"
  shows "h' ⊢ get_child_nodes ptr →_r children"
proof -
  have "h ⊢ check_in_heap ptr →_r ()"
    using assms set_child_nodes_def invoke_ptr_in_heap
    by (metis (full_types) check_in_heap_ptr_in_heap is_OK_returns_heap_I is_OK_returns_result_E
        old.unit.exhaust)
  then have ptr_in_h: "ptr |∈| object_ptr_kinds h"
    by (simp add: check_in_heap_ptr_in_heap is_OK_returns_result_I)

  have "type_wf h'"
    apply(unfold type_wf_impl)
    apply(rule subst[where P=id, OF type_wf_preserved[OF set_child_nodes_writes assms(3),
            unfolded all_args_def], simplified])
    by(auto simp add: all_args_def assms(2)[unfolded type_wf_impl] set_child_nodes_locs_def
        CD.set_child_nodes_locs_def split: if_splits)
  have "h' ⊢ check_in_heap ptr →_r ()"
    using check_in_heap_reads set_child_nodes_writes assms(3) ‹h ⊢ check_in_heap ptr →_r ()›
    apply(rule reads_writes_separate_forwards)
    apply(auto simp add: all_args_def set_child_nodes_locs_def CD.set_child_nodes_locs_def)[1]
    done
  then have "ptr |∈| object_ptr_kinds h'"
    using check_in_heap_ptr_in_heap by blast
  with assms ptr_in_h ‹type_wf h'› show ?thesis
    apply(auto simp add: type_wf_impl known_ptr_impl  get_child_nodes_def a_get_child_nodes_tups_def
        set_child_nodes_def a_set_child_nodes_tups_def del: bind_pure_returns_result_I2 intro!: bind_pure_returns_r
```

```
    apply(split CD.get_child_nodes_splits, (rule conjI impI)+)+
     apply(split CD.set_child_nodes_splits)+
      apply(auto simp add: CD.set_child_nodes_get_child_nodes type_wf_impl CD.type_wf_impl
        dest: ShadowRootClass.type_wf_Document)[1]
     apply(auto simp add: CD.set_child_nodes_get_child_nodes type_wf_impl CD.type_wf_impl
        dest: ShadowRootClass.type_wf_Document)[1]
    apply(split CD.set_child_nodes_splits)+
    by(auto simp add: known_ptr_impl CD.known_ptr_impl set_child_nodes_shadow_root_ptr_def
        get_child_nodes_shadow_root_ptr_def CD.type_wf_impl ShadowRootClass.type_wf_Document dest: known_ptr_new_shado
qed

lemma set_child_nodes_get_child_nodes_different_pointers:
  assumes "ptr ≠ ptr'"
  assumes "w ∈ set_child_nodes_locs ptr"
  assumes "h ⊢ w →_h h'"
  assumes "r ∈ get_child_nodes_locs ptr'"
  shows "r h h'"
  using assms
  apply(auto simp add: set_child_nodes_locs_def CD.set_child_nodes_locs_def
      get_child_nodes_locs_def CD.get_child_nodes_locs_def)[1]
  by(auto simp add: all_args_def elim!: is_document_ptr_kind_obtains is_shadow_root_ptr_kind_obtains
      is_element_ptr_kind_obtains split: if_splits option.splits)

end

interpretation
  i_set_child_nodes_get_child_nodes?: l_set_child_nodes_get_child_nodes_Shadow_DOM type_wf known_ptr
  DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes get_child_nodes_locs
  Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs set_child_nodes
  set_child_nodes_locs Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs
  using instances
  by(auto simp add: l_set_child_nodes_get_child_nodes_Shadow_DOM_def )
declare l_set_child_nodes_get_child_nodes_Shadow_DOM_axioms[instances]

lemma set_child_nodes_get_child_nodes_is_l_set_child_nodes_get_child_nodes [instances]:
  "l_set_child_nodes_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs set_child_nodes
set_child_nodes_locs"
  apply(auto simp add: instances l_set_child_nodes_get_child_nodes_def l_set_child_nodes_get_child_nodes_axioms_de
  using set_child_nodes_get_child_nodes apply fast
  using set_child_nodes_get_child_nodes_different_pointers apply fast
  done
```

set_tag_type

```
locale l_set_tag_name_Shadow_DOM =
  CD: l_set_tag_name_Core_DOM type_wf_Core_DOM set_tag_name set_tag_name_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and set_tag_name :: "(_) element_ptr ⇒ tag_name ⇒ (_, unit) dom_prog"
    and set_tag_name_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin
lemmas set_tag_name_def = CD.set_tag_name_impl[unfolded CD.a_set_tag_name_def set_tag_name_def]
lemmas set_tag_name_locs_def = CD.set_tag_name_locs_impl[unfolded CD.a_set_tag_name_locs_def
    set_tag_name_locs_def]

lemma set_tag_name_ok:
  "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (set_tag_name element_ptr tag)"
  apply(unfold type_wf_impl)
  unfolding set_tag_name_impl[unfolded a_set_tag_name_def] using get_M_Element_ok put_M_Element_ok
  using CD.set_tag_name_ok CD.type_wf_impl ShadowRootClass.type_wf_Document by blast
```

```
lemma set_tag_name_writes:
  "writes (set_tag_name_locs element_ptr) (set_tag_name element_ptr tag) h h'"
  using CD.set_tag_name_writes .

lemma set_tag_name_pointers_preserved:
  assumes "w ∈ set_tag_name_locs element_ptr"
  assumes "h ⊢ w →h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  using assms
  by(simp add: CD.set_tag_name_pointers_preserved)

lemma set_tag_name_typess_preserved:
  assumes "w ∈ set_tag_name_locs element_ptr"
  assumes "h ⊢ w →h h'"
  shows "type_wf h = type_wf h'"
  apply(unfold type_wf_impl)
  apply(rule type_wf_preserved[OF writes_singleton2 assms(2)])
  using assms(1) set_tag_name_locs_def
  by(auto simp add: all_args_def set_tag_name_locs_def
      split: if_splits)
end

interpretation i_set_tag_name?: l_set_tag_name_Shadow_DOM type_wf DocumentClass.type_wf set_tag_name
  set_tag_name_locs
  by(auto simp add: l_set_tag_name_Shadow_DOM_def l_set_tag_name_Shadow_DOM_axioms_def instances)
declare l_set_tag_name_Shadow_DOM_axioms [instances]

lemma set_tag_name_is_l_set_tag_name [instances]: "l_set_tag_name type_wf set_tag_name set_tag_name_locs"
  apply(auto simp add: l_set_tag_name_def)[1]

  using set_tag_name_writes apply fast
  using set_tag_name_ok apply fast
  using set_tag_name_pointers_preserved apply (fast, fast)
  using set_tag_name_typess_preserved apply (fast, fast)
  done
```

get_child_nodes

```
locale l_set_tag_name_get_child_nodes_Shadow_DOM =
  l_set_tag_name_Shadow_DOM +
  l_get_child_nodes_Shadow_DOM +
  CD: l_set_tag_name_get_child_nodes_Core_DOM type_wf_Core_DOM set_tag_name set_tag_name_locs
  known_ptr_Core_DOM get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
begin
lemma set_tag_name_get_child_nodes:
  "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
  apply(auto simp add: get_child_nodes_locs_def)[1]
    apply(auto simp add: set_tag_name_locs_def all_args_def)[1]
  using CD.set_tag_name_get_child_nodes apply(blast)
  using CD.set_tag_name_get_child_nodes apply(blast)
  done
end

interpretation
  i_set_tag_name_get_child_nodes?: l_set_tag_name_get_child_nodes_Shadow_DOM type_wf DocumentClass.type_wf
  set_tag_name set_tag_name_locs known_ptr DocumentClass.known_ptr get_child_nodes
  get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
  by unfold_locales
declare l_set_tag_name_get_child_nodes_Shadow_DOM_axioms[instances]

lemma set_tag_name_get_child_nodes_is_l_set_tag_name_get_child_nodes [instances]:
  "l_set_tag_name_get_child_nodes type_wf set_tag_name set_tag_name_locs known_ptr get_child_nodes
                                    get_child_nodes_locs"
  using set_tag_name_is_l_set_tag_name get_child_nodes_is_l_get_child_nodes
```

```
  apply(simp add: l_set_tag_name_get_child_nodes_def
      l_set_tag_name_get_child_nodes_axioms_def)
  using set_tag_name_get_child_nodes
  by fast
```

get_shadow_root

```
locale l_get_shadow_root_Shadow_DOM_defs
begin
definition a_get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
  where
    "a_get_shadow_root element_ptr = get_M element_ptr shadow_root_opt"

definition a_get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  where
    "a_get_shadow_root_locs element_ptr ≡ {preserved (get_M element_ptr shadow_root_opt)}"
end

global_interpretation l_get_shadow_root_Shadow_DOM_defs
  defines get_shadow_root = a_get_shadow_root
    and get_shadow_root_locs = a_get_shadow_root_locs
  .

locale l_get_shadow_root_defs =
  fixes get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
  fixes get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_shadow_root_Shadow_DOM =
  l_get_shadow_root_Shadow_DOM_defs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes get_shadow_root_impl: "get_shadow_root = a_get_shadow_root"
  assumes get_shadow_root_locs_impl: "get_shadow_root_locs = a_get_shadow_root_locs"
begin
lemmas get_shadow_root_def = get_shadow_root_impl[unfolded get_shadow_root_def a_get_shadow_root_def]
lemmas get_shadow_root_locs_def = get_shadow_root_locs_impl[unfolded get_shadow_root_locs_def a_get_shadow_root_l

lemma get_shadow_root_ok: "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (get_shadow_root
element_ptr)"
  unfolding get_shadow_root_def type_wf_impl
  using ShadowRootMonad.get_M_Element_ok by blast

lemma get_shadow_root_pure [simp]: "pure (get_shadow_root element_ptr) h"
  unfolding get_shadow_root_def by simp

lemma get_shadow_root_ptr_in_heap:
  assumes "h ⊢ get_shadow_root element_ptr →_r children"
  shows "element_ptr |∈| element_ptr_kinds h"
  using assms
  by(auto simp add: get_shadow_root_def get_M_Element_ptr_in_heap dest: is_OK_returns_result_I)

lemma get_shadow_root_reads: "reads (get_shadow_root_locs element_ptr) (get_shadow_root element_ptr) h
h'"
  by(simp add: get_shadow_root_def get_shadow_root_locs_def reads_bind_pure reads_insert_writes_set_right)
end

interpretation i_get_shadow_root?: l_get_shadow_root_Shadow_DOM type_wf  get_shadow_root get_shadow_root_locs
  using instances
  by (auto simp add: l_get_shadow_root_Shadow_DOM_def)
```

```
declare l_get_shadow_root_Shadow_DOM_axioms [instances]

locale l_get_shadow_root = l_type_wf + l_get_shadow_root_defs +
  assumes get_shadow_root_reads: "reads (get_shadow_root_locs element_ptr) (get_shadow_root element_ptr)
h h'"
  assumes get_shadow_root_ok: "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (get_shadow_root
element_ptr)"
  assumes get_shadow_root_ptr_in_heap: "h ⊢ ok (get_shadow_root element_ptr) ⟹ element_ptr |∈| element_ptr_kind
h"
  assumes get_shadow_root_pure [simp]: "pure (get_shadow_root element_ptr) h"

lemma get_shadow_root_is_l_get_shadow_root [instances]: "l_get_shadow_root type_wf get_shadow_root get_shadow_roo
  using instances
  apply(auto simp add: l_get_shadow_root_def)[1]
  using get_shadow_root_reads apply blast
  using get_shadow_root_ok apply blast
  using get_shadow_root_ptr_in_heap apply blast
  done
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_get_shadow_root_Shadow_DOM =
  l_set_disconnected_nodes_Core_DOM type_wf_Core_DOM set_disconnected_nodes set_disconnected_nodes_locs
+
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma set_disconnected_nodes_get_shadow_root:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  by(auto simp add: set_disconnected_nodes_locs_def get_shadow_root_locs_def all_args_def)
end

locale l_set_disconnected_nodes_get_shadow_root = l_set_disconnected_nodes_defs + l_get_shadow_root_defs
+
  assumes set_disconnected_nodes_get_shadow_root: "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h'
⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_disconnected_nodes_get_shadow_root?: l_set_disconnected_nodes_get_shadow_root_Shadow_DOM type_wf
  DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs get_shadow_root get_shadow_root_locs
  by(auto simp add: l_set_disconnected_nodes_get_shadow_root_Shadow_DOM_def instances)
declare l_set_disconnected_nodes_get_shadow_root_Shadow_DOM_axioms[instances]

lemma set_disconnected_nodes_get_shadow_root_is_l_set_disconnected_nodes_get_shadow_root [instances]:
  "l_set_disconnected_nodes_get_shadow_root set_disconnected_nodes_locs get_shadow_root_locs"
  apply(auto simp add: l_set_disconnected_nodes_get_shadow_root_def)[1]
  using set_disconnected_nodes_get_shadow_root apply fast
  done
```

set_tag_type

```
locale l_set_tag_name_get_shadow_root_Core_DOM =
  l_set_tag_name_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM
begin
lemma set_tag_name_get_shadow_root:
  "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  by(auto simp add: set_tag_name_locs_def
      get_shadow_root_locs_def all_args_def
      intro: element_put_get_preserved[where setter=tag_name_update and getter=shadow_root_opt])
end
```

```
locale l_set_tag_name_get_shadow_root = l_set_tag_name + l_get_shadow_root +
  assumes set_tag_name_get_shadow_root:
    "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_tag_name_get_shadow_root?: l_set_tag_name_get_shadow_root_Core_DOM type_wf DocumentClass.type_wf
  set_tag_name set_tag_name_locs
  get_shadow_root get_shadow_root_locs
  apply(auto simp add: l_set_tag_name_get_shadow_root_Core_DOM_def instances)[1]
  using l_set_tag_name_Shadow_DOM_axioms
  by unfold_locales
declare l_set_tag_name_get_shadow_root_Core_DOM_axioms[instances]

lemma set_tag_name_get_shadow_root_is_l_set_tag_name_get_shadow_root [instances]:
  "l_set_tag_name_get_shadow_root type_wf set_tag_name set_tag_name_locs get_shadow_root
                                  get_shadow_root_locs"
  using set_tag_name_is_l_set_tag_name get_shadow_root_is_l_get_shadow_root
  apply(simp add: l_set_tag_name_get_shadow_root_def l_set_tag_name_get_shadow_root_axioms_def)
  using set_tag_name_get_shadow_root
  by fast
```

set_child_nodes

```
locale l_set_child_nodes_get_shadow_root_Shadow_DOM =
  l_set_child_nodes_Shadow_DOM type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM set_child_nodes
  set_child_nodes_locs set_child_nodes_Core_DOM set_child_nodes_locs_Core_DOM +
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma set_child_nodes_get_shadow_root: "∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_shadow_root_
ptr'. r h h'))"
  apply(auto simp add: set_child_nodes_locs_def get_shadow_root_locs_def CD.set_child_nodes_locs_def all_args_def)
  by(auto intro!: element_put_get_preserved[where getter=shadow_root_opt and setter=RElement.child_nodes_update])
end

locale l_set_child_nodes_get_shadow_root = l_set_child_nodes_defs + l_get_shadow_root_defs +
  assumes set_child_nodes_get_shadow_root: "∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_shadow_r
ptr'. r h h'))"

interpretation
  i_set_child_nodes_get_shadow_root?: l_set_child_nodes_get_shadow_root_Shadow_DOM type_wf known_ptr
  DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
  Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_shadow_root
  get_shadow_root_locs
  by(auto simp add: l_set_child_nodes_get_shadow_root_Shadow_DOM_def instances)
declare l_set_child_nodes_get_shadow_root_Shadow_DOM_axioms[instances]

lemma set_child_nodes_get_shadow_root_is_l_set_child_nodes_get_shadow_root [instances]:
  "l_set_child_nodes_get_shadow_root set_child_nodes_locs get_shadow_root_locs"
  apply(auto simp add: l_set_child_nodes_get_shadow_root_def)[1]
  using set_child_nodes_get_shadow_root apply fast
  done
```

delete_shadow_root

```
locale l_delete_shadow_root_get_shadow_root_Shadow_DOM =
```

```
  l_get_shadow_root_Shadow_DOM
begin
lemma get_shadow_root_delete_shadow_root:  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  by(auto simp add: get_shadow_root_locs_def delete_shadow_root_get_M_Element)
end

locale l_delete_shadow_root_get_shadow_root = l_get_shadow_root_defs +
  assumes get_shadow_root_delete_shadow_root:  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
interpretation l_delete_shadow_root_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  by(auto simp add: l_delete_shadow_root_get_shadow_root_Shadow_DOM_def instances)

lemma l_delete_shadow_root_get_shadow_root_get_shadow_root_locs [instances]: "l_delete_shadow_root_get_shadow_roo
get_shadow_root_locs"
  apply(auto simp add: l_delete_shadow_root_get_shadow_root_def)[1]
  using get_shadow_root_delete_shadow_root apply fast
  done
```

new_character_data

```
locale l_new_character_data_get_shadow_root_Shadow_DOM =
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_shadow_root_new_character_data:
  "h ⊢ new_character_data →_r new_character_data_ptr ⟹ h ⊢ new_character_data →_h h'
     ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  by (auto simp add: get_shadow_root_locs_def new_character_data_get_M_Object new_character_data_get_M_Element
      split: prod.splits if_splits option.splits
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)
end

locale l_new_character_data_get_shadow_root = l_new_character_data + l_get_shadow_root +
  assumes get_shadow_root_new_character_data:
    "h ⊢ new_character_data →_r new_character_data_ptr
          ⟹ h ⊢ new_character_data →_h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"


interpretation i_new_character_data_get_shadow_root?:
  l_new_character_data_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  by(unfold_locales)
declare l_new_character_data_get_shadow_root_Shadow_DOM_axioms [instances]

lemma new_character_data_get_shadow_root_is_l_new_character_data_get_shadow_root [instances]:
  "l_new_character_data_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  using new_character_data_is_l_new_character_data get_shadow_root_is_l_get_shadow_root
  apply(auto simp add: l_new_character_data_get_shadow_root_def
      l_new_character_data_get_shadow_root_axioms_def instances)[1]
  using get_shadow_root_new_character_data
  by fast
```

new_document

```
locale l_new_document_get_shadow_root_Shadow_DOM =
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_shadow_root_new_document:
  "h ⊢ new_document →_r new_document_ptr ⟹ h ⊢ new_document →_h h'
     ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  by (auto simp add: get_shadow_root_locs_def new_document_get_M_Object new_document_get_M_Element
      split: prod.splits if_splits option.splits
```

```
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)
end

locale l_new_document_get_shadow_root = l_new_document + l_get_shadow_root +
  assumes get_shadow_root_new_document:
    "h ⊢ new_document →r new_document_ptr
            ⟹ h ⊢ new_document →h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"


interpretation i_new_document_get_shadow_root?:
  l_new_document_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  by(unfold_locales)
declare l_new_document_get_shadow_root_Shadow_DOM_axioms [instances]

lemma new_document_get_shadow_root_is_l_new_document_get_shadow_root [instances]:
  "l_new_document_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  using new_document_is_l_new_document get_shadow_root_is_l_get_shadow_root
  apply(auto simp add: l_new_document_get_shadow_root_def l_new_document_get_shadow_root_axioms_def instances)[1]
  using get_shadow_root_new_document
  by fast
```

new_element

```
locale l_new_element_get_shadow_root_Shadow_DOM =
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_shadow_root_new_element:
  "ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →r new_element_ptr ⟹ h ⊢ new_element →h h'
     ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  by (auto simp add: get_shadow_root_locs_def new_element_get_M_Object new_element_get_M_Element
      new_element_get_M_Document split: prod.splits if_splits option.splits
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)

lemma new_element_no_shadow_root:
  "h ⊢ new_element →r new_element_ptr ⟹ h ⊢ new_element →h h'
   ⟹ h' ⊢ get_shadow_root new_element_ptr →r None"
  by(simp add: get_shadow_root_def new_element_shadow_root_opt)
end

locale l_new_element_get_shadow_root = l_new_element + l_get_shadow_root +
  assumes get_shadow_root_new_element:
    "ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →r new_element_ptr
            ⟹ h ⊢ new_element →h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  assumes new_element_no_shadow_root:
    "h ⊢ new_element →r new_element_ptr ⟹ h ⊢ new_element →h h'
            ⟹ h' ⊢ get_shadow_root new_element_ptr →r None"


interpretation i_new_element_get_shadow_root?:
  l_new_element_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  by(unfold_locales)
declare l_new_element_get_shadow_root_Shadow_DOM_axioms [instances]

lemma new_element_get_shadow_root_is_l_new_element_get_shadow_root [instances]:
  "l_new_element_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  using new_element_is_l_new_element get_shadow_root_is_l_get_shadow_root
  apply(auto simp add: l_new_element_get_shadow_root_def l_new_element_get_shadow_root_axioms_def instances)[1]
  using get_shadow_root_new_element new_element_no_shadow_root
  by fast+
```

new_shadow_root

```
locale l_new_shadow_root_get_shadow_root_Shadow_DOM =
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
```

```
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_shadow_root_new_shadow_root:
  "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
     ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  by (auto simp add: get_shadow_root_locs_def new_shadow_root_get_M_Object new_shadow_root_get_M_Element
      split: prod.splits if_splits option.splits
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)
end

locale l_new_shadow_root_get_shadow_root = l_get_shadow_root +
  assumes get_shadow_root_new_shadow_root:
    "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
          ⟹ h ⊢ new_ShadowRoot_M →_h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"

interpretation i_new_shadow_root_get_shadow_root?:
  l_new_shadow_root_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  by(unfold_locales)
declare l_new_shadow_root_get_shadow_root_Shadow_DOM_axioms [instances]

lemma new_shadow_root_get_shadow_root_is_l_new_shadow_root_get_shadow_root [instances]:
  "l_new_shadow_root_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  using get_shadow_root_is_l_get_shadow_root
  apply(auto simp add: l_new_shadow_root_get_shadow_root_def l_new_shadow_root_get_shadow_root_axioms_def
instances)[1]
  using get_shadow_root_new_shadow_root
  by fast
```

set_shadow_root

```
locale l_set_shadow_root_Shadow_DOM_defs
begin
definition a_set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
  where
    "a_set_shadow_root element_ptr = put_M element_ptr shadow_root_opt_update"

definition a_set_shadow_root_locs :: "(_) element_ptr ⇒ ((_, unit) dom_prog) set"
  where
    "a_set_shadow_root_locs element_ptr ≡ all_args (put_M element_ptr shadow_root_opt_update)"
end

global_interpretation l_set_shadow_root_Shadow_DOM_defs
  defines set_shadow_root = a_set_shadow_root
    and set_shadow_root_locs = a_set_shadow_root_locs
  .

locale l_set_shadow_root_defs =
  fixes set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
  fixes set_shadow_root_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set"


locale l_set_shadow_root_Shadow_DOM =
  l_type_wf type_wf +
  l_set_shadow_root_defs set_shadow_root set_shadow_root_locs +
  l_set_shadow_root_Shadow_DOM_defs
  for type_wf :: "(_) heap ⇒ bool"
    and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes set_shadow_root_impl: "set_shadow_root = a_set_shadow_root"
  assumes set_shadow_root_locs_impl: "set_shadow_root_locs = a_set_shadow_root_locs"
```

```
begin
lemmas set_shadow_root_def = set_shadow_root_impl[unfolded set_shadow_root_def a_set_shadow_root_def]
lemmas set_shadow_root_locs_def = set_shadow_root_locs_impl[unfolded set_shadow_root_locs_def a_set_shadow_root_l

lemma set_shadow_root_ok: "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (set_shadow_root
element_ptr tag)"
  apply(unfold type_wf_impl)
  unfolding set_shadow_root_def using get_M_Element_ok put_M_Element_ok
  by (simp add: ShadowRootMonad.put_M_Element_ok)

lemma set_shadow_root_ptr_in_heap:
  "h ⊢ ok (set_shadow_root element_ptr shadow_root) ⟹ element_ptr |∈| element_ptr_kinds h"
  by(simp add: set_shadow_root_def ElementMonad.put_M_ptr_in_heap)

lemma set_shadow_root_writes: "writes (set_shadow_root_locs element_ptr) (set_shadow_root element_ptr tag)
h h'"
  by(auto simp add: set_shadow_root_def set_shadow_root_locs_def intro: writes_bind_pure)

lemma set_shadow_root_pointers_preserved:
  assumes "w ∈ set_shadow_root_locs element_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  using assms(1) object_ptr_kinds_preserved[OF writes_singleton2 assms(2)]
  by(auto simp add: all_args_def set_shadow_root_locs_def split: if_splits)

lemma set_shadow_root_types_preserved:
  assumes "w ∈ set_shadow_root_locs element_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  apply(unfold type_wf_impl)
  using assms(1) type_wf_preserved[OF writes_singleton2 assms(2)]
  by(auto simp add: all_args_def set_shadow_root_locs_def split: if_splits)
end

interpretation i_set_shadow_root?: l_set_shadow_root_Shadow_DOM type_wf set_shadow_root set_shadow_root_locs
  by (auto simp add: l_set_shadow_root_Shadow_DOM_def instances)
declare l_set_shadow_root_Shadow_DOM_axioms [instances]

locale l_set_shadow_root = l_type_wf +l_set_shadow_root_defs +
  assumes set_shadow_root_writes:
    "writes (set_shadow_root_locs element_ptr) (set_shadow_root element_ptr disc_nodes) h h'"
  assumes set_shadow_root_ok:
    "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (set_shadow_root element_ptr shadow_root)"
  assumes set_shadow_root_ptr_in_heap:
    "h ⊢ ok (set_shadow_root element_ptr shadow_root) ⟹ element_ptr |∈| element_ptr_kinds h"
  assumes set_shadow_root_pointers_preserved:
    "w ∈ set_shadow_root_locs element_ptr ⟹ h ⊢ w →_h h' ⟹ object_ptr_kinds h = object_ptr_kinds h'"
  assumes set_shadow_root_types_preserved:
    "w ∈ set_shadow_root_locs element_ptr ⟹ h ⊢ w →_h h' ⟹ type_wf h = type_wf h'"

lemma set_shadow_root_is_l_set_shadow_root [instances]: "l_set_shadow_root type_wf set_shadow_root set_shadow_roo
  apply(auto simp add: l_set_shadow_root_def instances)[1]
  using set_shadow_root_writes apply blast
  using set_shadow_root_ok apply (blast)
  using set_shadow_root_ptr_in_heap apply blast
  using set_shadow_root_pointers_preserved apply(blast, blast)
  using set_shadow_root_types_preserved apply(blast, blast)
  done
```

get_shadow_root

```
locale l_set_shadow_root_get_shadow_root_Shadow_DOM =
  l_set_shadow_root_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM
begin
```

```
lemma set_shadow_root_get_shadow_root:
  "type_wf h ⟹ h ⊢ set_shadow_root ptr shadow_root_ptr_opt →h h' ⟹
h' ⊢ get_shadow_root ptr →r shadow_root_ptr_opt"
  by(auto simp add: set_shadow_root_def get_shadow_root_def)

lemma set_shadow_root_get_shadow_root_different_pointers:
  "ptr ≠ ptr' ⟹ ∀w ∈ set_shadow_root_locs ptr.
(h ⊢ w →h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  by(auto simp add: set_shadow_root_locs_def get_shadow_root_locs_def all_args_def)
end

interpretation
  i_set_shadow_root_get_shadow_root?: l_set_shadow_root_get_shadow_root_Shadow_DOM type_wf
  set_shadow_root set_shadow_root_locs get_shadow_root get_shadow_root_locs
  apply(auto simp add: l_set_shadow_root_get_shadow_root_Shadow_DOM_def instances)[1]
  by(unfold_locales)
declare l_set_shadow_root_get_shadow_root_Shadow_DOM_axioms[instances]

locale l_set_shadow_root_get_shadow_root = l_type_wf + l_set_shadow_root_defs + l_get_shadow_root_defs +
  assumes set_shadow_root_get_shadow_root:
    "type_wf h ⟹ h ⊢ set_shadow_root ptr shadow_root_ptr_opt →h h' ⟹
h' ⊢ get_shadow_root ptr →r shadow_root_ptr_opt"
  assumes set_shadow_root_get_shadow_root_different_pointers:
    "ptr ≠ ptr' ⟹ w ∈ set_shadow_root_locs ptr ⟹ h ⊢ w →h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹
r h h'"

lemma set_shadow_root_get_shadow_root_is_l_set_shadow_root_get_shadow_root [instances]:
  "l_set_shadow_root_get_shadow_root type_wf set_shadow_root set_shadow_root_locs get_shadow_root
get_shadow_root_locs"
  apply(auto simp add: l_set_shadow_root_get_shadow_root_def instances)[1]
  using set_shadow_root_get_shadow_root apply fast
  using set_shadow_root_get_shadow_root_different_pointers apply fast
  done
```

set_mode

```
locale l_set_mode_Shadow_DOM_defs
begin
definition a_set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"
  where
    "a_set_mode shadow_root_ptr = put_M shadow_root_ptr mode_update"

definition a_set_mode_locs :: "(_) shadow_root_ptr ⇒ ((_, unit) dom_prog) set"
  where
    "a_set_mode_locs shadow_root_ptr ≡ all_args (put_M shadow_root_ptr mode_update)"
end

global_interpretation l_set_mode_Shadow_DOM_defs
  defines set_mode = a_set_mode
    and set_mode_locs = a_set_mode_locs
  .

locale l_set_mode_defs =
  fixes set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"
  fixes set_mode_locs :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog set"


locale l_set_mode_Shadow_DOM =
  l_type_wf type_wf +
  l_set_mode_defs set_mode set_mode_locs +
  l_set_mode_Shadow_DOM_defs
  for type_wf :: "(_) heap ⇒ bool"
    and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"
```

```
    and set_mode_locs :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes set_mode_impl: "set_mode = a_set_mode"
  assumes set_mode_locs_impl: "set_mode_locs = a_set_mode_locs"
begin
lemmas set_mode_def = set_mode_impl[unfolded set_mode_def a_set_mode_def]
lemmas set_mode_locs_def = set_mode_locs_impl[unfolded set_mode_locs_def a_set_mode_locs_def]

lemma set_mode_ok: "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹
h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode)"
  apply(unfold type_wf_impl)
  unfolding set_mode_def using get_M_ShadowRoot_ok put_M_ShadowRoot_ok
  by (simp add: ShadowRootMonad.put_M_ShadowRoot_ok)

lemma set_mode_ptr_in_heap:
  "h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode) ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  by(simp add: set_mode_def put_M_ptr_in_heap)

lemma set_mode_writes: "writes (set_mode_locs shadow_root_ptr) (set_mode shadow_root_ptr shadow_root_mode)
h h'"
  by(auto simp add: set_mode_def set_mode_locs_def intro: writes_bind_pure)

lemma set_mode_pointers_preserved:
  assumes "w ∈ set_mode_locs element_ptr"
  assumes "h ⊢ w →h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  using assms(1) object_ptr_kinds_preserved[OF writes_singleton2 assms(2)]
  by(auto simp add: all_args_def set_mode_locs_def split: if_splits)

lemma set_mode_types_preserved:
  assumes "w ∈ set_mode_locs element_ptr"
  assumes "h ⊢ w →h h'"
  shows "type_wf h = type_wf h'"
  apply(unfold type_wf_impl)
  using assms(1) type_wf_preserved[OF writes_singleton2 assms(2)]
  by(auto simp add: all_args_def set_mode_locs_def split: if_splits)
end

interpretation i_set_mode?: l_set_mode_Shadow_DOM type_wf set_mode set_mode_locs
  by (auto simp add: l_set_mode_Shadow_DOM_def instances)
declare l_set_mode_Shadow_DOM_axioms [instances]

locale l_set_mode = l_type_wf +l_set_mode_defs +
  assumes set_mode_writes:
    "writes (set_mode_locs shadow_root_ptr) (set_mode shadow_root_ptr shadow_root_mode) h h'"
  assumes set_mode_ok:
    "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹ h ⊢ ok (set_mode shadow_root_ptr shadow_root_m
  assumes set_mode_ptr_in_heap:
    "h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode) ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  assumes set_mode_pointers_preserved:
    "w ∈ set_mode_locs shadow_root_ptr ⟹ h ⊢ w →h h' ⟹ object_ptr_kinds h = object_ptr_kinds h'"
  assumes set_mode_types_preserved:
    "w ∈ set_mode_locs shadow_root_ptr ⟹ h ⊢ w →h h' ⟹ type_wf h = type_wf h'"

lemma set_mode_is_l_set_mode [instances]: "l_set_mode type_wf set_mode set_mode_locs"
  apply(auto simp add: l_set_mode_def instances)[1]
  using set_mode_writes apply blast
  using set_mode_ok apply (blast)
  using set_mode_ptr_in_heap apply blast
  using set_mode_pointers_preserved apply(blast, blast)
  using set_mode_types_preserved apply(blast, blast)
  done
```

get_child_nodes **locale** *l_set_shadow_root_get_child_nodes*$_{Shadow_DOM}$ =
  l_get_child_nodes$_{Shadow_DOM}$ +
  l_set_shadow_root$_{Shadow_DOM}$
begin
lemma *set_shadow_root_get_child_nodes:*
  "∀ *w* ∈ *set_shadow_root_locs ptr. (h* ⊢ *w* →$_h$ *h'* ⟶ (∀ *r* ∈ *get_child_nodes_locs ptr'. r h h'))"*
  by *(auto simp add: get_child_nodes_locs_def set_shadow_root_locs_def CD.get_child_nodes_locs_def*
    *all_args_def intro: element_put_get_preserved[***where** *setter=shadow_root_opt_update])*
end

interpretation *i_set_shadow_root_get_child_nodes?:*
  l_set_shadow_root_get_child_nodes$_{Shadow_DOM}$ *type_wf known_ptr DocumentClass.type_wf*
  DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs set_shadow_root set_shadow_root_locs
  by *(unfold_locales)*
declare *l_set_shadow_root_get_child_nodes*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_shadow_root_get_child_nodes = l_set_shadow_root + l_get_child_nodes +*
  assumes *set_shadow_root_get_child_nodes:*
    "∀ *w* ∈ *set_shadow_root_locs ptr. (h* ⊢ *w* →$_h$ *h'* ⟶ (∀ *r* ∈ *get_child_nodes_locs ptr'. r h h'))"*

lemma *set_shadow_root_get_child_nodes_is_l_set_shadow_root_get_child_nodes [instances]:*
  "l_set_shadow_root_get_child_nodes type_wf set_shadow_root set_shadow_root_locs known_ptr
get_child_nodes get_child_nodes_locs"
  apply *(auto simp add: l_set_shadow_root_get_child_nodes_def l_set_shadow_root_get_child_nodes_axioms_def*
    instances)[1]
  using *set_shadow_root_get_child_nodes* **apply** *blast*
  done

get_shadow_root **locale** *l_set_mode_get_shadow_root*$_{Shadow_DOM}$ =
  l_set_mode$_{Shadow_DOM}$ +
  l_get_shadow_root$_{Shadow_DOM}$
begin
lemma *set_mode_get_shadow_root:*
  "∀ *w* ∈ *set_mode_locs ptr. (h* ⊢ *w* →$_h$ *h'* ⟶ (∀ *r* ∈ *get_shadow_root_locs ptr'. r h h'))"*
  by *(auto simp add: set_mode_locs_def get_shadow_root_locs_def all_args_def)*
end

interpretation
  i_set_mode_get_shadow_root?: l_set_mode_get_shadow_root$_{Shadow_DOM}$ *type_wf*
  set_mode set_mode_locs get_shadow_root
  get_shadow_root_locs
  by *unfold_locales*
declare *l_set_mode_get_shadow_root*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_mode_get_shadow_root = l_set_mode + l_get_shadow_root +*
  assumes *set_mode_get_shadow_root:*
    "∀ *w* ∈ *set_mode_locs ptr. (h* ⊢ *w* →$_h$ *h'* ⟶ (∀ *r* ∈ *get_shadow_root_locs ptr'. r h h'))"*

lemma *set_mode_get_shadow_root_is_l_set_mode_get_shadow_root [instances]:*
  "l_set_mode_get_shadow_root type_wf set_mode set_mode_locs get_shadow_root
    get_shadow_root_locs"
  using *set_mode_is_l_set_mode get_shadow_root_is_l_get_shadow_root*
  apply *(simp add: l_set_mode_get_shadow_root_def*
    l_set_mode_get_shadow_root_axioms_def)
  using *set_mode_get_shadow_root*
  by *fast*

get_child_nodes **locale** *l_set_mode_get_child_nodes*$_{Shadow_DOM}$ =
  l_set_mode$_{Shadow_DOM}$ +
  l_get_child_nodes$_{Shadow_DOM}$
begin
lemma *set_mode_get_child_nodes:*

```
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
  by(auto simp add: get_child_nodes_locs_def CD.get_child_nodes_locs_def set_mode_locs_def all_args_def)[1]
end

interpretation
  i_set_mode_get_child_nodes?: l_set_mode_get_child_nodes_Shadow_DOM type_wf set_mode set_mode_locs known_ptr
DocumentClass.type_wf
  DocumentClass.known_ptr get_child_nodes
  get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
  by unfold_locales
declare l_set_mode_get_child_nodes_Shadow_DOM_axioms[instances]

locale l_set_mode_get_child_nodes = l_set_mode + l_get_child_nodes +
  assumes set_mode_get_child_nodes:
    "∀w ∈ set_mode_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"

lemma set_mode_get_child_nodes_is_l_set_mode_get_child_nodes [instances]:
  "l_set_mode_get_child_nodes type_wf set_mode set_mode_locs known_ptr get_child_nodes
                                  get_child_nodes_locs"
  using set_mode_is_l_set_mode get_child_nodes_is_l_get_child_nodes
  apply(simp add: l_set_mode_get_child_nodes_def
      l_set_mode_get_child_nodes_axioms_def)
  using set_mode_get_child_nodes
  by fast
```

get_host

```
locale l_get_host_Shadow_DOM_defs =
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs
  for get_shadow_root :: "(_::linorder) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option)
prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_host :: "(_) shadow_root_ptr ⇒ (_, (_) element_ptr) dom_prog"
  where
    "a_get_host shadow_root_ptr = do {
      host_ptrs ← element_ptr_kinds_M ≫= filter_M (λelement_ptr. do {
        shadow_root_opt ← get_shadow_root element_ptr;
        return (shadow_root_opt = Some shadow_root_ptr)
      });
      (case host_ptrs of host_ptr#[] ⇒ return host_ptr | _ ⇒ error HierarchyRequestError)
    }"
definition "a_get_host_locs ≡ (⋃element_ptr. (get_shadow_root_locs element_ptr)) ∪
(⋃ptr. {preserved (get_M_Object ptr RObject.nothing)})"

end

global_interpretation l_get_host_Shadow_DOM_defs get_shadow_root get_shadow_root_locs
  defines get_host = "a_get_host"
    and get_host_locs = "a_get_host_locs"
  .

locale l_get_host_defs =
  fixes get_host :: "(_) shadow_root_ptr ⇒ (_, (_) element_ptr) dom_prog"
  fixes get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_host_Shadow_DOM =
  l_get_host_Shadow_DOM_defs +
  l_get_host_defs +
  l_get_shadow_root +
  assumes get_host_impl: "get_host = a_get_host"
  assumes get_host_locs_impl: "get_host_locs = a_get_host_locs"
```

```
begin
lemmas get_host_def = get_host_impl[unfolded a_get_host_def]
lemmas get_host_locs_def = get_host_locs_impl[unfolded a_get_host_locs_def]

lemma get_host_pure [simp]: "pure (get_host element_ptr) h"
  by(auto simp add: get_host_def intro!: bind_pure_I filter_M_pure_I split: list.splits)

lemma get_host_reads: "reads get_host_locs (get_host element_ptr) h h'"
  using get_shadow_root_reads[unfolded reads_def]
  by(auto simp add: get_host_def get_host_locs_def intro!: reads_bind_pure
      reads_subset[OF check_in_heap_reads] reads_subset[OF error_reads] reads_subset[OF get_shadow_root_reads]
      reads_subset[OF return_reads] reads_subset[OF element_ptr_kinds_M_reads] filter_M_reads filter_M_pure_I
      bind_pure_I split: list.splits)
end

locale l_get_host = l_get_host_defs +
  assumes get_host_pure [simp]: "pure (get_host element_ptr) h"
  assumes get_host_reads: "reads get_host_locs (get_host node_ptr) h h'"


interpretation i_get_host?: l_get_host_Shadow_DOM get_shadow_root get_shadow_root_locs get_host
  get_host_locs type_wf
  using instances
  by (simp add: l_get_host_Shadow_DOM_def l_get_host_Shadow_DOM_axioms_def get_host_def get_host_locs_def)
declare l_get_host_Shadow_DOM_axioms [instances]

lemma get_host_is_l_get_host [instances]: "l_get_host get_host get_host_locs"
  apply(auto simp add: l_get_host_def)[1]
  using get_host_reads apply fast
  done
```

get_mode

```
locale l_get_mode_Shadow_DOM_defs
begin
definition a_get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"
  where
    "a_get_mode shadow_root_ptr = get_M shadow_root_ptr mode"

definition a_get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  where
    "a_get_mode_locs shadow_root_ptr ≡ {preserved (get_M shadow_root_ptr mode)}"
end

global_interpretation l_get_mode_Shadow_DOM_defs
  defines get_mode = a_get_mode
    and get_mode_locs = a_get_mode_locs
  .

locale l_get_mode_defs =
  fixes get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"
  fixes get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_mode_Shadow_DOM =
  l_get_mode_Shadow_DOM_defs +
  l_get_mode_defs get_mode get_mode_locs +
  l_type_wf type_wf
  for get_mode :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, shadow_root_mode) prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and type_wf :: "(_) heap ⇒ bool" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes get_mode_impl: "get_mode = a_get_mode"
  assumes get_mode_locs_impl: "get_mode_locs = a_get_mode_locs"
```

```
begin
lemmas get_mode_def = get_mode_impl[unfolded get_mode_def a_get_mode_def]
lemmas get_mode_locs_def = get_mode_locs_impl[unfolded get_mode_locs_def a_get_mode_locs_def]

lemma get_mode_ok: "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹
h ⊢ ok (get_mode shadow_root_ptr)"
  unfolding get_mode_def type_wf_impl
  using ShadowRootMonad.get_M_ShadowRoot_ok by blast

lemma get_mode_pure [simp]: "pure (get_mode element_ptr) h"
  unfolding get_mode_def by simp

lemma get_mode_ptr_in_heap:
  assumes "h ⊢ get_mode shadow_root_ptr →_r children"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  using assms
  by(auto simp add: get_mode_def get_M_ShadowRoot_ptr_in_heap dest: is_OK_returns_result_I)

lemma get_mode_reads: "reads (get_mode_locs element_ptr) (get_mode element_ptr) h h'"
  by(simp add: get_mode_def get_mode_locs_def reads_bind_pure reads_insert_writes_set_right)
end

interpretation i_get_mode?: l_get_mode_Shadow_DOM get_mode get_mode_locs type_wf
  using instances
  by (auto simp add: l_get_mode_Shadow_DOM_def)
declare l_get_mode_Shadow_DOM_axioms [instances]

locale l_get_mode = l_type_wf + l_get_mode_defs +
  assumes get_mode_reads: "reads (get_mode_locs shadow_root_ptr) (get_mode shadow_root_ptr) h h'"
  assumes get_mode_ok: "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹
h ⊢ ok (get_mode shadow_root_ptr)"
  assumes get_mode_ptr_in_heap: "h ⊢ ok (get_mode shadow_root_ptr) ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds
h"
  assumes get_mode_pure [simp]: "pure (get_mode shadow_root_ptr) h"

lemma get_mode_is_l_get_mode [instances]: "l_get_mode type_wf get_mode get_mode_locs"
  apply(auto simp add: l_get_mode_def instances)[1]
  using get_mode_reads apply blast
  using get_mode_ok apply blast
  using get_mode_ptr_in_heap apply blast
  done
```

get_shadow_root_safe

```
locale l_get_shadow_root_safe_Shadow_DOM_defs =
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_get_mode_defs get_mode get_mode_locs
  for get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_shadow_root_safe :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
  where
    "a_get_shadow_root_safe element_ptr = do {
      shadow_root_ptr_opt ← get_shadow_root element_ptr;
      (case shadow_root_ptr_opt of
        Some shadow_root_ptr ⇒ do {
          mode ← get_mode shadow_root_ptr;
          (if mode = Open then
            return (Some shadow_root_ptr)
          else
            return None
```

```
        )
      } | None ⇒ return None)
  }"

definition a_get_shadow_root_safe_locs ::
  "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  where
    "a_get_shadow_root_safe_locs element_ptr shadow_root_ptr ≡
(get_shadow_root_locs element_ptr) ∪ (get_mode_locs shadow_root_ptr)"
end

global_interpretation l_get_shadow_root_safe_Shadow_DOM_defs get_shadow_root get_shadow_root_locs get_mode
get_mode_locs
  defines get_shadow_root_safe = a_get_shadow_root_safe
    and get_shadow_root_safe_locs = a_get_shadow_root_safe_locs
  .

locale l_get_shadow_root_safe_defs =
  fixes get_shadow_root_safe :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
  fixes get_shadow_root_safe_locs ::
    "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_shadow_root_safe_Shadow_DOM =
  l_get_shadow_root_safe_Shadow_DOM_defs get_shadow_root get_shadow_root_locs get_mode get_mode_locs +
  l_get_shadow_root_safe_defs get_shadow_root_safe get_shadow_root_safe_locs +
  l_get_shadow_root  type_wf get_shadow_root get_shadow_root_locs +
  l_get_mode  type_wf get_mode get_mode_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root_safe :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_safe_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒
bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes get_shadow_root_safe_impl: "get_shadow_root_safe = a_get_shadow_root_safe"
  assumes get_shadow_root_safe_locs_impl: "get_shadow_root_safe_locs = a_get_shadow_root_safe_locs"
begin
lemmas get_shadow_root_safe_def = get_shadow_root_safe_impl[unfolded get_shadow_root_safe_def
    a_get_shadow_root_safe_def]
lemmas get_shadow_root_safe_locs_def = get_shadow_root_safe_locs_impl[unfolded get_shadow_root_safe_locs_def
    a_get_shadow_root_safe_locs_def]

lemma get_shadow_root_safe_pure [simp]: "pure (get_shadow_root_safe element_ptr) h"
  apply(auto simp add: get_shadow_root_safe_def)[1]
  by (smt bind_returns_heap_E is_OK_returns_heap_E local.get_mode_pure local.get_shadow_root_pure
      option.case_eq_if pure_def pure_returns_heap_eq return_pure)
end

interpretation i_get_shadow_root_safe?: l_get_shadow_root_safe_Shadow_DOM  type_wf get_shadow_root_safe
  get_shadow_root_safe_locs get_shadow_root get_shadow_root_locs get_mode get_mode_locs
  using instances
  by (auto simp add: l_get_shadow_root_safe_Shadow_DOM_def l_get_shadow_root_safe_Shadow_DOM_axioms_def
      get_shadow_root_safe_def get_shadow_root_safe_locs_def)
declare l_get_shadow_root_safe_Shadow_DOM_axioms [instances]

locale l_get_shadow_root_safe = l_get_shadow_root_safe_defs +
  assumes get_shadow_root_safe_pure [simp]: "pure (get_shadow_root_safe element_ptr) h"

lemma get_shadow_root_safe_is_l_get_shadow_root_safe [instances]: "l_get_shadow_root_safe get_shadow_root_safe"
  using instances
```

```
  apply(auto simp add: l_get_shadow_root_safe_def)[1]
  done
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_Shadow_DOM =
  CD: l_set_disconnected_nodes_Core_DOM type_wf_Core_DOM set_disconnected_nodes set_disconnected_nodes_locs
+
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin
lemma set_disconnected_nodes_ok:
  "type_wf h ⟹ document_ptr |∈| document_ptr_kinds h ⟹ h ⊢ ok (set_disconnected_nodes document_ptr
node_ptrs)"
  using CD.set_disconnected_nodes_ok CD.type_wf_impl ShadowRootClass.type_wf_defs local.type_wf_impl
  by blast

lemma set_disconnected_nodes_typess_preserved:
  assumes "w ∈ set_disconnected_nodes_locs object_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  using assms(1) type_wf_preserved[OF writes_singleton2 assms(2)]
  apply(unfold type_wf_impl)
  by(auto simp add: all_args_def CD.set_disconnected_nodes_locs_def
      intro: put_M_Document_disconnected_nodes_type_wf_preserved split: if_splits)
end

interpretation i_set_disconnected_nodes?: l_set_disconnected_nodes_Shadow_DOM type_wf DocumentClass.type_wf
  set_disconnected_nodes set_disconnected_nodes_locs
  by(auto simp add: l_set_disconnected_nodes_Shadow_DOM_def l_set_disconnected_nodes_Shadow_DOM_axioms_def
instances)
declare l_set_disconnected_nodes_Shadow_DOM_axioms [instances]

lemma set_disconnected_nodes_is_l_set_disconnected_nodes [instances]:
  "l_set_disconnected_nodes type_wf set_disconnected_nodes set_disconnected_nodes_locs"
  apply(auto simp add: l_set_disconnected_nodes_def)[1]
        apply (simp add: i_set_disconnected_nodes.set_disconnected_nodes_writes)
  using set_disconnected_nodes_ok apply blast
      apply (simp add: i_set_disconnected_nodes.set_disconnected_nodes_ptr_in_heap)
  using i_set_disconnected_nodes.set_disconnected_nodes_pointers_preserved apply (blast, blast)
  using set_disconnected_nodes_typess_preserved apply(blast, blast)
  done
```

get_child_nodes

```
locale l_set_disconnected_nodes_get_child_nodes_Shadow_DOM =
  l_set_disconnected_nodes_Core_DOM type_wf_Core_DOM set_disconnected_nodes set_disconnected_nodes_locs
+
  l_get_child_nodes_Shadow_DOM type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes
  get_child_nodes_locs get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
```

```
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin

lemma set_disconnected_nodes_get_child_nodes:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
  by(auto simp add: set_disconnected_nodes_locs_def get_child_nodes_locs_def CD.get_child_nodes_locs_def
      all_args_def)
end

interpretation i_set_disconnected_nodes_get_child_nodes?: l_set_disconnected_nodes_get_child_nodes_Shadow_DOM
  type_wf set_disconnected_nodes set_disconnected_nodes_locs known_ptr DocumentClass.type_wf
  DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
  Core_DOM_Functions.get_child_nodes_locs
  by(auto simp add: l_set_disconnected_nodes_get_child_nodes_Shadow_DOM_def instances)
declare l_set_disconnected_nodes_get_child_nodes_Shadow_DOM_axioms[instances]

lemma set_disconnected_nodes_get_child_nodes_is_l_set_disconnected_nodes_get_child_nodes [instances]:
  "l_set_disconnected_nodes_get_child_nodes set_disconnected_nodes_locs get_child_nodes_locs"
  apply(auto simp add: l_set_disconnected_nodes_get_child_nodes_def)[1]
  using set_disconnected_nodes_get_child_nodes apply fast
  done
```

get_host

```
locale l_set_disconnected_nodes_get_host_Shadow_DOM =
  l_set_disconnected_nodes_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM +
  l_get_host_Shadow_DOM
begin
lemma set_disconnected_nodes_get_host:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_host_locs. r h h'))"
  by(auto simp add: CD.set_disconnected_nodes_locs_def get_shadow_root_locs_def get_host_locs_def all_args_def)
end

interpretation i_set_disconnected_nodes_get_host?: l_set_disconnected_nodes_get_host_Shadow_DOM
  type_wf DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs get_shadow_root
  get_shadow_root_locs get_host get_host_locs
  by(auto simp add: l_set_disconnected_nodes_get_host_Shadow_DOM_def instances)
declare l_set_disconnected_nodes_get_host_Shadow_DOM_axioms [instances]

locale l_set_disconnected_nodes_get_host = l_set_disconnected_nodes_defs + l_get_host_defs +
  assumes set_disconnected_nodes_get_host:
    "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_host_locs. r h h'))"

lemma set_disconnected_nodes_get_host_is_l_set_disconnected_nodes_get_host [instances]:
  "l_set_disconnected_nodes_get_host set_disconnected_nodes_locs get_host_locs"
  apply(auto simp add: l_set_disconnected_nodes_get_host_def instances)[1]
  using set_disconnected_nodes_get_host
  by fast
```

get_tag_name

```
locale l_get_tag_name_Shadow_DOM =
  CD: l_get_tag_name_Core_DOM type_wf_Core_DOM get_tag_name get_tag_name_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ (_, tag_name) dom_prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin

lemma get_tag_name_ok:
  "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (get_tag_name element_ptr)"
  apply(unfold type_wf_impl get_tag_name_impl[unfolded a_get_tag_name_def])
```

```
  using CD.get_tag_name_ok CD.type_wf_impl ShadowRootClass.type_wf_Document
  by blast
end

interpretation i_get_tag_name?: l_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name get_tag_name_
  by(auto simp add: l_get_tag_name_Shadow_DOM_def l_get_tag_name_Shadow_DOM_axioms_def instances)
declare l_get_tag_name_Shadow_DOM_axioms [instances]

lemma get_tag_name_is_l_get_tag_name [instances]: "l_get_tag_name type_wf get_tag_name get_tag_name_locs"
  apply(auto simp add: l_get_tag_name_def)[1]
  using get_tag_name_reads apply fast
  using get_tag_name_ok apply fast
  done
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_get_tag_name_Shadow_DOM =
  l_set_disconnected_nodes_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_disconnected_nodes_get_tag_name:
  "∀ w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀ r ∈ get_tag_name_locs ptr'. r h h'))"
  by(auto simp add: CD.set_disconnected_nodes_locs_def CD.get_tag_name_locs_def all_args_def)
end

interpretation i_set_disconnected_nodes_get_tag_name?: l_set_disconnected_nodes_get_tag_name_Shadow_DOM
  type_wf DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs get_tag_name
  get_tag_name_locs
  by(auto simp add: l_set_disconnected_nodes_get_tag_name_Shadow_DOM_def instances)
declare l_set_disconnected_nodes_get_tag_name_Shadow_DOM_axioms [instances]

lemma set_disconnected_nodes_get_tag_name_is_l_set_disconnected_nodes_get_tag_name [instances]:
  "l_set_disconnected_nodes_get_tag_name type_wf set_disconnected_nodes set_disconnected_nodes_locs
get_tag_name get_tag_name_locs"
  apply(auto simp add: l_set_disconnected_nodes_get_tag_name_def
      l_set_disconnected_nodes_get_tag_name_axioms_def instances)[1]
  using set_disconnected_nodes_get_tag_name
  by fast
```

set_child_nodes

```
locale l_set_child_nodes_get_tag_name_Shadow_DOM =
  l_set_child_nodes_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_child_nodes_get_tag_name:
  "∀ w ∈ set_child_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀ r ∈ get_tag_name_locs ptr'. r h h'))"
  by(auto simp add: CD.set_child_nodes_locs_def set_child_nodes_locs_def CD.get_tag_name_locs_def
      all_args_def intro: element_put_get_preserved[where getter=tag_name and setter=RElement.child_nodes_update]
end

interpretation i_set_child_nodes_get_tag_name?: l_set_child_nodes_get_tag_name_Shadow_DOM type_wf known_ptr
  DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
  Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_tag_name get_tag_name_locs
  by(auto simp add: l_set_child_nodes_get_tag_name_Shadow_DOM_def instances)
declare l_set_child_nodes_get_tag_name_Shadow_DOM_axioms [instances]

lemma set_child_nodes_get_tag_name_is_l_set_child_nodes_get_tag_name [instances]:
  "l_set_child_nodes_get_tag_name type_wf set_child_nodes set_child_nodes_locs get_tag_name get_tag_name_locs"
  apply(auto simp add: l_set_child_nodes_get_tag_name_def l_set_child_nodes_get_tag_name_axioms_def instances)[1]
  using set_child_nodes_get_tag_name
  by fast
```

delete_shadow_root

```
locale l_delete_shadow_root_get_tag_name_Shadow_DOM =
  l_get_tag_name_Shadow_DOM
begin
```

```
lemma get_tag_name_delete_shadow_root: "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
  by(auto simp add: CD.get_tag_name_locs_def delete_shadow_root_get_M_Element)
end

locale l_delete_shadow_root_get_tag_name = l_get_tag_name_defs +
  assumes get_tag_name_delete_shadow_root: "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
interpretation l_delete_shadow_root_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  by(auto simp add: l_delete_shadow_root_get_tag_name_Shadow_DOM_def instances)

lemma l_delete_shadow_root_get_tag_name_get_tag_name_locs [instances]: "l_delete_shadow_root_get_tag_name
get_tag_name_locs"
  apply(auto simp add: l_delete_shadow_root_get_tag_name_def)[1]
  using get_tag_name_delete_shadow_root apply fast
  done
```

set_shadow_root

```
locale l_set_shadow_root_get_tag_name_Shadow_DOM =
  l_set_shadow_root_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_shadow_root_get_tag_name:
  "∀ w ∈ set_shadow_root_locs ptr. (h ⊢ w →_h h' ⟶ (∀ r ∈ get_tag_name_locs ptr'. r h h'))"
  by(auto simp add: set_shadow_root_locs_def CD.get_tag_name_locs_def all_args_def element_put_get_preserved[where
setter=shadow_root_opt_update])
end

interpretation
  i_set_shadow_root_get_tag_name?: l_set_shadow_root_get_tag_name_Shadow_DOM type_wf set_shadow_root
  set_shadow_root_locs DocumentClass.type_wf get_tag_name get_tag_name_locs
  apply(auto simp add: l_set_shadow_root_get_tag_name_Shadow_DOM_def instances)[1]
  by(unfold_locales)
declare l_set_shadow_root_get_tag_name_Shadow_DOM_axioms[instances]

locale l_set_shadow_root_get_tag_name = l_set_shadow_root_defs + l_get_tag_name_defs +
  assumes set_shadow_root_get_tag_name: "∀ w ∈ set_shadow_root_locs ptr. (h ⊢ w →_h h' ⟶ (∀ r ∈ get_tag_name_lo
ptr'. r h h'))"

lemma set_shadow_root_get_tag_name_is_l_set_shadow_root_get_tag_name [instances]:
  "l_set_shadow_root_get_tag_name set_shadow_root_locs get_tag_name_locs"
  using set_shadow_root_is_l_set_shadow_root get_tag_name_is_l_get_tag_name
  apply(simp add: l_set_shadow_root_get_tag_name_def )
  using set_shadow_root_get_tag_name
  by fast
```

new_element

```
locale l_new_element_get_tag_name_Shadow_DOM =
  l_get_tag_name_Shadow_DOM type_wf type_wf_Core_DOM get_tag_name get_tag_name_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ (_, tag_name) dom_prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_tag_name_new_element:
  "ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →_r new_element_ptr ⟹ h ⊢ new_element →_h h'
     ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
  by (auto simp add: CD.get_tag_name_locs_def new_element_get_M_Object new_element_get_M_Element
      new_element_get_M_Document split: prod.splits if_splits option.splits
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)

lemma new_element_empty_tag_name:
  "h ⊢ new_element →_r new_element_ptr ⟹ h ⊢ new_element →_h h'
   ⟹ h' ⊢ get_tag_name new_element_ptr →_r ''''"
```

```
  by(simp add: CD.get_tag_name_def new_element_tag_name)
end

locale l_new_element_get_tag_name = l_new_element + l_get_tag_name +
  assumes get_tag_name_new_element:
    "ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →r new_element_ptr
          ⟹ h ⊢ new_element →h h' ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
  assumes new_element_empty_tag_name:
    "h ⊢ new_element →r new_element_ptr ⟹ h ⊢ new_element →h h'
          ⟹ h' ⊢ get_tag_name new_element_ptr →r ''''"


interpretation i_new_element_get_tag_name?:
  l_new_element_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name get_tag_name_locs
  by(auto simp add: l_new_element_get_tag_name_Shadow_DOM_def instances)
declare l_new_element_get_tag_name_Shadow_DOM_axioms [instances]

lemma new_element_get_tag_name_is_l_new_element_get_tag_name [instances]:
  "l_new_element_get_tag_name type_wf get_tag_name get_tag_name_locs"
  using new_element_is_l_new_element get_tag_name_is_l_get_tag_name
  apply(auto simp add: l_new_element_get_tag_name_def l_new_element_get_tag_name_axioms_def instances)[1]
  using get_tag_name_new_element new_element_empty_tag_name
  by fast+
```

get_shadow_root

```
locale l_set_mode_get_tag_name_Shadow_DOM =
  l_set_mode_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_mode_get_tag_name:
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
  by(auto simp add: set_mode_locs_def CD.get_tag_name_locs_def all_args_def)
end

interpretation
  i_set_mode_get_tag_name?: l_set_mode_get_tag_name_Shadow_DOM type_wf
  set_mode set_mode_locs  DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  by unfold_locales
declare l_set_mode_get_tag_name_Shadow_DOM_axioms[instances]

locale l_set_mode_get_tag_name = l_set_mode + l_get_tag_name +
  assumes set_mode_get_tag_name:
    "∀w ∈ set_mode_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"

lemma set_mode_get_tag_name_is_l_set_mode_get_tag_name [instances]:
  "l_set_mode_get_tag_name type_wf set_mode set_mode_locs get_tag_name
                                    get_tag_name_locs"
  using set_mode_is_l_set_mode get_tag_name_is_l_get_tag_name
  apply(simp add: l_set_mode_get_tag_name_def
      l_set_mode_get_tag_name_axioms_def)
  using set_mode_get_tag_name
  by fast
```

new_document

```
locale l_new_document_get_tag_name_Shadow_DOM =
  l_get_tag_name_Shadow_DOM type_wf type_wf_Core_DOM get_tag_name get_tag_name_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, tag_name) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_tag_name_new_document:
  "h ⊢ new_document →r new_document_ptr ⟹ h ⊢ new_document →h h'
      ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
```

```
  by(auto simp add: CD.get_tag_name_locs_def new_document_get_M_Element)
end

locale l_new_document_get_tag_name = l_get_tag_name_defs +
  assumes get_tag_name_new_document:
    "h ⊢ new_document →r new_document_ptr ⟹ h ⊢ new_document →h h'
      ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"

interpretation i_new_document_get_tag_name?:
  l_new_document_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  by unfold_locales
declare l_new_document_get_tag_name_Shadow_DOM_def[instances]

lemma new_document_get_tag_name_is_l_new_document_get_tag_name [instances]:
  "l_new_document_get_tag_name get_tag_name_locs"
  unfolding l_new_document_get_tag_name_def
  unfolding get_tag_name_locs_def
  using new_document_get_M_Element by blast
```

new_shadow_root

```
locale l_new_shadow_root_get_tag_name_Shadow_DOM =
  l_get_tag_name_Shadow_DOM
begin
lemma get_tag_name_new_shadow_root:
  "h ⊢ new_ShadowRoot_M →r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →h h'
    ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
  by (auto simp add: CD.get_tag_name_locs_def new_shadow_root_get_M_Object new_shadow_root_get_M_Element
      split: prod.splits if_splits option.splits
      elim!: bind_returns_result_E bind_returns_heap_E intro: is_element_ptr_kind_obtains)
end

locale l_new_shadow_root_get_tag_name = l_get_tag_name +
  assumes get_tag_name_new_shadow_root:
    "h ⊢ new_ShadowRoot_M →r new_shadow_root_ptr
          ⟹ h ⊢ new_ShadowRoot_M →h h' ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"

interpretation i_new_shadow_root_get_tag_name?:
  l_new_shadow_root_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name get_tag_name_locs
  by(unfold_locales)
declare l_new_shadow_root_get_tag_name_Shadow_DOM_axioms [instances]

lemma new_shadow_root_get_tag_name_is_l_new_shadow_root_get_tag_name [instances]:
  "l_new_shadow_root_get_tag_name type_wf get_tag_name get_tag_name_locs"
  using get_tag_name_is_l_get_tag_name
  apply(auto simp add: l_new_shadow_root_get_tag_name_def l_new_shadow_root_get_tag_name_axioms_def instances)[1]
  using get_tag_name_new_shadow_root
  by fast
```

new_character_data

```
locale l_new_character_data_get_tag_name_Shadow_DOM =
  l_get_tag_name_Shadow_DOM type_wf type_wf_Core_DOM get_tag_name get_tag_name_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, tag_name) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_tag_name_new_character_data:
  "h ⊢ new_character_data →r new_character_data_ptr ⟹ h ⊢ new_character_data →h h'
      ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
  by(auto simp add: CD.get_tag_name_locs_def new_character_data_get_M_Element)
end

locale l_new_character_data_get_tag_name = l_get_tag_name_defs +
  assumes get_tag_name_new_character_data:
```

```
    "h ⊢ new_character_data →r new_character_data_ptr ⟹ h ⊢ new_character_data →h h'
      ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"

interpretation i_new_character_data_get_tag_name?:
  l_new_character_data_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  by unfold_locales
declare l_new_character_data_get_tag_name_Shadow_DOM_def[instances]

lemma new_character_data_get_tag_name_is_l_new_character_data_get_tag_name [instances]:
  "l_new_character_data_get_tag_name get_tag_name_locs"
  unfolding l_new_character_data_get_tag_name_def
  unfolding get_tag_name_locs_def
  using new_character_data_get_M_Element by blast
```

get_tag_type

```
locale l_set_tag_name_get_tag_name_Shadow_DOM = l_get_tag_name_Shadow_DOM
  + l_set_tag_name_Shadow_DOM
begin
lemma set_tag_name_get_tag_name:
  assumes "h ⊢ CD.a_set_tag_name element_ptr tag →h h'"
  shows "h' ⊢ CD.a_get_tag_name element_ptr →r tag"
  using assms
  by(auto simp add: CD.a_get_tag_name_def CD.a_set_tag_name_def)

lemma set_tag_name_get_tag_name_different_pointers:
  assumes "ptr ≠ ptr'"
  assumes "w ∈ CD.a_set_tag_name_locs ptr"
  assumes "h ⊢ w →h h'"
  assumes "r ∈ CD.a_get_tag_name_locs ptr'"
  shows "r h h'"
  using assms
  by(auto simp add: all_args_def CD.a_set_tag_name_locs_def CD.a_get_tag_name_locs_def
      split: if_splits option.splits )
end

interpretation i_set_tag_name_get_tag_name?:
  l_set_tag_name_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs set_tag_name set_tag_name_locs
  by unfold_locales
declare l_set_tag_name_get_tag_name_Shadow_DOM_axioms[instances]

lemma set_tag_name_get_tag_name_is_l_set_tag_name_get_tag_name [instances]:
  "l_set_tag_name_get_tag_name  type_wf get_tag_name get_tag_name_locs
                                                  set_tag_name set_tag_name_locs"
  using set_tag_name_is_l_set_tag_name get_tag_name_is_l_get_tag_name
  apply(simp add: l_set_tag_name_get_tag_name_def
      l_set_tag_name_get_tag_name_axioms_def)
  using set_tag_name_get_tag_name
    set_tag_name_get_tag_name_different_pointers
  by fast+
```

attach_shadow_root

```
locale l_attach_shadow_root_Shadow_DOM_defs =
  l_set_shadow_root_defs set_shadow_root set_shadow_root_locs +
  l_set_mode_defs set_mode set_mode_locs +
  l_get_tag_name_defs get_tag_name get_tag_name_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs
  for set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ ((_) heap, exception, unit) prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ ((_) heap, exception, unit) prog"
    and set_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_tag_name :: "(_) element_ptr ⇒ (_, char list) dom_prog"
```

```
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_attach_shadow_root :: "(_) element_ptr ⇒ shadow_root_mode ⇒ (_, (_) shadow_root_ptr) dom_prog"
  where
    "a_attach_shadow_root element_ptr shadow_root_mode = do {
      tag ← get_tag_name element_ptr;
      (if tag ∉ safe_shadow_root_element_types then error HierarchyRequestError else return ());
      prev_shadow_root ← get_shadow_root element_ptr;
      (if prev_shadow_root ≠ None then error HierarchyRequestError else return ());
      new_shadow_root_ptr ← new_ShadowRoot_M;
      set_mode new_shadow_root_ptr shadow_root_mode;
      set_shadow_root element_ptr (Some new_shadow_root_ptr);
      return new_shadow_root_ptr
    }"
end

locale l_attach_shadow_root_defs =
  fixes attach_shadow_root :: "(_) element_ptr ⇒ shadow_root_mode ⇒ (_, (_) shadow_root_ptr) dom_prog"

global_interpretation l_attach_shadow_root_Shadow_DOM_defs set_shadow_root set_shadow_root_locs set_mode
  set_mode_locs get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_locs
  defines attach_shadow_root = a_attach_shadow_root
  .

locale l_attach_shadow_root_Shadow_DOM =
  l_attach_shadow_root_Shadow_DOM_defs set_shadow_root set_shadow_root_locs set_mode set_mode_locs get_tag_name
get_tag_name_locs get_shadow_root get_shadow_root_locs +
  l_attach_shadow_root_defs attach_shadow_root +
  l_set_shadow_root_Shadow_DOM type_wf set_shadow_root set_shadow_root_locs +
  l_set_mode type_wf set_mode set_mode_locs +
  l_get_tag_name type_wf get_tag_name get_tag_name_locs +
  l_get_shadow_root type_wf get_shadow_root get_shadow_root_locs +
  l_known_ptr known_ptr
  for known_ptr :: "(_) object_ptr ⇒ bool"
    and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ ((_) heap, exception, unit)
prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ ((_) heap, exception, unit) prog"
    and set_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, unit) prog set"
    and attach_shadow_root :: "(_) element_ptr ⇒ shadow_root_mode ⇒ ((_) heap, exception, (_) shadow_root_ptr)
prog"
    and type_wf :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ (_, char list) dom_prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
  assumes attach_shadow_root_impl: "attach_shadow_root = a_attach_shadow_root"
begin
lemmas attach_shadow_root_def = a_attach_shadow_root_def[folded attach_shadow_root_impl]

lemma attach_shadow_root_element_ptr_in_heap:
  assumes "h ⊢ ok (attach_shadow_root element_ptr shadow_root_mode)"
  shows "element_ptr |∈| element_ptr_kinds h"
proof -
  obtain h' where "h ⊢ attach_shadow_root element_ptr shadow_root_mode →_h h'"
    using assms by auto
  then
  obtain h2 h3 new_shadow_root_ptr where
    h2: "h ⊢ new_ShadowRoot_M →_h h2" and
    new_shadow_root_ptr: "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr" and
```

```
    h3: "h2 ⊢ set_mode new_shadow_root_ptr shadow_root_mode →h h3" and
    "h3 ⊢ set_shadow_root element_ptr (Some new_shadow_root_ptr) →h h'"
    by(auto simp add: attach_shadow_root_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_tag_name_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_shadow_root_pure, rotated] split: if_splits)

  then have "element_ptr |∈| element_ptr_kinds h3"
    using set_shadow_root_ptr_in_heap by blast

  moreover
  have "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
    using h2 newShadowRoot_M_new_ptr new_shadow_root_ptr by auto

  moreover
  have "object_ptr_kinds h2 = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
          OF set_mode_writes h3])
    using set_mode_pointers_preserved
      apply blast
    by (auto simp add: reflp_def transp_def)
  ultimately
  show ?thesis
    by (metis (no_types, lifting) cast_document_ptr_not_node_ptr(2) element_ptr_kinds_commutes
        finsertE funion_finsert_right node_ptr_kinds_commutes sup_bot.right_neutral)
qed

lemma create_shadow_root_known_ptr:
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →r new_shadow_root_ptr"
  shows "known_ptr (castshadow_root_ptr2object_ptr new_shadow_root_ptr)"
  using assms
  by(auto simp add: attach_shadow_root_def known_ptr_impl ShadowRootClass.a_known_ptr_def
      newShadowRoot_M_def newShadowRoot_def Let_def elim!: bind_returns_result_E)
end

locale l_attach_shadow_root = l_attach_shadow_root_defs

interpretation
  i_attach_shadow_root?: l_attach_shadow_rootShadow_DOM known_ptr set_shadow_root set_shadow_root_locs
  set_mode set_mode_locs attach_shadow_root type_wf get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_
  by(auto simp add: l_attach_shadow_rootShadow_DOM_def l_attach_shadow_rootShadow_DOM_axioms_def
      attach_shadow_root_def instances)
declare l_attach_shadow_rootShadow_DOM_axioms [instances]
```

get_parent

```
global_interpretation l_get_parentCore_DOM_defs get_child_nodes get_child_nodes_locs
  defines get_parent = "l_get_parentCore_DOM_defs.a_get_parent get_child_nodes"
    and get_parent_locs = "l_get_parentCore_DOM_defs.a_get_parent_locs get_child_nodes_locs"
  .

interpretation i_get_parent?: l_get_parentCore_DOM known_ptr type_wf get_child_nodes
  get_child_nodes_locs known_ptrs get_parent get_parent_locs
  by(simp add: l_get_parentCore_DOM_def l_get_parentCore_DOM_axioms_def get_parent_def
      get_parent_locs_def instances)
declare l_get_parentCore_DOM_axioms [instances]

lemma get_parent_is_l_get_parent [instances]: "l_get_parent type_wf known_ptr known_ptrs get_parent
get_parent_locs get_child_nodes get_child_nodes_locs"
  apply(simp add: l_get_parent_def l_get_parent_axioms_def instances)
  using get_parent_reads get_parent_ok get_parent_ptr_in_heap get_parent_pure get_parent_parent_in_heap
get_parent_child_dual
  using get_parent_reads_pointers
  by blast
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_get_parent_Shadow_DOM =
  l_set_disconnected_nodes_get_child_nodes
  + l_set_disconnected_nodes_Shadow_DOM
  + l_get_parent_Core_DOM
begin
lemma set_disconnected_nodes_get_parent [simp]: "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h'
⟶ (∀r ∈ get_parent_locs. r h h'))"
  by(auto simp add: get_parent_locs_def CD.set_disconnected_nodes_locs_def all_args_def)
end
interpretation i_set_disconnected_nodes_get_parent?:
  l_set_disconnected_nodes_get_parent_Shadow_DOM set_disconnected_nodes set_disconnected_nodes_locs
  get_child_nodes get_child_nodes_locs type_wf DocumentClass.type_wf known_ptr known_ptrs get_parent
  get_parent_locs
  by (simp add: l_set_disconnected_nodes_get_parent_Shadow_DOM_def instances)
declare l_set_disconnected_nodes_get_parent_Core_DOM_axioms[instances]

lemma set_disconnected_nodes_get_parent_is_l_set_disconnected_nodes_get_parent [instances]:
  "l_set_disconnected_nodes_get_parent set_disconnected_nodes_locs get_parent_locs"
  by(simp add: l_set_disconnected_nodes_get_parent_def)
```

get_root_node

```
global_interpretation l_get_root_node_Core_DOM_defs get_parent get_parent_locs
  defines get_root_node = "l_get_root_node_Core_DOM_defs.a_get_root_node get_parent"
    and get_root_node_locs = "l_get_root_node_Core_DOM_defs.a_get_root_node_locs get_parent_locs"
    and get_ancestors = "l_get_root_node_Core_DOM_defs.a_get_ancestors get_parent"
    and get_ancestors_locs = "l_get_root_node_Core_DOM_defs.a_get_ancestors_locs get_parent_locs"
  .
declare a_get_ancestors.simps [code]

interpretation i_get_root_node?: l_get_root_node_Core_DOM type_wf known_ptr known_ptrs get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_ancestors get_ancestors_locs get_root_node
  get_root_node_locs
  by(simp add: l_get_root_node_Core_DOM_def l_get_root_node_Core_DOM_axioms_def get_root_node_def
      get_root_node_locs_def get_ancestors_def get_ancestors_locs_def instances)
declare l_get_root_node_Core_DOM_axioms [instances]

lemma get_ancestors_is_l_get_ancestors [instances]: "l_get_ancestors get_ancestors"
  apply(auto simp add: l_get_ancestors_def)[1]
  using get_ancestors_ptr_in_heap apply fast
  using get_ancestors_ptr apply fast
  done

lemma get_root_node_is_l_get_root_node [instances]: "l_get_root_node get_root_node get_parent"
  by (simp add: l_get_root_node_def Shadow_DOM.i_get_root_node.get_root_node_no_parent)
```

get_root_node_si

```
locale l_get_root_node_si_Shadow_DOM_defs =
  l_get_parent_defs get_parent get_parent_locs +
  l_get_host_defs get_host get_host_locs
  for get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_::linorder) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
begin
partial_function (dom_prog) a_get_ancestors_si :: "(_::linorder) object_ptr ⇒ (_, (_) object_ptr list)
dom_prog"
  where
    "a_get_ancestors_si ptr = do {
      check_in_heap ptr;
      ancestors ← (case cast_object_ptr2node_ptr ptr of
        Some node_ptr ⇒ do {
```

```
        parent_ptr_opt ← get_parent node_ptr;
        (case parent_ptr_opt of
          Some parent_ptr ⇒ a_get_ancestors_si parent_ptr
        | None ⇒ return [])
      }
    | None ⇒ (case cast ptr of
        Some shadow_root_ptr ⇒ do {
          host ← get_host shadow_root_ptr;
          a_get_ancestors_si (cast host)
        } |
        None ⇒ return []));
    return (ptr # ancestors)
  }"

definition "a_get_ancestors_si_locs = get_parent_locs ∪ get_host_locs"

definition a_get_root_node_si :: "(_) object_ptr ⇒ (_, (_) object_ptr) dom_prog"
  where
    "a_get_root_node_si ptr = do {
      ancestors ← a_get_ancestors_si ptr;
      return (last ancestors)
    }"
definition "a_get_root_node_si_locs = a_get_ancestors_si_locs"
end

locale l_get_ancestors_si_defs =
  fixes get_ancestors_si :: "(_::linorder) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  fixes get_ancestors_si_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_root_node_si_defs =
  fixes get_root_node_si :: "(_) object_ptr ⇒ (_, (_) object_ptr) dom_prog"
  fixes get_root_node_si_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_root_node_si_Shadow_DOM =
  l_get_parent +
  l_get_host +
  l_get_root_node_si_Shadow_DOM_defs +
  l_get_ancestors_si_defs +
  l_get_root_node_si_defs +
  assumes get_ancestors_si_impl: "get_ancestors_si = a_get_ancestors_si"
  assumes get_ancestors_si_locs_impl: "get_ancestors_si_locs = a_get_ancestors_si_locs"
  assumes get_root_node_si_impl: "get_root_node_si = a_get_root_node_si"
  assumes get_root_node_si_locs_impl: "get_root_node_si_locs = a_get_root_node_si_locs"
begin
lemmas get_ancestors_si_def = a_get_ancestors_si.simps[folded get_ancestors_si_impl]
lemmas get_ancestors_si_locs_def = a_get_ancestors_si_locs_def[folded get_ancestors_si_locs_impl]
lemmas get_root_node_si_def = a_get_root_node_si_def[folded get_root_node_si_impl get_ancestors_si_impl]
lemmas get_root_node_si_locs_def =
  a_get_root_node_si_locs_def[folded get_root_node_si_locs_impl get_ancestors_si_locs_impl]

lemma get_ancestors_si_pure [simp]:
  "pure (get_ancestors_si ptr) h"
proof -
  have "∀ptr h h' x. h ⊢ get_ancestors_si ptr →r x ⟶ h ⊢ get_ancestors_si ptr →h h' ⟶ h = h'"
  proof (induct rule: a_get_ancestors_si.fixp_induct[folded get_ancestors_si_impl])
    case 1
    then show ?case
      by(rule admissible_dom_prog)
  next
    case 2
    then show ?case
      by simp
  next
```

```
    case (3 f)
    then show ?case
      using get_parent_pure get_host_pure
      apply(auto simp add: pure_returns_heap_eq pure_def split: option.splits
          elim!: bind_returns_heap_E bind_returns_result_E
          dest!: pure_returns_heap_eq[rotated, OF check_in_heap_pure])[1]
        apply (meson option.simps(3) returns_result_eq)
       apply(metis get_parent_pure pure_returns_heap_eq)
      apply(metis get_host_pure pure_returns_heap_eq)
      done
  qed
  then show ?thesis
    by (meson pure_eq_iff)
qed


lemma get_root_node_si_pure [simp]: "pure (get_root_node_si ptr) h"
  by(auto simp add: get_root_node_si_def bind_pure_I)


lemma get_ancestors_si_ptr_in_heap:
  assumes "h ⊢ ok (get_ancestors_si ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms
  by(auto simp add: get_ancestors_si_def check_in_heap_ptr_in_heap elim!: bind_is_OK_E
      dest: is_OK_returns_result_I)

lemma get_ancestors_si_ptr:
  assumes "h ⊢ get_ancestors_si ptr →r ancestors"
  shows "ptr ∈ set ancestors"
  using assms
  by(simp add: get_ancestors_si_def) (auto  elim!: bind_returns_result_E2 split: option.splits
      intro!: bind_pure_I)


lemma get_ancestors_si_never_empty:
  assumes "h ⊢ get_ancestors_si child →r ancestors"
  shows "ancestors ≠ []"
  using assms
  apply(simp add: get_ancestors_si_def)
  by(auto elim!: bind_returns_result_E2 split: option.splits)


lemma get_root_node_si_no_parent:
  "h ⊢ get_parent node_ptr →r None ⟹ h ⊢ get_root_node_si (cast node_ptr) →r cast node_ptr"
  apply(auto simp add: check_in_heap_def get_root_node_si_def get_ancestors_si_def
      intro!: bind_pure_returns_result_I )[1]
  using get_parent_ptr_in_heap by blast


lemma get_root_node_si_root_not_shadow_root:
  assumes "h ⊢ get_root_node_si ptr →r root"
  shows "¬ is_shadow_root_ptr_object_ptr root"
  using assms
proof(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)
  fix y
  assume "h ⊢ get_ancestors_si ptr →r y"
    and "is_shadow_root_ptr_object_ptr (last y)"
    and "root = last y"
  then
  show False
  proof(induct y arbitrary: ptr)
    case Nil
```

```
    then show ?case
      using assms(1) get_ancestors_si_never_empty by blast
  next
    case (Cons a x)
    then show ?case
      apply(auto simp add: get_ancestors_si_def[of ptr] elim!: bind_returns_result_E2
          split: option.splits if_splits)[1]
      using get_ancestors_si_never_empty apply blast
      using Cons.prems(2) apply auto[1]
      using ‹is_shadow_root_ptr_object_ptr (last y)› ‹root = last y› by auto
  qed
qed
end

global_interpretation l_get_root_node_si_Shadow_DOM_defs get_parent get_parent_locs get_host get_host_locs
  defines get_root_node_si = a_get_root_node_si
    and get_root_node_si_locs = a_get_root_node_si_locs
    and get_ancestors_si = a_get_ancestors_si
    and get_ancestors_si_locs = a_get_ancestors_si_locs
  .
declare a_get_ancestors_si.simps [code]

interpretation
  i_get_root_node_si?: l_get_root_node_si_Shadow_DOM type_wf known_ptr known_ptrs get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_host get_host_locs get_ancestors_si
  get_ancestors_si_locs get_root_node_si get_root_node_si_locs
  apply(auto simp add: l_get_root_node_si_Shadow_DOM_def l_get_root_node_si_Shadow_DOM_axioms_def instances)[1]
  by(auto simp add: get_root_node_si_def get_root_node_si_locs_def get_ancestors_si_def get_ancestors_si_locs_def)
declare l_get_root_node_si_Shadow_DOM_axioms[instances]

lemma get_ancestors_si_is_l_get_ancestors [instances]: "l_get_ancestors get_ancestors_si"
  unfolding l_get_ancestors_def
  using get_ancestors_si_pure get_ancestors_si_ptr get_ancestors_si_ptr_in_heap
  by blast

lemma get_root_node_si_is_l_get_root_node [instances]: "l_get_root_node get_root_node_si get_parent"
  apply(simp add: l_get_root_node_def)
  using get_root_node_si_no_parent
  by fast
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_get_ancestors_si_Core_DOM =
  l_set_disconnected_nodes_get_parent
  + l_get_root_node_si_Shadow_DOM
  + l_set_disconnected_nodes_Shadow_DOM
  + l_set_disconnected_nodes_get_host
begin
lemma set_disconnected_nodes_get_ancestors_si:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_ancestors_si_locs. r h h'))"
  by(auto simp add: get_parent_locs_def set_disconnected_nodes_locs_def
      set_disconnected_nodes_get_host get_ancestors_si_locs_def all_args_def)
end

locale l_set_disconnected_nodes_get_ancestors_si = l_set_disconnected_nodes_defs + l_get_ancestors_si_defs
+
  assumes set_disconnected_nodes_get_ancestors_si:
    "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_ancestors_si_locs. r h h'))"

interpretation
  i_set_disconnected_nodes_get_ancestors_si?: l_set_disconnected_nodes_get_ancestors_si_Core_DOM
  set_disconnected_nodes set_disconnected_nodes_locs get_parent get_parent_locs type_wf known_ptr
  known_ptrs get_child_nodes get_child_nodes_locs get_host get_host_locs get_ancestors_si
  get_ancestors_si_locs get_root_node_si get_root_node_si_locs DocumentClass.type_wf
```

```
  by (auto simp add: l_set_disconnected_nodes_get_ancestors_si_Core_DOM_def instances)
declare l_set_disconnected_nodes_get_ancestors_si_Core_DOM_axioms[instances]


lemma set_disconnected_nodes_get_ancestors_si_is_l_set_disconnected_nodes_get_ancestors_si [instances]:
  "l_set_disconnected_nodes_get_ancestors_si set_disconnected_nodes_locs get_ancestors_si_locs"
  using instances
  apply(simp add: l_set_disconnected_nodes_get_ancestors_si_def)
  using set_disconnected_nodes_get_ancestors_si
  by fast
```

get_attribute

```
lemma get_attribute_is_l_get_attribute [instances]: "l_get_attribute type_wf get_attribute get_attribute_locs"
  apply(auto simp add: l_get_attribute_def)[1]
  using i_get_attribute.get_attribute_reads apply fast
  using type_wf_Document i_get_attribute.get_attribute_ok apply blast
  using i_get_attribute.get_attribute_ptr_in_heap apply fast
  done
```

to_tree_order

```
global_interpretation l_to_tree_order_Core_DOM_defs get_child_nodes get_child_nodes_locs defines
  to_tree_order = "l_to_tree_order_Core_DOM_defs.a_to_tree_order get_child_nodes" .
declare a_to_tree_order.simps [code]

interpretation i_to_tree_order?: l_to_tree_order_Core_DOM ShadowRootClass.known_ptr
  ShadowRootClass.type_wf Shadow_DOM.get_child_nodes Shadow_DOM.get_child_nodes_locs to_tree_order
  by(auto simp add: l_to_tree_order_Core_DOM_def l_to_tree_order_Core_DOM_axioms_def to_tree_order_def
instances)
declare l_to_tree_order_Core_DOM_axioms [instances]
```

to_tree_order_si

```
locale l_to_tree_order_si_Core_DOM_defs =
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
partial_function (dom_prog) a_to_tree_order_si :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  where
    "a_to_tree_order_si ptr = (do {
      children ← get_child_nodes ptr;
      shadow_root_part ← (case cast ptr of
        Some element_ptr ⇒ do {
          shadow_root_opt ← get_shadow_root element_ptr;
          (case shadow_root_opt of
            Some shadow_root_ptr ⇒ return [cast shadow_root_ptr]
            | None ⇒ return [])
        } |
        None ⇒ return []);
      treeorders ← map_M a_to_tree_order_si ((map (cast) children) @ shadow_root_part);
      return (ptr # concat treeorders)
    })"
end

locale l_to_tree_order_si_defs =
  fixes to_tree_order_si :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"

global_interpretation l_to_tree_order_si_Core_DOM_defs get_child_nodes get_child_nodes_locs
  get_shadow_root get_shadow_root_locs
```

```
  defines to_tree_order_si = "a_to_tree_order_si" .
declare a_to_tree_order_si.simps [code]


locale l_to_tree_order_si_Shadow_DOM =
  l_to_tree_order_si_defs +
  l_to_tree_order_si_Core_DOM_defs +
  l_get_child_nodes +
  l_get_shadow_root +
  assumes to_tree_order_si_impl: "to_tree_order_si = a_to_tree_order_si"
begin
lemmas to_tree_order_si_def = a_to_tree_order_si.simps[folded to_tree_order_si_impl]

lemma to_tree_order_si_pure [simp]: "pure (to_tree_order_si ptr) h"
proof -
  have "∀ptr h h' x. h ⊢ to_tree_order_si ptr →r x ⟶ h ⊢ to_tree_order_si ptr →h h' ⟶ h = h'"
  proof (induct rule: a_to_tree_order_si.fixp_induct[folded to_tree_order_si_impl])
    case 1
    then show ?case
      by (rule admissible_dom_prog)
  next
    case 2
    then show ?case
      by simp
  next
    case (3 f)
    then have "⋀x h. pure (f x) h"
      by (metis is_OK_returns_heap_E is_OK_returns_result_E pure_def)
    then have "⋀xs h. pure (map_M f xs) h"
      by(rule map_M_pure_I)
    then show ?case
      by(auto elim!: bind_returns_heap_E2 split: option.splits)
  qed
  then show ?thesis
    unfolding pure_def
    by (metis is_OK_returns_heap_E is_OK_returns_result_E)
qed
end

interpretation i_to_tree_order_si?: l_to_tree_order_si_Shadow_DOM to_tree_order_si get_child_nodes
  get_child_nodes_locs get_shadow_root get_shadow_root_locs type_wf known_ptr
  by(auto simp add: l_to_tree_order_si_Shadow_DOM_def l_to_tree_order_si_Shadow_DOM_axioms_def
      to_tree_order_si_def instances)
declare l_to_tree_order_si_Shadow_DOM_axioms [instances]
```

first_in_tree_order

```
global_interpretation l_first_in_tree_order_Core_DOM_defs to_tree_order defines
  first_in_tree_order = "l_first_in_tree_order_Core_DOM_defs.a_first_in_tree_order to_tree_order" .

interpretation i_first_in_tree_order?: l_first_in_tree_order_Core_DOM to_tree_order first_in_tree_order
  by(auto simp add: l_first_in_tree_order_Core_DOM_def first_in_tree_order_def)
declare l_first_in_tree_order_Core_DOM_axioms [instances]

lemma to_tree_order_is_l_to_tree_order [instances]: "l_to_tree_order to_tree_order"
  by(auto simp add: l_to_tree_order_def)
```

first_in_tree_order

```
global_interpretation l_dummy defines
  first_in_tree_order_si = "l_first_in_tree_order_Core_DOM_defs.a_first_in_tree_order to_tree_order_si"
  .
```

get_element_by

```
global_interpretation l_get_element_by_Core_DOM_defs to_tree_order first_in_tree_order get_attribute get_attribute
defines
  get_element_by_id = "l_get_element_by_Core_DOM_defs.a_get_element_by_id first_in_tree_order get_attribute"
and
  get_elements_by_class_name = "l_get_element_by_Core_DOM_defs.a_get_elements_by_class_name to_tree_order
get_attribute" and
  get_elements_by_tag_name = "l_get_element_by_Core_DOM_defs.a_get_elements_by_tag_name to_tree_order" .

interpretation
  i_get_element_by?: l_get_element_by_Core_DOM to_tree_order first_in_tree_order get_attribute
  get_attribute_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name type_wf
  by(auto simp add: l_get_element_by_Core_DOM_def l_get_element_by_Core_DOM_axioms_def get_element_by_id_def
      get_elements_by_class_name_def get_elements_by_tag_name_def instances)
declare l_get_element_by_Core_DOM_axioms[instances]


lemma get_element_by_is_l_get_element_by [instances]: "l_get_element_by get_element_by_id get_elements_by_tag_nam
to_tree_order"
  apply(auto simp add: l_get_element_by_def)[1]
  using get_element_by_id_result_in_tree_order apply fast
  done
```

get_element_by_si

```
global_interpretation l_dummy defines
  get_element_by_id_si = "l_get_element_by_Core_DOM_defs.a_get_element_by_id first_in_tree_order_si get_attribute"
and
  get_elements_by_class_name_si = "l_get_element_by_Core_DOM_defs.a_get_elements_by_class_name to_tree_order_si
get_attribute" and
  get_elements_by_tag_name_si = "l_get_element_by_Core_DOM_defs.a_get_elements_by_tag_name to_tree_order_si"
  .
```

find_slot

```
locale l_find_slot_Shadow_DOM_defs =
  l_get_parent_defs get_parent get_parent_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_get_mode_defs get_mode get_mode_locs +
  l_get_attribute_defs get_attribute get_attribute_locs +
  l_get_tag_name_defs get_tag_name get_tag_name_locs +
  l_first_in_tree_order_defs first_in_tree_order
  for get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_::linorder) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_mode :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, shadow_root_mode) prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_attribute :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, char list option) prog"
    and get_attribute_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and first_in_tree_order ::
    "(_) object_ptr ⇒ ((_) object_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog) ⇒
((_) heap, exception, (_) element_ptr option) prog"
begin
definition a_find_slot :: "bool ⇒ (_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"
  where
    "a_find_slot open_flag slotable = do {
      parent_opt ← get_parent slotable;
      (case parent_opt of
        Some parent ⇒
```

```
        if is_element_ptr_kind parent
        then do {
          shadow_root_ptr_opt ← get_shadow_root (the (cast parent));
          (case shadow_root_ptr_opt of
            Some shadow_root_ptr ⇒ do {
              shadow_root_mode ← get_mode shadow_root_ptr;
              if open_flag ∧ shadow_root_mode ≠ Open
              then return None
              else first_in_tree_order (cast shadow_root_ptr) (λptr. if is_element_ptr_kind ptr
                then do {
                  tag ← get_tag_name (the (cast ptr));
                  name_attr ← get_attribute (the (cast ptr)) ''name'';
                  slotable_name_attr ← (if is_element_ptr_kind slotable
                    then get_attribute (the (cast slotable)) ''slot'' else return None);
                  (if (tag = ''slot'' ∧ (name_attr = slotable_name_attr ∨
                    (name_attr = None ∧ slotable_name_attr = Some '''') ∨
                    (name_attr = Some '''' ∧ slotable_name_attr = None)))
                  then return (Some (the (cast ptr)))
                  else return None)}
                else return None)}
            | None ⇒ return None)}
        else return None
    | _ ⇒ return None)}"

definition a_assigned_slot :: "(_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"
  where
    "a_assigned_slot = a_find_slot True"
end

global_interpretation l_find_slot_Shadow_DOM_defs get_parent get_parent_locs get_shadow_root
  get_shadow_root_locs get_mode get_mode_locs get_attribute get_attribute_locs get_tag_name
  get_tag_name_locs first_in_tree_order
  defines find_slot = a_find_slot
    and assigned_slot = a_assigned_slot
  .

locale l_find_slot_defs =
  fixes find_slot :: "bool ⇒ (_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"
    and assigned_slot :: "(_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"

locale l_find_slot_Shadow_DOM =
  l_find_slot_Shadow_DOM_defs +
  l_find_slot_defs +
  l_get_parent +
  l_get_shadow_root +
  l_get_mode +
  l_get_attribute +
  l_get_tag_name +
  l_to_tree_order +
  l_first_in_tree_order_Core_DOM +
  assumes find_slot_impl: "find_slot = a_find_slot"
  assumes assigned_slot_impl: "assigned_slot = a_assigned_slot"
begin
lemmas find_slot_def = find_slot_impl[unfolded a_find_slot_def]
lemmas assigned_slot_def = assigned_slot_impl[unfolded a_assigned_slot_def]

lemma find_slot_ptr_in_heap:
  assumes "h ⊢ find_slot open_flag slotable →r slot_opt"
  shows "slotable |∈| node_ptr_kinds h"
  using assms
  apply(auto simp add: find_slot_def elim!: bind_returns_result_E2)[1]
  using get_parent_ptr_in_heap by blast
```

```
lemma find_slot_slot_in_heap:
  assumes "h ⊢ find_slot open_flag slotable →r Some slot"
  shows "slot |∈| element_ptr_kinds h"
  using assms
  apply(auto simp add: find_slot_def first_in_tree_order_def elim!: bind_returns_result_E2
      map_filter_M_pure_E[where y=slot] split: option.splits if_splits list.splits intro!: map_filter_M_pure
      bind_pure_I)[1]
  using get_tag_name_ptr_in_heap by blast+

lemma find_slot_pure [simp]: "pure (find_slot open_flag slotable) h"
  by(auto simp add: find_slot_def first_in_tree_order_def  intro!: bind_pure_I map_filter_M_pure
      split: option.splits list.splits)
end

interpretation i_find_slot?: l_find_slot_Shadow_DOM get_parent get_parent_locs get_shadow_root
  get_shadow_root_locs get_mode get_mode_locs get_attribute get_attribute_locs get_tag_name
  get_tag_name_locs first_in_tree_order find_slot assigned_slot type_wf known_ptr known_ptrs
  get_child_nodes get_child_nodes_locs to_tree_order
  by (auto simp add: find_slot_def assigned_slot_def l_find_slot_Shadow_DOM_def
      l_find_slot_Shadow_DOM_axioms_def instances)
declare l_find_slot_Shadow_DOM_axioms [instances]

locale l_find_slot = l_find_slot_defs +
  assumes find_slot_ptr_in_heap: "h ⊢ find_slot open_flag slotable →r slot_opt ⟹ slotable |∈| node_ptr_kinds
h"
  assumes find_slot_slot_in_heap: "h ⊢ find_slot open_flag slotable →r Some slot ⟹ slot |∈| element_ptr_kinds
h"
  assumes find_slot_pure [simp]: "pure (find_slot open_flag slotable) h"

lemma find_slot_is_l_find_slot [instances]: "l_find_slot find_slot"
  apply(auto simp add: l_find_slot_def)[1]
  using find_slot_ptr_in_heap apply fast
  using find_slot_slot_in_heap apply fast
  done
```

get_disconnected_nodes

```
locale l_get_disconnected_nodes_Shadow_DOM =
  CD: l_get_disconnected_nodes_Core_DOM type_wf_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs
+
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin

lemma get_disconnected_nodes_ok:
  "type_wf h ⟹ document_ptr |∈| document_ptr_kinds h ⟹ h ⊢ ok (get_disconnected_nodes document_ptr)"
  apply(unfold type_wf_impl get_disconnected_nodes_impl[unfolded a_get_disconnected_nodes_def])
  using CD.get_disconnected_nodes_ok CD.type_wf_impl ShadowRootClass.type_wf_Document
  by blast
end

interpretation i_get_disconnected_nodes?: l_get_disconnected_nodes_Shadow_DOM type_wf
  DocumentClass.type_wf get_disconnected_nodes get_disconnected_nodes_locs
  by(auto simp add: l_get_disconnected_nodes_Shadow_DOM_def l_get_disconnected_nodes_Shadow_DOM_axioms_def
instances)
declare l_get_disconnected_nodes_Shadow_DOM_axioms [instances]

lemma get_disconnected_nodes_is_l_get_disconnected_nodes [instances]:
  "l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs"
```

```
  apply(auto simp add: l_get_disconnected_nodes_def)[1]
  using i_get_disconnected_nodes.get_disconnected_nodes_reads apply fast
  using get_disconnected_nodes_ok apply fast
  using i_get_disconnected_nodes.get_disconnected_nodes_ptr_in_heap apply fast
  done
```

set_child_nodes

```
locale l_set_child_nodes_get_disconnected_nodes_Shadow_DOM =
  l_set_child_nodes_Shadow_DOM +
  l_get_disconnected_nodes_Shadow_DOM
begin
lemma set_child_nodes_get_disconnected_nodes:
  "∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
  by(auto simp add: set_child_nodes_locs_def CD.set_child_nodes_locs_def
      CD.get_disconnected_nodes_locs_def all_args_def elim: get_M_document_put_M_shadow_root
      split: option.splits)
end

interpretation
  i_set_child_nodes_get_disconnected_nodes?: l_set_child_nodes_get_disconnected_nodes_Shadow_DOM type_wf
  known_ptr DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
  Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  apply(auto simp add: l_set_child_nodes_get_disconnected_nodes_Shadow_DOM_def instances)[1]
  by(unfold_locales)
declare l_set_child_nodes_get_disconnected_nodes_Shadow_DOM_axioms[instances]

lemma set_child_nodes_get_disconnected_nodes_is_l_set_child_nodes_get_disconnected_nodes [instances]:
  "l_set_child_nodes_get_disconnected_nodes type_wf set_child_nodes set_child_nodes_locs
get_disconnected_nodes get_disconnected_nodes_locs"
  using set_child_nodes_is_l_set_child_nodes get_disconnected_nodes_is_l_get_disconnected_nodes
  apply(simp add: l_set_child_nodes_get_disconnected_nodes_def l_set_child_nodes_get_disconnected_nodes_axioms_def
)
  using set_child_nodes_get_disconnected_nodes
  by fast
```

set_disconnected_nodes

```
lemma set_disconnected_nodes_get_disconnected_nodes_l_set_disconnected_nodes_get_discon
[instances]:
  "l_set_disconnected_nodes_get_disconnected_nodes ShadowRootClass.type_wf get_disconnected_nodes
get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs"
  apply(auto simp add: l_set_disconnected_nodes_get_disconnected_nodes_def
      l_set_disconnected_nodes_get_disconnected_nodes_axioms_def instances)[1]
  using i_set_disconnected_nodes_get_disconnected_nodes.set_disconnected_nodes_get_disconnected_nodes
   apply fast
  using i_set_disconnected_nodes_get_disconnected_nodes.set_disconnected_nodes_get_disconnected_nodes_different_poi
  apply fast
  done
```

delete_shadow_root

```
locale l_delete_shadow_root_get_disconnected_nodes_Shadow_DOM =
  l_get_disconnected_nodes_Shadow_DOM
begin
lemma get_disconnected_nodes_delete_shadow_root:
  "cast shadow_root_ptr ≠ ptr' ⟹ h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
  by(auto simp add: CD.get_disconnected_nodes_locs_def delete_shadow_root_get_M_Document)
end

locale l_delete_shadow_root_get_disconnected_nodes = l_get_disconnected_nodes_defs +
  assumes get_disconnected_nodes_delete_shadow_root:
    "cast shadow_root_ptr ≠ ptr' ⟹ h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
interpretation l_delete_shadow_root_get_disconnected_nodes_Shadow_DOM type_wf DocumentClass.type_wf
  get_disconnected_nodes get_disconnected_nodes_locs
```

```
  by(auto simp add: l_delete_shadow_root_get_disconnected_nodes_Shadow_DOM_def instances)

lemma l_delete_shadow_root_get_disconnected_nodes_get_disconnected_nodes_locs [instances]: "l_delete_shadow_root_
get_disconnected_nodes_locs"
  apply(auto simp add: l_delete_shadow_root_get_disconnected_nodes_def)[1]
  using get_disconnected_nodes_delete_shadow_root apply fast
  done
```

set_shadow_root

```
locale l_set_shadow_root_get_disconnected_nodes_Shadow_DOM =
  l_set_shadow_root_Shadow_DOM +
  l_get_disconnected_nodes_Shadow_DOM
begin
lemma set_shadow_root_get_disconnected_nodes:
  "∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
  by(auto simp add: set_shadow_root_locs_def CD.get_disconnected_nodes_locs_def all_args_def)
end

interpretation
  i_set_shadow_root_get_disconnected_nodes?: l_set_shadow_root_get_disconnected_nodes_Shadow_DOM type_wf
  set_shadow_root set_shadow_root_locs DocumentClass.type_wf get_disconnected_nodes get_disconnected_nodes_locs
  apply(auto simp add: l_set_shadow_root_get_disconnected_nodes_Shadow_DOM_def instances)[1]
  by(unfold_locales)
declare l_set_shadow_root_get_disconnected_nodes_Shadow_DOM_axioms[instances]

locale l_set_shadow_root_get_disconnected_nodes = l_set_shadow_root_defs + l_get_disconnected_nodes_defs
+
  assumes set_shadow_root_get_disconnected_nodes:
    "∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"

lemma set_shadow_root_get_disconnected_nodes_is_l_set_shadow_root_get_disconnected_nodes [instances]:
  "l_set_shadow_root_get_disconnected_nodes set_shadow_root_locs get_disconnected_nodes_locs"
  using set_shadow_root_is_l_set_shadow_root get_disconnected_nodes_is_l_get_disconnected_nodes
  apply(simp add: l_set_shadow_root_get_disconnected_nodes_def )
  using set_shadow_root_get_disconnected_nodes
  by fast
```

set_mode

```
locale l_set_mode_get_disconnected_nodes_Shadow_DOM =
  l_set_mode_Shadow_DOM +
  l_get_disconnected_nodes_Shadow_DOM
begin
lemma set_mode_get_disconnected_nodes:
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
  by(auto simp add: set_mode_locs_def
      CD.get_disconnected_nodes_locs_impl[unfolded CD.a_get_disconnected_nodes_locs_def]
      all_args_def)
end

interpretation
  i_set_mode_get_disconnected_nodes?: l_set_mode_get_disconnected_nodes_Shadow_DOM type_wf
  set_mode set_mode_locs DocumentClass.type_wf  get_disconnected_nodes
  get_disconnected_nodes_locs
  by unfold_locales
declare l_set_mode_get_disconnected_nodes_Shadow_DOM_axioms[instances]

locale l_set_mode_get_disconnected_nodes = l_set_mode + l_get_disconnected_nodes +
  assumes set_mode_get_disconnected_nodes:
    "∀w ∈ set_mode_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"

lemma set_mode_get_disconnected_nodes_is_l_set_mode_get_disconnected_nodes [instances]:
  "l_set_mode_get_disconnected_nodes type_wf set_mode set_mode_locs get_disconnected_nodes
                                     get_disconnected_nodes_locs"
  using set_mode_is_l_set_mode get_disconnected_nodes_is_l_get_disconnected_nodes
  apply(simp add: l_set_mode_get_disconnected_nodes_def
```

```
    l_set_mode_get_disconnected_nodes_axioms_def)
  using set_mode_get_disconnected_nodes
  by fast
```

new_shadow_root

```
locale l_new_shadow_root_get_disconnected_nodes_Shadow_DOM =
  l_get_disconnected_nodes_Shadow_DOM type_wf type_wf_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_disconnected_nodes_new_shadow_root_different_pointers:
  "cast new_shadow_root_ptr ≠ ptr' ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M
→_h h'
    ⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
  by(auto simp add: CD.get_disconnected_nodes_locs_def new_shadow_root_get_M_Document)

lemma new_shadow_root_no_disconnected_nodes:
  "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
    ⟹ h' ⊢ get_disconnected_nodes (cast new_shadow_root_ptr) →_r []"
  by(simp add: CD.get_disconnected_nodes_def new_shadow_root_disconnected_nodes)

end

interpretation i_new_shadow_root_get_disconnected_nodes?:
  l_new_shadow_root_get_disconnected_nodes_Shadow_DOM type_wf DocumentClass.type_wf get_disconnected_nodes
  get_disconnected_nodes_locs
  by unfold_locales
declare l_new_shadow_root_get_disconnected_nodes_Shadow_DOM_axioms[instances]

locale l_new_shadow_root_get_disconnected_nodes = l_get_disconnected_nodes_defs +
  assumes get_disconnected_nodes_new_shadow_root_different_pointers:
    "cast new_shadow_root_ptr ≠ ptr' ⟹ h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M
→_h h'
    ⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
  assumes new_shadow_root_no_disconnected_nodes:
    "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
    ⟹ h' ⊢ get_disconnected_nodes (cast new_shadow_root_ptr) →_r []"

lemma new_shadow_root_get_disconnected_nodes_is_l_new_shadow_root_get_disconnected_nodes [instances]:
  "l_new_shadow_root_get_disconnected_nodes get_disconnected_nodes get_disconnected_nodes_locs"
  apply (auto simp add: l_new_shadow_root_get_disconnected_nodes_def)[1]
  using get_disconnected_nodes_new_shadow_root_different_pointers apply fast
  using new_shadow_root_no_disconnected_nodes apply blast
  done
```

remove_shadow_root

```
locale l_remove_shadow_root_Shadow_DOM_defs =
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_set_shadow_root_defs set_shadow_root set_shadow_root_locs +
  l_get_disconnected_nodes_defs get_disconnected_nodes get_disconnected_nodes_locs
  for get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ ((_) heap, exception, unit)
prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
```

```
definition a_remove_shadow_root :: "(_) element_ptr ⇒ (_, unit) dom_prog" where
  "a_remove_shadow_root element_ptr = do {
    shadow_root_ptr_opt ← get_shadow_root element_ptr;
    (case shadow_root_ptr_opt of
      Some shadow_root_ptr ⇒ do {
        children ← get_child_nodes (cast shadow_root_ptr);
        disconnected_nodes ← get_disconnected_nodes (cast shadow_root_ptr);
        (if children = [] ∧ disconnected_nodes = []
        then do {
          set_shadow_root element_ptr None;
          delete_M shadow_root_ptr
        } else do {
          error HierarchyRequestError
        })
      } |
      None ⇒ error HierarchyRequestError)
    }"

definition a_remove_shadow_root_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_, unit) dom_prog) set"
  where
    "a_remove_shadow_root_locs element_ptr shadow_root_ptr ≡ set_shadow_root_locs element_ptr ∪ {delete_M
shadow_root_ptr}"
end

global_interpretation l_remove_shadow_root_Shadow_DOM_defs get_child_nodes get_child_nodes_locs
  get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  defines remove_shadow_root = "a_remove_shadow_root"
    and remove_shadow_root_locs = a_remove_shadow_root_locs
  .

locale l_remove_shadow_root_defs =
  fixes remove_shadow_root :: "(_) element_ptr ⇒ (_, unit) dom_prog"
  fixes remove_shadow_root_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_, unit) dom_prog) set"


locale l_remove_shadow_root_Shadow_DOM =
  l_remove_shadow_root_Shadow_DOM_defs +
  l_remove_shadow_root_defs +
  l_get_shadow_root_Shadow_DOM +
  l_set_shadow_root_Shadow_DOM +
  l_get_child_nodes +
  l_get_disconnected_nodes +
  assumes remove_shadow_root_impl: "remove_shadow_root = a_remove_shadow_root"
  assumes remove_shadow_root_locs_impl: "remove_shadow_root_locs = a_remove_shadow_root_locs"
begin
lemmas remove_shadow_root_def =
  remove_shadow_root_impl[unfolded remove_shadow_root_def a_remove_shadow_root_def]
lemmas remove_shadow_root_locs_def =
  remove_shadow_root_locs_impl[unfolded remove_shadow_root_locs_def a_remove_shadow_root_locs_def]

lemma remove_shadow_root_writes:
  "writes (remove_shadow_root_locs element_ptr (the |h ⊢ get_shadow_root element_ptr|_r))
(remove_shadow_root element_ptr) h h'"
  apply(auto simp add: remove_shadow_root_locs_def remove_shadow_root_def all_args_def
      writes_union_right_I writes_union_left_I set_shadow_root_writes
      intro!: writes_bind writes_bind_pure[OF get_shadow_root_pure] writes_bind_pure[OF get_child_nodes_pure]
      intro: writes_subset[OF set_shadow_root_writes] writes_subset[OF writes_singleton2] split: option.splits)[1]
  using  writes_union_left_I[OF set_shadow_root_writes]
   apply (metis inf_sup_aci(5) insert_is_Un)
  using writes_union_right_I[OF writes_singleton[of delete_ShadowRoot_M]]
  by (smt insert_is_Un writes_singleton2 writes_union_left_I)
```

```
end

interpretation i_remove_shadow_root?: l_remove_shadow_root_Shadow_DOM get_child_nodes get_child_nodes_locs
  get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes
  get_disconnected_nodes_locs remove_shadow_root remove_shadow_root_locs type_wf known_ptr
  by(auto simp add: l_remove_shadow_root_Shadow_DOM_def l_remove_shadow_root_Shadow_DOM_axioms_def
      remove_shadow_root_def remove_shadow_root_locs_def instances)
declare l_remove_shadow_root_Shadow_DOM_axioms [instances]
```

get_child_nodes

```
locale l_remove_shadow_root_get_child_nodes_Shadow_DOM =
  l_get_child_nodes_Shadow_DOM +
  l_remove_shadow_root_Shadow_DOM
begin

lemma remove_shadow_root_get_child_nodes_different_pointers:
  assumes "ptr ≠ cast shadow_root_ptr"
  assumes "w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr"
  assumes "h ⊢ w →_h h'"
  assumes "r ∈ get_child_nodes_locs ptr"
  shows "r h h'"
  using assms
  by(auto simp add: all_args_def get_child_nodes_locs_def CD.get_child_nodes_locs_def
      remove_shadow_root_locs_def set_shadow_root_locs_def
      delete_shadow_root_get_M_Object delete_shadow_root_get_M_ShadowRoot
      intro: is_shadow_root_ptr_kind_obtains
      simp add: delete_shadow_root_get_M_ShadowRoot delete_shadow_root_get_M_Element
      delete_shadow_root_get_M_Document[rotated] element_put_get_preserved[where setter=shadow_root_opt_update]
      elim: is_document_ptr_kind_obtains is_shadow_root_ptr_kind_obtains
      split: if_splits option.splits)[1]
end

locale l_remove_shadow_root_get_child_nodes = l_get_child_nodes_defs + l_remove_shadow_root_defs +
  assumes remove_shadow_root_get_child_nodes_different_pointers:
    "ptr ≠ cast shadow_root_ptr ⟹ w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr ⟹ h ⊢ w
→_h h' ⟹
r ∈ get_child_nodes_locs ptr ⟹ r h h'"

interpretation
  i_remove_shadow_root_get_child_nodes?: l_remove_shadow_root_get_child_nodes_Shadow_DOM type_wf
  known_ptr DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes get_child_nodes_locs
  Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs get_shadow_root
  get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes get_disconnected_nodes_locs
  remove_shadow_root remove_shadow_root_locs
  by(auto simp add: l_remove_shadow_root_get_child_nodes_Shadow_DOM_def instances)
declare l_remove_shadow_root_get_child_nodes_Shadow_DOM_axioms[instances]

lemma remove_shadow_root_get_child_nodes_is_l_remove_shadow_root_get_child_nodes [instances]:
  "l_remove_shadow_root_get_child_nodes get_child_nodes_locs remove_shadow_root_locs"
  apply(auto simp add: l_remove_shadow_root_get_child_nodes_def instances )[1]
  using remove_shadow_root_get_child_nodes_different_pointers apply fast
  done
```

get_tag_name

```
locale l_remove_shadow_root_get_tag_name_Shadow_DOM =
  l_get_tag_name_Shadow_DOM +
  l_remove_shadow_root_Shadow_DOM
begin

lemma remove_shadow_root_get_tag_name:
  assumes "w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr"
  assumes "h ⊢ w →_h h'"
  assumes "r ∈ get_tag_name_locs ptr"
  shows "r h h'"
  using assms
```

```
  by(auto simp add: all_args_def remove_shadow_root_locs_def set_shadow_root_locs_def
      CD.get_tag_name_locs_def delete_shadow_root_get_M_Element
      element_put_get_preserved[where setter=shadow_root_opt_update] split: if_splits option.splits)
end

locale l_remove_shadow_root_get_tag_name = l_get_tag_name_defs + l_remove_shadow_root_defs +
  assumes remove_shadow_root_get_tag_name:
    "w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr ⟹ h ⊢ w →_h h' ⟹ r ∈ get_tag_name_locs
ptr ⟹
r h h'"

interpretation
  i_remove_shadow_root_get_tag_name?: l_remove_shadow_root_get_tag_name_Shadow_DOM type_wf
  DocumentClass.type_wf get_tag_name get_tag_name_locs get_child_nodes get_child_nodes_locs
  get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes
  get_disconnected_nodes_locs remove_shadow_root remove_shadow_root_locs known_ptr
  by(auto simp add: l_remove_shadow_root_get_tag_name_Shadow_DOM_def instances)
declare l_remove_shadow_root_get_tag_name_Shadow_DOM_axioms[instances]

lemma remove_shadow_root_get_tag_name_is_l_remove_shadow_root_get_tag_name [instances]:
  "l_remove_shadow_root_get_tag_name get_tag_name_locs remove_shadow_root_locs"
  apply(auto simp add: l_remove_shadow_root_get_tag_name_def instances )[1]
  using remove_shadow_root_get_tag_name apply fast
  done
```

get_owner_document

```
locale l_get_owner_document_Shadow_DOM_defs =
  l_get_host_defs get_host get_host_locs +
  CD: l_get_owner_document_Core_DOM_defs get_root_node get_root_node_locs get_disconnected_nodes get_disconnected_
  for get_root_node :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_owner_document_shadow_root_ptr :: "(_) shadow_root_ptr ⇒ unit ⇒ (_, (_) document_ptr) dom_prog"
  where
    "a_get_owner_document_shadow_root_ptr shadow_root_ptr = CD.a_get_owner_document_document_ptr (cast shadow_root_ptr)

definition a_get_owner_document_tups :: "(((_) object_ptr ⇒ bool) × ((_) object_ptr ⇒ unit
  ⇒ (_, (_) document_ptr) dom_prog)) list"
  where
    "a_get_owner_document_tups = [(is_shadow_root_ptr, a_get_owner_document_shadow_root_ptr ∘ the ∘ cast)]"

definition a_get_owner_document :: "(_::linorder) object_ptr ⇒ (_, (_) document_ptr) dom_prog"
  where
    "a_get_owner_document ptr = invoke (CD.a_get_owner_document_tups @ a_get_owner_document_tups) ptr ()"
end

global_interpretation l_get_owner_document_Shadow_DOM_defs get_root_node get_root_node_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_host get_host_locs
  defines get_owner_document_tups = "l_get_owner_document_Shadow_DOM_defs.a_get_owner_document_tups"
    and get_owner_document =
    "l_get_owner_document_Shadow_DOM_defs.a_get_owner_document get_root_node get_disconnected_nodes"
    and get_owner_document_shadow_root_ptr =
    "l_get_owner_document_Shadow_DOM_defs.a_get_owner_document_shadow_root_ptr"
    and get_owner_document_tups_Core_DOM =
    "l_get_owner_document_Core_DOM_defs.a_get_owner_document_tups get_root_node get_disconnected_nodes"
    and get_owner_document_node_ptr =
    "l_get_owner_document_Core_DOM_defs.a_get_owner_document_node_ptr get_root_node get_disconnected_nodes"
  .
```

```
locale l_get_owner_document_Shadow_DOM =
  l_get_owner_document_Shadow_DOM_defs get_root_node get_root_node_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_host get_host_locs +
  l_get_owner_document_defs get_owner_document +
  l_get_host get_host get_host_locs +
  CD: l_get_owner_document_Core_DOM
  get_parent get_parent_locs known_ptr_Core_DOM type_wf_Core_DOM get_disconnected_nodes
  get_disconnected_nodes_locs get_root_node get_root_node_locs get_owner_document_Core_DOM
  for known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_owner_document_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_owner_document :: "(_) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes get_owner_document_impl: "get_owner_document = a_get_owner_document"
begin
lemmas get_owner_document_def = a_get_owner_document_def[folded get_owner_document_impl]

lemma get_owner_document_pure [simp]:
  "pure (get_owner_document ptr) h"
proof -
  have "⋀shadow_root_ptr. pure (a_get_owner_document_shadow_root_ptr shadow_root_ptr ()) h"
    apply(auto simp add: a_get_owner_document_shadow_root_ptr_def intro!: bind_pure_I filter_M_pure_I
        split: option.splits)[1]
    by(auto simp add: CD.a_get_owner_document_document_ptr_def intro!: bind_pure_I filter_M_pure_I
        split: option.splits)
  then show ?thesis
    apply(auto simp add: get_owner_document_def)[1]
    apply(split CD.get_owner_document_splits, rule conjI)+
     apply(simp)
    apply(auto simp add: a_get_owner_document_tups_def)[1]
    apply(split invoke_splits, rule conjI)+
     apply simp
    by(auto intro!: bind_pure_I)
qed

lemma get_owner_document_ptr_in_heap:
  assumes "h ⊢ ok (get_owner_document ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms
  by(auto simp add: get_owner_document_def invoke_ptr_in_heap dest: is_OK_returns_heap_I)
end

interpretation
  i_get_owner_document?: l_get_owner_document_Shadow_DOM DocumentClass.known_ptr get_parent get_parent_locs
  DocumentClass.type_wf get_disconnected_nodes get_disconnected_nodes_locs get_root_node get_root_node_locs
CD.a_get_owner_document get_host get_host_locs get_owner_document get_child_nodes get_child_nodes_locs
  using get_child_nodes_is_l_get_child_nodes[unfolded ShadowRootClass.known_ptr_defs]
  by(auto simp add:  instances l_get_owner_document_Shadow_DOM_def l_get_owner_document_Shadow_DOM_axioms_def
      l_get_owner_document_Core_DOM_def l_get_owner_document_Core_DOM_axioms_def get_owner_document_def
Core_DOM_Functions.get_owner_document_def)
declare l_get_owner_document_Shadow_DOM_axioms [instances]

lemma get_owner_document_is_l_get_owner_document [instances]: "l_get_owner_document get_owner_document"
```

```
  apply(auto simp add: l_get_owner_document_def)[1]
  using get_owner_document_ptr_in_heap apply fast
  done
```

remove_child

```
global_interpretation l_remove_child_Core_DOM_defs get_child_nodes get_child_nodes_locs set_child_nodes
  set_child_nodes_locs get_parent get_parent_locs get_owner_document get_disconnected_nodes get_disconnected_nodes_
set_disconnected_nodes set_disconnected_nodes_locs
  defines remove = "l_remove_child_Core_DOM_defs.a_remove get_child_nodes set_child_nodes get_parent
get_owner_document get_disconnected_nodes set_disconnected_nodes"
    and remove_child = "l_remove_child_Core_DOM_defs.a_remove_child get_child_nodes set_child_nodes
get_owner_document get_disconnected_nodes set_disconnected_nodes"
    and remove_child_locs = "l_remove_child_Core_DOM_defs.a_remove_child_locs set_child_nodes_locs
set_disconnected_nodes_locs"
  .

interpretation i_remove_child?: l_remove_child_Core_DOM Shadow_DOM.get_child_nodes
  Shadow_DOM.get_child_nodes_locs Shadow_DOM.set_child_nodes Shadow_DOM.set_child_nodes_locs
  Shadow_DOM.get_parent Shadow_DOM.get_parent_locs
  Shadow_DOM.get_owner_document get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs remove_child remove_child_locs remove
  ShadowRootClass.type_wf
  ShadowRootClass.known_ptr ShadowRootClass.known_ptrs
  by(auto simp add: l_remove_child_Core_DOM_def l_remove_child_Core_DOM_axioms_def remove_child_def
      remove_child_locs_def remove_def instances)
declare l_remove_child_Core_DOM_axioms [instances]
```

get_disconnected_document

```
locale l_get_disconnected_document_Core_DOM_defs =
  l_get_disconnected_nodes_defs get_disconnected_nodes get_disconnected_nodes_locs
  for get_disconnected_nodes :: "(_::linorder) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list)
prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_disconnected_document :: "(_) node_ptr ⇒ (_, (_) document_ptr) dom_prog"
  where
    "a_get_disconnected_document node_ptr = do {
      check_in_heap (cast node_ptr);
      ptrs ← document_ptr_kinds_M;
      candidates ← filter_M (λdocument_ptr. do {
        disconnected_nodes ← get_disconnected_nodes document_ptr;
        return (node_ptr ∈ set disconnected_nodes)
      }) ptrs;
      (case candidates of
        Cons document_ptr [] ⇒ return document_ptr |
        _ ⇒ error HierarchyRequestError
      )
    }"
definition "a_get_disconnected_document_locs =
(⋃document_ptr. get_disconnected_nodes_locs document_ptr) ∪ (⋃ptr. {preserved (get_M_Object ptr RObject.nothing)})
end

locale l_get_disconnected_document_defs =
  fixes get_disconnected_document :: "(_) node_ptr ⇒ (_, (_::linorder) document_ptr) dom_prog"
  fixes get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_disconnected_document_Core_DOM =
  l_get_disconnected_document_Core_DOM_defs +
  l_get_disconnected_document_defs +
  l_get_disconnected_nodes +
  assumes get_disconnected_document_impl: "get_disconnected_document = a_get_disconnected_document"
```

```
  assumes get_disconnected_document_locs_impl: "get_disconnected_document_locs = a_get_disconnected_document_locs'
begin
lemmas get_disconnected_document_def =
  get_disconnected_document_impl[unfolded a_get_disconnected_document_def]
lemmas get_disconnected_document_locs_def =
  get_disconnected_document_locs_impl[unfolded a_get_disconnected_document_locs_def]


lemma get_disconnected_document_pure [simp]: "pure (get_disconnected_document ptr) h"
  using get_disconnected_nodes_pure
  by(auto simp add: get_disconnected_document_def intro!: bind_pure_I filter_M_pure_I split: list.splits)

lemma get_disconnected_document_ptr_in_heap [simp]:
  "h ⊢ ok (get_disconnected_document node_ptr) ⟹ node_ptr |∈| node_ptr_kinds h"
  using get_disconnected_document_def is_OK_returns_result_I check_in_heap_ptr_in_heap
  by (metis (no_types, lifting) bind_returns_heap_E get_disconnected_document_pure
      node_ptr_kinds_commutes pure_pure)

lemma get_disconnected_document_disconnected_document_in_heap:
  assumes "h ⊢ get_disconnected_document child_node →r disconnected_document"
  shows "disconnected_document |∈| document_ptr_kinds h"
  using assms get_disconnected_nodes_pure
  by(auto simp add: get_disconnected_document_def elim!: bind_returns_result_E2
      dest!: filter_M_not_more_elements[where x=disconnected_document]
      intro!: filter_M_pure_I bind_pure_I
      split: if_splits list.splits)

lemma get_disconnected_document_reads:
  "reads get_disconnected_document_locs (get_disconnected_document node_ptr) h h'"
  using get_disconnected_nodes_reads[unfolded reads_def]
  by(auto simp add: get_disconnected_document_def get_disconnected_document_locs_def
      intro!: reads_bind_pure reads_subset[OF check_in_heap_reads]
      reads_subset[OF error_reads]
      reads_subset[OF get_disconnected_nodes_reads] reads_subset[OF return_reads]
      reads_subset[OF document_ptr_kinds_M_reads] filter_M_reads filter_M_pure_I bind_pure_I
      split: list.splits)

end

locale l_get_disconnected_document = l_get_disconnected_document_defs +
  assumes get_disconnected_document_reads:
    "reads get_disconnected_document_locs (get_disconnected_document node_ptr) h h'"
  assumes get_disconnected_document_ptr_in_heap:
    "h ⊢ ok (get_disconnected_document node_ptr) ⟹ node_ptr |∈| node_ptr_kinds h"
  assumes get_disconnected_document_pure [simp]:
    "pure (get_disconnected_document node_ptr) h"
  assumes get_disconnected_document_disconnected_document_in_heap:
    "h ⊢ get_disconnected_document child_node →r disconnected_document ⟹
disconnected_document |∈| document_ptr_kinds h"

global_interpretation l_get_disconnected_documentCore_DOM_defs get_disconnected_nodes
  get_disconnected_nodes_locs defines
  get_disconnected_document = "l_get_disconnected_documentCore_DOM_defs.a_get_disconnected_document
get_disconnected_nodes" and
  get_disconnected_document_locs = "l_get_disconnected_documentCore_DOM_defs.a_get_disconnected_document_locs
get_disconnected_nodes_locs" .

interpretation i_get_disconnected_document?: l_get_disconnected_documentCore_DOM
  get_disconnected_nodes get_disconnected_nodes_locs get_disconnected_document get_disconnected_document_locs
type_wf
  by(auto simp add: l_get_disconnected_documentCore_DOM_def l_get_disconnected_documentCore_DOM_axioms_def
      get_disconnected_document_def get_disconnected_document_locs_def instances)
declare l_get_disconnected_documentCore_DOM_axioms [instances]
```

```
lemma get_disconnected_document_is_l_get_disconnected_document [instances]:
  "l_get_disconnected_document get_disconnected_document get_disconnected_document_locs"
  apply(auto simp add: l_get_disconnected_document_def instances)[1]
  using get_disconnected_document_ptr_in_heap get_disconnected_document_pure
    get_disconnected_document_disconnected_document_in_heap get_disconnected_document_reads
  by blast+
```

get_disconnected_nodes

```
locale l_set_tag_name_get_disconnected_nodes_Shadow_DOM =
  l_set_tag_name_Shadow_DOM +
  l_get_disconnected_nodes_Shadow_DOM
begin
lemma set_tag_name_get_disconnected_nodes:
  "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
  by(auto simp add: CD.set_tag_name_locs_impl[unfolded CD.a_set_tag_name_locs_def]
      CD.get_disconnected_nodes_locs_impl[unfolded CD.a_get_disconnected_nodes_locs_def]
      all_args_def)
end

interpretation
  i_set_tag_name_get_disconnected_nodes?: l_set_tag_name_get_disconnected_nodes_Shadow_DOM type_wf DocumentClass.t
  set_tag_name set_tag_name_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  by unfold_locales
declare l_set_tag_name_get_disconnected_nodes_Core_DOM_axioms[instances]

lemma set_tag_name_get_disconnected_nodes_is_l_set_tag_name_get_disconnected_nodes [instances]:
  "l_set_tag_name_get_disconnected_nodes type_wf set_tag_name set_tag_name_locs get_disconnected_nodes
                                         get_disconnected_nodes_locs"
  using set_tag_name_is_l_set_tag_name get_disconnected_nodes_is_l_get_disconnected_nodes
  apply(simp add: l_set_tag_name_get_disconnected_nodes_def
      l_set_tag_name_get_disconnected_nodes_axioms_def)
  using set_tag_name_get_disconnected_nodes
  by fast
```

get_ancestors_di

```
locale l_get_ancestors_di_Shadow_DOM_defs =
  l_get_parent_defs get_parent get_parent_locs +
  l_get_host_defs get_host get_host_locs +
  l_get_disconnected_document_defs get_disconnected_document get_disconnected_document_locs
  for get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_::linorder) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
begin
partial_function (dom_prog) a_get_ancestors_di :: "(_::linorder) object_ptr ⇒ (_, (_) object_ptr list)
dom_prog"
  where
    "a_get_ancestors_di ptr = do {
      check_in_heap ptr;
      ancestors ← (case cast_object_ptr2node_ptr ptr of
        Some node_ptr ⇒ do {
          parent_ptr_opt ← get_parent node_ptr;
          (case parent_ptr_opt of
            Some parent_ptr ⇒ a_get_ancestors_di parent_ptr
          | None ⇒ do {
              document_ptr ← get_disconnected_document node_ptr;
              a_get_ancestors_di (cast document_ptr)
            })
        }
```

```
      | None ⇒ (case cast ptr of
          Some shadow_root_ptr ⇒ do {
            host ← get_host shadow_root_ptr;
            a_get_ancestors_di (cast host)
          } |
          None ⇒ return []));
      return (ptr # ancestors)
    }"

definition "a_get_ancestors_di_locs = get_parent_locs ∪ get_host_locs ∪ get_disconnected_document_locs"
end

locale l_get_ancestors_di_defs =
  fixes get_ancestors_di :: "(_::linorder) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  fixes get_ancestors_di_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_ancestors_di_Shadow_DOM =
  l_get_parent +
  l_get_host +
  l_get_disconnected_document +
  l_get_ancestors_di_Shadow_DOM_defs +
  l_get_ancestors_di_defs +
  assumes get_ancestors_di_impl: "get_ancestors_di = a_get_ancestors_di"
  assumes get_ancestors_di_locs_impl: "get_ancestors_di_locs = a_get_ancestors_di_locs"
begin
lemmas get_ancestors_di_def = a_get_ancestors_di.simps[folded get_ancestors_di_impl]
lemmas get_ancestors_di_locs_def = a_get_ancestors_di_locs_def[folded get_ancestors_di_locs_impl]


lemma get_ancestors_di_pure [simp]:
  "pure (get_ancestors_di ptr) h"
proof -
  have "∀ptr h h' x. h ⊢ get_ancestors_di ptr →r x ⟶ h ⊢ get_ancestors_di ptr →h h' ⟶ h = h'"
  proof (induct rule: a_get_ancestors_di.fixp_induct[folded get_ancestors_di_impl])
    case 1
    then show ?case
      by(rule admissible_dom_prog)
  next
    case 2
    then show ?case
      by simp
  next
    case (3 f)
    then show ?case
      using get_parent_pure get_host_pure get_disconnected_document_pure
      apply(auto simp add: pure_returns_heap_eq pure_def split: option.splits elim!: bind_returns_heap_E
          bind_returns_result_E dest!: pure_returns_heap_eq[rotated, OF check_in_heap_pure])[1]
          apply (metis is_OK_returns_result_I returns_heap_eq returns_result_eq)
         apply (meson option.simps(3) returns_result_eq)
        apply (meson option.simps(3) returns_result_eq)
       apply(metis get_parent_pure pure_returns_heap_eq)
      apply(metis get_host_pure pure_returns_heap_eq)
      done
  qed
  then show ?thesis
    by (meson pure_eq_iff)
qed

lemma get_ancestors_di_ptr:
  assumes "h ⊢ get_ancestors_di ptr →r ancestors"
  shows "ptr ∈ set ancestors"
  using assms
  by(simp add: get_ancestors_di_def) (auto  elim!: bind_returns_result_E2 split: option.splits
```

```
    intro!: bind_pure_I)

lemma get_ancestors_di_ptr_in_heap:
  assumes "h ⊢ ok (get_ancestors_di ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms
  by(auto simp add: get_ancestors_di_def check_in_heap_ptr_in_heap elim!: bind_is_OK_E
      dest: is_OK_returns_result_I)

lemma get_ancestors_di_never_empty:
  assumes "h ⊢ get_ancestors_di child →r ancestors"
  shows "ancestors ≠ []"
  using assms
  apply(simp add: get_ancestors_di_def)
  by(auto elim!: bind_returns_result_E2 split: option.splits intro!: bind_pure_I)
end

global_interpretation l_get_ancestors_di_Shadow_DOM_defs get_parent get_parent_locs get_host get_host_locs
  get_disconnected_document get_disconnected_document_locs
  defines get_ancestors_di = a_get_ancestors_di
    and get_ancestors_di_locs = a_get_ancestors_di_locs .
declare a_get_ancestors_di.simps [code]

interpretation i_get_ancestors_di?: l_get_ancestors_di_Shadow_DOM
  type_wf known_ptr known_ptrs get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_host get_host_locs get_disconnected_document get_disconnected_document_locs get_ancestors_di get_ancestors_di
  by(auto simp add: l_get_ancestors_di_Shadow_DOM_def l_get_ancestors_di_Shadow_DOM_axioms_def
      get_ancestors_di_def get_ancestors_di_locs_def instances)
declare l_get_ancestors_di_Shadow_DOM_axioms [instances]

lemma get_ancestors_di_is_l_get_ancestors [instances]: "l_get_ancestors get_ancestors_di"
  apply(auto simp add: l_get_ancestors_def)[1]
  using get_ancestors_di_ptr_in_heap apply fast
  using get_ancestors_di_ptr apply fast
  done
```

adopt_node

```
locale l_adopt_node_Shadow_DOM_defs =
  CD: l_adopt_node_Core_DOM_defs get_owner_document get_parent get_parent_locs remove_child
  remove_child_locs get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs +
  l_get_ancestors_di_defs get_ancestors_di get_ancestors_di_locs
  for get_owner_document :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and remove_child :: "(_) object_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_child_locs :: "(_) object_ptr ⇒ (_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_ancestors_di :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_di_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_adopt_node :: "(_) document_ptr ⇒ (_) node_ptr ⇒ (_, unit) dom_prog"
  where
    "a_adopt_node document node = do {
      ancestors ← get_ancestors_di (cast document);
      (if cast node ∈ set ancestors
        then error HierarchyRequestError
        else CD.a_adopt_node document node)}"
```

```
definition a_adopt_node_locs ::
  "(_) object_ptr option ⇒ (_) document_ptr ⇒ (_) document_ptr ⇒ (_, unit) dom_prog set"
  where "a_adopt_node_locs = CD.a_adopt_node_locs"
end

locale l_adopt_node_Shadow_DOM =
  l_adopt_node_Shadow_DOM_defs get_owner_document get_parent get_parent_locs remove_child
  remove_child_locs get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs get_ancestors_di get_ancestors_di_locs +
  l_adopt_node_defs adopt_node adopt_node_locs +
  l_get_ancestors_di_Shadow_DOM type_wf known_ptr known_ptrs get_parent get_parent_locs
  get_child_nodes get_child_nodes_locs get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs get_ancestors_di get_ancestors_di_locs +
  CD: l_adopt_node_Core_DOM get_owner_document get_parent get_parent_locs remove_child
  remove_child_locs get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs adopt_node_Core_DOM adopt_node_locs_Core_DOM
  known_ptr type_wf get_child_nodes get_child_nodes_locs known_ptrs set_child_nodes
  set_child_nodes_locs remove
  for get_owner_document :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and remove_child :: "(_) object_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_child_locs :: "(_) object_ptr ⇒ (_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_ancestors_di :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_di_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and adopt_node :: "(_) document_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and adopt_node_locs ::
    "(_) object_ptr option ⇒ (_) document_ptr ⇒ (_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and adopt_node_Core_DOM :: "(_) document_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and adopt_node_locs_Core_DOM ::
    "(_) object_ptr option ⇒ (_) document_ptr ⇒ (_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and known_ptrs :: "(_) heap ⇒ bool"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and remove :: "(_) node_ptr ⇒ ((_) heap, exception, unit) prog" +
  assumes adopt_node_impl: "adopt_node = a_adopt_node"
  assumes adopt_node_locs_impl: "adopt_node_locs = a_adopt_node_locs"
begin
lemmas adopt_node_def = a_adopt_node_def[folded adopt_node_impl CD.adopt_node_impl]
lemmas adopt_node_locs_def = a_adopt_node_locs_def[folded adopt_node_locs_impl CD.adopt_node_locs_impl]

lemma adopt_node_writes:
  "writes (adopt_node_locs |h ⊢ get_parent node|_r
           |h ⊢ get_owner_document (cast node)|_r document_ptr) (adopt_node document_ptr node) h h'"
  by(auto simp add: CD.adopt_node_writes adopt_node_def CD.adopt_node_impl[symmetric]
      adopt_node_locs_def CD.adopt_node_locs_impl[symmetric]
      intro!: writes_bind_pure[OF get_ancestors_di_pure])

lemma adopt_node_pointers_preserved:
```

```
  "w ∈ adopt_node_locs parent owner_document document_ptr
   ⟹ h ⊢ w →h h' ⟹ object_ptr_kinds h = object_ptr_kinds h'"
  using CD.adopt_node_locs_impl CD.adopt_node_pointers_preserved local.adopt_node_locs_def by blast
lemma adopt_node_types_preserved:
  "w ∈ adopt_node_locs parent owner_document document_ptr
   ⟹ h ⊢ w →h h' ⟹ type_wf h = type_wf h'"
  using CD.adopt_node_locs_impl CD.adopt_node_types_preserved local.adopt_node_locs_def by blast
lemma adopt_node_child_in_heap:
  "h ⊢ ok (adopt_node document_ptr child) ⟹ child |∈| node_ptr_kinds h"
  by (smt CD.adopt_node_child_in_heap CD.adopt_node_impl bind_is_OK_E error_returns_heap
      is_OK_returns_heap_E l_adopt_node_Shadow_DOM.adopt_node_def l_adopt_node_Shadow_DOM_axioms
      local.get_ancestors_di_pure pure_returns_heap_eq)
lemma adopt_node_children_subset:
  "h ⊢ adopt_node owner_document node →h h' ⟹ h ⊢ get_child_nodes ptr →r children
     ⟹ h' ⊢ get_child_nodes ptr →r children'
     ⟹ known_ptrs h ⟹ type_wf h ⟹ set children' ⊆ set children"
  by (smt CD.adopt_node_children_subset CD.adopt_node_impl bind_returns_heap_E error_returns_heap
      local.adopt_node_def local.get_ancestors_di_pure pure_returns_heap_eq)
end

global_interpretation l_adopt_node_Shadow_DOM_defs get_owner_document get_parent get_parent_locs
  remove_child remove_child_locs get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs get_ancestors_di get_ancestors_di_locs
  defines adopt_node = "a_adopt_node"
    and adopt_node_locs = "a_adopt_node_locs"
    and adopt_node_Core_DOM = "CD.a_adopt_node"
    and adopt_node_locs_Core_DOM = "CD.a_adopt_node_locs"
  .
interpretation i_adopt_node_Core_DOM: l_adopt_node_Core_DOM
  get_owner_document get_parent get_parent_locs remove_child remove_child_locs
  get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs
  adopt_node_Core_DOM adopt_node_locs_Core_DOM known_ptr type_wf get_child_nodes get_child_nodes_locs
  known_ptrs set_child_nodes set_child_nodes_locs remove
  by(auto simp add: l_adopt_node_Core_DOM_def l_adopt_node_Core_DOM_axioms_def adopt_node_Core_DOM_def
      adopt_node_locs_Core_DOM_def instances)
declare i_adopt_node_Core_DOM.l_adopt_node_Core_DOM_axioms [instances]

interpretation i_adopt_node?: l_adopt_node_Shadow_DOM
  get_owner_document get_parent get_parent_locs remove_child remove_child_locs get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs get_ancestors_di
  get_ancestors_di_locs adopt_node adopt_node_locs CD.a_adopt_node CD.a_adopt_node_locs known_ptr
  type_wf get_child_nodes get_child_nodes_locs known_ptrs set_child_nodes set_child_nodes_locs
  get_host get_host_locs get_disconnected_document get_disconnected_document_locs remove
  by(auto simp add: l_adopt_node_Shadow_DOM_def l_adopt_node_Shadow_DOM_axioms_def adopt_node_def
      adopt_node_locs_def instances)
declare l_adopt_node_Shadow_DOM_axioms [instances]

lemma adopt_node_is_l_adopt_node [instances]: "l_adopt_node type_wf known_ptr known_ptrs get_parent
adopt_node adopt_node_locs get_child_nodes get_owner_document"
  apply(auto simp add: l_adopt_node_def l_adopt_node_axioms_def instances)[1]
  using adopt_node_writes apply fast
  using adopt_node_pointers_preserved apply (fast, fast)
  using adopt_node_types_preserved apply (fast, fast)
  using adopt_node_child_in_heap apply fast
  using adopt_node_children_subset apply fast
  done
```

get_shadow_root

```
locale l_adopt_node_get_shadow_root_Shadow_DOM =
  l_set_child_nodes_get_shadow_root_Shadow_DOM +
  l_set_disconnected_nodes_get_shadow_root_Shadow_DOM +
  l_adopt_node_Shadow_DOM
begin
lemma adopt_node_get_shadow_root:
```

```
  "∀w ∈ adopt_node_locs parent owner_document document_ptr. (h ⊢ w →h h' ⟶
(∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  by(auto simp add: adopt_node_locs_def CD.adopt_node_locs_def CD.remove_child_locs_def
      all_args_def set_disconnected_nodes_get_shadow_root set_child_nodes_get_shadow_root)
end

locale l_adopt_node_get_shadow_root = l_adopt_node_defs + l_get_shadow_root_defs +
  assumes adopt_node_get_shadow_root:
    "∀w ∈ adopt_node_locs parent owner_document document_ptr. (h ⊢ w →h h' ⟶
(∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation i_adopt_node_get_shadow_root?: l_adopt_node_get_shadow_rootShadow_DOM
  type_wf known_ptr DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
  Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_shadow_root
  get_shadow_root_locs set_disconnected_nodes set_disconnected_nodes_locs get_owner_document
  get_parent get_parent_locs remove_child remove_child_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_ancestors_di get_ancestors_di_locs adopt_node adopt_node_locs
  adopt_nodeCore_DOM adopt_node_locsCore_DOM get_child_nodes get_child_nodes_locs known_ptrs get_host
  get_host_locs get_disconnected_document get_disconnected_document_locs remove
  by(auto simp add: l_adopt_node_get_shadow_rootShadow_DOM_def instances)
declare l_adopt_node_get_shadow_rootShadow_DOM_axioms [instances]

interpretation i_adopt_node_get_shadow_root?: l_adopt_node_get_shadow_rootShadow_DOM
  type_wf known_ptr DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes
  set_child_nodes_locs Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs
  get_shadow_root get_shadow_root_locs set_disconnected_nodes set_disconnected_nodes_locs
  get_owner_document get_parent get_parent_locs remove_child remove_child_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_ancestors_di get_ancestors_di_locs adopt_node adopt_node_locs
  adopt_nodeCore_DOM adopt_node_locsCore_DOM get_child_nodes get_child_nodes_locs known_ptrs get_host
  get_host_locs get_disconnected_document get_disconnected_document_locs remove
  by(auto simp add: l_adopt_node_get_shadow_rootShadow_DOM_def instances)
declare l_adopt_node_get_shadow_rootShadow_DOM_axioms [instances]

lemma adopt_node_get_shadow_root_is_l_adopt_node_get_shadow_root [instances]:
  "l_adopt_node_get_shadow_root adopt_node_locs get_shadow_root_locs"
  apply(auto simp add: l_adopt_node_get_shadow_root_def)[1]
  using adopt_node_get_shadow_root apply fast
  done
```

insert_before

```
global_interpretation l_insert_beforeCore_DOM_defs get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs set_child_nodes set_child_nodes_locs get_ancestors_di get_ancestors_di_locs
  adopt_node adopt_node_locs set_disconnected_nodes set_disconnected_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_owner_document
  defines
    next_sibling = a_next_sibling and
    insert_node = a_insert_node and
    ensure_pre_insertion_validity = a_ensure_pre_insertion_validity and
    insert_before = a_insert_before and
    insert_before_locs = a_insert_before_locs
  .
global_interpretation l_append_childCore_DOM_defs insert_before
  defines append_child = "l_append_childCore_DOM_defs.a_append_child insert_before"
  .

interpretation i_insert_before?: l_insert_beforeCore_DOM get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs set_child_nodes set_child_nodes_locs get_ancestors_di get_ancestors_di_locs
  adopt_node adopt_node_locs set_disconnected_nodes set_disconnected_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_owner_document insert_before insert_before_locs append_child type_wf
  known_ptr known_ptrs
  by(auto simp add: l_insert_beforeCore_DOM_def l_insert_beforeCore_DOM_axioms_def insert_before_def
      insert_before_locs_def instances)
```

```
declare l_insert_before_Core_DOM_axioms [instances]

interpretation i_append_child?: l_append_child_Core_DOM append_child insert_before insert_before_locs
  by(simp add: l_append_child_Core_DOM_def instances append_child_def)
declare l_append_child_Core_DOM_axioms[instances]
```

get_assigned_nodes

```
fun map_filter_M2 :: "('x ⇒ ('heap, 'e, 'y option) prog) ⇒ 'x list
  ⇒ ('heap, 'e, 'y list) prog"
  where
    "map_filter_M2 f [] = return []" |
    "map_filter_M2 f (x # xs) = do {
      res ← f x;
      remainder ← map_filter_M2 f xs;
      return ((case res of  Some r ⇒ [r] | None ⇒ []) @ remainder)
    }"
lemma map_filter_M2_pure [simp]:
  assumes "⋀x. x ∈ set xs ⟹ pure (f x) h"
  shows "pure (map_filter_M2 f xs) h"
  using assms
  apply(induct xs arbitrary: h)
  by(auto elim!: bind_returns_result_E2 intro!: bind_pure_I)

lemma map_filter_pure_no_monad:
  assumes "⋀x. x ∈ set xs ⟹ pure (f x) h"
  assumes "h ⊢ map_filter_M2 f xs →r ys"
  shows
    "ys = map the (filter (λx. x ≠ None) (map (λx. |h ⊢ f x|r) xs))" and
    "⋀x. x ∈ set xs ⟹ h ⊢ ok (f x)"
  using assms
   apply(induct xs arbitrary: h ys)
  by(auto elim!: bind_returns_result_E2)

lemma map_filter_pure_foo:
  assumes "⋀x. x ∈ set xs ⟹ pure (f x) h"
  assumes "h ⊢ map_filter_M2 f xs →r ys"
  assumes "y ∈ set ys"
  obtains x where "h ⊢ f x →r Some y" and "x ∈ set xs"
  using assms
  apply(induct xs arbitrary: ys)
  by(auto elim!: bind_returns_result_E2)


lemma map_filter_M2_in_result:
  assumes "h ⊢ map_filter_M2 P xs →r ys"
  assumes "a ∈ set xs"
  assumes "⋀x. x ∈ set xs ⟹ pure (P x) h"
  assumes "h ⊢ P a →r Some b"
  shows "b ∈ set ys"
  using assms
  apply(induct xs arbitrary: h ys)
  by(auto elim!: bind_returns_result_E2 )


locale l_assigned_nodes_Shadow_DOM_defs =
  l_get_tag_name_defs get_tag_name get_tag_name_locs +
  l_get_root_node_defs get_root_node get_root_node_locs +
  l_get_host_defs get_host get_host_locs +
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  l_find_slot_defs find_slot assigned_slot +
  l_remove_defs remove +
  l_insert_before_defs insert_before insert_before_locs +
```

```
  l_append_child_defs append_child +
  l_remove_shadow_root_defs remove_shadow_root remove_shadow_root_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and find_slot :: "bool ⇒ (_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and assigned_slot :: "(_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and remove :: "(_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and insert_before ::
    "(_) object_ptr ⇒ (_) node_ptr ⇒ (_) node_ptr option ⇒ ((_) heap, exception, unit) prog"
    and insert_before_locs ::
    "(_) object_ptr ⇒ (_) object_ptr option ⇒ (_) document_ptr ⇒ (_) document_ptr ⇒ (_, unit) dom_prog
set"
    and append_child :: "(_) object_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root_locs ::
    "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap, exception, unit) prog set"
begin
definition a_assigned_nodes :: "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"
  where
    "a_assigned_nodes slot = do {
      tag ← get_tag_name slot;
      (if tag ≠ ''slot''
      then error HierarchyRequestError
      else return ());
      root ← get_root_node (cast slot);
      if is_shadow_root_ptr_kind root
      then do {
        host ← get_host (the (cast root));
        children ← get_child_nodes (cast host);
        filter_M (λslotable. do {
          found_slot ← find_slot False slotable;
          return (found_slot = Some slot)}) children}
      else return []}"

partial_function (dom_prog) a_assigned_nodes_flatten ::
  "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"
  where
    "a_assigned_nodes_flatten slot = do {
      tag ← get_tag_name slot;
      (if tag ≠ ''slot''
      then error HierarchyRequestError
      else return ());
      root ← get_root_node (cast slot);
      (if is_shadow_root_ptr_kind root
      then do {
        slotables ← a_assigned_nodes slot;
        slotables_or_child_nodes ← (if slotables = []
        then do {
          get_child_nodes (cast slot)
        } else do {
          return slotables
        });
        list_of_lists ← map_M (λnode_ptr. do {
          (case cast node_ptr of
            Some element_ptr ⇒ do {
              tag ← get_tag_name element_ptr;
              (if tag = ''slot''
```

```
            then do {
              root ← get_root_node (cast element_ptr);
              (if is_shadow_root_ptr_kind root
              then do {
                a_assigned_nodes_flatten element_ptr
              } else do {
                return [node_ptr]
              })
            } else do {
              return [node_ptr]
            })
          }
          | None ⇒ return [node_ptr])
      }) slotables_or_child_nodes;
      return (concat list_of_lists)
    } else return [])
  }"
```

definition *a_flatten_dom :: "(_, unit) dom_prog"* **where**

```
  "a_flatten_dom = do {
    tups ← element_ptr_kinds_M ≫= map_filter_M2 (λelement_ptr. do {
      tag ← get_tag_name element_ptr;
      assigned_nodes ← a_assigned_nodes element_ptr;
      (if tag = ''slot'' ∧ assigned_nodes ≠ []
      then return (Some (element_ptr, assigned_nodes)) else return None)});
    forall_M (λ(slot, assigned_nodes). do {
      get_child_nodes (cast slot) ≫= forall_M remove;
      forall_M (append_child (cast slot)) assigned_nodes
    }) tups;
    shadow_root_ptr_kinds_M ≫= forall_M (λshadow_root_ptr. do {
      host ← get_host shadow_root_ptr;
      get_child_nodes (cast host) ≫= forall_M remove;
      get_child_nodes (cast shadow_root_ptr) ≫= forall_M (append_child (cast host));
      remove_shadow_root host
    });
    return ()
  }"
```

end

global_interpretation *l_assigned_nodes$_{Shadow_DOM}$_defs get_child_nodes get_child_nodes_locs*
get_tag_name get_tag_name_locs get_root_node get_root_node_locs get_host get_host_locs
find_slot assigned_slot remove insert_before insert_before_locs append_child remove_shadow_root
remove_shadow_root_locs
defines *assigned_nodes =*
"l_assigned_nodes$_{Shadow_DOM}$_defs.a_assigned_nodes get_child_nodes get_tag_name get_root_node get_host find_slot"
and *assigned_nodes_flatten =*
"l_assigned_nodes$_{Shadow_DOM}$_defs.a_assigned_nodes_flatten get_child_nodes get_tag_name get_root_node get_host find_slot"
and *flatten_dom =*
"l_assigned_nodes$_{Shadow_DOM}$_defs.a_flatten_dom get_child_nodes get_tag_name get_root_node get_host find_slot remove append_child remove_shadow_root"
.

declare *a_assigned_nodes_flatten.simps [code]*

locale *l_assigned_nodes_defs =*
fixes *assigned_nodes :: "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"*
fixes *assigned_nodes_flatten :: "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"*
fixes *flatten_dom :: "(_, unit) dom_prog"*

locale *l_assigned_nodes$_{Shadow_DOM}$ =*
l_assigned_nodes_defs

```
  assigned_nodes assigned_nodes_flatten flatten_dom
  + l_assigned_nodes_Shadow_DOM_defs
  get_child_nodes get_child_nodes_locs get_tag_name get_tag_name_locs get_root_node
  get_root_node_locs get_host get_host_locs find_slot assigned_slot remove insert_before
  insert_before_locs append_child remove_shadow_root remove_shadow_root_locs

  + l_get_shadow_root
  type_wf get_shadow_root get_shadow_root_locs
  + l_set_shadow_root
  type_wf set_shadow_root set_shadow_root_locs
  + l_remove
  + l_insert_before
  insert_before insert_before_locs
  + l_find_slot
  find_slot assigned_slot
  + l_get_tag_name
  type_wf get_tag_name get_tag_name_locs
  + l_get_root_node
  get_root_node get_root_node_locs get_parent get_parent_locs
  + l_get_host
  get_host get_host_locs
  + l_get_child_nodes
  type_wf known_ptr get_child_nodes get_child_nodes_locs
  + l_to_tree_order
  to_tree_order
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and assigned_nodes :: "(_) element_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and assigned_nodes_flatten :: "(_) element_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and flatten_dom :: "((_) heap, exception, unit) prog"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and find_slot :: "bool ⇒ (_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and assigned_slot :: "(_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and remove :: "(_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and insert_before :: "(_) object_ptr ⇒ (_) node_ptr ⇒ (_) node_ptr option ⇒ ((_) heap, exception,
unit) prog"
    and insert_before_locs ::
    "(_) object_ptr ⇒ (_) object_ptr option ⇒ (_) document_ptr ⇒ (_) document_ptr ⇒ (_, unit) dom_prog
set"
    and append_child :: "(_) object_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap, exception, unit)
prog set"
    and type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ ((_) heap, exception, unit)
prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog" +
  assumes assigned_nodes_impl: "assigned_nodes = a_assigned_nodes"
  assumes flatten_dom_impl: "flatten_dom = a_flatten_dom"
begin
lemmas assigned_nodes_def = assigned_nodes_impl[unfolded a_assigned_nodes_def]
lemmas flatten_dom_def = flatten_dom_impl[unfolded a_flatten_dom_def, folded assigned_nodes_impl]
```

```
lemma assigned_nodes_pure [simp]: "pure (assigned_nodes slot) h"
  by(auto simp add: assigned_nodes_def intro!: bind_pure_I filter_M_pure_I)

lemma assigned_nodes_ptr_in_heap:
  assumes "h ⊢ ok (assigned_nodes slot)"
  shows "slot |∈| element_ptr_kinds h"
  using assms
  apply(auto simp add: assigned_nodes_def)[1]
  by (meson bind_is_OK_E is_OK_returns_result_I local.get_tag_name_ptr_in_heap)

lemma assigned_nodes_slot_is_slot:
  assumes "h ⊢ ok (assigned_nodes slot)"
  shows "h ⊢ get_tag_name slot →r ''slot''"
  using assms
  by(auto simp add: assigned_nodes_def elim!: bind_is_OK_E split: if_splits)

lemma assigned_nodes_different_ptr:
  assumes "h ⊢ assigned_nodes slot →r nodes"
  assumes "h ⊢ assigned_nodes slot' →r nodes'"
  assumes "slot ≠ slot'"
  shows "set nodes ∩ set nodes' = {}"
proof (rule ccontr)
  assume "set nodes ∩ set nodes' ≠ {} "
  then obtain common_ptr where "common_ptr ∈ set nodes" and "common_ptr ∈ set nodes'"
    by auto

  have "h ⊢ find_slot False common_ptr →r Some slot"
    using ‹common_ptr ∈ set nodes›
    using assms(1)
    by(auto simp add: assigned_nodes_def elim!: bind_returns_result_E2 split: if_splits
        dest!: filter_M_holds_for_result[where x=common_ptr] intro!: bind_pure_I)
  moreover
  have "h ⊢ find_slot False common_ptr →r Some slot'"
    using ‹common_ptr ∈ set nodes'›
    using assms(2)
    by(auto simp add: assigned_nodes_def elim!: bind_returns_result_E2 split: if_splits
        dest!: filter_M_holds_for_result[where x=common_ptr] intro!: bind_pure_I)
  ultimately
  show False
    using assms(3)
    by (meson option.inject returns_result_eq)
qed
end

interpretation i_assigned_nodes?: l_assigned_nodes_Shadow_DOM known_ptr assigned_nodes
  assigned_nodes_flatten flatten_dom get_child_nodes get_child_nodes_locs get_tag_name
  get_tag_name_locs get_root_node get_root_node_locs get_host get_host_locs find_slot assigned_slot
  remove insert_before insert_before_locs append_child remove_shadow_root remove_shadow_root_locs
  type_wf get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_parent
  get_parent_locs to_tree_order
  by(auto simp add: instances l_assigned_nodes_Shadow_DOM_def l_assigned_nodes_Shadow_DOM_axioms_def
      assigned_nodes_def flatten_dom_def)
declare l_assigned_nodes_Shadow_DOM_axioms [instances]

locale l_assigned_nodes = l_assigned_nodes_defs +
  assumes assigned_nodes_pure [simp]: "pure (assigned_nodes slot) h"
  assumes assigned_nodes_ptr_in_heap: "h ⊢ ok (assigned_nodes slot) ⟹ slot |∈| element_ptr_kinds h"
  assumes assigned_nodes_slot_is_slot: "h ⊢ ok (assigned_nodes slot) ⟹ h ⊢ get_tag_name slot →r ''slot''"
  assumes assigned_nodes_different_ptr:
    "h ⊢ assigned_nodes slot →r nodes ⟹ h ⊢ assigned_nodes slot' →r nodes' ⟹ slot ≠ slot' ⟹
set nodes ∩ set nodes' = {}"
```

```
lemma assigned_nodes_is_l_assigned_nodes [instances]: "l_assigned_nodes assigned_nodes"
  apply(auto simp add: l_assigned_nodes_def)[1]
  using assigned_nodes_ptr_in_heap apply fast
  using assigned_nodes_slot_is_slot apply fast
  using assigned_nodes_different_ptr apply fast
  done
```

set_val

```
locale l_set_val_Shadow_DOM =
  CD: l_set_val_Core_DOM type_wf_Core_DOM set_val set_val_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and set_val :: "(_) character_data_ptr ⇒ char list ⇒ (_, unit) dom_prog"
    and set_val_locs :: "(_) character_data_ptr ⇒ (_, unit) dom_prog set"  +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin

lemma set_val_ok:
  "type_wf h ⟹ character_data_ptr |∈| character_data_ptr_kinds h ⟹
h ⊢ ok (set_val character_data_ptr tag)"
  using CD.set_val_ok CD.type_wf_impl ShadowRootClass.type_wf_Document local.type_wf_impl by blast

lemma set_val_writes: "writes (set_val_locs character_data_ptr) (set_val character_data_ptr tag) h h'"
  using CD.set_val_writes .

lemma set_val_pointers_preserved:
  assumes "w ∈ set_val_locs character_data_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  using assms CD.set_val_pointers_preserved by simp

lemma set_val_typess_preserved:
  assumes "w ∈ set_val_locs character_data_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  apply(unfold type_wf_impl)
  using assms(1) type_wf_preserved[OF writes_singleton2 assms(2)]
  by(auto simp add: all_args_def CD.set_val_locs_impl[unfolded CD.a_set_val_locs_def] split: if_splits)
end

interpretation
  i_set_val?: l_set_val_Shadow_DOM type_wf DocumentClass.type_wf set_val set_val_locs
  apply(unfold_locales)
  by (auto simp add: set_val_def set_val_locs_def)
declare l_set_val_Shadow_DOM_axioms[instances]

lemma set_val_is_l_set_val [instances]: "l_set_val type_wf set_val set_val_locs"
  apply(simp add: l_set_val_def)
  using set_val_ok set_val_writes set_val_pointers_preserved set_val_typess_preserved
  by blast
```

get_shadow_root

```
locale l_set_val_get_shadow_root_Shadow_DOM =
  l_set_val_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM
begin
lemma set_val_get_shadow_root:
  "∀w ∈ set_val_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  by(auto simp add: CD.set_val_locs_impl[unfolded CD.a_set_val_locs_def]
      get_shadow_root_locs_def all_args_def)
end
```

```
locale l_set_val_get_shadow_root = l_set_val + l_get_shadow_root +
  assumes set_val_get_shadow_root:
    "∀w ∈ set_val_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_val_get_shadow_root?: l_set_val_get_shadow_root_Shadow_DOM type_wf DocumentClass.type_wf
  set_val set_val_locs
  get_shadow_root get_shadow_root_locs
  apply(auto simp add: l_set_val_get_shadow_root_Shadow_DOM_def  instances)[1]
  using l_set_val_Shadow_DOM_axioms
  by unfold_locales
declare l_set_val_get_shadow_root_Shadow_DOM_axioms[instances]

lemma set_val_get_shadow_root_is_l_set_val_get_shadow_root [instances]:
  "l_set_val_get_shadow_root type_wf set_val set_val_locs get_shadow_root
                                 get_shadow_root_locs"
  using set_val_is_l_set_val get_shadow_root_is_l_get_shadow_root
  apply(simp add: l_set_val_get_shadow_root_def l_set_val_get_shadow_root_axioms_def)
  using set_val_get_shadow_root
  by fast
```

get_tag_type

```
locale l_set_val_get_tag_name_Shadow_DOM =
  l_set_val_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_val_get_tag_name:
  "∀w ∈ set_val_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
  by(auto simp add: CD.set_val_locs_impl[unfolded CD.a_set_val_locs_def]
      CD.get_tag_name_locs_impl[unfolded CD.a_get_tag_name_locs_def]
      all_args_def)
end

locale l_set_val_get_tag_name = l_set_val + l_get_tag_name +
  assumes set_val_get_tag_name:
    "∀w ∈ set_val_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"

interpretation
  i_set_val_get_tag_name?: l_set_val_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf set_val
  set_val_locs get_tag_name get_tag_name_locs
  by unfold_locales
declare l_set_val_get_tag_name_Shadow_DOM_axioms[instances]

lemma set_val_get_tag_name_is_l_set_val_get_tag_name [instances]:
  "l_set_val_get_tag_name type_wf set_val set_val_locs get_tag_name get_tag_name_locs"
  using set_val_is_l_set_val get_tag_name_is_l_get_tag_name
  apply(simp add: l_set_val_get_tag_name_def l_set_val_get_tag_name_axioms_def)
  using set_val_get_tag_name
  by fast
```

create_character_data

```
locale l_create_character_data_Shadow_DOM =
  CD: l_create_character_data_Core_DOM _ _ _ _ _ _ type_wf_Core_DOM _ known_ptr_Core_DOM +
  l_known_ptr known_ptr
  for known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
begin

lemma create_character_data_document_in_heap:
  assumes "h ⊢ ok (create_character_data document_ptr text)"
  shows "document_ptr |∈| document_ptr_kinds h"
```

```
  using assms CD.create_character_data_document_in_heap by simp

lemma create_character_data_known_ptr:
  assumes "h ⊢ create_character_data document_ptr text →r new_character_data_ptr"
  shows "known_ptr (cast new_character_data_ptr)"
  using assms CD.create_character_data_known_ptr
  by(simp add: known_ptr_impl CD.known_ptr_impl ShadowRootClass.a_known_ptr_def)
end

locale l_create_character_data = l_create_character_data_defs

interpretation
  i_create_character_data?: l_create_character_data_Shadow_DOM get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs set_val
  set_val_locs create_character_data known_ptr DocumentClass.type_wf DocumentClass.known_ptr
  by(auto simp add: l_create_character_data_Shadow_DOM_def l_create_character_data_Shadow_DOM_axioms_def
instances)
declare l_create_character_data_Core_DOM_axioms [instances]
```

create_element

```
locale l_create_element_Shadow_DOM =
  CD: l_create_element_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs set_tag_name set_tag_name_locs type_wf_Core_DOM create_element known_ptr_Core_DOM
+
  l_known_ptr known_ptr
  for get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_tag_name :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
    and set_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and type_wf :: "(_) heap ⇒ bool"
    and create_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
begin
lemmas create_element_def = CD.create_element_def

lemma create_element_document_in_heap:
  assumes "h ⊢ ok (create_element document_ptr tag)"
  shows "document_ptr |∈| document_ptr_kinds h"
  using CD.create_element_document_in_heap assms .

lemma create_element_known_ptr:
  assumes "h ⊢ create_element document_ptr tag →r new_element_ptr"
  shows "known_ptr (cast new_element_ptr)"
proof -
  have "is_element_ptr new_element_ptr"
    using assms
    apply(auto simp add: create_element_def elim!: bind_returns_result_E)[1]
    using new_element_is_element_ptr
    by blast
  then show ?thesis
    by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
        CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs)
qed
end

interpretation
```

```
  i_create_element?: l_create_element_Shadow_DOM get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs set_tag_name set_tag_name_locs type_wf
  create_element known_ptr DocumentClass.type_wf DocumentClass.known_ptr
  by(auto simp add: l_create_element_Shadow_DOM_def l_create_element_Shadow_DOM_axioms_def instances)
declare l_create_element_Shadow_DOM_axioms[instances]
```

2.3.2 A wellformed heap (Core DOM)

wellformed_heap

```
locale l_heap_is_wellformed_Shadow_DOM_defs =
  CD: l_heap_is_wellformed_Core_DOM_defs get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_get_tag_name_defs get_tag_name get_tag_name_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_host_shadow_root_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
  where
    "a_host_shadow_root_rel h = (λ(x, y). (cast x, cast y)) ‘ {(host, shadow_root).
      host |∈| element_ptr_kinds h ∧ |h ⊢ get_shadow_root host|_r = Some shadow_root}"

lemma a_host_shadow_root_rel_code [code]: "a_host_shadow_root_rel h = set (concat (map
  (λhost. (case |h ⊢ get_shadow_root host|_r of
    Some shadow_root ⇒ [(cast host, cast shadow_root)] |
    None ⇒ []))
  (sorted_list_of_fset (element_ptr_kinds h)))
)"
  by(auto simp add: a_host_shadow_root_rel_def)

definition a_ptr_disconnected_node_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
  where
    "a_ptr_disconnected_node_rel h = (λ(x, y). (cast x, cast y)) ‘ {(document_ptr, disconnected_node).
      document_ptr |∈| document_ptr_kinds h ∧ disconnected_node ∈ set |h ⊢ get_disconnected_nodes document_ptr|_r}

lemma a_ptr_disconnected_node_rel_code [code]: "a_ptr_disconnected_node_rel h = set (concat (map
  (λptr. map
    (λnode. (cast ptr, cast node))
    |h ⊢ get_disconnected_nodes ptr|_r)
  (sorted_list_of_fset (document_ptr_kinds h)))
)"
  by(auto simp add: a_ptr_disconnected_node_rel_def)

definition a_all_ptrs_in_heap :: "(_) heap ⇒ bool" where
  "a_all_ptrs_in_heap h = ((∀host shadow_root_ptr.
    (h ⊢ get_shadow_root host →_r Some shadow_root_ptr) ⟶
      shadow_root_ptr |∈| shadow_root_ptr_kinds h))"

definition a_distinct_lists :: "(_) heap ⇒ bool"
  where
    "a_distinct_lists h = distinct (concat (
      map (λelement_ptr. (case |h ⊢ get_shadow_root element_ptr|_r of
 Some shadow_root_ptr ⇒ [shadow_root_ptr] | None ⇒ []))
        |h ⊢ element_ptr_kinds_M|_r
    ))"
```

```
definition a_shadow_root_valid :: "(_) heap ⇒ bool" where
  "a_shadow_root_valid h = (∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h).
     (∃host ∈ fset(element_ptr_kinds h).
       |h ⊢ get_tag_name host|r ∈ safe_shadow_root_element_types ∧
       |h ⊢ get_shadow_root host|r = Some shadow_root_ptr))"

definition a_heap_is_wellformed :: "(_) heap ⇒ bool"
  where
    "a_heap_is_wellformed h ⟷ CD.a_heap_is_wellformed h ∧
      acyclic (CD.a_parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h) ∧
      a_all_ptrs_in_heap h ∧
      a_distinct_lists h ∧
      a_shadow_root_valid h"
end

global_interpretation l_heap_is_wellformed_Shadow_DOM_defs get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs
  get_tag_name get_tag_name_locs
  defines heap_is_wellformed = a_heap_is_wellformed
    and parent_child_rel = CD.a_parent_child_rel
    and host_shadow_root_rel = a_host_shadow_root_rel
    and ptr_disconnected_node_rel = a_ptr_disconnected_node_rel
    and all_ptrs_in_heap = a_all_ptrs_in_heap
    and distinct_lists = a_distinct_lists
    and shadow_root_valid = a_shadow_root_valid
    and heap_is_wellformed_Core_DOM = CD.a_heap_is_wellformed
    and parent_child_rel_Core_DOM = CD.a_parent_child_rel
    and acyclic_heap_Core_DOM = CD.a_acyclic_heap
    and all_ptrs_in_heap_Core_DOM = CD.a_all_ptrs_in_heap
    and distinct_lists_Core_DOM = CD.a_distinct_lists
    and owner_document_valid_Core_DOM = CD.a_owner_document_valid
  .

interpretation i_heap_is_wellformed_Core_DOM: l_heap_is_wellformed_Core_DOM known_ptr type_wf
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  heap_is_wellformed_Core_DOM parent_child_rel
  by (auto simp add: l_heap_is_wellformed_Core_DOM_def l_heap_is_wellformed_Core_DOM_axioms_def
      heap_is_wellformed_Core_DOM_def parent_child_rel_def instances)
declare i_heap_is_wellformed_Core_DOM.l_heap_is_wellformed_Core_DOM_axioms[instances]

lemma heap_is_wellformed_Core_DOM_is_l_heap_is_wellformed [instances]:
  "l_heap_is_wellformed type_wf known_ptr heap_is_wellformed_Core_DOM parent_child_rel get_child_nodes
get_disconnected_nodes"
  apply(auto simp add: l_heap_is_wellformed_def)[1]
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_children_in_heap apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_disc_nodes_in_heap apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_one_parent apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_one_disc_parent apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_children_disc_nodes_different apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_disconnected_nodes_distinct apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_children_distinct apply blast
  using i_heap_is_wellformed_Core_DOM.heap_is_wellformed_children_disc_nodes apply blast
  using i_heap_is_wellformed_Core_DOM.parent_child_rel_child apply (blast, blast)
  using i_heap_is_wellformed_Core_DOM.parent_child_rel_finite apply blast
  using i_heap_is_wellformed_Core_DOM.parent_child_rel_acyclic apply blast
  using i_heap_is_wellformed_Core_DOM.parent_child_rel_node_ptr apply blast
  using i_heap_is_wellformed_Core_DOM.parent_child_rel_parent_in_heap apply blast
  using i_heap_is_wellformed_Core_DOM.parent_child_rel_child_in_heap apply blast
  done


locale l_heap_is_wellformed_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM_defs
```

```
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs
  + CD: l_heap_is_wellformed_Core_DOM
  known_ptr type_wf get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs heap_is_wellformed_Core_DOM parent_child_rel
  + l_heap_is_wellformed_defs
  heap_is_wellformed parent_child_rel
  + l_get_host_Shadow_DOM
  get_shadow_root get_shadow_root_locs get_host get_host_locs type_wf
  + l_get_disconnected_document_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs
  get_disconnected_document get_disconnected_document_locs type_wf
  + l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes heap_is_wellformed_impl: "heap_is_wellformed = a_heap_is_wellformed"
begin
lemmas heap_is_wellformed_def = heap_is_wellformed_impl[unfolded a_heap_is_wellformed_def,
    folded CD.heap_is_wellformed_impl CD.parent_child_rel_impl]

lemma a_distinct_lists_code [code]: "a_all_ptrs_in_heap h = ((∀host ∈ fset (element_ptr_kinds h).
h ⊢ ok (get_shadow_root host) ⟶ (case |h ⊢ get_shadow_root host|_r of
    Some shadow_root_ptr ⇒ shadow_root_ptr |∈| shadow_root_ptr_kinds h |
    None ⇒ True)))"
  apply(auto simp add: a_all_ptrs_in_heap_def split: option.splits)[1]
  by (meson is_OK_returns_result_I local.get_shadow_root_ptr_in_heap select_result_I2)

lemma get_shadow_root_shadow_root_ptr_in_heap:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  using assms
  by(auto simp add: heap_is_wellformed_def a_all_ptrs_in_heap_def)

lemma get_host_ptr_in_heap:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_host shadow_root_ptr →_r host"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  using assms get_shadow_root_shadow_root_ptr_in_heap
  by(auto simp add: get_host_def elim!: bind_returns_result_E2 dest!: filter_M_holds_for_result
      intro!: bind_pure_I split: list.splits)

lemma shadow_root_same_host:
  assumes "heap_is_wellformed h" and "type_wf h"
  assumes "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  assumes "h ⊢ get_shadow_root host' →_r Some shadow_root_ptr"
  shows "host = host'"
proof (rule ccontr)
  assume " host ≠ host'"
```

```
  have "host |∈| element_ptr_kinds h"
    using assms(3)
    by (meson is_OK_returns_result_I local.get_shadow_root_ptr_in_heap)
  moreover
  have "host' |∈| element_ptr_kinds h"
    using assms(4)
    by (meson is_OK_returns_result_I local.get_shadow_root_ptr_in_heap)
  ultimately show False
    using assms
    apply(auto simp add: heap_is_wellformed_def a_distinct_lists_def)[1]
    apply(drule distinct_concat_map_E(1)[where x=host and y=host'])
       apply(simp)
      apply(simp)
    using ‹host ≠ host'› apply(simp)
    apply(auto)[1]
    done
qed

lemma shadow_root_host_dual:
  assumes "h ⊢ get_host shadow_root_ptr →r host"
  shows "h ⊢ get_shadow_root host →r Some shadow_root_ptr"
  using assms
  by(auto simp add: get_host_def dest: filter_M_holds_for_result elim!: bind_returns_result_E2
      intro!: bind_pure_I split: list.splits)

lemma disc_doc_disc_node_dual:
  assumes "h ⊢ get_disconnected_document disc_node →r disc_doc"
  obtains disc_nodes where "h ⊢ get_disconnected_nodes disc_doc →r disc_nodes" and
    "disc_node ∈ set disc_nodes"
  using assms get_disconnected_nodes_pure
  by(auto simp add: get_disconnected_document_def bind_pure_I
      dest!: filter_M_holds_for_result
      elim!: bind_returns_result_E2
      intro!: filter_M_pure_I
      split: if_splits list.splits)

lemma get_host_valid_tag_name:
  assumes "heap_is_wellformed h" and "type_wf h"
  assumes "h ⊢ get_host shadow_root_ptr →r host"
  assumes "h ⊢ get_tag_name host →r tag"
  shows "tag ∈ safe_shadow_root_element_types"
proof -
  obtain host' where "host' |∈| element_ptr_kinds h" and
    "|h ⊢ get_tag_name host'|r ∈ safe_shadow_root_element_types"
    and "h ⊢ get_shadow_root host' →r Some shadow_root_ptr"
    using assms
    apply(auto simp add: heap_is_wellformed_def a_shadow_root_valid_def)[1]
    by (smt (z3) assms(1) get_host_ptr_in_heap local.get_shadow_root_ok returns_result_select_result)
  then have "host = host'"
    by (meson assms(1) assms(2) assms(3) shadow_root_host_dual shadow_root_same_host)
  then show ?thesis
    by (smt ‹⋀thesis. (⋀host'. ⟦host' |∈| element_ptr_kinds h; |h ⊢ get_tag_name host'|r ∈
safe_shadow_root_element_types; h ⊢ get_shadow_root host' →r Some shadow_root_ptr⟧ ⟹ thesis) ⟹
thesis› ‹h ⊢ get_shadow_root host' →r Some shadow_root_ptr› assms(1) assms(2) assms(4)
        select_result_I2 shadow_root_same_host)
qed


lemma a_host_shadow_root_rel_finite: "finite (a_host_shadow_root_rel h)"
proof -
  have "a_host_shadow_root_rel h = (⋃host ∈ fset (element_ptr_kinds h).
(case |h ⊢ get_shadow_root host|r of Some shadow_root ⇒ {(cast host, cast shadow_root)} | None ⇒ {}))"
    by(auto simp add: a_host_shadow_root_rel_def split: option.splits)
```

```
  moreover have "finite (⋃host ∈ fset (element_ptr_kinds h). (case |h ⊢ get_shadow_root host|_r of
Some shadow_root ⇒ {(cast_element_ptr2object_ptr host, cast_shadow_root_ptr2object_ptr shadow_root)} | None ⇒
{}))"
    by(auto split: option.splits)
  ultimately show ?thesis
    by auto
qed

lemma a_ptr_disconnected_node_rel_finite: "finite (a_ptr_disconnected_node_rel h)"
proof -
  have "a_ptr_disconnected_node_rel h = (⋃owner_document ∈ set |h ⊢ document_ptr_kinds_M|_r.
                (⋃disconnected_node ∈ set |h ⊢ get_disconnected_nodes owner_document|_r.
{(cast owner_document, cast disconnected_node)}))"
    by(auto simp add: a_ptr_disconnected_node_rel_def)
  moreover have "finite (⋃owner_document ∈ set |h ⊢ document_ptr_kinds_M|_r.
                  (⋃disconnected_node ∈ set |h ⊢ get_disconnected_nodes owner_document|_r.
{(cast_document_ptr2object_ptr owner_document, cast_node_ptr2object_ptr disconnected_node)}))"
    by simp
  ultimately show ?thesis
    by simp
qed


lemma heap_is_wellformed_children_in_heap:
  "heap_is_wellformed h ⟹ h ⊢ get_child_nodes ptr →_r children ⟹ child ∈ set children ⟹
child |∈| node_ptr_kinds h"
  using CD.heap_is_wellformed_children_in_heap local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_disc_nodes_in_heap:
  "heap_is_wellformed h ⟹ h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes ⟹
node ∈ set disc_nodes ⟹ node |∈| node_ptr_kinds h"
  using CD.heap_is_wellformed_disc_nodes_in_heap local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_one_parent: "heap_is_wellformed h ⟹
h ⊢ get_child_nodes ptr →_r children ⟹ h ⊢ get_child_nodes ptr' →_r children' ⟹
set children ∩ set children' ≠ {} ⟹ ptr = ptr'"
  using CD.heap_is_wellformed_one_parent local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_one_disc_parent: "heap_is_wellformed h ⟹
h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes ⟹
h ⊢ get_disconnected_nodes document_ptr' →_r disc_nodes' ⟹ set disc_nodes ∩ set disc_nodes' ≠ {} ⟹
document_ptr = document_ptr'"
  using CD.heap_is_wellformed_one_disc_parent local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_children_disc_nodes_different: "heap_is_wellformed h ⟹
h ⊢ get_child_nodes ptr →_r children ⟹ h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes ⟹
set children ∩ set disc_nodes = {}"
  using CD.heap_is_wellformed_children_disc_nodes_different local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_disconnected_nodes_distinct: "heap_is_wellformed h ⟹
h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes ⟹ distinct disc_nodes"
  using CD.heap_is_wellformed_disconnected_nodes_distinct local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_children_distinct: "heap_is_wellformed h ⟹
h ⊢ get_child_nodes ptr →_r children ⟹ distinct children"
  using CD.heap_is_wellformed_children_distinct local.heap_is_wellformed_def by blast
lemma heap_is_wellformed_children_disc_nodes: "heap_is_wellformed h ⟹
node_ptr |∈| node_ptr_kinds h ⟹  ¬(∃parent ∈ fset (object_ptr_kinds h).
node_ptr ∈ set |h ⊢ get_child_nodes parent|_r) ⟹ (∃document_ptr ∈ fset (document_ptr_kinds h).
node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|_r)"
  using CD.heap_is_wellformed_children_disc_nodes local.heap_is_wellformed_def by blast
lemma parent_child_rel_finite: "heap_is_wellformed h ⟹ finite (parent_child_rel h)"
  using CD.parent_child_rel_finite by blast
lemma parent_child_rel_acyclic: "heap_is_wellformed h ⟹ acyclic (parent_child_rel h)"
  using CD.parent_child_rel_acyclic heap_is_wellformed_def by blast
lemma parent_child_rel_child_in_heap: "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptr parent ⟹
(parent, child_ptr) ∈ parent_child_rel h ⟹ child_ptr |∈| object_ptr_kinds h"
  using CD.parent_child_rel_child_in_heap local.heap_is_wellformed_def by blast
end
```

interpretation *i_heap_is_wellformed?: l_heap_is_wellformed$_{Shadow_DOM}$ get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf heap_is_wellformed parent_child_rel heap_is_wellformed$_{Core_DOM}$ get_host get_host_locs get_disconnected_document get_disconnected_document_locs*
by *(auto simp add: l_heap_is_wellformed$_{Shadow_DOM}$_def l_heap_is_wellformed$_{Shadow_DOM}$_axioms_def l_heap_is_wellformed$_{Core_DOM}$_def l_heap_is_wellformed$_{Core_DOM}$_axioms_def heap_is_wellformed$_{Core_DOM}$_def parent_child_rel_def heap_is_wellformed_def instances)*
declare *l_heap_is_wellformed$_{Shadow_DOM}$_axioms [instances]*

lemma *heap_is_wellformed_is_l_heap_is_wellformed [instances]: "l_heap_is_wellformed ShadowRootClass.type_wf ShadowRootClass.known_ptr Shadow_DOM.heap_is_wellformed Shadow_DOM.parent_child_rel Shadow_DOM.get_child_nodes get_disconnected_nodes"*
apply *(auto simp add: l_heap_is_wellformed_def instances)[1]*
using *heap_is_wellformed_children_in_heap* **apply** *metis*
using *heap_is_wellformed_disc_nodes_in_heap* **apply** *metis*
using *heap_is_wellformed_one_parent* **apply** *blast*
using *heap_is_wellformed_one_disc_parent* **apply** *blast*
using *heap_is_wellformed_children_disc_nodes_different* **apply** *blast*
using *heap_is_wellformed_disconnected_nodes_distinct* **apply** *metis*
using *heap_is_wellformed_children_distinct* **apply** *metis*
using *heap_is_wellformed_children_disc_nodes* **apply** *metis*
using *i_heap_is_wellformed$_{Core_DOM}$.parent_child_rel_child* **apply** *(blast, blast)*
using *i_heap_is_wellformed$_{Core_DOM}$.parent_child_rel_finite* **apply** *blast*
using *parent_child_rel_acyclic* **apply** *blast*
using *i_heap_is_wellformed$_{Core_DOM}$.parent_child_rel_node_ptr* **apply** *blast*
using *i_heap_is_wellformed$_{Core_DOM}$.parent_child_rel_parent_in_heap* **apply** *blast*
using *parent_child_rel_child_in_heap* **apply** *metis*
done

get_parent

interpretation *i_get_parent_wf$_{Core_DOM}$: l_get_parent_wf$_{Core_DOM}$ known_ptr type_wf get_child_nodes get_child_nodes_locs known_ptrs get_parent get_parent_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_disconnected_nodes*
by *(simp add: l_get_parent_wf$_{Core_DOM}$_def instances)*
declare *i_get_parent_wf$_{Core_DOM}$.l_get_parent_wf$_{Core_DOM}$_axioms[instances]*

interpretation *i_get_parent_wf2$_{Core_DOM}$: l_get_parent_wf2$_{Core_DOM}$ known_ptr type_wf get_child_nodes get_child_nodes_locs known_ptrs get_parent get_parent_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_disconnected_nodes get_disconnected_nodes_locs*
by *(auto simp add: l_get_parent_wf2$_{Core_DOM}$_def instances)*
declare *i_get_parent_wf2$_{Core_DOM}$.l_get_parent_wf2$_{Core_DOM}$_axioms[instances]*

lemma *get_parent_wf$_{Core_DOM}$_is_l_get_parent_wf [instances]: "l_get_parent_wf type_wf known_ptr known_ptrs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes get_parent"*
apply *(auto simp add: l_get_parent_wf_def l_get_parent_wf_axioms_def instances)[1]*
using *i_get_parent_wf2$_{Core_DOM}$.child_parent_dual* **apply** *fast*
using *i_get_parent_wf2$_{Core_DOM}$.heap_wellformed_induct* **apply** *metis*
using *i_get_parent_wf2$_{Core_DOM}$.heap_wellformed_induct_rev* **apply** *metis*
using *i_get_parent_wf2$_{Core_DOM}$.parent_child_rel_parent* **apply** *fast*
done

get_disconnected_nodes

set_disconnected_nodes

get_disconnected_nodes **interpretation** *i_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$: l_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$ known_ptr type_wf get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes*
by *(simp add: l_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$_def instances)*
declare *i_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$.l_set_disconnected_nodes_get_disconnected_nodes*

lemma *set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$_is_l_set_disconnected_nodes_get_disconnected_node*
[instances]:
"l_set_disconnected_nodes_get_disconnected_nodes_wf type_wf known_ptr heap_is_wellformed$_{Core_DOM}$
parent_child_rel get_child_nodes get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
set_disconnected_nodes_locs"
apply(*auto simp add: l_set_disconnected_nodes_get_disconnected_nodes_wf_def*
l_set_disconnected_nodes_get_disconnected_nodes_wf_axioms_def instances)[1]
using *i_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$.remove_from_disconnected_nodes_removes*
apply *fast*
done

get_root_node

interpretation *i_get_root_node_wf$_{Core_DOM}$:*
l_get_root_node_wf$_{Core_DOM}$ known_ptr type_wf known_ptrs heap_is_wellformed$_{Core_DOM}$ parent_child_rel
get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_parent
get_parent_locs get_ancestors get_ancestors_locs get_root_node get_root_node_locs
by(*simp add: l_get_root_node_wf$_{Core_DOM}$_def instances)*
declare *i_get_root_node_wf$_{Core_DOM}$.l_get_root_node_wf$_{Core_DOM}$_axioms[instances]*

lemma *get_ancestors_wf$_{Core_DOM}$_is_l_get_ancestors_wf [instances]:*
"l_get_ancestors_wf heap_is_wellformed$_{Core_DOM}$ parent_child_rel known_ptr known_ptrs type_wf
get_ancestors get_ancestors_locs get_child_nodes get_parent"
apply(*auto simp add: l_get_ancestors_wf_def l_get_ancestors_wf_axioms_def instances)[1]*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_never_empty* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_ok* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_reads* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_ptrs_in_heap* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_remains_not_in_ancestors* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_also_parent* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_obtains_children* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_parent_child_rel* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_parent_child_rel* **apply** *blast*
done

lemma *get_root_node_wf$_{Core_DOM}$_is_l_get_root_node_wf [instances]:*
"l_get_root_node_wf heap_is_wellformed$_{Core_DOM}$ get_root_node type_wf known_ptr known_ptrs
get_ancestors get_parent"
apply(*auto simp add: l_get_root_node_wf_def l_get_root_node_wf_axioms_def instances)[1]*
using *i_get_root_node_wf$_{Core_DOM}$.get_root_node_ok* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_root_node_ptr_in_heap* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_root_node_root_in_heap* **apply** *blast*
using *i_get_root_node_wf$_{Core_DOM}$.get_ancestors_same_root_node* **apply**(*blast, blast)*
using *i_get_root_node_wf$_{Core_DOM}$.get_root_node_same_no_parent* **apply** *blast*

using *i_get_root_node_wf$_{Core_DOM}$.get_root_node_parent_same* **apply** *(blast, blast)*
done

to_tree_order

interpretation *i_to_tree_order_wf$_{Core_DOM}$: l_to_tree_order_wf$_{Core_DOM}$ known_ptr type_wf get_child_nodes*
get_child_nodes_locs to_tree_order known_ptrs get_parent get_parent_locs heap_is_wellformed$_{Core_DOM}$
parent_child_rel get_disconnected_nodes get_disconnected_nodes_locs
apply(*simp add: l_to_tree_order_wf$_{Core_DOM}$_def instances)*
done
declare *i_to_tree_order_wf$_{Core_DOM}$.l_to_tree_order_wf$_{Core_DOM}$_axioms [instances]*

lemma *to_tree_order_wf$_{Core_DOM}$_is_l_to_tree_order_wf [instances]:*
"l_to_tree_order_wf heap_is_wellformed$_{Core_DOM}$ parent_child_rel type_wf known_ptr known_ptrs
to_tree_order get_parent get_child_nodes"
apply(*auto simp add: l_to_tree_order_wf_def l_to_tree_order_wf_axioms_def instances)[1]*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_ok* **apply** *blast*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_ptrs_in_heap* **apply** *blast*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_parent_child_rel* **apply**(*blast, blast)*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_child2* **apply** *blast*

using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_node_ptrs* **apply** *blast*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_child* **apply** *blast*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_ptr_in_result* **apply** *blast*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_parent* **apply** *blast*
using *i_to_tree_order_wf$_{Core_DOM}$.to_tree_order_subset* **apply** *blast*
done

get_root_node **interpretation** *i_to_tree_order_wf_get_root_node_wf$_{Core_DOM}$: l_to_tree_order_wf_get_root_node_wf$_{C}$*
known_ptr type_wf known_ptrs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes
get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_parent get_parent_locs
get_ancestors get_ancestors_locs get_root_node get_root_node_locs to_tree_order
by*(auto simp add: l_to_tree_order_wf_get_root_node_wf$_{Core_DOM}$_def instances)*
declare *i_to_tree_order_wf_get_root_node_wf$_{Core_DOM}$.l_to_tree_order_wf_get_root_node_wf$_{Core_DOM}$_axioms [instances]*

lemma *to_tree_order_wf_get_root_node_wf$_{Core_DOM}$_is_l_to_tree_order_wf_get_root_node_wf [instances]:*
"l_to_tree_order_wf_get_root_node_wf type_wf known_ptr known_ptrs to_tree_order get_root_node heap_is_wellformed
apply*(auto simp add: l_to_tree_order_wf_get_root_node_wf_def l_to_tree_order_wf_get_root_node_wf_axioms_def instances)[1]*
using *i_to_tree_order_wf_get_root_node_wf$_{Core_DOM}$.to_tree_order_get_root_node* **apply** *blast*
using *i_to_tree_order_wf_get_root_node_wf$_{Core_DOM}$.to_tree_order_same_root* **apply** *blast*
done

remove_child

interpretation *i_remove_child_wf2$_{Core_DOM}$: l_remove_child_wf2$_{Core_DOM}$ get_child_nodes get_child_nodes_locs*
set_child_nodes set_child_nodes_locs get_parent
get_parent_locs get_owner_document get_disconnected_nodes get_disconnected_nodes_locs
set_disconnected_nodes
set_disconnected_nodes_locs remove_child remove_child_locs remove type_wf known_ptr known_ptrs
heap_is_wellformed$_{Core_DOM}$
parent_child_rel
by *unfold_locales*
declare *i_remove_child_wf2$_{Core_DOM}$.l_remove_child_wf2$_{Core_DOM}$_axioms [instances]*

lemma *remove_child_wf2$_{Core_DOM}$_is_l_remove_child_wf2 [instances]:*
"l_remove_child_wf2 type_wf known_ptr known_ptrs remove_child heap_is_wellformed$_{Core_DOM}$ get_child_nodes remove"
apply*(auto simp add: l_remove_child_wf2_def l_remove_child_wf2_axioms_def instances)[1]*
using *i_remove_child_wf2$_{Core_DOM}$.remove_child_heap_is_wellformed_preserved* **apply***(fast, fast, fast)*
using *i_remove_child_wf2$_{Core_DOM}$.remove_heap_is_wellformed_preserved* **apply***(fast, fast, fast)*
using *i_remove_child_wf2$_{Core_DOM}$.remove_child_removes_child* **apply** *fast*
using *i_remove_child_wf2$_{Core_DOM}$.remove_child_removes_first_child* **apply** *fast*
using *i_remove_child_wf2$_{Core_DOM}$.remove_removes_child* **apply** *fast*
using *i_remove_child_wf2$_{Core_DOM}$.remove_for_all_empty_children* **apply** *fast*
done

2.3.3 A wellformed heap

get_parent

interpretation *i_get_parent_wf?: l_get_parent_wf$_{Core_DOM}$ known_ptr type_wf get_child_nodes*
get_child_nodes_locs known_ptrs get_parent get_parent_locs heap_is_wellformed parent_child_rel
get_disconnected_nodes
using *instances*
by*(simp add: l_get_parent_wf$_{Core_DOM}$_def)*
declare *l_get_parent_wf$_{Core_DOM}$_axioms[instances]*

lemma *get_parent_wf_is_l_get_parent_wf [instances]: "l_get_parent_wf ShadowRootClass.type_wf ShadowRootClass.known_ptr ShadowRootClass.known_ptrs heap_is_wellformed parent_child_rel Shadow_DOM.get_child_nodes Shadow_DOM.get_parent"*
apply*(auto simp add: l_get_parent_wf_def l_get_parent_wf_axioms_def instances)[1]*
using *child_parent_dual* **apply** *blast*

```
  using heap_wellformed_induct apply metis
  using heap_wellformed_induct_rev apply metis
  using parent_child_rel_parent apply metis
  done
```

remove_shadow_root

```
locale l_remove_shadow_root_wf_Shadow_DOM =
  l_get_tag_name +
  l_get_disconnected_nodes  +
  l_set_shadow_root_get_tag_name +
  l_get_child_nodes  +
  l_heap_is_wellformed_Shadow_DOM +
  l_remove_shadow_root_Shadow_DOM +
  l_delete_shadow_root_get_disconnected_nodes +
  l_delete_shadow_root_get_child_nodes +
  l_set_shadow_root_get_disconnected_nodes +
  l_set_shadow_root_get_child_nodes +
  l_delete_shadow_root_get_tag_name +
  l_set_shadow_root_get_shadow_root +
  l_delete_shadow_root_get_shadow_root +
  l_get_parent_Core_DOM
begin

lemma remove_shadow_root_preserves:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_shadow_root ptr →_h h'"
  shows "known_ptrs h'" and "type_wf h'" "heap_is_wellformed h'"
proof -
  obtain shadow_root_ptr h2 where
    "h ⊢ get_shadow_root ptr →_r Some shadow_root_ptr" and
    "h ⊢ get_child_nodes (cast shadow_root_ptr) →_r []" and
    "h ⊢  get_disconnected_nodes (cast shadow_root_ptr) →_r []" and
    h2: "h ⊢ set_shadow_root ptr None →_h h2" and
    h': "h2 ⊢ delete_M shadow_root_ptr →_h h'"
    using assms(4)
    by(auto simp add: remove_shadow_root_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_shadow_root_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        split: option.splits if_splits)

  have "type_wf h2"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_shadow_root_writes h2]
    using ‹type_wf h› set_shadow_root_types_preserved
    by(auto simp add: reflp_def transp_def)
  then show "type_wf h'"
    using h' delete_shadow_root_type_wf_preserved local.type_wf_impl
    by blast

  have object_ptr_kinds_eq_h: "object_ptr_kinds h = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
          OF set_shadow_root_writes h2])
    using set_shadow_root_pointers_preserved
      apply blast
    by (auto simp add: reflp_def transp_def)
  have node_ptr_kinds_eq_h: "node_ptr_kinds h = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h
    by (simp add: node_ptr_kinds_def)
  have element_ptr_kinds_eq_h: "element_ptr_kinds h = element_ptr_kinds h2"
    using node_ptr_kinds_eq_h
    by (simp add: element_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h = document_ptr_kinds h2"
```

```
    using object_ptr_kinds_eq_h
    by (simp add: document_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_h: "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h2"
    using object_ptr_kinds_eq_h
    by (simp add: document_ptr_kinds_eq_h shadow_root_ptr_kinds_def)

  have "known_ptrs h2"
    using ‹known_ptrs h› object_ptr_kinds_eq_h known_ptrs_subset
    by blast

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h' |⊆| object_ptr_kinds h2"
    using h' delete_shadow_root_pointers
    by auto
  have object_ptr_kinds_eq2_h2: "object_ptr_kinds h2 = object_ptr_kinds h' |∪| {|cast shadow_root_ptr|}"
    using h' delete_shadow_root_pointers
    by auto
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h2 = node_ptr_kinds h'"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def delete_shadow_root_pointers[OF h'])
  have element_ptr_kinds_eq_h2: "element_ptr_kinds h2 = element_ptr_kinds h'"
    using node_ptr_kinds_eq_h2
    by (simp add: element_ptr_kinds_def)
  have document_ptr_kinds_eq_h2: "document_ptr_kinds h2 = document_ptr_kinds h' |∪| {|cast shadow_root_ptr|}"
    using object_ptr_kinds_eq_h2
    by (auto simp add: document_ptr_kinds_def delete_shadow_root_pointers[OF h'])
  then
  have document_ptr_kinds_eq2_h2: "document_ptr_kinds h' |⊆| document_ptr_kinds h2"
    using h' delete_shadow_root_pointers
    by auto
  have shadow_root_ptr_kinds_eq_h2: "shadow_root_ptr_kinds h' |⊆| shadow_root_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    apply(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)[1]
    by auto
  have shadow_root_ptr_kinds_eq2_h2: "shadow_root_ptr_kinds h2 = shadow_root_ptr_kinds h' |∪| {|shadow_root_ptr|}"
    using object_ptr_kinds_eq2_h2 document_ptr_kinds_eq_h2
    by (auto simp add: shadow_root_ptr_kinds_def)

  show "known_ptrs h'"
    using object_ptr_kinds_eq_h2 ‹known_ptrs h2› known_ptrs_subset
    by blast


  have disconnected_nodes_eq_h:
    "⋀doc_ptr disc_nodes. h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes =
h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads set_shadow_root_writes h2 set_shadow_root_get_disconnected_nodes
    by(rule reads_writes_preserved)
  then have disconnected_nodes_eq2_h:
    "⋀doc_ptr. |h ⊢ get_disconnected_nodes doc_ptr|r = |h2 ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force

  have disconnected_nodes_eq_h2:
    "⋀doc_ptr disc_nodes. doc_ptr ≠ cast shadow_root_ptr ⟹ h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
=
h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads get_disconnected_nodes_delete_shadow_root[rotated, OF h']
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by(metis (no_types, lifting))+
  then have disconnected_nodes_eq2_h2:
    "⋀doc_ptr. doc_ptr ≠ cast shadow_root_ptr ⟹ |h2 ⊢ get_disconnected_nodes doc_ptr|r =
|h' ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force
```

```
  have tag_name_eq_h:
    "⋀doc_ptr disc_nodes. h ⊢ get_tag_name doc_ptr →r disc_nodes =
h2 ⊢ get_tag_name doc_ptr →r disc_nodes"
    using get_tag_name_reads set_shadow_root_writes h2 set_shadow_root_get_tag_name
    by(rule reads_writes_preserved)
  then have tag_name_eq2_h: "⋀doc_ptr. |h ⊢ get_tag_name doc_ptr|r = |h2 ⊢ get_tag_name doc_ptr|r"
    using select_result_eq by force

  have tag_name_eq_h2:
    "⋀doc_ptr disc_nodes. h2 ⊢ get_tag_name doc_ptr →r disc_nodes = h' ⊢ get_tag_name doc_ptr →r disc_nodes"
    using get_tag_name_reads get_tag_name_delete_shadow_root[OF h']
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have tag_name_eq2_h2: "⋀doc_ptr. |h2 ⊢ get_tag_name doc_ptr|r = |h' ⊢ get_tag_name doc_ptr|r"
    using select_result_eq by force


  have children_eq_h:
    "⋀ptr' children. h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads set_shadow_root_writes h2 set_shadow_root_get_child_nodes
    by(rule reads_writes_preserved)

  then have children_eq2_h: "⋀ptr'. |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force


  have children_eq_h2:
    "⋀ptr' children. ptr' ≠ cast shadow_root_ptr ⟹ h2 ⊢ get_child_nodes ptr' →r children =
h' ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads h' get_child_nodes_delete_shadow_root
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h2:
    "⋀ptr'. ptr' ≠ cast shadow_root_ptr ⟹ |h2 ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force

  have "cast shadow_root_ptr |∉| object_ptr_kinds h'"
    using h' delete_ShadowRoot_M_ptr_not_in_heap
    by auto

  have get_shadow_root_eq_h:
    "⋀shadow_root_opt ptr'. ptr ≠ ptr' ⟹ h ⊢ get_shadow_root ptr' →r shadow_root_opt =
h2 ⊢ get_shadow_root ptr' →r shadow_root_opt"
    using get_shadow_root_reads set_shadow_root_writes h2
    apply(rule reads_writes_preserved)
    using set_shadow_root_get_shadow_root_different_pointers
    by fast

  have get_shadow_root_eq_h2:
    "⋀shadow_root_opt ptr'. h2 ⊢ get_shadow_root ptr' →r shadow_root_opt =
h' ⊢ get_shadow_root ptr' →r shadow_root_opt"
    using get_shadow_root_reads get_shadow_root_delete_shadow_root[OF h']
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then
  have get_shadow_root_eq2_h2: "⋀ptr'. |h2 ⊢ get_shadow_root ptr'|r = |h' ⊢ get_shadow_root ptr'|r"
    using select_result_eq by force



  have "acyclic (parent_child_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)
```

```
moreover
have "parent_child_rel h = parent_child_rel h2"
  by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h children_eq2_h)
moreover
have "parent_child_rel h' ⊆ parent_child_rel h2"
  using object_ptr_kinds_eq_h2
  apply(auto simp add: CD.parent_child_rel_def)[1]
  by (metis ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'› children_eq2_h2)
ultimately
have "CD.a_acyclic_heap h'"
  using acyclic_subset
  by (auto simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)

moreover
have "CD.a_all_ptrs_in_heap h"
  using ‹heap_is_wellformed h›
  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
hence "CD.a_all_ptrs_in_heap h2"
  by (simp add: children_eq2_h disconnected_nodes_eq2_h document_ptr_kinds_eq_h
      l_heap_is_wellformed_Core_DOM_defs.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h
      object_ptr_kinds_eq_h)
hence "CD.a_all_ptrs_in_heap h'"
  by (metis (no_types, opaque_lifting)
      ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'› children_eq2_h2
      delete_ShadowRoot_M_ptr_not_in_heap disconnected_nodes_eq2_h2 document_ptr_kinds_eq2_h2
      fsubsetD h' CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h2 object_ptr_kinds_eq_h2
      shadow_root_ptr_kinds_commutes)

moreover
have "CD.a_distinct_lists h"
  using ‹heap_is_wellformed h›
  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
then have "CD.a_distinct_lists h2"
  by(auto simp add: CD.a_distinct_lists_def object_ptr_kinds_eq_h document_ptr_kinds_eq_h
      children_eq2_h disconnected_nodes_eq2_h)
then have "CD.a_distinct_lists h'"
  apply(auto simp add: CD.a_distinct_lists_def document_ptr_kinds_eq_h2 disconnected_nodes_eq2_h2)[1]
    apply(auto simp add:  intro!: distinct_concat_map_I)[1]
     apply(case_tac "x = cast shadow_root_ptr")
  using ‹cast shadow_root_ptr |∉| object_ptr_kinds h'› apply simp
  using children_eq_h2 concat_map_all_distinct[of "(λptr. |h2 ⊢ get_child_nodes ptr|_r)"]
     apply (metis (no_types, lifting) children_eq2_h2 finite_fset fset_mp
      object_ptr_kinds_eq_h2 set_sorted_list_of_set)
    apply(case_tac "x = cast shadow_root_ptr")
  using ‹cast shadow_root_ptr |∉| object_ptr_kinds h'› apply simp
    apply(case_tac "y = cast shadow_root_ptr")
  using ‹cast shadow_root_ptr |∉| object_ptr_kinds h'› apply simp
  using children_eq_h2 distinct_concat_map_E(1)[of "(λptr. |h2 ⊢ get_child_nodes ptr|_r)"]
    apply (smt (verit) IntI children_eq2_h2 empty_iff finite_fset fset_mp
      object_ptr_kinds_eq_h2 set_sorted_list_of_set)

   apply(auto simp add:  intro!: distinct_concat_map_I)[1]
    apply(case_tac "x = cast shadow_root_ptr")
  using ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'› document_ptr_kinds_commutes
     apply blast
    apply (metis (mono_tags, lifting) ‹local.CD.a_distinct_lists h2› ‹type_wf h'›
      disconnected_nodes_eq_h2 is_OK_returns_result_E local.CD.distinct_lists_disconnected_nodes
      local.get_disconnected_nodes_ok select_result_I2)
   apply(case_tac "x = cast shadow_root_ptr")
  using ‹cast shadow_root_ptr |∉| object_ptr_kinds h'› apply simp
   apply(case_tac "y = cast shadow_root_ptr")
  using ‹cast shadow_root_ptr |∉| object_ptr_kinds h'› apply simp
proof -
```

```
    fix x  and y and xa
    assume a1: "x |∈| document_ptr_kinds h'"
    assume a2: "y |∈| document_ptr_kinds h'"
    assume a3: "x ≠ y"
    assume a4: "x ≠ cast_shadow_root_ptr2document_ptr shadow_root_ptr"
    assume a5: "y ≠ cast_shadow_root_ptr2document_ptr shadow_root_ptr"
    assume a6: "xa ∈ set |h' ⊢ get_disconnected_nodes x|_r"
    assume a7: "xa ∈ set |h' ⊢ get_disconnected_nodes y|_r"
    assume "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
(insort (cast_shadow_root_ptr2document_ptr shadow_root_ptr) (sorted_list_of_set (fset (document_ptr_kinds h')
-
{cast_shadow_root_ptr2document_ptr shadow_root_ptr})))))"
    then show False
      using a7 a6 a5 a4 a3 a2 a1 by (metis (no_types) IntI
          distinct_concat_map_E(1)[of "(λptr. |h2 ⊢ get_disconnected_nodes ptr|_r)"] disconnected_nodes_eq2_h2
          empty_iff finite_fset finsert.rep_eq insert_iff set_sorted_list_of_set
          sorted_list_of_set_insert_remove)
  next
    fix x xa xb
    assume 0: "distinct (concat (map (λptr. |h2 ⊢ get_child_nodes ptr|_r)
(sorted_list_of_set (fset (object_ptr_kinds h2)))))"
      and 1: "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
(insort (cast_shadow_root_ptr2document_ptr shadow_root_ptr) (sorted_list_of_set (fset (document_ptr_kinds h')
-
{cast_shadow_root_ptr2document_ptr shadow_root_ptr})))))"
      and 2: "(⋃x∈fset (object_ptr_kinds h2). set |h2 ⊢ get_child_nodes x|_r) ∩
(⋃x∈set (insort (cast_shadow_root_ptr2document_ptr shadow_root_ptr) (sorted_list_of_set (fset (document_ptr_kinds
h') -
{cast_shadow_root_ptr2document_ptr shadow_root_ptr}))). set |h2 ⊢ get_disconnected_nodes x|_r) = {}"
      and 3: "xa |∈| object_ptr_kinds h'"
      and 4: "x ∈ set |h' ⊢ get_child_nodes xa|_r"
      and 5: "xb |∈| document_ptr_kinds h'"
      and 6: "x ∈ set |h' ⊢ get_disconnected_nodes xb|_r"
    then show "False"
      apply(cases "xa = cast shadow_root_ptr")
      using ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'› apply blast
      apply(cases "xb = cast shadow_root_ptr")
      using ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'› document_ptr_kinds_commutes
       apply blast
      by (metis (no_types, opaque_lifting) ‹local.CD.a_distinct_lists h2› ‹type_wf h'› children_eq2_h2
          disconnected_nodes_eq_h2 fset_rev_mp is_OK_returns_result_E local.CD.distinct_lists_no_parent
          local.get_disconnected_nodes_ok object_ptr_kinds_eq_h2 select_result_I2)
  qed
  moreover
  have "CD.a_owner_document_valid h"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_owner_document_valid h2"
    by(auto simp add: CD.a_owner_document_valid_def object_ptr_kinds_eq_h document_ptr_kinds_eq_h
        node_ptr_kinds_eq_h children_eq2_h disconnected_nodes_eq2_h)
  then have "CD.a_owner_document_valid h'"
    apply(auto simp add: CD.a_owner_document_valid_def  document_ptr_kinds_eq_h2 node_ptr_kinds_eq_h2
        disconnected_nodes_eq2_h2)[1]
    by (smt (z3) ‹h ⊢ get_child_nodes (cast_shadow_root_ptr2object_ptr shadow_root_ptr) →_r []›
        ‹h ⊢ get_disconnected_nodes (cast_shadow_root_ptr2document_ptr shadow_root_ptr) →_r []› ‹local.CD.a_distinct_l
h›
        children_eq2_h children_eq2_h2 disconnected_nodes_eq2_h disconnected_nodes_eq2_h2 finsert_iff
        funion_finsert_right local.CD.distinct_lists_no_parent object_ptr_kinds_eq2_h2 object_ptr_kinds_eq_h
        select_result_I2 sup_bot.comm_neutral)

  ultimately have "heap_is_wellformed_Core_DOM h'"
    by(simp add: CD.heap_is_wellformed_def)
```

```
  moreover
  have "acyclic (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using ‹heap_is_wellformed h›
    by(simp add: heap_is_wellformed_def)
  then
  have "acyclic (parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2)"
  proof -
    have "a_host_shadow_root_rel h2 ⊆ a_host_shadow_root_rel h"
      apply(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h)[1]
      apply(case_tac "aa = ptr")
       apply(simp)
       apply (metis (no_types, lifting) ‹type_wf h2› assms(2) h2 local.get_shadow_root_ok
          local.type_wf_impl option.distinct(1) returns_result_eq returns_result_select_result
          set_shadow_root_get_shadow_root)
      using get_shadow_root_eq_h
      by (metis (mono_tags, lifting) ‹type_wf h2› image_eqI is_OK_returns_result_E
          local.get_shadow_root_ok mem_Collect_eq prod.simps(2) select_result_I2)
    moreover have "a_ptr_disconnected_node_rel h = a_ptr_disconnected_node_rel h2"
      by (simp add: a_ptr_disconnected_node_rel_def disconnected_nodes_eq2_h document_ptr_kinds_eq_h)
    ultimately show ?thesis
      using ‹parent_child_rel h = parent_child_rel h2›
      by (smt ‹acyclic (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪
local.a_ptr_disconnected_node_rel h)› acyclic_subset subset_refl sup_mono)
  qed
  then
  have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h')"
  proof -
    have "a_host_shadow_root_rel h' ⊆ a_host_shadow_root_rel h2"
      by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h2 get_shadow_root_eq2_h2)
    moreover have "a_ptr_disconnected_node_rel h2 = a_ptr_disconnected_node_rel h'"
      apply(simp add: a_ptr_disconnected_node_rel_def disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2)
      by (metis (no_types, lifting) ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'›
          ‹h ⊢ get_child_nodes (cast_shadow_root_ptr2object_ptr shadow_root_ptr) →_r []›
          ‹h ⊢ get_disconnected_nodes (cast_shadow_root_ptr2document_ptr shadow_root_ptr) →_r []› ‹local.CD.a_distinct
h›
          disconnected_nodes_eq2_h disconnected_nodes_eq2_h2 document_ptr_kinds_commutes is_OK_returns_result_I
          local.CD.distinct_lists_no_parent local.get_disconnected_nodes_ptr_in_heap select_result_I2)
    ultimately show ?thesis
      using ‹parent_child_rel h' ⊆ parent_child_rel h2›
        ‹acyclic (parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2)›
      using acyclic_subset order_refl sup_mono
      by (metis (no_types, opaque_lifting))
  qed

  moreover
  have "a_all_ptrs_in_heap h"
    using ‹heap_is_wellformed h›
    by(simp add: heap_is_wellformed_def)
  then
  have "a_all_ptrs_in_heap h2"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h)[1]
    apply(case_tac "host = ptr")
     apply(simp)
     apply (metis assms(2) h2 local.type_wf_impl option.distinct(1) returns_result_eq
        set_shadow_root_get_shadow_root)
    using get_shadow_root_eq_h
    by fastforce
  then
  have "a_all_ptrs_in_heap h'"
    apply(auto simp add: a_all_ptrs_in_heap_def get_shadow_root_eq_h2)[1]
    apply(auto simp add: shadow_root_ptr_kinds_eq2_h2)[1]
    by (metis (no_types, lifting) ‹h ⊢ get_shadow_root ptr →_r Some shadow_root_ptr› assms(1) assms(2)
        get_shadow_root_eq_h get_shadow_root_eq_h2 h2 local.shadow_root_same_host local.type_wf_impl
```

```
        option.distinct(1) select_result_I2 set_shadow_root_get_shadow_root)

  moreover
  have "a_distinct_lists h"
    using ‹heap_is_wellformed h›
    by(simp add: heap_is_wellformed_def)
  then
  have "a_distinct_lists h2"
    apply(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h)[1]
    apply(auto intro!: distinct_concat_map_I split: option.splits)[1]
    apply(case_tac "x = ptr")
     apply(simp)
     apply (metis (no_types, opaque_lifting) assms(2) h2 is_OK_returns_result_I
        l_set_shadow_root_get_shadow_root.set_shadow_root_get_shadow_root
        l_set_shadow_root_get_shadow_root_axioms local.type_wf_impl option.discI returns_result_eq
        returns_result_select_result)

    apply(case_tac "y = ptr")
     apply(simp)
     apply (metis (no_types, opaque_lifting) assms(2) h2 is_OK_returns_result_I
        l_set_shadow_root_get_shadow_root.set_shadow_root_get_shadow_root
        l_set_shadow_root_get_shadow_root_axioms local.type_wf_impl option.discI returns_result_eq
        returns_result_select_result)
    by (metis ‹type_wf h2› assms(1) assms(2) get_shadow_root_eq_h local.get_shadow_root_ok
        local.shadow_root_same_host returns_result_select_result)

  then
  have "a_distinct_lists h'"
    by(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h2 get_shadow_root_eq2_h2)

  moreover
  have "a_shadow_root_valid h"
    using ‹heap_is_wellformed h›
    by(simp add: heap_is_wellformed_def)
  then
  have "a_shadow_root_valid h'"
    apply(auto simp add: a_shadow_root_valid_def shadow_root_ptr_kinds_eq_h element_ptr_kinds_eq_h
        tag_name_eq2_h)[1]
    apply(simp add: shadow_root_ptr_kinds_eq2_h2 element_ptr_kinds_eq_h2 tag_name_eq2_h2)
    using get_shadow_root_eq_h get_shadow_root_eq_h2
    by (smt (z3) ‹cast_shadow_root_ptr2object_ptr shadow_root_ptr |∉| object_ptr_kinds h'›
        ‹h ⊢ get_shadow_root ptr →_r Some shadow_root_ptr› assms(2) document_ptr_kinds_commutes
        element_ptr_kinds_eq_h element_ptr_kinds_eq_h2 local.get_shadow_root_ok
        option.inject returns_result_select_result select_result_I2 shadow_root_ptr_kinds_commutes)

  ultimately show "heap_is_wellformed h'"
    by(simp add: heap_is_wellformed_def)
qed
end
interpretation i_remove_shadow_root_wf?: l_remove_shadow_root_wf_Shadow_DOM
  type_wf get_tag_name get_tag_name_locs get_disconnected_nodes get_disconnected_nodes_locs
  set_shadow_root set_shadow_root_locs known_ptr get_child_nodes get_child_nodes_locs get_shadow_root
  get_shadow_root_locs heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_host
  get_host_locs get_disconnected_document get_disconnected_document_locs remove_shadow_root
  remove_shadow_root_locs known_ptrs get_parent get_parent_locs
  by(auto simp add: l_remove_shadow_root_wf_Shadow_DOM_def instances)
declare l_remove_shadow_root_wf_Shadow_DOM_axioms [instances]
```

get_root_node

```
interpretation i_get_root_node_wf?:
  l_get_root_node_wf_Core_DOM known_ptr type_wf known_ptrs heap_is_wellformed parent_child_rel
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_parent
```

```
  get_parent_locs get_ancestors get_ancestors_locs get_root_node get_root_node_locs
  by(simp add: l_get_root_node_wf_Core_DOM_def instances)
declare l_get_root_node_wf_Core_DOM_axioms[instances]

lemma get_ancestors_wf_is_l_get_ancestors_wf [instances]:
  "l_get_ancestors_wf heap_is_wellformed parent_child_rel known_ptr known_ptrs type_wf get_ancestors
get_ancestors_locs get_child_nodes get_parent"
  apply(auto simp add: l_get_ancestors_wf_def l_get_ancestors_wf_axioms_def instances)[1]
  using get_ancestors_never_empty apply blast
  using get_ancestors_ok apply blast
  using get_ancestors_reads apply blast
  using get_ancestors_ptrs_in_heap apply blast
  using get_ancestors_remains_not_in_ancestors apply blast
  using get_ancestors_also_parent apply blast
  using get_ancestors_obtains_children apply blast
  using get_ancestors_parent_child_rel apply blast
  using get_ancestors_parent_child_rel apply blast
  done

lemma get_root_node_wf_is_l_get_root_node_wf [instances]:
  "l_get_root_node_wf heap_is_wellformed get_root_node type_wf known_ptr known_ptrs get_ancestors get_parent"
  using known_ptrs_is_l_known_ptrs
  apply(auto simp add: l_get_root_node_wf_def l_get_root_node_wf_axioms_def)[1]
  using get_root_node_ok apply blast
  using get_root_node_ptr_in_heap apply blast
  using get_root_node_root_in_heap apply blast
  using get_ancestors_same_root_node apply(blast, blast)
  using get_root_node_same_no_parent apply blast

  using get_root_node_parent_same apply (blast, blast)
  done
```

get_parent_get_host_get_disconnected_document

```
locale l_get_parent_get_host_get_disconnected_document_wf_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs
  known_ptr type_wf heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs
  get_disconnected_document get_disconnected_document_locs +
  l_get_disconnected_document get_disconnected_document get_disconnected_document_locs +
  l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_get_parent_wf type_wf known_ptr known_ptrs heap_is_wellformed parent_child_rel get_child_nodes
  get_child_nodes_locs get_parent get_parent_locs +
  l_get_shadow_root type_wf get_shadow_root get_shadow_root_locs +
  l_get_host get_host get_host_locs +
  l_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
```

```
    and known_ptrs :: "(_) heap ⇒ bool"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma a_host_shadow_root_rel_shadow_root:
  "h ⊢ get_shadow_root host →r shadow_root_option ⟹ shadow_root_option = Some shadow_root ⟷
((cast host, cast shadow_root) ∈ a_host_shadow_root_rel h)"
  apply(auto simp add: a_host_shadow_root_rel_def)[1]
  by(metis (mono_tags, lifting) case_prodI is_OK_returns_result_I
      l_get_shadow_root.get_shadow_root_ptr_in_heap local.l_get_shadow_root_axioms mem_Collect_eq
      pair_imageI select_result_I2)

lemma a_host_shadow_root_rel_host:
  "heap_is_wellformed h ⟹ h ⊢ get_host shadow_root →r host ⟹
((cast host, cast shadow_root) ∈ a_host_shadow_root_rel h)"
  apply(auto simp add: a_host_shadow_root_rel_def)[1]
  using shadow_root_host_dual
  by (metis (no_types, lifting) Collect_cong a_host_shadow_root_rel_shadow_root
      local.a_host_shadow_root_rel_def split_cong)

lemma a_ptr_disconnected_node_rel_disconnected_node:
  "h ⊢ get_disconnected_nodes document →r disc_nodes ⟹ node_ptr ∈ set disc_nodes ⟷
(cast document, cast node_ptr) ∈ a_ptr_disconnected_node_rel h"
  apply(auto simp add: a_ptr_disconnected_node_rel_def)[1]
  by (smt CD.get_disconnected_nodes_ptr_in_heap case_prodI is_OK_returns_result_I mem_Collect_eq
      pair_imageI select_result_I2)

lemma a_ptr_disconnected_node_rel_document:
  "heap_is_wellformed h ⟹ h ⊢ get_disconnected_document node_ptr →r document ⟹
(cast document, cast node_ptr) ∈ a_ptr_disconnected_node_rel h"
  apply(auto simp add: a_ptr_disconnected_node_rel_def)[1]
  using disc_doc_disc_node_dual
  by (metis (no_types, lifting) local.a_ptr_disconnected_node_rel_def
      a_ptr_disconnected_node_rel_disconnected_node)

lemma heap_wellformed_induct_si [consumes 1, case_names step]:
  assumes "heap_is_wellformed h"
  assumes "⋀parent. (⋀children child. h ⊢ get_child_nodes parent →r children ⟹ child ∈ set children
⟹
P (cast child))
        ⟹ (⋀shadow_root host. parent = cast host ⟹ h ⊢ get_shadow_root host →r Some shadow_root ⟹
 P (cast shadow_root))
        ⟹ (⋀owner_document disc_nodes node_ptr. parent = cast owner_document ⟹
h ⊢ get_disconnected_nodes owner_document →r disc_nodes ⟹ node_ptr ∈ set disc_nodes ⟹ P (cast node_ptr))
      ⟹ P parent"
  shows "P ptr"
proof -
  fix ptr
  have "finite (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using a_host_shadow_root_rel_finite a_ptr_disconnected_node_rel_finite
    using local.CD.parent_child_rel_finite local.CD.parent_child_rel_impl
    by auto
  then
  have "wf ((parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)⁻¹)"
    using assms(1)
    apply(simp add: heap_is_wellformed_def)
    by (simp add: finite_acyclic_wf_converse local.CD.parent_child_rel_impl)
  then show "?thesis"
  proof (induct rule: wf_induct_rule)
    case (less parent)
    then show ?case
      apply(auto)[1]
      using assms a_ptr_disconnected_node_rel_disconnected_node a_host_shadow_root_rel_shadow_root
```

```
        local.CD.parent_child_rel_child
      by blast
  qed
qed

lemma heap_wellformed_induct_rev_si [consumes 1, case_names step]:
  assumes "heap_is_wellformed h"
  assumes "⋀child. (⋀parent child_node. child = cast child_node ⟹
h ⊢ get_parent child_node →r Some parent ⟹ P parent)
      ⟹ (⋀host shadow_root. child = cast shadow_root ⟹ h ⊢ get_host shadow_root →r host ⟹
P (cast host))
      ⟹ (⋀disc_doc disc_node. child = cast disc_node ⟹
h ⊢ get_disconnected_document disc_node →r disc_doc⟹ P (cast disc_doc))
      ⟹ P child"
  shows "P ptr"
proof -
  fix ptr
  have "finite (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using a_host_shadow_root_rel_finite a_ptr_disconnected_node_rel_finite
    using local.CD.parent_child_rel_finite local.CD.parent_child_rel_impl
    by auto
  then
  have "wf (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using assms(1)
    apply(simp add: heap_is_wellformed_def)
    by (simp add: finite_acyclic_wf)
  then show "?thesis"
  proof (induct rule: wf_induct_rule)
    case (less parent)
    then show ?case
      apply(auto)[1]
      using parent_child_rel_parent a_host_shadow_root_rel_host a_ptr_disconnected_node_rel_document
      using assms(1) assms(2) by auto
  qed
qed
end

interpretation i_get_parent_get_host_get_disconnected_document_wf?:
  l_get_parent_get_host_get_disconnected_document_wf_Shadow_DOM
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf heap_is_wellformed
  parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs known_ptrs get_parent get_parent_locs
  by(auto simp add: l_get_parent_get_host_get_disconnected_document_wf_Shadow_DOM_def instances)
declare l_get_parent_get_host_get_disconnected_document_wf_Shadow_DOM_axioms [instances]


locale l_get_parent_get_host_wf =
  l_heap_is_wellformed_defs +
  l_get_parent_defs +
  l_get_shadow_root_defs +
  l_get_host_defs +
  l_get_child_nodes_defs +
  l_get_disconnected_document_defs +
  l_get_disconnected_nodes_defs +
  assumes heap_wellformed_induct_si [consumes 1, case_names step]:
    "heap_is_wellformed h
    ⟹ (⋀parent. (⋀children child. h ⊢ get_child_nodes parent →r children ⟹
child ∈ set children ⟹ P (cast child))
        ⟹ (⋀shadow_root host. parent = cast host ⟹
h ⊢ get_shadow_root host →r Some shadow_root ⟹  P (cast shadow_root))
        ⟹ (⋀owner_document disc_nodes node_ptr. parent = cast owner_document ⟹
h ⊢ get_disconnected_nodes owner_document →r disc_nodes ⟹ node_ptr ∈ set disc_nodes ⟹
```

```
P (cast node_ptr))
      ⟹ P parent)
    ⟹ P ptr"
  assumes heap_wellformed_induct_rev_si [consumes 1, case_names step]:
    "heap_is_wellformed h
    ⟹ (⋀child. (⋀parent child_node. child = cast child_node ⟹
h ⊢ get_parent child_node →r Some parent ⟹ P parent)
      ⟹ (⋀host shadow_root. child = cast shadow_root ⟹
h ⊢ get_host shadow_root →r host ⟹ P (cast host))
      ⟹ (⋀disc_doc disc_node. child = cast disc_node ⟹
h ⊢ get_disconnected_document disc_node →r disc_doc ⟹ P (cast disc_doc))
      ⟹ P child)
    ⟹ P ptr"

lemma l_get_parent_get_host_wf_is_get_parent_get_host_wf [instances]:
  "l_get_parent_get_host_wf heap_is_wellformed get_parent get_shadow_root get_host get_child_nodes
get_disconnected_document get_disconnected_nodes"
  using heap_wellformed_induct_si heap_wellformed_induct_rev_si
  using l_get_parent_get_host_wf_def by blast
```

get_host

```
locale l_get_host_wf_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM  get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs
  known_ptr type_wf heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs
+
  l_type_wf type_wf +
  l_get_host_Shadow_DOM get_shadow_root get_shadow_root_locs get_host get_host_locs type_wf +
  l_get_shadow_root type_wf get_shadow_root get_shadow_root_locs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
begin

lemma get_host_ok [simp]:
  assumes "heap_is_wellformed h"
  assumes "type_wf h"
  assumes "known_ptrs h"
  assumes "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  shows "h ⊢ ok (get_host shadow_root_ptr)"
proof -
  obtain host where host: "host |∈| element_ptr_kinds h"
    and "|h ⊢ get_tag_name host|r ∈ safe_shadow_root_element_types"
    and shadow_root: "h ⊢ get_shadow_root host →r Some shadow_root_ptr"
    using assms(1) assms(4) get_shadow_root_ok assms(2)
    apply(auto simp add: heap_is_wellformed_def a_shadow_root_valid_def)[1]
    by (smt (z3) returns_result_select_result)
```

```
  obtain host_candidates where
    host_candidates: "h ⊢ filter_M (λelement_ptr. Heap_Error_Monad.bind (get_shadow_root element_ptr)
(λshadow_root_opt. return (shadow_root_opt = Some shadow_root_ptr)))
                (sorted_list_of_set (fset (element_ptr_kinds h)))
          →r host_candidates"
    apply(drule is_OK_returns_result_E[rotated])
    using get_shadow_root_ok assms(2)
    by(auto intro!: filter_M_is_OK_I bind_pure_I bind_is_OK_I2)
  then have "host_candidates = [host]"
    apply(rule filter_M_ex1)
    using host apply(auto)[1]
       apply (smt (verit) assms(1) assms(2) bind_pure_returns_result_I2 bind_returns_result_E host
        local.get_shadow_root_ok local.get_shadow_root_pure local.shadow_root_same_host return_returns_result
        returns_result_eq shadow_root sorted_list_of_fset.rep_eq sorted_list_of_fset_simps(1))
      apply (simp add: bind_pure_I)
     apply(auto intro!: bind_pure_returns_result_I)[1]
    apply (smt (verit) assms(2) bind_pure_returns_result_I2 host local.get_shadow_root_ok
        local.get_shadow_root_pure return_returns_result returns_result_eq shadow_root)
    done

  then
  show ?thesis
    using host_candidates host assms(1) get_shadow_root_ok
    apply(auto simp add: get_host_def known_ptrs_known_ptr
        intro!: bind_is_OK_pure_I filter_M_pure_I filter_M_is_OK_I bind_pure_I split: list.splits)[1]
    using assms(2) apply blast
     apply (meson list.distinct(1) returns_result_eq)
    by (meson list.distinct(1) list.inject returns_result_eq)
qed
end

interpretation i_get_host_wf?: l_get_host_wf_Shadow_DOM
  get_disconnected_document get_disconnected_document_locs known_ptr known_ptrs type_wf get_host
  get_host_locs get_shadow_root get_shadow_root_locs get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_tag_name get_tag_name_locs heap_is_wellformed
  parent_child_rel heap_is_wellformed_Core_DOM
  by(auto simp add: l_get_host_wf_Shadow_DOM_def instances)
declare l_get_host_wf_Shadow_DOM_axioms [instances]

locale l_get_host_wf = l_heap_is_wellformed_defs + l_known_ptrs + l_type_wf + l_get_host_defs +
  assumes get_host_ok: "heap_is_wellformed h ⟹ known_ptrs h ⟹ type_wf h ⟹
shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹ h ⊢ ok (get_host shadow_root_ptr)"

lemma get_host_wf_is_l_get_host_wf [instances]: "l_get_host_wf heap_is_wellformed known_ptr
known_ptrs type_wf get_host"
  by(auto simp add: l_get_host_wf_def l_get_host_wf_axioms_def instances)
```

get_root_node_si

```
locale l_get_root_node_si_wf_Shadow_DOM =
  l_get_root_node_si_Shadow_DOM +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_parent_wf +
  l_get_parent_get_host_wf +
  l_get_host_wf
begin
lemma get_root_node_ptr_in_heap:
  assumes "h ⊢ ok (get_root_node_si ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms
  unfolding get_root_node_si_def
  using get_ancestors_si_ptr_in_heap
  by auto
```

```
lemma get_ancestors_si_ok:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
    and "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_ancestors_si ptr)"
proof (insert assms(1) assms(4), induct rule: heap_wellformed_induct_rev_si)
  case (step child)
  then show ?case
    using assms(2) assms(3)
    apply(auto simp add: get_ancestors_si_def[of child] assms(1) get_parent_parent_in_heap
        intro!: bind_is_OK_pure_I split: option.splits)[1]
    using local.get_parent_ok apply blast
    using get_host_ok assms(1) apply blast
    by (meson assms(1) is_OK_returns_result_I local.get_shadow_root_ptr_in_heap
        local.shadow_root_host_dual)
qed

lemma get_ancestors_si_remains_not_in_ancestors:
  assumes "heap_is_wellformed h"
    and "heap_is_wellformed h'"
    and "h ⊢ get_ancestors_si ptr →r ancestors"
    and "h' ⊢ get_ancestors_si ptr →r ancestors'"
    and "⋀p children children'. h ⊢ get_child_nodes p →r children
        ⟹ h' ⊢ get_child_nodes p →r children' ⟹ set children' ⊆ set children"
    and "⋀p shadow_root_option shadow_root_option'. h ⊢ get_shadow_root p →r shadow_root_option ⟹
h' ⊢ get_shadow_root p →r shadow_root_option' ⟹ (if shadow_root_option = None
then shadow_root_option' = None else shadow_root_option' = None ∨ shadow_root_option' = shadow_root_option)"
    and "node ∉ set ancestors"
    and object_ptr_kinds_eq3: "object_ptr_kinds h = object_ptr_kinds h'"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
    and type_wf': "type_wf h'"
  shows "node ∉ set ancestors'"
proof -
  have object_ptr_kinds_M_eq:
    "⋀ptrs. h ⊢ object_ptr_kinds_M →r ptrs = h' ⊢ object_ptr_kinds_M →r ptrs"
    using object_ptr_kinds_eq3
    by(simp add: object_ptr_kinds_M_defs)
  then have object_ptr_kinds_eq: "|h ⊢ object_ptr_kinds_M|r = |h' ⊢ object_ptr_kinds_M|r"
    by(simp)

  show ?thesis
  proof (insert assms(1) assms(3) assms(4) assms(7), induct ptr arbitrary: ancestors ancestors'
      rule: heap_wellformed_induct_rev_si)
    case (step child)

    obtain ancestors_remains where ancestors_remains:
      "ancestors = child # ancestors_remains"
      using ‹h ⊢ get_ancestors_si child →r ancestors› get_ancestors_si_never_empty
      by(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2 split: option.splits)
    obtain ancestors_remains' where ancestors_remains':
      "ancestors' = child # ancestors_remains'"
      using ‹h' ⊢ get_ancestors_si child →r ancestors'› get_ancestors_si_never_empty
      by(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2 split: option.splits)
    have "child |∈| object_ptr_kinds h"
      using local.get_ancestors_si_ptr_in_heap object_ptr_kinds_eq3 step.prems(2) by fastforce

    have "node ≠ child"
      using ancestors_remains step.prems(3) by auto

    have 1: "⋀p parent. h' ⊢ get_parent p →r Some parent ⟹ h ⊢ get_parent p →r Some parent"
    proof -
```

```
      fix p parent
      assume "h' ⊢ get_parent p →r Some parent"
      then obtain children' where
        children': "h' ⊢ get_child_nodes parent →r children'" and
        p_in_children': "p ∈ set children'"
        using get_parent_child_dual by blast
      obtain children where children: "h ⊢ get_child_nodes parent →r children"
        using get_child_nodes_ok assms(1) get_child_nodes_ptr_in_heap object_ptr_kinds_eq children'
          known_ptrs
        using  type_wf type_wf'
        by (metis ‹h' ⊢ get_parent p →r Some parent› get_parent_parent_in_heap is_OK_returns_result_E
            local.known_ptrs_known_ptr object_ptr_kinds_eq3)
      have "p ∈ set children"
        using assms(5) children children' p_in_children'
        by blast
      then show "h ⊢ get_parent p →r Some parent"
        using child_parent_dual assms(1) children known_ptrs  type_wf  by blast
    qed

    have 2: "⋀p host. h' ⊢ get_host p →r host ⟹ h ⊢ get_host p →r host"
    proof -
      fix p host
      assume "h' ⊢ get_host p →r host"
      then have "h' ⊢ get_shadow_root host →r Some p"
        using local.shadow_root_host_dual by blast
      then have "h ⊢ get_shadow_root host →r Some p"
        by (metis assms(6) element_ptr_kinds_commutes is_OK_returns_result_I local.get_shadow_root_ok
            local.get_shadow_root_ptr_in_heap node_ptr_kinds_commutes object_ptr_kinds_eq3 option.distinct(1)
            returns_result_select_result type_wf)
      then show "h ⊢ get_host p →r host"
        by (metis assms(1) is_OK_returns_result_E known_ptrs local.get_host_ok
            local.get_shadow_root_shadow_root_ptr_in_heap local.shadow_root_host_dual local.shadow_root_same_host
            type_wf)

    qed


    show ?case
    proof (cases "cast_object_ptr2node_ptr child")
      case None
      then show ?thesis
        using step(4) step(5) ‹node ≠ child›
        apply(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2
            split: option.splits)[1]
        by (metis "2" assms(1) shadow_root_same_host list.set_intros(2) shadow_root_host_dual
            step.hyps(2) step.prems(3) type_wf)
    next
      case (Some node_child)
      then
      show ?thesis
        using step(4) step(5) ‹node ≠ child›
        apply(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2
            split: option.splits)[1]
         apply (meson "1" option.distinct(1) returns_result_eq)
        by (metis "1" list.set_intros(2) option.inject returns_result_eq step.hyps(1) step.prems(3))
    qed
  qed
qed


lemma get_ancestors_si_ptrs_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_ancestors_si ptr →r ancestors"
```

```
  assumes "ptr' ∈ set ancestors"
  shows "ptr' |∈| object_ptr_kinds h"
proof (insert assms(4) assms(5), induct ancestors arbitrary: ptr)
  case Nil
  then show ?case
    by(auto)
next
  case (Cons a ancestors)
  then obtain x where x: "h ⊢ get_ancestors_si x →r a # ancestors"
    by(auto simp add: get_ancestors_si_def[of a] elim!: bind_returns_result_E2 split: option.splits)
  then have "x = a"
    by(auto simp add: get_ancestors_si_def[of x] elim!: bind_returns_result_E2 split: option.splits)
  then show ?case
  proof (cases "ptr' = a")
    case True
    then show ?thesis
      using Cons.hyps Cons.prems(2) get_ancestors_si_ptr_in_heap x
      using ‹x = a› by blast
  next
    case False
    obtain ptr'' where ptr'': "h ⊢ get_ancestors_si ptr'' →r ancestors"
      using ‹ h ⊢ get_ancestors_si x →r a # ancestors› Cons.prems(2) False
      apply(auto simp add: get_ancestors_si_def elim!: bind_returns_result_E2)[1]
       apply(auto elim!: bind_returns_result_E2 split: option.splits intro!: bind_pure_I)[1]
      apply(auto elim!: bind_returns_result_E2 split: option.splits intro!: bind_pure_I)[1]
       apply (metis local.get_ancestors_si_def)
      by (simp add: local.get_ancestors_si_def)
    then show ?thesis
      using Cons.hyps Cons.prems(2) False by auto
  qed
qed

lemma get_ancestors_si_reads:
  assumes "heap_is_wellformed h"
  shows "reads get_ancestors_si_locs (get_ancestors_si node_ptr) h h'"
proof (insert assms(1), induct rule: heap_wellformed_induct_rev_si)
  case (step child)
  then show ?case
    using [[simproc del: Product_Type.unit_eq]] get_parent_reads[unfolded reads_def]
      get_host_reads[unfolded reads_def]
    apply(simp (no_asm) add: get_ancestors_si_def)
    by(auto simp add: get_ancestors_si_locs_def get_parent_reads_pointers
        intro!: reads_bind_pure reads_subset[OF check_in_heap_reads] reads_subset[OF return_reads]
        reads_subset[OF get_parent_reads] reads_subset[OF get_child_nodes_reads]
        reads_subset[OF get_host_reads]
        split: option.splits)
qed


lemma get_ancestors_si_subset:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_si ptr →r ancestors"
    and "ancestor ∈ set ancestors"
    and "h ⊢ get_ancestors_si ancestor →r ancestor_ancestors"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "set ancestor_ancestors ⊆ set ancestors"
proof (insert assms(1) assms(2) assms(3), induct ptr arbitrary: ancestors
    rule: heap_wellformed_induct_rev_si)
  case (step child)
  have "child |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr_in_heap step(4) by auto
```

```
obtain tl_ancestors where tl_ancestors: "ancestors = child # tl_ancestors"
  using step(4)
  by(auto simp add: get_ancestors_si_def[of child] intro!: bind_pure_I
      elim!: bind_returns_result_E2 split: option.splits)
show ?case
proof (induct "cast_object_ptr2node_ptr child")
  case None
  show ?case
  proof (induct "cast_object_ptr2shadow_root_ptr child")
    case None
    then show ?case
      using step(4) ‹None = cast_object_ptr2node_ptr child›
      apply(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2)[1]
      by (metis (no_types, lifting) assms(4) empty_iff empty_set select_result_I2 set_ConsD
          step.prems(1) step.prems(2))
  next
    case (Some shadow_root_child)
    then
    have "cast shadow_root_child |∈| document_ptr_kinds h"
      using  ‹child |∈| object_ptr_kinds h›
      apply(auto simp add: document_ptr_kinds_def image_iff Bex_def split: option.splits)[1]
      by (metis (mono_tags) shadow_root_ptr_casts_commute)
    then
    have "shadow_root_child |∈| shadow_root_ptr_kinds h"
      using shadow_root_ptr_kinds_commutes by blast
    obtain host where host: "h ⊢ get_host shadow_root_child →_r host"
      using get_host_ok assms
      by (meson ‹shadow_root_child |∈| shadow_root_ptr_kinds h› is_OK_returns_result_E)
    then
    have "h ⊢ get_ancestors_si (cast host) →_r tl_ancestors"
      using Some step(4) tl_ancestors None
      by(auto simp add: get_ancestors_si_def[of child] intro!: bind_pure_returns_result_I
          elim!: bind_returns_result_E2 split: option.splits dest: returns_result_eq)
    then
    show ?case
      using step(2) Some host step(5) tl_ancestors
      using assms(4) dual_order.trans eq_iff returns_result_eq set_ConsD set_subset_Cons
        shadow_root_ptr_casts_commute document_ptr_casts_commute step.prems(1)
      by (smt case_optionE local.shadow_root_host_dual option.case_distrib option.distinct(1))
  qed
next
  case (Some child_node)
  note s1 = Some
  obtain parent_opt where parent_opt: "h ⊢ get_parent child_node →_r parent_opt"
    using ‹child |∈| object_ptr_kinds h› assms(1) Some[symmetric] get_parent_ok[OF type_wf known_ptrs]
    by (metis (no_types, lifting) is_OK_returns_result_E known_ptrs get_parent_ok
        l_get_parent_Core_DOM_axioms node_ptr_casts_commute node_ptr_kinds_commutes)
  then show ?case
  proof (induct parent_opt)
    case None
    then have "ancestors = [child]"
      using step(4) s1
      apply(simp add: get_ancestors_si_def)
      by(auto elim!: bind_returns_result_E2 split: option.splits dest: returns_result_eq)
    show ?case
      using step(4) step(5)
      apply(auto simp add: ‹ancestors = [child]›)[1]
      using assms(4) returns_result_eq by fastforce
  next
    case (Some parent)
    then
```

```
      have "h ⊢ get_ancestors_si parent →r tl_ancestors"
        using s1 tl_ancestors step(4)
        by(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2
            split: option.splits dest: returns_result_eq)
      show ?case
        by (metis (no_types, lifting) Some.prems ‹h ⊢ get_ancestors_si parent →r tl_ancestors›
            assms(4) eq_iff node_ptr_casts_commute order_trans s1 select_result_I2 set_ConsD set_subset_Cons
            step.hyps(1) step.prems(1) step.prems(2) tl_ancestors)
    qed
  qed
qed

lemma get_ancestors_si_also_parent:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_si some_ptr →r ancestors"
    and "cast child ∈ set ancestors"
    and "h ⊢ get_parent child →r Some parent"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "parent ∈ set ancestors"
proof -
  obtain child_ancestors where child_ancestors: "h ⊢ get_ancestors_si (cast child) →r child_ancestors"
    by (meson assms(1) assms(4) get_ancestors_si_ok is_OK_returns_result_I known_ptrs
        local.get_parent_ptr_in_heap node_ptr_kinds_commutes returns_result_select_result
        type_wf)
  then have "parent ∈ set child_ancestors"
    apply(simp add: get_ancestors_si_def)
    by(auto elim!: bind_returns_result_E2 split: option.splits dest!: returns_result_eq[OF assms(4)]
        get_ancestors_si_ptr)
  then show ?thesis
    using assms child_ancestors get_ancestors_si_subset by blast
qed

lemma get_ancestors_si_also_host:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_si some_ptr →r ancestors"
    and "cast shadow_root ∈ set ancestors"
    and "h ⊢ get_host shadow_root →r host"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "cast host ∈ set ancestors"
proof -
  obtain child_ancestors where child_ancestors: "h ⊢ get_ancestors_si (cast shadow_root) →r child_ancestors"
    by (meson assms(1) assms(2) assms(3) get_ancestors_si_ok get_ancestors_si_ptrs_in_heap
        is_OK_returns_result_E known_ptrs type_wf)
  then have "cast host ∈ set child_ancestors"
    apply(simp add: get_ancestors_si_def)
    by(auto elim!: bind_returns_result_E2 split: option.splits dest!: returns_result_eq[OF assms(4)]
        get_ancestors_si_ptr)
  then show ?thesis
    using assms child_ancestors get_ancestors_si_subset by blast
qed


lemma get_ancestors_si_obtains_children_or_shadow_root:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
    and "h ⊢ get_ancestors_si ptr →r ancestors"
    and "ancestor ≠ ptr"
    and "ancestor ∈ set ancestors"
  shows "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →r children ⟶
ancestor_child ∈ set children ⟶ cast ancestor_child ∈ set ancestors ⟶ thesis) ⟶ thesis)
      ∨ ((∀ancestor_element shadow_root. ancestor = cast ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶ cast shadow_root ∈ set ancestors ⟶ thesis)
```

```
⟶
thesis)"
proof (insert assms(4) assms(5) assms(6), induct ptr arbitrary: ancestors
    rule: heap_wellformed_induct_rev_si[OF assms(1)])
  case (1 child)
  then show ?case
  proof (cases "cast_object_ptr2node_ptr child")
    case None
    then obtain shadow_root where shadow_root: "child = cast_shadow_root_ptr2object_ptr shadow_root"
      using 1(4) 1(5) 1(6)
      by(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2
          split: option.splits)
    then obtain host where host: "h ⊢ get_host shadow_root →_r host"
      by (metis "1.prems"(1) assms(1) assms(2) assms(3) document_ptr_kinds_commutes
          get_ancestors_si_ptrs_in_heap is_OK_returns_result_E local.get_ancestors_si_ptr local.get_host_ok
          shadow_root_ptr_kinds_commutes)
    then obtain host_ancestors where host_ancestors: "h ⊢ get_ancestors_si (cast_element_ptr2object_ptr host)
→_r host_ancestors"
      by (metis "1.prems"(1) assms(1) assms(2) assms(3) get_ancestors_si_also_host get_ancestors_si_ok
          get_ancestors_si_ptrs_in_heap is_OK_returns_result_E local.get_ancestors_si_ptr shadow_root)
    then have "ancestors = cast shadow_root # host_ancestors"
      using 1(4) 1(5) 1(3) None shadow_root host
      by(auto simp add: get_ancestors_si_def[of child, simplified shadow_root]
          elim!: bind_returns_result_E2 dest!: returns_result_eq[OF host] split: option.splits)
    then show ?thesis
    proof (cases "ancestor = cast host")
      case True
      then show ?thesis
        using "1.prems"(1) host local.get_ancestors_si_ptr local.shadow_root_host_dual shadow_root
        by blast
    next
      case False
      have "ancestor ∈ set ancestors"
        using host host_ancestors 1(3) get_ancestors_si_also_host assms(1) assms(2) assms(3)
        using "1.prems"(3) by blast
      then have "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →_r children ⟶
ancestor_child ∈ set children ⟶ cast_node_ptr2object_ptr ancestor_child ∈ set host_ancestors ⟶ thesis)
⟶
thesis) ∨
          ((∀ancestor_element shadow_root. ancestor = cast_element_ptr2object_ptr ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →_r Some shadow_root ⟶
cast_shadow_root_ptr2object_ptr shadow_root ∈ set host_ancestors ⟶ thesis) ⟶ thesis)"
        using "1.hyps"(2) "1.prems"(2) False ‹ancestors = cast_shadow_root_ptr2object_ptr shadow_root # host_ancestors›
          host host_ancestors shadow_root
        by auto
      then show ?thesis
        using ‹ancestors = cast_shadow_root_ptr2object_ptr shadow_root # host_ancestors› by auto
    qed
  next
    case (Some child_node)
    then obtain parent where parent: "h ⊢ get_parent child_node →_r Some parent"
      using 1(4) 1(5) 1(6)
      by(auto simp add: get_ancestors_si_def[of child] elim!: bind_returns_result_E2
          split: option.splits)
    then obtain parent_ancestors where parent_ancestors: "h ⊢ get_ancestors_si parent →_r parent_ancestors"
      by (meson assms(1) assms(2) assms(3) get_ancestors_si_ok is_OK_returns_result_E
          local.get_parent_parent_in_heap)
    then have "ancestors = cast child_node # parent_ancestors"
      using 1(4) 1(5) 1(3) Some
      by(auto simp add: get_ancestors_si_def[of child, simplified Some]
          elim!: bind_returns_result_E2 dest!: returns_result_eq[OF parent] split: option.splits)
    then show ?thesis
    proof (cases "ancestor = parent")
```

```
      case True
      then show ?thesis
        by (metis (no_types, lifting) "1.prems"(1) Some local.get_ancestors_si_ptr
            local.get_parent_child_dual node_ptr_casts_commute parent)
    next
      case False
      have "ancestor ∈ set ancestors"
        by (simp add: "1.prems"(3))
      then have "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →r children ⟶
ancestor_child ∈ set children ⟶ cast_node_ptr2object_ptr ancestor_child ∈ set parent_ancestors ⟶ thesis)
⟶
thesis) ∨
         ((∀ancestor_element shadow_root. ancestor = cast_element_ptr2object_ptr ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶
cast_shadow_root_ptr2object_ptr shadow_root ∈ set parent_ancestors ⟶ thesis) ⟶ thesis)"
        using "1.hyps"(1) "1.prems"(2) False Some ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors›
          parent parent_ancestors
        by auto
      then show ?thesis
        using ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors› by auto
    qed
  qed
qed

lemma a_host_shadow_root_rel_shadow_root:
  "h ⊢ get_shadow_root host →r Some shadow_root ⟹ (cast host, cast shadow_root) ∈ a_host_shadow_root_rel
h"
  by(auto simp add: is_OK_returns_result_I get_shadow_root_ptr_in_heap a_host_shadow_root_rel_def)

lemma get_ancestors_si_parent_child_a_host_shadow_root_rel:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
  assumes "h ⊢ get_ancestors_si child →r ancestors"
  shows "(ptr, child) ∈ (parent_child_rel h ∪ a_host_shadow_root_rel h)* ⟷ ptr ∈ set ancestors"
proof
  assume "(ptr, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h)* "
  then show "ptr ∈ set ancestors"
  proof (induct ptr rule: heap_wellformed_induct_si[OF assms(1)])
    case (1 ptr)
    then show ?case
    proof (cases "ptr = child")
      case True
      then show ?thesis
        using assms(4) local.get_ancestors_si_ptr by blast
    next
      case False
      obtain ptr_child where
        ptr_child: "(ptr, ptr_child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h) ∧
(ptr_child, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h)*"
        using converse_rtranclE[OF 1(4)] ‹ptr ≠ child›
        by metis
      then show ?thesis
      proof(cases "(ptr, ptr_child) ∈ parent_child_rel h")
        case True

        then obtain ptr_child_node
          where ptr_child_ptr_child_node: "ptr_child = cast_node_ptr2object_ptr ptr_child_node"
          using ptr_child node_ptr_casts_commute3 CD.parent_child_rel_node_ptr
          by (metis)
        then obtain children where
          children: "h ⊢ get_child_nodes ptr →r children" and
          ptr_child_node: "ptr_child_node ∈ set children"
        proof -
          assume a1: "⋀children. ⟦h ⊢ get_child_nodes ptr →r children; ptr_child_node ∈ set children⟧
```

```
                  ⟹ thesis"

          have "ptr |∈| object_ptr_kinds h"
            using CD.parent_child_rel_parent_in_heap True by blast
          moreover have "ptr_child_node ∈ set |h ⊢ get_child_nodes ptr|_r"
            by (metis True assms(2) assms(3) calculation local.CD.parent_child_rel_child
                local.get_child_nodes_ok local.known_ptrs_known_ptr ptr_child_ptr_child_node
                returns_result_select_result)
          ultimately show ?thesis
            using a1 get_child_nodes_ok ‹type_wf h› ‹known_ptrs h›
            by (meson local.known_ptrs_known_ptr returns_result_select_result)
        qed
        moreover have "(cast_node_ptr2object_ptr ptr_child_node, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_
h)*"
          using ptr_child True ptr_child_ptr_child_node by auto
        ultimately have "cast_node_ptr2object_ptr ptr_child_node ∈ set ancestors"
          using 1 by auto
        moreover have "h ⊢ get_parent ptr_child_node →_r Some ptr"
          using assms(1) children ptr_child_node child_parent_dual
          using ‹known_ptrs h› ‹type_wf h› by blast
        ultimately show ?thesis
          using get_ancestors_si_also_parent assms ‹type_wf h› by blast
      next
        case False
        then
        obtain host where host: "ptr = cast_element_ptr2object_ptr host"
          using ptr_child
          by(auto simp add: a_host_shadow_root_rel_def)
        then obtain shadow_root where shadow_root: "h ⊢ get_shadow_root host →_r Some shadow_root"
          and ptr_child_shadow_root: "ptr_child = cast shadow_root"
          using ptr_child False
          apply(auto simp add: a_host_shadow_root_rel_def)[1]
          by (metis (no_types, lifting) assms(3) local.get_shadow_root_ok select_result_I)

        moreover have "(cast shadow_root, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel
h)*"
          using ptr_child ptr_child_shadow_root by blast
        ultimately have "cast shadow_root ∈ set ancestors"
          using "1.hyps"(2) host by blast
        moreover have "h ⊢ get_host shadow_root →_r host"
          by (metis assms(1) assms(2) assms(3) is_OK_returns_result_E local.get_host_ok
              local.get_shadow_root_shadow_root_ptr_in_heap local.shadow_root_host_dual local.shadow_root_same_host
              shadow_root)
        ultimately show ?thesis
          using get_ancestors_si_also_host assms(1) assms(2) assms(3) assms(4) host
          by blast
      qed
    qed
  qed
next
  assume "ptr ∈ set ancestors"
  then show "(ptr, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h)*"
  proof (induct ptr rule: heap_wellformed_induct_si[OF assms(1)])
    case (1 ptr)
    then show ?case
    proof (cases "ptr = child")
      case True
      then show ?thesis
        by simp
    next
      case False
      have "⋀thesis. ((∀children ancestor_child. h ⊢ get_child_nodes ptr →_r children ⟶
ancestor_child ∈ set children ⟶ cast ancestor_child ∈ set ancestors ⟶ thesis) ⟶ thesis)
```

```
      ∨ ((∀ ancestor_element shadow_root. ptr = cast ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶ cast shadow_root ∈ set ancestors ⟶ thesis)
⟶
thesis)"
        using "1.prems" False assms(1) assms(2) assms(3) assms(4) get_ancestors_si_obtains_children_or_shadow_root
        by blast
      then show ?thesis
      proof (cases "∀ thesis. ((∀ children ancestor_child. h ⊢ get_child_nodes ptr →r children ⟶
ancestor_child ∈ set children ⟶ cast ancestor_child ∈ set ancestors ⟶ thesis) ⟶ thesis)")
        case True
        then obtain children ancestor_child
          where "h ⊢ get_child_nodes ptr →r children"
            and "ancestor_child ∈ set children"
            and "cast ancestor_child ∈ set ancestors"
          by blast
        then show ?thesis
          by (meson "1.hyps"(1) in_rtrancl_UnI local.CD.parent_child_rel_child r_into_rtrancl rtrancl_trans)
      next
        case False
        obtain ancestor_element shadow_root
          where "ptr = cast ancestor_element"
            and "h ⊢ get_shadow_root ancestor_element →r Some shadow_root"
            and "cast shadow_root ∈ set ancestors"
          using False ‹⋀thesis. ((∀ children ancestor_child. h ⊢ get_child_nodes ptr →r children ⟶
ancestor_child ∈ set children ⟶ cast_node_ptr2object_ptr ancestor_child ∈ set ancestors ⟶ thesis) ⟶
thesis) ∨
((∀ ancestor_element shadow_root. ptr = cast_element_ptr2object_ptr ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶
cast_shadow_root_ptr2object_ptr shadow_root ∈ set ancestors ⟶ thesis) ⟶ thesis)›
          by blast

        then show ?thesis
          using 1(2) a_host_shadow_root_rel_shadow_root
          apply(simp)
          by (meson Un_iff converse_rtrancl_into_rtrancl)
      qed
    qed
  qed
qed

lemma get_root_node_si_root_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node_si ptr →r root"
  shows "root |∈| object_ptr_kinds h"
  using assms
  apply(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)[1]
  by (simp add: get_ancestors_si_never_empty get_ancestors_si_ptrs_in_heap)

lemma get_root_node_si_same_no_parent:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node_si ptr →r cast child"
  shows "h ⊢ get_parent child →r None"
proof (insert assms(1) assms(4), induct ptr rule: heap_wellformed_induct_rev_si)
  case (step c)
  then show ?case
  proof (cases "cast_object_ptr2node_ptr c")
    case None
    then show ?thesis
      using step(4)
      by(auto simp add: get_root_node_si_def get_ancestors_si_def[of c] elim!: bind_returns_result_E2
          split: if_splits option.splits intro!: step(2) bind_pure_returns_result_I)
  next
    case (Some child_node)
```

```
    note s = this
    then obtain parent_opt where parent_opt: "h ⊢ get_parent child_node →r parent_opt"
      using step(4)
      apply(auto simp add: get_root_node_si_def get_ancestors_si_def intro!: bind_pure_I
          elim!: bind_returns_result_E2)[1]
      by(auto split: option.splits)
    then show ?thesis
    proof(induct parent_opt)
      case None
      then show ?case
        using Some get_root_node_si_no_parent returns_result_eq step.prems by fastforce
    next
      case (Some parent)
      then show ?case
        using step(4) s
        apply(auto simp add: get_root_node_si_def get_ancestors_si_def[of c]
            elim!: bind_returns_result_E2 split: option.splits list.splits if_splits)[1]
        using assms(1) get_ancestors_si_never_empty apply blast
        by(auto simp add: get_root_node_si_def dest: returns_result_eq
            intro!: step(1) bind_pure_returns_result_I)
    qed
  qed
qed

lemma get_root_node_si_parent_child_a_host_shadow_root_rel:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
  assumes "h ⊢ get_root_node_si ptr →r root"
  shows "(root, ptr) ∈ (parent_child_rel h ∪ a_host_shadow_root_rel h)*"
  using assms
  using get_ancestors_si_parent_child_a_host_shadow_root_rel get_ancestors_si_never_empty
  by(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2 intro!: bind_pure_returns_result_I)
end

interpretation i_get_root_node_si_wf?: l_get_root_node_si_wf_Shadow_DOM
  type_wf known_ptr known_ptrs get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_host get_host_locs get_ancestors_si get_ancestors_si_locs get_root_node_si get_root_node_si_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name
  get_tag_name_locs heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_disconnected_document
  get_disconnected_document_locs
  by(auto simp add: l_get_root_node_si_wf_Shadow_DOM_def instances)
declare l_get_root_node_si_wf_Shadow_DOM_axioms [instances]
```

get_disconnected_document

```
locale l_get_disconnected_document_wf_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM +
  l_get_disconnected_document_Core_DOM +
  l_get_parent_wf +
  l_get_parent
begin

lemma get_disconnected_document_ok:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_parent node_ptr →r None"
  shows "h ⊢ ok (get_disconnected_document node_ptr)"
proof -
  have "node_ptr |∈| node_ptr_kinds h"
    by (meson assms(4) is_OK_returns_result_I local.get_parent_ptr_in_heap)
  have "¬(∃parent ∈ fset (object_ptr_kinds h). node_ptr ∈ set |h ⊢ get_child_nodes parent|r)"
    apply(auto)[1]
    using assms(4) child_parent_dual[OF assms(1)]
      assms(1) assms(2) assms(3) known_ptrs_known_ptr option.simps(3)
      returns_result_eq returns_result_select_result
```

```
    by (metis (no_types, lifting) CD.get_child_nodes_ok)
  then
  have "(∃document_ptr ∈ fset (document_ptr_kinds h). node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r)
    using heap_is_wellformed_children_disc_nodes
    using ‹node_ptr |∈| node_ptr_kinds h› assms(1) by blast
  then obtain some_owner_document where
    "some_owner_document ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))" and
    "node_ptr ∈ set |h ⊢ get_disconnected_nodes some_owner_document|r"
    by auto

  have h5: "∃!x. x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h))) ∧ h ⊢ Heap_Error_Monad.bind
(get_disconnected_nodes x)
                  (λchildren. return (node_ptr ∈ set children)) →r True"
    apply(auto intro!: bind_pure_returns_result_I)[1]
     apply (smt (verit) CD.get_disconnected_nodes_ok CD.get_disconnected_nodes_pure
        ‹∃document_ptr∈fset (document_ptr_kinds h). node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r›
        assms(2) bind_pure_returns_result_I2 return_returns_result select_result_I2)

    apply(auto elim!: bind_returns_result_E2 intro!: bind_pure_returns_result_I)[1]
    using heap_is_wellformed_one_disc_parent assms(1)
    by blast
  let ?filter_M = "filter_M
            (λdocument_ptr.
                Heap_Error_Monad.bind (get_disconnected_nodes document_ptr)
                 (λdisconnected_nodes. return (node_ptr ∈ set disconnected_nodes)))
            (sorted_list_of_set (fset (document_ptr_kinds h)))"
  have "h ⊢ ok (?filter_M)"
    using CD.get_disconnected_nodes_ok
    by (smt CD.get_disconnected_nodes_pure DocumentMonad.ptr_kinds_M_ptr_kinds
        DocumentMonad.ptr_kinds_ptr_kinds_M assms(2) bind_is_OK_pure_I bind_pure_I document_ptr_kinds_M_def
        filter_M_is_OK_I l_ptr_kinds_M.ptr_kinds_M_ok return_ok return_pure returns_result_select_result)
  then
  obtain candidates where candidates: "h ⊢ filter_M
            (λdocument_ptr.
                Heap_Error_Monad.bind (get_disconnected_nodes document_ptr)
                 (λdisconnected_nodes. return (node_ptr ∈ set disconnected_nodes)))
            (sorted_list_of_set (fset (document_ptr_kinds h)))
       →r candidates"
    by auto
  have "candidates = [some_owner_document]"
    apply(rule filter_M_ex1[OF candidates ‹some_owner_document ∈ set (sorted_list_of_set (fset (document_ptr_kinds
h)))› h5])
    using ‹node_ptr ∈ set |h ⊢ get_disconnected_nodes some_owner_document|r›
      ‹some_owner_document ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))›
    by(auto simp add: CD.get_disconnected_nodes_ok assms(2) intro!: bind_pure_I
        intro!: bind_pure_returns_result_I)
  then show ?thesis
    using candidates  ‹node_ptr |∈| node_ptr_kinds h›
    apply(auto simp add: get_disconnected_document_def intro!: bind_is_OK_pure_I filter_M_pure_I bind_pure_I
        split: list.splits)[1]
     apply (meson not_Cons_self2 returns_result_eq)
    by (meson list.distinct(1) list.inject returns_result_eq)
qed
end

interpretation i_get_disconnected_document_wf?: l_get_disconnected_document_wf_Shadow_DOM
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf
  heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs
  get_disconnected_document get_disconnected_document_locs known_ptrs get_parent get_parent_locs
  by(auto simp add: l_get_disconnected_document_wf_Shadow_DOM_def instances)
declare l_get_disconnected_document_wf_Shadow_DOM_axioms [instances]
```

get_ancestors_di

```
locale l_get_ancestors_di_wf_Shadow_DOM =
  l_get_ancestors_di_Shadow_DOM +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_parent_wf +
  l_get_parent_get_host_wf +
  l_get_host_wf +
  l_get_disconnected_document_wf_Shadow_DOM +
  l_get_parent_get_host_get_disconnected_document_wf_Shadow_DOM
begin
lemma get_ancestors_di_ok:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
    and "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_ancestors_di ptr)"
proof (insert assms(1) assms(4), induct rule: heap_wellformed_induct_rev_si)
  case (step child)
  then show ?case
    using assms(2) assms(3)
    apply(auto simp add: get_ancestors_di_def[of child] assms(1) get_parent_parent_in_heap
        intro!: bind_is_OK_pure_I bind_pure_I split: option.splits)[1]
    using local.get_parent_ok apply blast
    using assms(1) get_disconnected_document_ok apply blast
      apply(simp add: get_ancestors_di_def )
      apply(auto intro!: bind_is_OK_pure_I split: option.splits)[1]
         apply (metis (no_types, lifting) bind_is_OK_E document_ptr_kinds_commutes is_OK_returns_heap_I
        local.get_ancestors_di_def local.get_disconnected_document_disconnected_document_in_heap step.hyps(3))
        apply (metis (no_types, lifting) bind_is_OK_E document_ptr_kinds_commutes is_OK_returns_heap_I
        local.get_ancestors_di_def local.get_disconnected_document_disconnected_document_in_heap step.hyps(3))
    using assms(1) local.get_disconnected_document_disconnected_document_in_heap local.get_host_ok
      shadow_root_ptr_kinds_commutes apply blast
      apply (smt assms(1) bind_returns_heap_E document_ptr_casts_commute2 is_OK_returns_heap_E
        is_OK_returns_heap_I l_heap_is_wellformed_Shadow_DOM.shadow_root_same_host
        local.get_disconnected_document_disconnected_document_in_heap local.get_host_pure
        local.l_heap_is_wellformed_Shadow_DOM_axioms local.shadow_root_host_dual option.simps(4) option.simps(5)
        pure_returns_heap_eq shadow_root_ptr_casts_commute2)
    using get_host_ok assms(1) apply blast
    by (meson assms(1) is_OK_returns_result_I local.get_shadow_root_ptr_in_heap local.shadow_root_host_dual)
qed

lemma get_ancestors_di_ptrs_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_ancestors_di ptr →r ancestors"
  assumes "ptr' ∈ set ancestors"
  shows "ptr' |∈| object_ptr_kinds h"
proof (insert assms(4) assms(5), induct ancestors arbitrary: ptr)
  case Nil
  then show ?case
    by(auto)
next
  case (Cons a ancestors)
  then obtain x where x: "h ⊢ get_ancestors_di x →r a # ancestors"
    by(auto simp add: get_ancestors_di_def[of a] elim!: bind_returns_result_E2 split: option.splits)
  then have "x = a"
    by(auto simp add: get_ancestors_di_def[of x] intro!: bind_pure_I elim!: bind_returns_result_E2
        split: option.splits)
  then show ?case
  proof (cases "ptr' = a")
    case True
    then show ?thesis
      using Cons.hyps Cons.prems(2) get_ancestors_di_ptr_in_heap x
      using ‹x = a› by blast
  next
```

```
    case False
    obtain ptr'' where ptr'': "h ⊢ get_ancestors_di ptr'' →r ancestors"
      using ‹ h ⊢ get_ancestors_di x →r a # ancestors› Cons.prems(2) False
      apply(auto simp add: get_ancestors_di_def elim!: bind_returns_result_E2)[1]
       apply(auto elim!: bind_returns_result_E2 split: option.splits intro!: bind_pure_I)[1]
      apply(auto elim!: bind_returns_result_E2 split: option.splits intro!: bind_pure_I)[1]
        apply (metis (no_types, lifting) local.get_ancestors_di_def)
       apply (metis (no_types, lifting) local.get_ancestors_di_def)
      by (simp add: local.get_ancestors_di_def)
    then show ?thesis
      using Cons.hyps Cons.prems(2) False by auto
  qed
qed

lemma get_ancestors_di_reads:
  assumes "heap_is_wellformed h"
  shows "reads get_ancestors_di_locs (get_ancestors_di node_ptr) h h'"
proof (insert assms(1), induct rule: heap_wellformed_induct_rev_si)
  case (step child)
  then show ?case
    using  get_parent_reads[unfolded reads_def]
      get_host_reads[unfolded reads_def] get_disconnected_document_reads[unfolded reads_def]
    apply(auto simp add: get_ancestors_di_def[of child])[1]
    by(auto simp add: get_ancestors_di_locs_def get_parent_reads_pointers
        intro!: bind_pure_I reads_bind_pure reads_subset[OF check_in_heap_reads] reads_subset[OF return_reads]
        reads_subset[OF get_parent_reads] reads_subset[OF get_child_nodes_reads]
        reads_subset[OF get_host_reads] reads_subset[OF get_disconnected_document_reads]
        split: option.splits list.splits
        )

qed

lemma get_ancestors_di_subset:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_di ptr →r ancestors"
    and "ancestor ∈ set ancestors"
    and "h ⊢ get_ancestors_di ancestor →r ancestor_ancestors"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "set ancestor_ancestors ⊆ set ancestors"
proof (insert assms(1) assms(2) assms(3), induct ptr arbitrary: ancestors
    rule: heap_wellformed_induct_rev_si)
  case (step child)
  have "child |∈| object_ptr_kinds h"
    using get_ancestors_di_ptr_in_heap step(4) by auto



  obtain tl_ancestors where tl_ancestors: "ancestors = child # tl_ancestors"
    using step(4)
    by(auto simp add: get_ancestors_di_def[of child] intro!: bind_pure_I
        elim!: bind_returns_result_E2 split: option.splits)
  show ?case
  proof (induct "cast_object_ptr2node_ptr child")
    case None
    show ?case
    proof (induct "cast_object_ptr2shadow_root_ptr child")
      case None
      then show ?case
        using step(4) ‹None = cast_object_ptr2node_ptr child›
        apply(auto simp add: get_ancestors_di_def[of child] elim!: bind_returns_result_E2)[1]
        by (metis (no_types, lifting) assms(4) empty_iff empty_set select_result_I2 set_ConsD
            step.prems(1) step.prems(2))
```

```
    next
      case (Some shadow_root_child)
      then
      have "cast shadow_root_child |∈| document_ptr_kinds h"
        using ‹child |∈| object_ptr_kinds h›
        apply(auto simp add: document_ptr_kinds_def image_iff Bex_def split: option.splits)[1]
        by (metis (mono_tags, lifting) shadow_root_ptr_casts_commute)
      then
      have "shadow_root_child |∈| shadow_root_ptr_kinds h"
        using shadow_root_ptr_kinds_commutes by blast
      obtain host where host: "h ⊢ get_host shadow_root_child →r host"
        using get_host_ok assms
        by (meson ‹shadow_root_child |∈| shadow_root_ptr_kinds h› is_OK_returns_result_E)
      then
      have "h ⊢ get_ancestors_di (cast host) →r tl_ancestors"
        using Some step(4) tl_ancestors None
        by(auto simp add: get_ancestors_di_def[of child] intro!: bind_pure_returns_result_I
            elim!: bind_returns_result_E2 split: option.splits dest: returns_result_eq)
      then
      show ?case
        using step(2) Some host step(5) tl_ancestors
        using assms(4) dual_order.trans eq_iff returns_result_eq set_ConsD set_subset_Cons
          shadow_root_ptr_casts_commute document_ptr_casts_commute step.prems(1)
        by (smt case_optionE local.shadow_root_host_dual option.case_distrib option.distinct(1))
    qed
  next
    case (Some child_node)
    note s1 = Some
    obtain parent_opt where parent_opt: "h ⊢ get_parent child_node →r parent_opt"
      using ‹child |∈| object_ptr_kinds h› assms(1) Some[symmetric] get_parent_ok[OF type_wf known_ptrs]
      by (metis (no_types, lifting) is_OK_returns_result_E known_ptrs get_parent_ok
          l_get_parent_Core_DOM_axioms node_ptr_casts_commute node_ptr_kinds_commutes)
    then show ?case
    proof (induct parent_opt)
      case None
      then obtain disc_doc where disc_doc: "h ⊢ get_disconnected_document child_node →r disc_doc"
        and "h ⊢ get_ancestors_di (cast disc_doc) →r tl_ancestors"
        using step(4) s1 tl_ancestors
        apply(simp add: get_ancestors_di_def[of child])
        by(auto elim!: bind_returns_result_E2 intro!: bind_pure_I split: option.splits dest: returns_result_eq)
      then show ?thesis
        using step(3) step(4) step(5)
        by (metis (no_types, lifting) assms(4) dual_order.trans eq_iff node_ptr_casts_commute s1
            select_result_I2 set_ConsD set_subset_Cons tl_ancestors)
    next
      case (Some parent)
      then
      have "h ⊢ get_ancestors_di parent →r tl_ancestors"
        using s1 tl_ancestors step(4)
        by(auto simp add: get_ancestors_di_def[of child] elim!: bind_returns_result_E2
            split: option.splits dest: returns_result_eq)
      show ?case
        by (metis (no_types, lifting) Some.prems ‹h ⊢ get_ancestors_di parent →r tl_ancestors›
            assms(4) eq_iff node_ptr_casts_commute order_trans s1 select_result_I2 set_ConsD set_subset_Cons
            step.hyps(1) step.prems(1) step.prems(2) tl_ancestors)
    qed
  qed
qed

lemma get_ancestors_di_also_parent:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_di some_ptr →r ancestors"
    and "cast child ∈ set ancestors"
```

```
    and "h ⊢ get_parent child →r Some parent"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "parent ∈ set ancestors"
proof -
  obtain child_ancestors where child_ancestors: "h ⊢ get_ancestors_di (cast child) →r child_ancestors"
    by (meson assms(1) assms(4) get_ancestors_di_ok is_OK_returns_result_I known_ptrs
        local.get_parent_ptr_in_heap node_ptr_kinds_commutes returns_result_select_result
        type_wf)
  then have "parent ∈ set child_ancestors"
    apply(simp add: get_ancestors_di_def)
    by(auto elim!: bind_returns_result_E2 split: option.splits dest!: returns_result_eq[OF assms(4)]
        get_ancestors_di_ptr)
  then show ?thesis
    using assms child_ancestors get_ancestors_di_subset by blast
qed

lemma get_ancestors_di_also_host:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_di some_ptr →r ancestors"
    and "cast shadow_root ∈ set ancestors"
    and "h ⊢ get_host shadow_root →r host"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "cast host ∈ set ancestors"
proof -
  obtain child_ancestors where child_ancestors: "h ⊢ get_ancestors_di (cast shadow_root) →r child_ancestors"
    by (meson assms(1) assms(2) assms(3) get_ancestors_di_ok get_ancestors_di_ptrs_in_heap
        is_OK_returns_result_E known_ptrs type_wf)
  then have "cast host ∈ set child_ancestors"
    apply(simp add: get_ancestors_di_def)
    by(auto elim!: bind_returns_result_E2 split: option.splits dest!: returns_result_eq[OF assms(4)]
        get_ancestors_di_ptr)
  then show ?thesis
    using assms child_ancestors get_ancestors_di_subset by blast
qed

lemma get_ancestors_di_also_disconnected_document:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_di some_ptr →r ancestors"
    and "cast disc_node ∈ set ancestors"
    and "h ⊢ get_disconnected_document disc_node →r disconnected_document"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
    and "h ⊢ get_parent disc_node →r None"
  shows "cast disconnected_document ∈ set ancestors"
proof -
  obtain child_ancestors where child_ancestors: "h ⊢ get_ancestors_di (cast disc_node) →r child_ancestors"
    by (meson assms(1) assms(2) assms(3) get_ancestors_di_ok get_ancestors_di_ptrs_in_heap
        is_OK_returns_result_E known_ptrs type_wf)
  then have "cast disconnected_document ∈ set child_ancestors"
    apply(simp add: get_ancestors_di_def)
    by(auto elim!: bind_returns_result_E2 intro!: bind_pure_I split: option.splits
        dest!: returns_result_eq[OF assms(4)] returns_result_eq[OF assms(7)]
        get_ancestors_di_ptr)
  then show ?thesis
    using assms child_ancestors get_ancestors_di_subset by blast
qed

lemma disc_node_disc_doc_dual:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_parent node_ptr →r None"
  assumes "h ⊢ get_disconnected_nodes disc_doc →r disc_nodes"
```

```
  assumes "node_ptr ∈ set disc_nodes"
  shows "h ⊢ get_disconnected_document node_ptr →r disc_doc"
proof -
  have "node_ptr |∈| node_ptr_kinds h"
    by (meson assms(4) is_OK_returns_result_I local.get_parent_ptr_in_heap)
  then
  have "¬(∃parent ∈ fset (object_ptr_kinds h). node_ptr ∈ set |h ⊢ get_child_nodes parent|r)"
    apply(auto)[1]
    using child_parent_dual[OF assms(1)]
      assms known_ptrs_known_ptr option.simps(3)
      returns_result_eq returns_result_select_result
    by (metis (no_types, lifting) get_child_nodes_ok)
  then
  have "(∃document_ptr ∈ fset (document_ptr_kinds h). node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r)
    using heap_is_wellformed_children_disc_nodes
    using ‹node_ptr |∈| node_ptr_kinds h› assms(1) by blast

  then have "disc_doc ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))" and
    "node_ptr ∈ set |h ⊢ get_disconnected_nodes disc_doc|r"
    using CD.get_disconnected_nodes_ptr_in_heap assms(5)
      assms(6) by auto


  have h5: "∃!x. x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h))) ∧
h ⊢ Heap_Error_Monad.bind (get_disconnected_nodes x)
                  (λchildren. return (node_ptr ∈ set children)) →r True"
    apply(auto intro!: bind_pure_returns_result_I)[1]
     apply (smt (verit) CD.get_disconnected_nodes_ok CD.get_disconnected_nodes_pure
        ‹∃document_ptr∈fset (document_ptr_kinds h). node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r›
        assms(2) bind_pure_returns_result_I2 return_returns_result select_result_I2)

    apply(auto elim!: bind_returns_result_E2 intro!: bind_pure_returns_result_I)[1]
    using heap_is_wellformed_one_disc_parent assms(1)
    by blast
  let ?filter_M = "filter_M
            (λdocument_ptr.
                Heap_Error_Monad.bind (get_disconnected_nodes document_ptr)
                 (λdisconnected_nodes. return (node_ptr ∈ set disconnected_nodes)))
            (sorted_list_of_set (fset (document_ptr_kinds h)))"
  have "h ⊢ ok (?filter_M)"
    using CD.get_disconnected_nodes_ok
    by (smt CD.get_disconnected_nodes_pure DocumentMonad.ptr_kinds_M_ptr_kinds
        DocumentMonad.ptr_kinds_ptr_kinds_M assms(2) bind_is_OK_pure_I bind_pure_I document_ptr_kinds_M_def
        filter_M_is_OK_I l_ptr_kinds_M.ptr_kinds_M_ok return_ok return_pure returns_result_select_result)
  then
  obtain candidates where candidates: "h ⊢ ?filter_M →r candidates"
    by auto
  have "candidates = [disc_doc]"
    apply(rule filter_M_ex1[OF candidates ‹disc_doc ∈
set (sorted_list_of_set (fset (document_ptr_kinds h)))› h5])
    using ‹node_ptr ∈ set |h ⊢ get_disconnected_nodes disc_doc|r›
      ‹disc_doc ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))›
    by(auto simp add: CD.get_disconnected_nodes_ok assms(2) intro!: bind_pure_I
        intro!: bind_pure_returns_result_I)

  then
  show ?thesis
    using ‹node_ptr |∈| node_ptr_kinds h› candidates
    by(auto simp add: bind_pure_I get_disconnected_document_def
        elim!: bind_returns_result_E2
        intro!: bind_pure_returns_result_I filter_M_pure_I)
qed
```

```
lemma get_ancestors_di_obtains_children_or_shadow_root_or_disconnected_node:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
    and "h ⊢ get_ancestors_di ptr →r ancestors"
    and "ancestor ≠ ptr"
    and "ancestor ∈ set ancestors"
  shows "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →r children ⟶
ancestor_child ∈ set children ⟶ cast ancestor_child ∈ set ancestors ⟶ thesis) ⟶ thesis)
      ∨ ((∀ancestor_element shadow_root. ancestor = cast ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶ cast shadow_root ∈ set ancestors ⟶ thesis)
⟶
thesis)
      ∨ ((∀disc_doc disc_nodes disc_node. ancestor = cast disc_doc ⟶
h ⊢ get_disconnected_nodes disc_doc →r disc_nodes ⟶ disc_node ∈ set disc_nodes ⟶
cast disc_node ∈ set ancestors ⟶ thesis) ⟶ thesis)"
proof (insert assms(4) assms(5) assms(6), induct ptr arbitrary: ancestors
    rule: heap_wellformed_induct_rev_si[OF assms(1)])
  case (1 child)
  then show ?case
  proof (cases "castobject_ptr2node_ptr child")
    case None
    then obtain shadow_root where shadow_root: "child = castshadow_root_ptr2object_ptr shadow_root"
      using 1(4) 1(5) 1(6)
      by(auto simp add: get_ancestors_di_def[of child] elim!: bind_returns_result_E2
          split: option.splits)
    then obtain host where host: "h ⊢ get_host shadow_root →r host"
      by (metis "1.prems"(1) assms(1) assms(2) assms(3) document_ptr_kinds_commutes
          get_ancestors_di_ptrs_in_heap is_OK_returns_result_E local.get_ancestors_di_ptr local.get_host_ok
          shadow_root_ptr_kinds_commutes)
    then obtain host_ancestors where host_ancestors:
      "h ⊢ get_ancestors_di (castelement_ptr2object_ptr host) →r host_ancestors"
      by (metis "1.prems"(1) assms(1) assms(2) assms(3) get_ancestors_di_also_host get_ancestors_di_ok
          get_ancestors_di_ptrs_in_heap is_OK_returns_result_E local.get_ancestors_di_ptr shadow_root)
    then have "ancestors = cast shadow_root # host_ancestors"
      using 1(4) 1(5) 1(3) None shadow_root host
      by(auto simp add: get_ancestors_di_def[of child, simplified shadow_root]
          elim!: bind_returns_result_E2 dest!: returns_result_eq[OF host] split: option.splits)
    then show ?thesis
    proof (cases "ancestor = cast host")
      case True
      then show ?thesis
        using "1.prems"(1) host local.get_ancestors_di_ptr local.shadow_root_host_dual shadow_root
        by blast
    next
      case False
      have "ancestor ∈ set ancestors"
        using host host_ancestors 1(3) get_ancestors_di_also_host assms(1) assms(2) assms(3)
        using "1.prems"(3) by blast
      then have "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →r children ⟶
ancestor_child ∈ set children ⟶ castnode_ptr2object_ptr ancestor_child ∈ set host_ancestors ⟶ thesis)
⟶
thesis) ∨
          ((∀ancestor_element shadow_root. ancestor = castelement_ptr2object_ptr ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶
castshadow_root_ptr2object_ptr shadow_root ∈ set host_ancestors ⟶ thesis) ⟶ thesis)
  ∨ ((∀disc_doc disc_nodes disc_node. ancestor = cast disc_doc ⟶
h ⊢ get_disconnected_nodes disc_doc →r disc_nodes ⟶ disc_node ∈ set disc_nodes ⟶
cast disc_node ∈ set host_ancestors ⟶ thesis) ⟶ thesis)"
        using "1.hyps"(2) "1.prems"(2) False ‹ancestors = castshadow_root_ptr2object_ptr shadow_root # host_ancestors›
          host host_ancestors shadow_root
        by auto
      then show ?thesis
        using ‹ancestors = castshadow_root_ptr2object_ptr shadow_root # host_ancestors›
        by auto
```

```
    qed
  next
    case (Some child_node)
    then obtain parent_opt where parent_opt: "h ⊢ get_parent child_node →r parent_opt"
      by (metis (no_types, lifting) "1.prems"(1) assms(1) assms(2) assms(3)
          get_ancestors_di_ptrs_in_heap is_OK_returns_result_E local.get_ancestors_di_ptr
          local.get_parent_ok node_ptr_casts_commute node_ptr_kinds_commutes)
    then show ?thesis
    proof (induct parent_opt)
      case None
      then obtain disc_doc where disc_doc: "h ⊢ get_disconnected_document child_node →r disc_doc"
        by (meson assms(1) assms(2) assms(3) is_OK_returns_result_E local.get_disconnected_document_ok)
      then obtain parent_ancestors where parent_ancestors: "h ⊢ get_ancestors_di (cast disc_doc) →r parent_ances
        by (meson assms(1) assms(2) assms(3) document_ptr_kinds_commutes is_OK_returns_result_E
            l_get_ancestors_di_wf_Shadow_DOM.get_ancestors_di_ok l_get_ancestors_di_wf_Shadow_DOM_axioms
            local.get_disconnected_document_disconnected_document_in_heap)
      then have "ancestors = cast child_node # parent_ancestors"
        using 1(4) 1(5) 1(6) Some ‹cast_object_ptr2node_ptr child = Some child_node›
        apply(auto simp add: get_ancestors_di_def[of child,
              simplified ‹cast_object_ptr2node_ptr child = Some child_node›] intro!: bind_pure_I
            elim!: bind_returns_result_E2 dest!: returns_result_eq[OF disc_doc] split: option.splits)[1]
         apply (simp add: returns_result_eq)
        by (meson None.prems option.distinct(1) returns_result_eq)
      then show ?thesis
      proof (cases "ancestor = cast disc_doc")
        case True
        then show ?thesis
          by (metis ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors› disc_doc
              list.set_intros(1) local.disc_doc_disc_node_dual)
      next
        case False
        have "ancestor ∈ set ancestors"
          by (simp add: "1.prems"(3))
        then have "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →r children ⟶
ancestor_child ∈ set children ⟶ cast_node_ptr2object_ptr ancestor_child ∈ set parent_ancestors ⟶ thesis)
⟶
thesis) ∨
          ((∀ancestor_element shadow_root. ancestor = cast_element_ptr2object_ptr ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶
cast_shadow_root_ptr2object_ptr shadow_root ∈ set parent_ancestors ⟶ thesis) ⟶ thesis)
  ∨ ((∀disc_doc disc_nodes disc_node. ancestor = cast disc_doc ⟶
h ⊢ get_disconnected_nodes disc_doc →r disc_nodes ⟶ disc_node ∈ set disc_nodes ⟶
cast disc_node ∈ set parent_ancestors ⟶ thesis) ⟶ thesis)"
          using "1.hyps"(3) "1.prems"(2) False Some ‹cast_object_ptr2node_ptr child = Some child_node›
            ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors› disc_doc parent_ancestors
          by auto
        then show ?thesis
          using ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors› by auto
      qed
    next
      case (Some option)
      then obtain parent where parent: "h ⊢ get_parent child_node →r Some parent"
        using 1(4) 1(5) 1(6)
        by(auto simp add: get_ancestors_di_def[of child] intro!: bind_pure_I
            elim!: bind_returns_result_E2 split: option.splits)
      then obtain parent_ancestors where parent_ancestors:
        "h ⊢ get_ancestors_di parent →r parent_ancestors"
        by (meson assms(1) assms(2) assms(3) get_ancestors_di_ok is_OK_returns_result_E
            local.get_parent_parent_in_heap)
      then have "ancestors = cast child_node # parent_ancestors"
        using 1(4) 1(5) 1(6) Some ‹cast_object_ptr2node_ptr child = Some child_node›
        by(auto simp add: get_ancestors_di_def[of child, simplified
              ‹cast_object_ptr2node_ptr child = Some child_node›] dest!:  elim!: bind_returns_result_E2
```

```
          dest!: returns_result_eq[OF parent] split: option.splits)
      then show ?thesis
      proof (cases "ancestor = parent")
        case True
        then show ?thesis
          by (metis ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors›
              list.set_intros(1) local.get_parent_child_dual parent)
      next
        case False
        have "ancestor ∈ set ancestors"
          by (simp add: "1.prems"(3))
        then have "((∀children ancestor_child. h ⊢ get_child_nodes ancestor →_r children ⟶
ancestor_child ∈ set children ⟶ cast_node_ptr2object_ptr ancestor_child ∈ set parent_ancestors ⟶ thesis)
⟶ thesis) ∨
          ((∀ancestor_element shadow_root. ancestor = cast_element_ptr2object_ptr ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →_r Some shadow_root ⟶ cast_shadow_root_ptr2object_ptr shadow_root ∈
set parent_ancestors ⟶ thesis) ⟶ thesis)
  ∨ ((∀disc_doc disc_nodes disc_node. ancestor = cast disc_doc ⟶
h ⊢ get_disconnected_nodes disc_doc →_r disc_nodes ⟶ disc_node ∈ set disc_nodes ⟶
cast disc_node ∈ set parent_ancestors ⟶ thesis) ⟶ thesis)"
          using "1.hyps"(1) "1.prems"(2) False Some ‹cast_object_ptr2node_ptr child = Some child_node›
            ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors› parent parent_ancestors
          by auto
        then show ?thesis
          using ‹ancestors = cast_node_ptr2object_ptr child_node # parent_ancestors›
          by auto
      qed
    qed
  qed
qed

lemma get_ancestors_di_parent_child_a_host_shadow_root_rel:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
  assumes "h ⊢ get_ancestors_di child →_r ancestors"
  shows "(ptr, child) ∈ (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*
⟷ ptr ∈ set ancestors"
proof
  assume "(ptr, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel
h)* "
  then show "ptr ∈ set ancestors"
  proof (induct ptr rule: heap_wellformed_induct_si[OF assms(1)])
    case (1 ptr)
    then show ?case
    proof (cases "ptr = child")
      case True
      then show ?thesis
        using assms(4) get_ancestors_di_ptr by blast
    next
      case False
      obtain ptr_child where
        ptr_child: "(ptr, ptr_child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_
h) ∧
(ptr_child, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel
h)*"
        using converse_rtranclE[OF 1(4)] ‹ptr ≠ child›
        by metis
      then show ?thesis
      proof(cases "(ptr, ptr_child) ∈ parent_child_rel h")
        case True

        then obtain ptr_child_node
          where ptr_child_ptr_child_node: "ptr_child = cast_node_ptr2object_ptr ptr_child_node"
          using ptr_child node_ptr_casts_commute3 CD.parent_child_rel_node_ptr
```

```
          by (metis)
        then obtain children where
          children: "h ⊢ get_child_nodes ptr →r children" and
          ptr_child_node: "ptr_child_node ∈ set children"
        proof -
          assume a1: "⋀children. ⟦h ⊢ get_child_nodes ptr →r children; ptr_child_node ∈ set children⟧
                  ⟹ thesis"

          have "ptr |∈| object_ptr_kinds h"
            using CD.parent_child_rel_parent_in_heap True by blast
          moreover have "ptr_child_node ∈ set |h ⊢ get_child_nodes ptr|r"
            by (metis True assms(2) assms(3) calculation local.CD.parent_child_rel_child
                local.get_child_nodes_ok local.known_ptrs_known_ptr ptr_child_ptr_child_node
                returns_result_select_result)
          ultimately show ?thesis
            using a1 get_child_nodes_ok ‹type_wf h› ‹known_ptrs h›
            by (meson local.known_ptrs_known_ptr returns_result_select_result)
        qed
        moreover have "(castnode_ptr2object_ptr ptr_child_node, child) ∈
(parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*"
          using ptr_child True ptr_child_ptr_child_node by auto
        ultimately have "castnode_ptr2object_ptr ptr_child_node ∈ set ancestors"
          using 1 by auto
        moreover have "h ⊢ get_parent ptr_child_node →r Some ptr"
          using assms(1) children ptr_child_node child_parent_dual
          using ‹known_ptrs h› ‹type_wf h› by blast
        ultimately show ?thesis
          using get_ancestors_di_also_parent assms ‹type_wf h› by blast
      next
        case False
        then show ?thesis
        proof (cases "(ptr, ptr_child) ∈ a_host_shadow_root_rel h")
          case True
          then
          obtain host where host: "ptr = castelement_ptr2object_ptr host"
            using ptr_child
            by(auto simp add: a_host_shadow_root_rel_def)
          then obtain shadow_root where shadow_root: "h ⊢ get_shadow_root host →r Some shadow_root"
            and ptr_child_shadow_root: "ptr_child = cast shadow_root"
            using False True
            apply(auto simp add: a_host_shadow_root_rel_def)[1]
            by (metis (no_types, lifting) assms(3) local.get_shadow_root_ok select_result_I)

          moreover have "(cast shadow_root, child) ∈
(parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*"
            using ptr_child ptr_child_shadow_root by blast
          ultimately have "cast shadow_root ∈ set ancestors"
            using "1.hyps"(2) host by blast
          moreover have "h ⊢ get_host shadow_root →r host"
            by (metis assms(1) assms(2) assms(3) is_OK_returns_result_E local.get_host_ok
                local.get_shadow_root_shadow_root_ptr_in_heap local.shadow_root_host_dual local.shadow_root_same_ho
                shadow_root)
          ultimately show ?thesis
            using get_ancestors_di_also_host assms(1) assms(2) assms(3) assms(4) host
            by blast
        next
          case False
          then
          obtain disc_doc where disc_doc: "ptr = castdocument_ptr2object_ptr disc_doc"
            using ptr_child ‹(ptr, ptr_child) ∉ parent_child_rel h›
            by(auto simp add: a_ptr_disconnected_node_rel_def)
          then obtain disc_node disc_nodes where disc_nodes:
            "h ⊢ get_disconnected_nodes disc_doc →r disc_nodes"
```

```
            and disc_node: "disc_node ∈ set disc_nodes"
            and ptr_child_disc_node: "ptr_child = cast disc_node"
            using False ‹(ptr, ptr_child) ∉ parent_child_rel h› ptr_child
            apply(auto simp add: a_ptr_disconnected_node_rel_def)[1]
            by (metis (no_types, lifting) CD.get_disconnected_nodes_ok assms(3)
                returns_result_select_result)

          moreover have "(cast disc_node, child) ∈
(parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*"
            using ptr_child ptr_child_disc_node by blast
          ultimately have "cast disc_node ∈ set ancestors"
            using "1.hyps"(3) disc_doc by blast
          moreover have "h ⊢ get_parent disc_node →r None"
            using ‹(ptr, ptr_child) ∉ parent_child_rel h›
            apply(auto simp add: parent_child_rel_def ptr_child_disc_node)[1]
            by (smt assms(1) assms(2) assms(3) assms(4) calculation disc_node disc_nodes
                get_ancestors_di_ptrs_in_heap is_OK_returns_result_E local.CD.a_heap_is_wellformed_def
                local.CD.distinct_lists_no_parent local.CD.heap_is_wellformed_impl local.get_parent_child_dual
                local.get_parent_ok local.get_parent_parent_in_heap local.heap_is_wellformed_def
                node_ptr_kinds_commutes select_result_I2 split_option_ex)
          then
          have "h ⊢ get_disconnected_document disc_node →r disc_doc"
            using disc_node_disc_doc_dual disc_nodes disc_node assms(1) assms(2) assms(3)
            by blast
          ultimately show ?thesis
            using ‹h ⊢ get_parent disc_node →r None› assms(1) assms(2) assms(3) assms(4)
              disc_doc get_ancestors_di_also_disconnected_document
            by blast
        qed
      qed
    qed
  qed
next
  assume "ptr ∈ set ancestors"
  then show "(ptr, child) ∈ (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel
h)*"
  proof (induct ptr rule: heap_wellformed_induct_si[OF assms(1)])
    case (1 ptr)
    then show ?case
    proof (cases "ptr = child")
      case True
      then show ?thesis
        by simp
    next
      case False
      have 2: "⋀thesis. ((∀children ancestor_child. h ⊢ get_child_nodes ptr →r children ⟶
ancestor_child ∈ set children ⟶ cast ancestor_child ∈ set ancestors ⟶ thesis) ⟶ thesis)
      ∨ ((∀ancestor_element shadow_root. ptr = cast ancestor_element ⟶
h ⊢ get_shadow_root ancestor_element →r Some shadow_root ⟶ cast shadow_root ∈ set ancestors ⟶ thesis)
⟶
thesis)
      ∨ ((∀disc_doc disc_nodes disc_node. ptr = cast disc_doc ⟶
h ⊢ get_disconnected_nodes disc_doc →r disc_nodes ⟶ disc_node ∈ set disc_nodes ⟶
cast disc_node ∈ set ancestors ⟶ thesis) ⟶ thesis)"
        using "1.prems" False assms(1) assms(2) assms(3) assms(4)
          get_ancestors_di_obtains_children_or_shadow_root_or_disconnected_node by blast
      then show ?thesis
      proof (cases "∀thesis. ((∀children ancestor_child. h ⊢ get_child_nodes ptr →r children ⟶
ancestor_child ∈ set children ⟶ cast ancestor_child ∈ set ancestors ⟶ thesis) ⟶ thesis)")
        case True
        then obtain children ancestor_child
          where "h ⊢ get_child_nodes ptr →r children"
            and "ancestor_child ∈ set children"
```

```
            and "cast ancestor_child ∈ set ancestors"
          by blast
        then show ?thesis
          by (meson "1.hyps"(1) in_rtrancl_UnI local.CD.parent_child_rel_child r_into_rtrancl
              rtrancl_trans)
      next
        case False
        note f1 = this
        then show ?thesis
        proof (cases "∀thesis. ((∀disc_doc disc_nodes disc_node. ptr = cast disc_doc ⟶
h ⊢ get_disconnected_nodes disc_doc →r disc_nodes ⟶ disc_node ∈ set disc_nodes ⟶
cast disc_node ∈ set ancestors ⟶ thesis) ⟶ thesis)")
          case True
          then obtain disc_doc disc_nodes disc_node
            where "ptr = cast disc_doc"
              and "h ⊢ get_disconnected_nodes disc_doc →r disc_nodes"
              and "disc_node ∈ set disc_nodes"
              and "cast disc_node ∈ set ancestors"
            by blast
          then show ?thesis
            by (meson "1.hyps"(3) in_rtrancl_UnI
                local.a_ptr_disconnected_node_rel_disconnected_node r_into_rtrancl rtrancl_trans)
        next
          case False
          then
          obtain ancestor_element shadow_root
            where "ptr = cast ancestor_element"
              and "h ⊢ get_shadow_root ancestor_element →r Some shadow_root"
              and "cast shadow_root ∈ set ancestors"
            using f1 2 by smt
          then show ?thesis
            by (meson "1.hyps"(2) in_rtrancl_UnI local.a_host_shadow_root_rel_shadow_root
                r_into_rtrancl rtrancl_trans)
        qed
      qed
    qed
  qed
qed
end
interpretation i_get_ancestors_di_wf?: l_get_ancestors_di_wf_Shadow_DOM
  type_wf known_ptr known_ptrs get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_host get_host_locs get_disconnected_document get_disconnected_document_locs get_ancestors_di
  get_ancestors_di_locs get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root
  get_shadow_root_locs get_tag_name get_tag_name_locs heap_is_wellformed parent_child_rel
  heap_is_wellformed_Core_DOM
  by(auto simp add: l_get_ancestors_di_wf_Shadow_DOM_def instances)
declare l_get_ancestors_di_wf_Shadow_DOM_axioms [instances]
```

get_owner_document

```
locale l_get_owner_document_wf_Shadow_DOM =
  l_get_disconnected_nodes +
  l_get_child_nodes +
  l_get_owner_document_Shadow_DOM +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_parent_wf +
  l_known_ptrs +
  l_get_root_node_wf_Core_DOM +
  l_get_parent_Core_DOM +
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
begin
lemma get_owner_document_disconnected_nodes:
  assumes "heap_is_wellformed h"
```

```
  assumes "h ⊢ get_disconnected_nodes document_ptr →r disc_nodes"
  assumes "node_ptr ∈ set disc_nodes"
  assumes known_ptrs: "known_ptrs h"
  assumes type_wf: "type_wf h"
  shows "h ⊢ get_owner_document (cast node_ptr) →r document_ptr"
proof -
  have 2: "node_ptr |∈| node_ptr_kinds h"
    using assms
    apply(auto simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.a_all_ptrs_in_heap_def)[1]
    using assms(1) local.heap_is_wellformed_disc_nodes_in_heap by blast
  have 3: "document_ptr |∈| document_ptr_kinds h"
    using assms(2) get_disconnected_nodes_ptr_in_heap by blast
  then have 4: "¬(∃parent_ptr. parent_ptr |∈| object_ptr_kinds h ∧ node_ptr ∈ set |h ⊢ get_child_nodes
parent_ptr|r)"
    using CD.distinct_lists_no_parent assms  unfolding heap_is_wellformed_def CD.heap_is_wellformed_def
    by simp
  moreover have "(∃document_ptr. document_ptr |∈| document_ptr_kinds h ∧
node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r) ∨
       (∃parent_ptr. parent_ptr |∈| object_ptr_kinds h ∧ node_ptr ∈ set |h ⊢ get_child_nodes parent_ptr|r)"
    using assms(1) 2 "3" assms(2) assms(3) by auto
  ultimately have 0: "∃!document_ptr∈set |h ⊢ document_ptr_kinds_M|r.
node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r"
    using concat_map_distinct assms(1)  known_ptrs_implies
    by (smt CD.heap_is_wellformed_one_disc_parent DocumentMonad.ptr_kinds_ptr_kinds_M
        disjoint_iff_not_equal local.get_disconnected_nodes_ok local.heap_is_wellformed_def
        returns_result_select_result type_wf)


  have "h ⊢ get_parent node_ptr →r None"
    using 4 2
    apply(auto simp add: get_parent_def intro!: bind_pure_returns_result_I filter_M_pure_I bind_pure_I )[1]
    apply(auto intro!:  filter_M_empty_I bind_pure_I bind_pure_returns_result_I)[1]
    using get_child_nodes_ok assms(4)  type_wf
    by (metis get_child_nodes_ok known_ptrs_known_ptr returns_result_select_result)


  then have 4: "h ⊢ get_root_node (cast node_ptr) →r cast node_ptr"
    using get_root_node_no_parent
    by simp
  obtain document_ptrs where document_ptrs: "h ⊢ document_ptr_kinds_M →r document_ptrs"
    by simp
  then have "h ⊢ ok (filter_M (λdocument_ptr. do {
          disconnected_nodes ← get_disconnected_nodes document_ptr;
          return (((cast_node_ptr2object_ptr node_ptr)) ∈ cast ‘ set disconnected_nodes)
        }) document_ptrs)"
    using assms(1) get_disconnected_nodes_ok type_wf
    by(auto intro!: bind_is_OK_I2 filter_M_is_OK_I bind_pure_I)
  then obtain candidates where
    candidates: "h ⊢ filter_M (λdocument_ptr. do {
          disconnected_nodes ← get_disconnected_nodes document_ptr;
          return (((cast_node_ptr2object_ptr node_ptr)) ∈ cast ‘ set disconnected_nodes)
        }) document_ptrs →r candidates"
    by auto

  have filter: "filter (λdocument_ptr. |h ⊢ do {
          disconnected_nodes ← get_disconnected_nodes document_ptr;
          return (cast_node_ptr2object_ptr node_ptr ∈ cast ‘ set disconnected_nodes)
        }|r) document_ptrs = [document_ptr]"
    apply(rule filter_ex1)
    using 0 document_ptrs apply(simp)[1]
       apply (smt "0" "3" assms bind_is_OK_pure_I bind_pure_returns_result_I bind_pure_returns_result_I2
        bind_returns_result_E2 bind_returns_result_E3 document_ptr_kinds_M_def get_disconnected_nodes_ok
        get_disconnected_nodes_pure image_eqI is_OK_returns_result_E l_ptr_kinds_M.ptr_kinds_ptr_kinds_M
```

```
return_ok
        return_returns_result returns_result_eq select_result_E select_result_I select_result_I2 select_result_I2)
    using assms(2) assms(3)
      apply (smt bind_is_OK_I2 bind_returns_result_E3 get_disconnected_nodes_pure image_eqI
        is_OK_returns_result_I return_ok return_returns_result select_result_E)
    using document_ptrs  3 apply(simp)
    using document_ptrs
    by simp
  have "h ⊢ filter_M (λdocument_ptr. do {
          disconnected_nodes ← get_disconnected_nodes document_ptr;
          return (((cast_node_ptr2object_ptr node_ptr)) ∈ cast ‘ set disconnected_nodes)
        }) document_ptrs →_r [document_ptr]"
    apply(rule filter_M_filter2)
    using get_disconnected_nodes_ok document_ptrs 3  assms(1)  type_wf filter
    by(auto intro: bind_pure_I bind_is_OK_I2)

  with 4 document_ptrs have "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →_r document_ptr"
    by(auto simp add: CD.a_get_owner_document_node_ptr_def intro!:  bind_pure_returns_result_I
        filter_M_pure_I bind_pure_I split: option.splits)
  moreover have "known_ptr (cast node_ptr)"
    using known_ptrs_known_ptr[OF known_ptrs, where ptr="cast_node_ptr2object_ptr node_ptr"] 2
      known_ptrs_implies by(simp)
  ultimately show ?thesis
    using 2
    apply(auto simp add: CD.a_get_owner_document_tups_def get_owner_document_def
        a_get_owner_document_tups_def known_ptr_impl)[1]
    apply(split invoke_splits, (rule conjI | rule impI)+)+
        apply(drule(1) known_ptr_not_shadow_root_ptr)
        apply(drule(1) known_ptr_not_document_ptr)
        apply(drule(1)  known_ptr_not_character_data_ptr)
        apply(drule(1)  known_ptr_not_element_ptr)
        apply(simp add: NodeClass.known_ptr_defs)
    by(auto split: option.splits intro!: bind_pure_returns_result_I)
qed

lemma in_disconnected_nodes_no_parent:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_parent node_ptr →_r None"
  assumes "h ⊢ get_owner_document (cast node_ptr) →_r owner_document"
  assumes "h ⊢ get_disconnected_nodes owner_document →_r disc_nodes"
  assumes "known_ptrs h"
  assumes "type_wf h"
  shows "node_ptr ∈ set disc_nodes"
proof -
  have "⋀parent. parent |∈| object_ptr_kinds h ⟹ node_ptr ∉ set |h ⊢ get_child_nodes parent|_r"
    using assms(2)
    by (meson get_child_nodes_ok assms(1) assms(5) assms(6) local.child_parent_dual
        local.known_ptrs_known_ptr option.distinct(1) returns_result_eq returns_result_select_result)
  then show ?thesis
    by (smt (verit) assms(1) assms(2) assms(3) assms(4) assms(5) assms(6)
        is_OK_returns_result_I local.get_disconnected_nodes_ok local.get_owner_document_disconnected_nodes
        local.get_parent_ptr_in_heap local.heap_is_wellformed_children_disc_nodes returns_result_select_result
        select_result_I2)
qed


lemma get_owner_document_owner_document_in_heap_node:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →_r owner_document"
  shows "owner_document |∈| document_ptr_kinds h"
proof -
  obtain root where
    root: "h ⊢ get_root_node (cast node_ptr) →_r root"
```

```
    using assms(4)
    by(auto simp add: CD.a_get_owner_document_node_ptr_def elim!: bind_returns_result_E2
        split: option.splits)

  then show ?thesis
  proof (cases "is_document_ptr root")
    case True
    then show ?thesis
      using assms(4) root
      apply(auto simp add: CD.a_get_owner_document_node_ptr_def elim!:  bind_returns_result_E2
          intro!: filter_M_pure_I bind_pure_I split: option.splits)[1]
       apply(drule(1) returns_result_eq) apply(auto)[1]
      using assms document_ptr_kinds_commutes get_root_node_root_in_heap
      by blast
  next
    case False
    have "known_ptr root"
      using assms local.get_root_node_root_in_heap local.known_ptrs_known_ptr root by blast
    have "root |∈| object_ptr_kinds h"
      using root
      using assms local.get_root_node_root_in_heap
      by blast

    show ?thesis
    proof (cases "is_shadow_root_ptr root")
      case True
      then show ?thesis
        using assms(4) root
        apply(auto simp add: CD.a_get_owner_document_node_ptr_def elim!:  bind_returns_result_E2
            intro!: filter_M_pure_I bind_pure_I split: option.splits)[1]
         apply(drule(1) returns_result_eq) apply(auto)[1]
        using assms document_ptr_kinds_commutes get_root_node_root_in_heap
        by blast
    next
      case False
      then have "is_node_ptr_kind root"
        using ‹¬ is_document_ptr_object_ptr root› ‹known_ptr root› ‹root |∈| object_ptr_kinds h›
        apply(simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
            CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs)
        using is_node_ptr_kind_none
        by force
      then
      have "(∃document_ptr ∈ fset (document_ptr_kinds h). root ∈ cast ‘ set |h ⊢ get_disconnected_nodes
document_ptr|_r)"
        using local.child_parent_dual local.get_child_nodes_ok local.get_root_node_same_no_parent
          local.heap_is_wellformed_children_disc_nodes local.known_ptrs_known_ptr node_ptr_casts_commute3
          node_ptr_inclusion node_ptr_kinds_commutes option.distinct(1) returns_result_eq
          returns_result_select_result root
        by (metis (no_types, opaque_lifting) assms ‹root |∈| object_ptr_kinds h›)
      then obtain some_owner_document where
        "some_owner_document |∈| document_ptr_kinds h" and
        "root ∈ cast ‘ set |h ⊢ get_disconnected_nodes some_owner_document|_r"
        by auto
      then
      obtain candidates where
        candidates: "h ⊢ filter_M
               (λdocument_ptr.
                   Heap_Error_Monad.bind (get_disconnected_nodes document_ptr)
                    (λdisconnected_nodes. return (root ∈ cast_node_ptr2object_ptr ‘ set disconnected_nodes)))
               (sorted_list_of_set (fset (document_ptr_kinds h)))
         →_r candidates"
        by (metis (no_types, lifting) assms bind_is_OK_I2 bind_pure_I filter_M_is_OK_I finite_fset
            is_OK_returns_result_E local.get_disconnected_nodes_ok local.get_disconnected_nodes_pure
```

```
        return_ok return_pure sorted_list_of_set(1))
      then have "some_owner_document ∈ set candidates"
        apply(rule filter_M_in_result_if_ok)
        using ‹some_owner_document |∈| document_ptr_kinds h›
          ‹root ∈ cast ‘ set |h ⊢ get_disconnected_nodes some_owner_document|r›
          apply(auto intro!: bind_pure_I bind_pure_returns_result_I)[1]
        using ‹some_owner_document |∈| document_ptr_kinds h›
          ‹root ∈ cast ‘ set |h ⊢ get_disconnected_nodes some_owner_document|r›
         apply(auto intro!: bind_pure_I bind_pure_returns_result_I)[1]
        using ‹some_owner_document |∈| document_ptr_kinds h›
          ‹root ∈ cast ‘ set |h ⊢ get_disconnected_nodes some_owner_document|r›
        apply(auto simp add: assms local.get_disconnected_nodes_ok
            intro!: bind_pure_I bind_pure_returns_result_I)[1]
        done

      then have "candidates ≠ []"
        by auto
      then have "owner_document ∈ set candidates"
        using assms(4) root
        apply(auto simp add: CD.a_get_owner_document_node_ptr_def elim!:  bind_returns_result_E2
            intro!: filter_M_pure_I bind_pure_I split: option.splits)[1]
         apply (metis candidates list.set_sel(1) returns_result_eq)
        by (metis ‹is_node_ptr_kind root› node_ptr_no_document_ptr_cast returns_result_eq)

      then show ?thesis
        using candidates
        by (meson bind_pure_I bind_returns_result_E2 filter_M_holds_for_result is_OK_returns_result_I
            local.get_disconnected_nodes_ptr_in_heap local.get_disconnected_nodes_pure return_pure)
    qed
  qed
qed

lemma get_owner_document_owner_document_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_owner_document ptr →r owner_document"
  shows "owner_document |∈| document_ptr_kinds h"
  using assms
  apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
      CD.a_get_owner_document_tups_def)[1]
  apply(split invoke_split_asm)+
proof -
  assume "h ⊢ invoke [] ptr () →r owner_document"
  then show "owner_document |∈| document_ptr_kinds h"
    by (meson invoke_empty is_OK_returns_result_I)
next
  assume "h ⊢ Heap_Error_Monad.bind (check_in_heap ptr)
          (λ_. (CD.a_get_owner_document_document_ptr ∘ the ∘ cast_object_ptr2document_ptr) ptr ())
    →r owner_document"
  then show "owner_document |∈| document_ptr_kinds h"
    by(auto simp add: CD.a_get_owner_document_document_ptr_def elim!: bind_returns_result_E2
        split: if_splits)
next
  assume 0: "heap_is_wellformed h"
    and 1: "type_wf h"
    and 2: "known_ptrs h"
    and 3: "¬ is_element_ptr_object_ptr ptr"
    and 4: "is_character_data_ptr_object_ptr ptr"
    and 5: "h ⊢ Heap_Error_Monad.bind (check_in_heap ptr)
(λ_. (CD.a_get_owner_document_node_ptr ∘ the ∘ cast_object_ptr2node_ptr) ptr ()) →r owner_document"
  then show ?thesis
    by (metis bind_returns_result_E2 check_in_heap_pure comp_apply
        get_owner_document_owner_document_in_heap_node)
next
```

```
  assume 0: "heap_is_wellformed h"
    and 1: "type_wf h"
    and 2: "known_ptrs h"
    and 3: "is_element_ptr_object_ptr ptr"
    and 4: "h ⊢ Heap_Error_Monad.bind (check_in_heap ptr)
(λ_. (CD.a_get_owner_document_node_ptr ∘ the ∘ cast_object_ptr2node_ptr) ptr ()) →_r owner_document"
  then show ?thesis
    by (metis bind_returns_result_E2 check_in_heap_pure comp_apply get_owner_document_owner_document_in_heap_node)
next
  assume 0: "heap_is_wellformed h"
    and 1: "type_wf h"
    and 2: "known_ptrs h"
    and 3: "¬ is_element_ptr_object_ptr ptr"
    and 4: "¬ is_character_data_ptr_object_ptr ptr"
    and 5: "¬ is_document_ptr_object_ptr ptr"
    and 6: "is_shadow_root_ptr_object_ptr ptr"
    and 7: "h ⊢ Heap_Error_Monad.bind (check_in_heap ptr)
(λ_. (local.a_get_owner_document_shadow_root_ptr ∘ the ∘ cast_object_ptr2shadow_root_ptr) ptr ()) →_r owner_document"
  then show "owner_document |∈| document_ptr_kinds h"
    by(auto simp add: CD.a_get_owner_document_document_ptr_def a_get_owner_document_shadow_root_ptr_def
        elim!: bind_returns_result_E2 split: if_splits)
qed

lemma get_owner_document_ok:
  assumes "heap_is_wellformed h"  "known_ptrs h" "type_wf h"
  assumes "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_owner_document ptr)"
proof -
  have "known_ptr ptr"
    using assms(2) assms(4) local.known_ptrs_known_ptr
    by blast
  then show ?thesis
    apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def CD.a_get_owner_document_tups_def)[1]
    apply(split invoke_splits, (rule conjI | rule impI)+)+
        apply(auto simp add: known_ptr_impl)[1]
    using NodeClass.a_known_ptr_def known_ptr_not_character_data_ptr known_ptr_not_document_ptr
      known_ptr_not_shadow_root_ptr known_ptr_not_element_ptr apply blast
    using assms(4)
       apply(auto simp add: get_root_node_def CD.a_get_owner_document_document_ptr_def
        CD.a_get_owner_document_node_ptr_def a_get_owner_document_shadow_root_ptr_def
        intro!: bind_is_OK_pure_I filter_M_pure_I bind_pure_I filter_M_is_OK_I
        split: option.splits)[1]
    using assms(4)
      apply(auto simp add: get_root_node_def CD.a_get_owner_document_document_ptr_def
        CD.a_get_owner_document_node_ptr_def a_get_owner_document_shadow_root_ptr_def intro!: bind_is_OK_pure_I
        filter_M_pure_I bind_pure_I filter_M_is_OK_I split: option.splits)[1]
    using assms(4)
     apply(auto simp add: assms(1) assms(2) assms(3) local.get_ancestors_ok get_disconnected_nodes_ok
        get_root_node_def CD.a_get_owner_document_document_ptr_def CD.a_get_owner_document_node_ptr_def
        a_get_owner_document_shadow_root_ptr_def intro!: bind_is_OK_pure_I filter_M_pure_I bind_pure_I
        filter_M_is_OK_I split: option.splits)[1]
    using assms(4)
    apply(auto simp add: assms(1) assms(2) assms(3) local.get_ancestors_ok get_disconnected_nodes_ok
        get_root_node_def CD.a_get_owner_document_document_ptr_def CD.a_get_owner_document_node_ptr_def
        a_get_owner_document_shadow_root_ptr_def intro!: bind_is_OK_pure_I filter_M_pure_I bind_pure_I
        filter_M_is_OK_I split: option.splits)[1]
    done
qed

lemma get_owner_document_child_same:
  assumes "heap_is_wellformed h"  "known_ptrs h" "type_wf h"
  assumes "h ⊢ get_child_nodes ptr →_r children"
  assumes "child ∈ set children"
```

```
  shows "h ⊢ get_owner_document ptr →r owner_document ⟷ h ⊢ get_owner_document (cast child) →r owner_documen
proof -
  have "ptr |∈| object_ptr_kinds h"
    by (meson assms(4) is_OK_returns_result_I local.get_child_nodes_ptr_in_heap)
  then have "known_ptr ptr"
    using assms(2) local.known_ptrs_known_ptr by blast

  have "cast child |∈| object_ptr_kinds h"
    using assms(1) assms(4) assms(5) local.heap_is_wellformed_children_in_heap node_ptr_kinds_commutes
    by blast
  then
  have "known_ptr (cast child)"
    using assms(2) local.known_ptrs_known_ptr by blast
  then have "is_character_data_ptr_object_ptr (cast child) ∨ is_element_ptr_object_ptr (cast child)"
    by(auto simp add: known_ptr_impl NodeClass.a_known_ptr_def ElementClass.a_known_ptr_def
        CharacterDataClass.a_known_ptr_def DocumentClass.a_known_ptr_def a_known_ptr_def
        split: option.splits)
  obtain root where root: "h ⊢ get_root_node ptr →r root"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.get_root_node_ok)
  then have "h ⊢ get_root_node (cast child) →r root"
    using assms(1) assms(2) assms(3) assms(4) assms(5) local.child_parent_dual local.get_root_node_parent_same
    by blast

  have "h ⊢ get_owner_document ptr →r owner_document ⟷ h ⊢ CD.a_get_owner_document_node_ptr child ()
→r owner_document"
  proof (cases "is_document_ptr ptr")
    case True
    then obtain document_ptr where document_ptr: "cast_document_ptr2object_ptr document_ptr = ptr"
      using case_optionE document_ptr_casts_commute by blast
    then have "root = cast document_ptr"
      using root
      by(auto simp add: get_root_node_def get_ancestors_def elim!: bind_returns_result_E2
          split: option.splits)

    then have "h ⊢ CD.a_get_owner_document_document_ptr document_ptr () →r owner_document ⟷
h ⊢ CD.a_get_owner_document_node_ptr child () →r owner_document"
      using document_ptr ‹h ⊢ get_root_node (cast child) →r root›[simplified ‹root = cast document_ptr›
          document_ptr]
      apply(auto simp add:  CD.a_get_owner_document_node_ptr_def CD.a_get_owner_document_document_ptr_def
          elim!: bind_returns_result_E2 dest!: bind_returns_result_E3[rotated, OF
            ‹h ⊢ get_root_node (cast child) →r root›[simplified ‹root = cast document_ptr› document_ptr],
rotated]
          intro!: bind_pure_returns_result_I filter_M_pure_I bind_pure_I split: if_splits option.splits)[1]
      using ‹ptr |∈| object_ptr_kinds h› document_ptr_kinds_commutes by blast
    then show ?thesis
      using ‹known_ptr ptr›
      apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def known_ptr_impl)[1]
       apply(split invoke_splits, ((rule conjI | rule impI)+)?)+
           apply(drule(1) known_ptr_not_shadow_root_ptr[folded known_ptr_impl])
           apply(drule(1) known_ptr_not_document_ptr)
           apply(drule(1)  known_ptr_not_character_data_ptr)
           apply(drule(1)  known_ptr_not_element_ptr)
           apply(simp add: NodeClass.known_ptr_defs)
      using ‹ptr |∈| object_ptr_kinds h› True
      by(auto simp add: document_ptr[symmetric] intro!: bind_pure_returns_result_I
          split: option.splits)
  next
    case False
    then show ?thesis
    proof (cases "is_shadow_root_ptr ptr")
      case True
```

```
      then obtain shadow_root_ptr where shadow_root_ptr: "cast_shadow_root_ptr2object_ptr shadow_root_ptr =
ptr"
        using case_optionE shadow_root_ptr_casts_commute
        by (metis (no_types, lifting) document_ptr_casts_commute3 is_document_ptr_kind_none option.case_eq_if)
      then have "root = cast shadow_root_ptr"
        using root
        by(auto simp add: get_root_node_def get_ancestors_def elim!: bind_returns_result_E2
            split: option.splits)

      then have "h ⊢ a_get_owner_document_shadow_root_ptr shadow_root_ptr () →_r owner_document ⟷
h ⊢ CD.a_get_owner_document_node_ptr child () →_r owner_document"
        using shadow_root_ptr ‹h ⊢ get_root_node (cast child) →_r root›[simplified ‹root = cast shadow_root_ptr›
            shadow_root_ptr]
        apply(auto simp add:  CD.a_get_owner_document_node_ptr_def a_get_owner_document_shadow_root_ptr_def
            CD.a_get_owner_document_document_ptr_def elim!: bind_returns_result_E2
            dest!: bind_returns_result_E3[rotated, OF ‹h ⊢ get_root_node (cast child) →_r root›[simplified
                ‹root = cast shadow_root_ptr› shadow_root_ptr], rotated] intro!: bind_pure_returns_result_I
            filter_M_pure_I bind_pure_I split: if_splits option.splits)[1]
        using ‹ptr |∈| object_ptr_kinds h› shadow_root_ptr_kinds_commutes document_ptr_kinds_commutes
        by blast
      then show ?thesis
        using ‹known_ptr ptr›
        apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
            CD.a_get_owner_document_tups_def known_ptr_impl)[1]
         apply(split invoke_splits, ((rule conjI | rule impI)+)?)+
             apply(drule(1) known_ptr_not_shadow_root_ptr[folded known_ptr_impl])
             apply(drule(1) known_ptr_not_document_ptr)
             apply(drule(1)  known_ptr_not_character_data_ptr)
             apply(drule(1)  known_ptr_not_element_ptr)
             apply(simp add: NodeClass.known_ptr_defs)
        using ‹ptr |∈| object_ptr_kinds h› True
        using False
        by(auto simp add: a_get_owner_document_shadow_root_ptr_def shadow_root_ptr[symmetric]
            intro!: bind_pure_returns_result_I split: option.splits)
    next
      case False
      then obtain node_ptr where node_ptr: "cast_node_ptr2object_ptr node_ptr = ptr"
        using ‹known_ptr ptr› ‹¬ is_document_ptr_object_ptr ptr›
        by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
            CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs split:
option.splits)
      then have "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →_r owner_document ⟷
h ⊢ CD.a_get_owner_document_node_ptr child () →_r owner_document"
        using root ‹h ⊢ get_root_node (cast child) →_r root›
        unfolding CD.a_get_owner_document_node_ptr_def
        by (meson bind_pure_returns_result_I bind_returns_result_E3 local.get_root_node_pure)
      then show ?thesis
        using ‹known_ptr ptr›
        apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
            CD.a_get_owner_document_tups_def known_ptr_impl)[1]
         apply(split invoke_splits, ((rule conjI | rule impI)+)?)+
             apply(drule(1) known_ptr_not_shadow_root_ptr[folded known_ptr_impl])
             apply(drule(1) known_ptr_not_document_ptr[folded known_ptr_impl])
             apply(drule(1)  known_ptr_not_character_data_ptr)
             apply(drule(1)  known_ptr_not_element_ptr)
             apply(simp add: NodeClass.known_ptr_defs)
        using ‹cast child |∈| object_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› False ‹¬ is_document_ptr_object_ptr
ptr›
            apply(auto simp add: node_ptr[symmetric] intro!: bind_pure_returns_result_I split: )[1]
        using ‹cast child |∈| object_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› False ‹¬ is_document_ptr_object_ptr
ptr›
           apply(auto simp add: node_ptr[symmetric] intro!: bind_pure_returns_result_I split: )[1]
        using ‹cast child |∈| object_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› False ‹¬ is_document_ptr_object_ptr
```

```
ptr›
          apply(auto simp add: node_ptr[symmetric] intro!: bind_pure_returns_result_I split: )[1]
        using ‹cast child |∈| object_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› False ‹¬ is_document_ptr_object_ptr
ptr›
         apply(auto simp add: node_ptr[symmetric] intro!: bind_pure_returns_result_I split: )[1]
        using ‹cast child |∈| object_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› False ‹¬ is_document_ptr_object_ptr
ptr›
        apply(auto simp add: node_ptr[symmetric] intro!: bind_pure_returns_result_I split: )[1]
        apply(split invoke_splits, ((rule conjI | rule impI)+)?)+
        by(auto simp add: node_ptr[symmetric] intro!: bind_pure_returns_result_I dest!: is_OK_returns_result_I)
    qed
  qed
  then show ?thesis
    using ‹is_character_data_ptr_object_ptr (cast_node_ptr2object_ptr child) ∨ is_element_ptr_object_ptr (cast_node_ptr2object
child)›
    using ‹cast child |∈| object_ptr_kinds h›
    by(auto simp add: get_owner_document_def[of "cast_node_ptr2object_ptr child"]
        a_get_owner_document_tups_def CD.a_get_owner_document_tups_def split: invoke_splits)
qed

lemma get_owner_document_rel:
  assumes "heap_is_wellformed h"  "known_ptrs h" "type_wf h"
  assumes "h ⊢ get_owner_document ptr →_r owner_document"
  assumes "ptr ≠ cast owner_document"
  shows "(cast owner_document, ptr) ∈ (parent_child_rel h ∪ a_ptr_disconnected_node_rel h)*"
proof -
  have "ptr |∈| object_ptr_kinds h"
    using assms
    by (meson is_OK_returns_result_I local.get_owner_document_ptr_in_heap)
  then
  have "known_ptr ptr"
    using known_ptrs_known_ptr[OF assms(2)] by simp
  have "is_node_ptr_kind ptr"
  proof (rule ccontr)
    assume "¬ is_node_ptr_kind ptr"
    then
    show False
      using assms(4) ‹known_ptr ptr›
      apply(auto simp add: known_ptr_impl get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def)[1]
      apply(split invoke_splits)+
          apply(drule(1) known_ptr_not_shadow_root_ptr)
          apply(drule(1) known_ptr_not_document_ptr)
          apply(drule(1)  known_ptr_not_character_data_ptr)
          apply(drule(1)  known_ptr_not_element_ptr)
          apply(simp add: NodeClass.known_ptr_defs)
      using ‹ptr |∈| object_ptr_kinds h› assms(5)
      by(auto simp add: CD.a_get_owner_document_document_ptr_def a_get_owner_document_shadow_root_ptr_def
          elim!: bind_returns_result_E2 split: if_splits option.splits)
  qed
  then obtain node_ptr where node_ptr: "ptr = cast_node_ptr2object_ptr node_ptr"
    by (metis node_ptr_casts_commute3)
  then have "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →_r owner_document"
    using assms(4) ‹known_ptr ptr›
    apply(auto simp add: known_ptr_impl get_owner_document_def a_get_owner_document_tups_def
        CD.a_get_owner_document_tups_def)[1]
    apply(split invoke_splits)+
        apply(drule(1) known_ptr_not_shadow_root_ptr)
        apply(drule(1) known_ptr_not_document_ptr)
        apply(drule(1)  known_ptr_not_character_data_ptr)
        apply(drule(1)  known_ptr_not_element_ptr)
        apply(simp add: NodeClass.known_ptr_defs)
    using ‹ptr |∈| object_ptr_kinds h›
```

```
    by (auto simp add: is_document_ptr_kind_none)
  then obtain root where root: "h ⊢ get_root_node (cast node_ptr) →r root"
    by(auto simp add: CD.a_get_owner_document_node_ptr_def elim!: bind_returns_result_E2)
  then have "root |∈| object_ptr_kinds h"
    using assms(1) assms(2) assms(3) local.get_root_node_root_in_heap by blast
  then
  have "known_ptr root"
    using ‹known_ptrs h› local.known_ptrs_known_ptr by blast

  have "(root, cast node_ptr) ∈ (parent_child_rel h ∪ a_ptr_disconnected_node_rel h)*"
    using root
    by (simp add: assms(1) assms(2) assms(3) in_rtrancl_UnI local.get_root_node_parent_child_rel)

  show ?thesis
  proof (cases "is_document_ptr_kind root")
    case True
    then have "root = cast owner_document"
      using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› root
      by(auto simp add: CD.a_get_owner_document_node_ptr_def is_document_ptr_kind_def
          dest!: bind_returns_result_E3[rotated, OF root, rotated] split: option.splits)
    then have "(root, cast node_ptr) ∈ (parent_child_rel h ∪ a_ptr_disconnected_node_rel h)*"
      using assms(1) assms(2) assms(3) in_rtrancl_UnI local.get_root_node_parent_child_rel root
      by blast
    then show ?thesis
      using ‹root = cast_document_ptr2object_ptr owner_document› node_ptr by blast
  next
    case False
    then obtain root_node where root_node: "root = cast_node_ptr2object_ptr root_node"
      using assms(2)   ‹root |∈| object_ptr_kinds h›
      by(auto simp add: known_ptr_impl ShadowRootClass.known_ptr_defs DocumentClass.known_ptr_defs
          CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
          dest!: known_ptrs_known_ptr split: option.splits)
    have "h ⊢ CD.a_get_owner_document_node_ptr root_node () →r owner_document"
      using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› root  False
      apply(auto simp add: root_node CD.a_get_owner_document_node_ptr_def elim!: bind_returns_result_E2
          dest!: bind_returns_result_E3[rotated, OF root, rotated] split: option.splits
          intro!: bind_pure_returns_result_I filter_M_pure_I bind_pure_I)[1]
      by (simp add: assms(1) assms(2) assms(3) local.get_root_node_no_parent local.get_root_node_same_no_parent)
    then
    have "h ⊢ get_owner_document root →r owner_document"
      using ‹known_ptr root›
      apply(auto simp add: get_owner_document_def CD.a_get_owner_document_tups_def
          a_get_owner_document_tups_def known_ptr_impl)[1]
      apply(split invoke_splits, ((rule conjI | rule impI)+)?)+
          apply(drule(1) known_ptr_not_shadow_root_ptr[folded known_ptr_impl])
          apply(drule(1) known_ptr_not_document_ptr)
          apply(drule(1)  known_ptr_not_character_data_ptr)
          apply(drule(1)  known_ptr_not_element_ptr)
          apply(simp add: NodeClass.known_ptr_defs)
      using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› root
        False ‹root |∈| object_ptr_kinds h›
         apply(auto intro!: bind_pure_returns_result_I split: option.splits)[1]
      using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› root
        False ‹root |∈| object_ptr_kinds h›
        apply(auto intro!: bind_pure_returns_result_I split: option.splits)[1]
      using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› root
        False ‹root |∈| object_ptr_kinds h›
       apply(auto simp add: root_node intro!: bind_pure_returns_result_I split: option.splits)[1]
      using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› root
        False ‹root |∈| object_ptr_kinds h›
      apply(auto simp add: root_node intro!: bind_pure_returns_result_I split: option.splits)[1]
      done
    have "¬ (∃parent∈fset (object_ptr_kinds h). root_node ∈ set |h ⊢ get_child_nodes parent|r)"
```

```
      using root_node
      by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3) local.child_parent_dual
          local.get_child_nodes_ok local.get_root_node_same_no_parent local.known_ptrs_known_ptr
          option.distinct(1) returns_result_eq returns_result_select_result root)
    have "root_node |∈| node_ptr_kinds h"
      using assms(1) assms(2) assms(3) local.get_root_node_root_in_heap node_ptr_kinds_commutes root root_node
      by blast
    then have "∃document_ptr∈fset (document_ptr_kinds h). root_node ∈ set |h ⊢ get_disconnected_nodes
document_ptr|_r"
      using ‹¬ (∃parent∈fset (object_ptr_kinds h). root_node ∈ set |h ⊢ get_child_nodes parent|_r)› assms(1)
        local.heap_is_wellformed_children_disc_nodes by blast
    then obtain disc_nodes document_ptr where "h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes"
      and "root_node ∈ set disc_nodes"
      by (meson assms(3) local.get_disconnected_nodes_ok returns_result_select_result)
    then have "document_ptr |∈| document_ptr_kinds h"
      by (meson is_OK_returns_result_I local.get_disconnected_nodes_ptr_in_heap)
    then have "document_ptr = owner_document"
      by (metis ‹h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes›
          ‹h ⊢ get_owner_document root →_r owner_document› ‹root_node ∈ set disc_nodes› assms(1) assms(2)
          assms(3) local.get_owner_document_disconnected_nodes returns_result_eq root_node)
    then have "(cast owner_document, cast root_node) ∈ a_ptr_disconnected_node_rel h"
      apply(auto simp add: a_ptr_disconnected_node_rel_def)[1]
      using ‹h ⊢ local.CD.a_get_owner_document_node_ptr node_ptr () →_r owner_document› assms(1)
        assms(2) assms(3) get_owner_document_owner_document_in_heap_node
      by (metis (no_types, lifting) ‹h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes›
          ‹root_node ∈ set disc_nodes› case_prodI mem_Collect_eq pair_imageI select_result_I2)
    moreover have "(cast root_node, cast node_ptr) ∈
(parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*"
      by (metis assms(1) assms(2) assms(3) in_rtrancl_UnI local.get_root_node_parent_child_rel
          root root_node)
    ultimately show ?thesis
      by (metis (no_types, lifting) assms(1) assms(2) assms(3) in_rtrancl_UnI
          local.get_root_node_parent_child_rel node_ptr r_into_rtrancl root root_node rtrancl_trans)
  qed
qed
end

interpretation i_get_owner_document_wf?: l_get_owner_document_wf_Shadow_DOM
  type_wf get_disconnected_nodes get_disconnected_nodes_locs known_ptr get_child_nodes
  get_child_nodes_locs DocumentClass.known_ptr get_parent get_parent_locs DocumentClass.type_wf
  get_root_node get_root_node_locs CD.a_get_owner_document get_host get_host_locs get_owner_document
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs heap_is_wellformed
  parent_child_rel heap_is_wellformed_Core_DOM get_disconnected_document get_disconnected_document_locs
  known_ptrs get_ancestors get_ancestors_locs
  by(auto simp add: l_get_owner_document_wf_Shadow_DOM_def l_get_owner_document_wf_Shadow_DOM_axioms_def
instances)
declare l_get_owner_document_wf_Shadow_DOM_axioms [instances]


lemma get_owner_document_wf_is_l_get_owner_document_wf [instances]:
  "l_get_owner_document_wf heap_is_wellformed type_wf known_ptr known_ptrs get_disconnected_nodes
get_owner_document get_parent get_child_nodes"
  apply(auto simp add: l_get_owner_document_wf_def l_get_owner_document_wf_axioms_def instances)[1]
  using get_owner_document_disconnected_nodes apply fast
  using in_disconnected_nodes_no_parent apply fast
  using get_owner_document_owner_document_in_heap apply fast
  using get_owner_document_ok apply fast
  using get_owner_document_child_same apply (fast, fast)
  done
```

get_owner_document locale l_get_owner_document_wf_get_root_node_wf_Shadow_DOM =
  l_get_owner_document_Shadow_DOM +
  l_get_root_node_Core_DOM +

```
  l_get_root_node_wf +
  l_get_owner_document_wf +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
begin

lemma get_root_node_document:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node ptr →r root"
  assumes "is_document_ptr_kind root"
  shows "h ⊢ get_owner_document ptr →r the (cast root)"
proof -
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    by (meson is_OK_returns_result_I local.get_root_node_ptr_in_heap)
  then have "known_ptr ptr"
    using assms(3) local.known_ptrs_known_ptr by blast
  {
    assume "is_document_ptr_kind ptr"
    then have "ptr = root"
      using assms(4)
      by(auto simp add: get_root_node_def get_ancestors_def elim!: bind_returns_result_E2
          split: option.splits)
    then have ?thesis
      using ‹is_document_ptr_kind ptr› ‹known_ptr ptr› ‹ptr |∈| object_ptr_kinds h›
      apply(auto simp add: known_ptr_impl get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def)[1]
      apply(split invoke_splits, (rule conjI | rule impI)+)+
          apply(drule(1) known_ptr_not_shadow_root_ptr[folded known_ptr_impl])
          apply(drule(1) known_ptr_not_document_ptr[folded known_ptr_impl])
          apply(drule(1) known_ptr_not_character_data_ptr)
          apply(drule(1) known_ptr_not_element_ptr)
          apply(simp add: NodeClass.known_ptr_defs)
      by(auto simp add: CD.a_get_owner_document_document_ptr_def a_get_owner_document_shadow_root_ptr_def
          intro!: bind_pure_returns_result_I split: option.splits)
  }
  moreover
  {
    assume "is_node_ptr_kind ptr"
    then have ?thesis
      using ‹known_ptr ptr› ‹ptr |∈| object_ptr_kinds h›
      apply(auto simp add: known_ptr_impl get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def)[1]
      apply(split invoke_splits, (rule conjI | rule impI)+)+
          apply(drule(1) known_ptr_not_shadow_root_ptr[folded known_ptr_impl])
          apply(drule(1) known_ptr_not_document_ptr[folded known_ptr_impl])
          apply(drule(1) known_ptr_not_character_data_ptr)
          apply(drule(1) known_ptr_not_element_ptr)
          apply(simp add: NodeClass.known_ptr_defs)
         apply(auto split: option.splits)[1]
      using ‹h ⊢ get_root_node ptr →r root› assms(5)
      by(auto simp add: CD.a_get_owner_document_node_ptr_def is_document_ptr_kind_def
          intro!: bind_pure_returns_result_I  split: option.splits)
  }
  ultimately
  show ?thesis
    using ‹known_ptr ptr›
    by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
        CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
        split: option.splits)
qed

lemma get_root_node_same_owner_document:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
```

```
  assumes "h ⊢ get_root_node ptr →r root"
  shows "h ⊢ get_owner_document ptr →r owner_document ⟷ h ⊢ get_owner_document root →r owner_document"
proof -
  have "ptr |∈| object_ptr_kinds h"
    by (meson assms(4) is_OK_returns_result_I local.get_root_node_ptr_in_heap)
  have "root |∈| object_ptr_kinds h"
    using assms(1) assms(2) assms(3) assms(4) local.get_root_node_root_in_heap by blast
  have "known_ptr ptr"
    using ‹ptr |∈| object_ptr_kinds h› assms(3) local.known_ptrs_known_ptr by blast
  have "known_ptr root"
    using ‹root |∈| object_ptr_kinds h› assms(3) local.known_ptrs_known_ptr by blast
  show ?thesis
  proof (cases "is_document_ptr_kind ptr")
    case True
    then
    have "ptr = root"
      using assms(4)
      apply(auto simp add: get_root_node_def elim!: bind_returns_result_E2)[1]
      by (metis document_ptr_casts_commute3 last_ConsL local.get_ancestors_not_node node_ptr_no_document_ptr_cast)
    then show ?thesis
      by auto
  next
    case False
    then have "is_node_ptr_kind ptr"
      using ‹known_ptr ptr›
      by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
          CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
          split: option.splits)
    then obtain node_ptr where node_ptr: "ptr = cast_node_ptr2object_ptr node_ptr"
      by (metis node_ptr_casts_commute3)
    show ?thesis
    proof
      assume "h ⊢ get_owner_document ptr →r owner_document"
      then have "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document"
        using node_ptr
        apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
            CD.a_get_owner_document_tups_def)[1]
        apply(split invoke_splits)+
            apply (meson invoke_empty is_OK_returns_result_I)
        by(auto  elim!: bind_returns_result_E2 split: option.splits)

      show "h ⊢ get_owner_document root →r owner_document"
      proof (cases "is_document_ptr_kind root")
        case True
        then show ?thesis
        proof (cases "is_shadow_root_ptr root")
          case True
          then
          have "is_shadow_root_ptr root"
            using True ‹known_ptr root›
            by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
                CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
                split: option.splits)
          have "root = cast owner_document"
            using ‹is_document_ptr_kind root›
            by (smt ‹h ⊢ get_owner_document ptr →r owner_document› assms(1) assms(2) assms(3) assms(4)
                document_ptr_casts_commute3 get_root_node_document returns_result_eq)
          then show ?thesis
            apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
                CD.a_get_owner_document_tups_def)[1]
            apply(split invoke_splits, (rule conjI | rule impI)+)+
            using ‹is_shadow_root_ptr root› apply blast
            using ‹root |∈| object_ptr_kinds h›
```

```
            apply(simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none)
            apply (metis ‹h ⊢ get_owner_document ptr →r owner_document› assms(1) assms(2) assms(3)
              case_optionE document_ptr_kinds_def is_shadow_root_ptr_kind_none l_get_owner_document_wf.get_owner_
local.l_get_owner_document_wf_axioms not_None_eq return_bind shadow_root_ptr_casts_commute3 shadow_root_ptr_kinds_c
shadow_root_ptr_kinds_def)
          using ‹root |∈| object_ptr_kinds h› document_ptr_kinds_commutes
            apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none intro!: bind_pure_returns_result_I)[1]
          using ‹root |∈| object_ptr_kinds h› document_ptr_kinds_commutes
           apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none intro!: bind_pure_returns_result_I)[1]
          using ‹root |∈| object_ptr_kinds h› document_ptr_kinds_commutes
          apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none intro!: bind_pure_returns_result_I)[1]
          done
      next
        case False
        then
        have "is_document_ptr root"
          using True ‹known_ptr root›
          by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
              CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs split:
option.splits)
        have "root = cast owner_document"
          using True
          by (smt ‹h ⊢ get_owner_document ptr →r owner_document› assms(1) assms(2) assms(3) assms(4)
              document_ptr_casts_commute3 get_root_node_document returns_result_eq)
        then show ?thesis
          apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
              CD.a_get_owner_document_tups_def)[1]
          apply(split invoke_splits, (rule conjI | rule impI)+)+
          using ‹is_document_ptr_object_ptr root› apply blast
          using ‹root |∈| object_ptr_kinds h›
            apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none)[1]
            apply (metis ‹h ⊢ get_owner_document ptr →r owner_document› assms(1) assms(2) assms(3)
              case_optionE document_ptr_kinds_def is_shadow_root_ptr_kind_none
              l_get_owner_document_wf.get_owner_document_owner_document_in_heap local.l_get_owner_document_wf_axi
              not_None_eq return_bind shadow_root_ptr_casts_commute3 shadow_root_ptr_kinds_commutes shadow_root_p
          using ‹root |∈| object_ptr_kinds h› document_ptr_kinds_commutes
            apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none intro!: bind_pure_returns_result_I)[1]
          using ‹root |∈| object_ptr_kinds h› document_ptr_kinds_commutes
           apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none intro!: bind_pure_returns_result_I)[1]
          using ‹root |∈| object_ptr_kinds h› document_ptr_kinds_commutes
          apply(auto simp add: a_get_owner_document_shadow_root_ptr_def CD.a_get_owner_document_document_ptr_def
              is_node_ptr_kind_none intro!: bind_pure_returns_result_I)[1]
          done
      qed
    next
      case False
      then have "is_node_ptr_kind root"
        using ‹known_ptr root›
        by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
            CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
            split: option.splits)
      then obtain root_node_ptr where root_node_ptr: "root = cast_node_ptr2object_ptr root_node_ptr"
        by (metis node_ptr_casts_commute3)
      then have "h ⊢ CD.a_get_owner_document_node_ptr root_node_ptr () →r owner_document"
        using ‹h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document› assms(4)
        apply(auto simp add: CD.a_get_owner_document_node_ptr_def elim!: bind_returns_result_E2
```

```
            intro!: bind_pure_returns_result_I filter_M_pure_I bind_pure_I split: option.splits)[1]
        apply (metis assms(1) assms(2) assms(3) local.get_root_node_no_parent
            local.get_root_node_same_no_parent node_ptr returns_result_eq)
      using ‹is_node_ptr_kind root› node_ptr returns_result_eq by fastforce
    then show ?thesis
      apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def)[1]
      apply(split invoke_splits, (rule conjI | rule impI)+)+
      using ‹is_node_ptr_kind root› ‹known_ptr root›
          apply(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
          CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
          split: option.splits)[1]
      using ‹is_node_ptr_kind root› ‹known_ptr root›
         apply(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
          CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
          split: option.splits)[1]
      using ‹is_node_ptr_kind root› ‹known_ptr root›
        apply(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
          CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
          split: option.splits)[1]
      using ‹root |∈| object_ptr_kinds h›
      by(auto simp add: root_node_ptr)
  qed
next
  assume "h ⊢ get_owner_document root →r owner_document"
  show "h ⊢ get_owner_document ptr →r owner_document"
  proof (cases "is_document_ptr_kind root")
    case True
    have "root = cast owner_document"
      using ‹h ⊢ get_owner_document root →r owner_document›
      apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def)[1]
      apply(split invoke_splits)+
          apply (meson invoke_empty is_OK_returns_result_I)
         apply(auto simp add: True CD.a_get_owner_document_document_ptr_def
          a_get_owner_document_shadow_root_ptr_def elim!: bind_returns_result_E2 split: if_splits option.splits)[
        apply(auto simp add: True CD.a_get_owner_document_document_ptr_def
          a_get_owner_document_shadow_root_ptr_def elim!: bind_returns_result_E2 split: if_splits option.splits)[
        apply (metis True cast_document_ptr_not_node_ptr(2) is_document_ptr_kind_obtains
          is_node_ptr_kind_none node_ptr_casts_commute3 option.case_eq_if)
      by (metis True cast_document_ptr_not_node_ptr(1) document_ptr_casts_commute3
          is_node_ptr_kind_none node_ptr_casts_commute3 option.case_eq_if)
    then show ?thesis
      using assms(1) assms(2) assms(3) assms(4) get_root_node_document
      by fastforce
  next
    case False
    then have "is_node_ptr_kind root"
      using ‹known_ptr root›
      by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
          CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs split:
option.splits)
    then obtain root_node_ptr where root_node_ptr: "root = cast_node_ptr2object_ptr root_node_ptr"
      by (metis node_ptr_casts_commute3)
    then have "h ⊢ CD.a_get_owner_document_node_ptr root_node_ptr () →r owner_document"
      using ‹h ⊢ get_owner_document root →r owner_document›
      apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
          CD.a_get_owner_document_tups_def)[1]
      apply(split invoke_splits)+
          apply (meson invoke_empty is_OK_returns_result_I)
      by(auto simp add: is_document_ptr_kind_none elim!: bind_returns_result_E2)
    then have "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →r owner_document"
      apply(auto simp add: CD.a_get_owner_document_node_ptr_def elim!: bind_returns_result_E2
```

```
            intro!: bind_pure_returns_result_I filter_M_pure_I bind_pure_I split: option.splits)[1]
        using assms(1) assms(2) assms(3) assms(4) local.get_root_node_no_parent
          local.get_root_node_same_no_parent node_ptr returns_result_eq root_node_ptr
        by fastforce+
      then show ?thesis
        apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def
            CD.a_get_owner_document_tups_def)[1]
        apply(split invoke_splits, (rule conjI | rule impI)+)+
        using node_ptr  ‹known_ptr ptr› ‹ptr |∈| object_ptr_kinds h›

        by(auto simp add: known_ptr_impl known_ptr_defs DocumentClass.known_ptr_defs
            CharacterDataClass.known_ptr_defs ElementClass.known_ptr_defs NodeClass.known_ptr_defs
            intro!: bind_pure_returns_result_I split: option.splits)
    qed
  qed
 qed
qed
end

interpretation i_get_owner_document_wf_get_root_node_wf?: l_get_owner_document_wf_get_root_node_wf_Shadow_DOM
  DocumentClass.known_ptr get_parent get_parent_locs DocumentClass.type_wf get_disconnected_nodes
  get_disconnected_nodes_locs get_root_node get_root_node_locs CD.a_get_owner_document
  get_host get_host_locs get_owner_document get_child_nodes get_child_nodes_locs type_wf known_ptr
  known_ptrs get_ancestors get_ancestors_locs heap_is_wellformed parent_child_rel
  by(auto simp add: l_get_owner_document_wf_get_root_node_wf_Shadow_DOM_def
      l_get_owner_document_wf_get_root_node_wf_Shadow_DOM_axioms_def instances)
declare l_get_owner_document_wf_get_root_node_wf_Shadow_DOM_axioms [instances]

lemma get_owner_document_wf_get_root_node_wf_is_l_get_owner_document_wf_get_root_node_wf [instances]:
  "l_get_owner_document_wf_get_root_node_wf heap_is_wellformed type_wf known_ptr known_ptrs get_root_node
get_owner_document"
  apply(auto simp add: l_get_owner_document_wf_get_root_node_wf_def l_get_owner_document_wf_get_root_node_wf_axiom
instances)[1]
  using get_root_node_document apply blast
  using get_root_node_same_owner_document apply (blast, blast)
  done
```

remove_child

```
locale l_remove_child_wf2_Shadow_DOM =
  l_set_disconnected_nodes_get_disconnected_nodes +
  l_get_child_nodes +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_owner_document_wf_Shadow_DOM +
  l_remove_child_Core_DOM +
  l_set_child_nodes_get_shadow_root +
  l_set_disconnected_nodes_get_shadow_root +
  l_set_child_nodes_get_tag_name +
  l_set_disconnected_nodes_get_tag_name +
  CD: l_remove_child_wf2_Core_DOM _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ heap_is_wellformed_Core_DOM
begin
lemma remove_child_preserves_type_wf:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_child ptr child →_h h'"
  shows "type_wf h'"
  using CD.remove_child_heap_is_wellformed_preserved(1) assms
  unfolding heap_is_wellformed_def
  by auto

lemma remove_child_preserves_known_ptrs:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_child ptr child →_h h'"
  shows "known_ptrs h'"
```

```
  using CD.remove_child_heap_is_wellformed_preserved(2) assms
  unfolding heap_is_wellformed_def
  by auto


lemma remove_child_heap_is_wellformed_preserved:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_child ptr child →h h'"
  shows "heap_is_wellformed h'"
proof -
  obtain owner_document children_h h2 disconnected_nodes_h where
    owner_document: "h ⊢ get_owner_document (cast_node_ptr2object_ptr child) →r owner_document" and
    children_h: "h ⊢ get_child_nodes ptr →r children_h" and
    child_in_children_h: "child ∈ set children_h" and
    disconnected_nodes_h: "h ⊢  get_disconnected_nodes owner_document →r disconnected_nodes_h" and
    h2: "h ⊢  set_disconnected_nodes owner_document (child # disconnected_nodes_h) →h h2" and
    h': "h2 ⊢ set_child_nodes ptr (remove1 child children_h) →h h'"
    using assms(4)
    apply(auto simp add: remove_child_def elim!: bind_returns_heap_E
        dest!: pure_returns_heap_eq[rotated, OF get_owner_document_pure]
        pure_returns_heap_eq[rotated, OF get_child_nodes_pure] split: if_splits)[1]
    using pure_returns_heap_eq by fastforce

  have "heap_is_wellformed_Core_DOM h'"
    using CD.remove_child_heap_is_wellformed_preserved(3) assms
    unfolding heap_is_wellformed_def
    by auto

  have "h ⊢ get_owner_document ptr →r owner_document"
    using owner_document children_h child_in_children_h
    using local.get_owner_document_child_same assms by blast

  have shadow_root_eq: "⋀ptr' shadow_root_ptr_opt. h ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt =
h' ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt"
    using get_shadow_root_reads remove_child_writes assms(4)
    apply(rule reads_writes_preserved)
    by(auto simp add: remove_child_locs_def set_child_nodes_get_shadow_root
        set_disconnected_nodes_get_shadow_root)
  then
  have shadow_root_eq2: "⋀ptr'. |h ⊢ get_shadow_root ptr'|r = |h' ⊢ get_shadow_root ptr'|r"
    by (meson select_result_eq)

  have tag_name_eq: "⋀ptr' tag. h ⊢ get_tag_name ptr' →r tag = h' ⊢ get_tag_name ptr' →r tag"
    using get_tag_name_reads remove_child_writes assms(4)
    apply(rule reads_writes_preserved)
    by(auto simp add: remove_child_locs_def set_child_nodes_get_tag_name
        set_disconnected_nodes_get_tag_name)
  then
  have tag_name_eq2: "⋀ptr'. |h ⊢ get_tag_name ptr'|r = |h' ⊢ get_tag_name ptr'|r"
    by (meson select_result_eq)

  have object_ptr_kinds_eq: "object_ptr_kinds h = object_ptr_kinds h'"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
          OF remove_child_writes assms(4)])
    unfolding remove_child_locs_def
    using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)

  have document_ptr_kinds_eq: "document_ptr_kinds h = document_ptr_kinds h'"
    using object_ptr_kinds_eq
    by(auto simp add: document_ptr_kinds_def document_ptr_kinds_def)

  have shadow_root_ptr_kinds_eq: "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"
```

```
    using object_ptr_kinds_eq
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  have element_ptr_kinds_eq: "element_ptr_kinds h = element_ptr_kinds h'"
    using object_ptr_kinds_eq
    by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)

  have "parent_child_rel h' ⊆ parent_child_rel h"
    using ‹heap_is_wellformed h› heap_is_wellformed_def
    using CD.remove_child_parent_child_rel_subset
    using ‹known_ptrs h› ‹type_wf h› assms(4)
    by simp

  have "known_ptr ptr"
    using assms(3)
    using children_h get_child_nodes_ptr_in_heap local.known_ptrs_known_ptr by blast
  have "type_wf h2"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'",
        OF set_disconnected_nodes_writes h2]
    using set_disconnected_nodes_types_preserved ‹type_wf h›
    by(auto simp add: reflp_def transp_def)

  have children_eq:
    "⋀ptr' children. ptr ≠ ptr' ⟹ h ⊢ get_child_nodes ptr' →r children =
h' ⊢ get_child_nodes ptr' →r children"
    apply(rule reads_writes_preserved[OF get_child_nodes_reads remove_child_writes assms(4)])
    unfolding remove_child_locs_def
    using set_disconnected_nodes_get_child_nodes set_child_nodes_get_child_nodes_different_pointers
    by fast
  then have children_eq2:
    "⋀ptr' children. ptr ≠ ptr' ⟹ |h ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have "h2 ⊢ get_child_nodes ptr →r children_h"
    apply(rule reads_writes_separate_forwards[OF get_child_nodes_reads
          set_disconnected_nodes_writes h2 children_h] )
    by (simp add: set_disconnected_nodes_get_child_nodes)

  have children_h': "h' ⊢ get_child_nodes ptr →r remove1 child children_h"
    using assms(4) owner_document h2 disconnected_nodes_h children_h
    apply(auto simp add: remove_child_def split: if_splits)[1]
    apply(drule bind_returns_heap_E3)
      apply(auto split: if_splits)[1]
     apply(simp)
    apply(auto split: if_splits)[1]
    apply(drule bind_returns_heap_E3)
      apply(auto)[1]
     apply(simp)
    apply(drule bind_returns_heap_E3)
      apply(auto)[1]
     apply(simp)
    apply(drule bind_returns_heap_E4)
      apply(auto)[1]
     apply simp
    using ‹type_wf h2› set_child_nodes_get_child_nodes ‹known_ptr ptr› h'
    by blast

  have disconnected_nodes_eq: "⋀ptr' disc_nodes. ptr' ≠ owner_document ⟹
h ⊢ get_disconnected_nodes ptr' →r disc_nodes = h2 ⊢ get_disconnected_nodes ptr' →r disc_nodes"
    using local.get_disconnected_nodes_reads set_disconnected_nodes_writes h2
    apply(rule reads_writes_preserved)
    by (metis local.set_disconnected_nodes_get_disconnected_nodes_different_pointers)
  then
  have disconnected_nodes_eq2: "⋀ptr'. ptr' ≠ owner_document ⟹
|h ⊢ get_disconnected_nodes ptr'|r = |h2 ⊢ get_disconnected_nodes ptr'|r"
```

```
    by (meson select_result_eq)
  have "h2 ⊢ get_disconnected_nodes owner_document →r child # disconnected_nodes_h"
    using h2 local.set_disconnected_nodes_get_disconnected_nodes
    by blast

  have disconnected_nodes_eq_h2:
    "⋀ptr' disc_nodes. h2 ⊢ get_disconnected_nodes ptr' →r disc_nodes = h' ⊢ get_disconnected_nodes ptr'
→r disc_nodes"
    using local.get_disconnected_nodes_reads set_child_nodes_writes h'
    apply(rule reads_writes_preserved)
    using local.set_child_nodes_get_disconnected_nodes by blast
  then
  have disconnected_nodes_eq2_h2: "⋀ptr'. |h2 ⊢ get_disconnected_nodes ptr'|r = |h' ⊢ get_disconnected_nodes
ptr'|r"
    by (meson select_result_eq)

  have "a_host_shadow_root_rel h' = a_host_shadow_root_rel h"
    by(auto simp add: a_host_shadow_root_rel_def shadow_root_eq2 element_ptr_kinds_eq)
  moreover
  have "(ptr, cast child) ∈ parent_child_rel h"
    using child_in_children_h children_h local.CD.parent_child_rel_child by blast
  moreover
  have "a_ptr_disconnected_node_rel h' = insert (cast owner_document, cast child) (a_ptr_disconnected_node_rel
h)"
    using ‹h2 ⊢ get_disconnected_nodes owner_document →r child # disconnected_nodes_h› disconnected_nodes_eq2
disconnected_nodes_h
    apply(auto simp add: a_ptr_disconnected_node_rel_def disconnected_nodes_eq2_h2[symmetric] document_ptr_kinds_e
      apply(case_tac "aa = owner_document")
       apply(auto)[1]
      apply(auto)[1]
     apply (metis (no_types, lifting) assms(4) case_prodI disconnected_nodes_eq_h2 h2
        is_OK_returns_heap_I local.remove_child_in_disconnected_nodes
        local.set_disconnected_nodes_ptr_in_heap mem_Collect_eq owner_document pair_imageI select_result_I2)
    by (metis (no_types, lifting) case_prodI list.set_intros(2) mem_Collect_eq pair_imageI select_result_I2)
  then
  have "a_ptr_disconnected_node_rel h' = a_ptr_disconnected_node_rel h ∪ {(cast owner_document, cast child)}"
    by auto
  moreover have "acyclic (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel
h)"
    using assms(1) local.heap_is_wellformed_def by blast
  moreover have "parent_child_rel h' = parent_child_rel h - {(ptr, cast child)}"
    apply(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq children_eq2)[1]
      apply (metis (no_types, lifting) children_eq2 children_h children_h' notin_set_remove1 select_result_I2)
    using ‹h2 ⊢ get_disconnected_nodes owner_document →r child # disconnected_nodes_h›
      ‹heap_is_wellformedCore_DOM h'› disconnected_nodes_eq_h2 local.CD.distinct_lists_no_parent
      local.CD.heap_is_wellformed_def apply auto[1]
    by (metis (no_types, lifting) children_eq2 children_h children_h' in_set_remove1 select_result_I2)

  moreover have "(cast owner_document, ptr) ∈ (parent_child_rel h ∪ a_ptr_disconnected_node_rel h)*"
    using ‹h ⊢ get_owner_document ptr →r owner_document› get_owner_document_rel
    using assms(1) assms(2) assms(3) by blast
  then have "(cast owner_document, ptr) ∈ (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_no
h)*"
    by (metis (no_types, lifting) in_rtrancl_UnI inf_sup_aci(5) inf_sup_aci(7))
  ultimately
  have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h')"
    by (smt Un_assoc Un_insert_left Un_insert_right acyclic_insert insert_Diff_single
        insert_absorb2 mk_disjoint_insert prod.inject rtrancl_Un_separator_converseE rtrancl_trans
        singletonD sup_bot.comm_neutral)

  show ?thesis
    using ‹heap_is_wellformed h›
    using ‹heap_is_wellformedCore_DOM h'›
```

```
    using ‹acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h')›
    apply(auto simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def
        host_shadow_root_rel_def a_all_ptrs_in_heap_def a_distinct_lists_def a_shadow_root_valid_def)[1]
    by(auto simp add: object_ptr_kinds_eq element_ptr_kinds_eq shadow_root_ptr_kinds_eq
        shadow_root_eq shadow_root_eq2 tag_name_eq tag_name_eq2)
qed

lemma remove_preserves_type_wf:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove child →h h'"
  shows "type_wf h'"
  using CD.remove_heap_is_wellformed_preserved(1) assms
  unfolding heap_is_wellformed_def
  by auto

lemma remove_preserves_known_ptrs:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove child →h h'"
  shows "known_ptrs h'"
  using CD.remove_heap_is_wellformed_preserved(2) assms
  unfolding heap_is_wellformed_def
  by auto


lemma remove_heap_is_wellformed_preserved:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove child →h h'"
  shows "heap_is_wellformed h'"
  using assms
  by(auto simp add: remove_def elim!: bind_returns_heap_E2
      intro: remove_child_heap_is_wellformed_preserved split: option.splits)

lemma remove_child_removes_child:
  "heap_is_wellformed h ⟹ h ⊢ remove_child ptr' child →h h' ⟹ h' ⊢ get_child_nodes ptr →r children
    ⟹ known_ptrs h ⟹ type_wf h
    ⟹ child ∉ set children"
  using CD.remove_child_removes_child local.heap_is_wellformed_def by blast
lemma remove_child_removes_first_child:
  "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_child_nodes ptr →r node_ptr # children
⟹
h ⊢ remove_child ptr node_ptr →h h' ⟹ h' ⊢ get_child_nodes ptr →r children"
  using CD.remove_child_removes_first_child local.heap_is_wellformed_def by blast
lemma remove_removes_child:
  "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_child_nodes ptr →r node_ptr # children
⟹
h ⊢ remove node_ptr →h h' ⟹ h' ⊢ get_child_nodes ptr →r children"
  using CD.remove_removes_child local.heap_is_wellformed_def by blast
lemma remove_for_all_empty_children:
  "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_child_nodes ptr →r children ⟹
h ⊢ forall_M remove children →h h' ⟹ h' ⊢ get_child_nodes ptr →r []"
  using CD.remove_for_all_empty_children local.heap_is_wellformed_def by blast

end

interpretation i_remove_child_wf2?: l_remove_child_wf2Shadow_DOM
  type_wf get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs known_ptr get_child_nodes get_child_nodes_locs get_shadow_root
  get_shadow_root_locs get_tag_name get_tag_name_locs heap_is_wellformed parent_child_rel
  heap_is_wellformedCore_DOM get_host get_host_locs get_disconnected_document get_disconnected_document_locs
  DocumentClass.known_ptr get_parent get_parent_locs DocumentClass.type_wf get_root_node get_root_node_locs
  CD.a_get_owner_document get_owner_document known_ptrs get_ancestors get_ancestors_locs set_child_nodes
  set_child_nodes_locs remove_child remove_child_locs remove
  by(auto simp add: l_remove_child_wf2Shadow_DOM_def instances)
```

```
declare l_remove_child_wf2Shadow_DOM_axioms [instances]


lemma remove_child_wf2_is_l_remove_child_wf2 [instances]:
  "l_remove_child_wf2 type_wf known_ptr known_ptrs remove_child heap_is_wellformed get_child_nodes remove"
  apply(auto simp add: l_remove_child_wf2_def l_remove_child_wf2_axioms_def instances)[1]
  using remove_child_preserves_type_wf apply fast
  using remove_child_preserves_known_ptrs apply fast
  using remove_child_heap_is_wellformed_preserved apply (fast)
  using remove_preserves_type_wf apply fast
  using remove_preserves_known_ptrs apply fast
  using remove_heap_is_wellformed_preserved apply (fast)
  using remove_child_removes_child apply fast
  using remove_child_removes_first_child apply fast
  using remove_removes_child apply fast
  using remove_for_all_empty_children apply fast
  done
```

adopt_node

```
locale l_adopt_node_wfShadow_DOM =
  l_adopt_nodeShadow_DOM +
  CD: l_adopt_node_wfCore_DOM _ _ _ _ _ _ _ _ _ adopt_nodeCore_DOM adopt_node_locsCore_DOM
begin
lemma adopt_node_removes_first_child: "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h
                        ⟹ h ⊢ adopt_node owner_document node →h h'
                        ⟹ h ⊢ get_child_nodes ptr' →r node # children
                        ⟹ h' ⊢ get_child_nodes ptr' →r children"
  by (smt CD.adopt_node_removes_first_child bind_returns_heap_E error_returns_heap
      l_adopt_nodeShadow_DOM.adopt_node_def local.CD.adopt_node_impl local.get_ancestors_di_pure
      local.l_adopt_nodeShadow_DOM_axioms pure_returns_heap_eq)
lemma adopt_node_document_in_heap: "heap_is_wellformed h ⟹ known_ptrs h ⟹ type_wf h
                       ⟹ h ⊢ ok (adopt_node owner_document node)
                       ⟹ owner_document |∈| document_ptr_kinds h"
  by (metis (no_types, lifting) bind_returns_heap_E document_ptr_kinds_commutes is_OK_returns_heap_E
      is_OK_returns_result_I local.adopt_node_def local.get_ancestors_di_ptr_in_heap)
end

locale l_adopt_node_wf2Shadow_DOM =
  l_get_child_nodes +
  l_get_disconnected_nodes +
  l_set_child_nodes_get_shadow_root +
  l_set_disconnected_nodes_get_shadow_root +
  l_set_child_nodes_get_tag_name +
  l_set_disconnected_nodes_get_tag_name +
  l_get_owner_document  +
  l_remove_childCore_DOM+
  l_heap_is_wellformedShadow_DOM +
  l_get_root_node +
  l_set_disconnected_nodes_get_child_nodes +
  l_get_owner_document_wf +
  l_remove_child_wf2 +
  l_adopt_node_wfShadow_DOM +
  l_adopt_nodeShadow_DOM +
  l_get_parent_wfCore_DOM +
  l_get_disconnected_document +
  l_get_ancestors_diShadow_DOM +
  l_get_ancestors_di_wfShadow_DOM
begin
lemma adopt_node_removes_child:
  assumes wellformed: "heap_is_wellformed h"
    and adopt_node: "h ⊢ adopt_node owner_document node_ptr →h h2"
    and children: "h2 ⊢ get_child_nodes ptr →r children"
```

```
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
  shows "node_ptr ∉ set children"
proof -
  obtain old_document parent_opt h' where
    old_document: "h ⊢ get_owner_document (cast node_ptr) →r old_document" and
    parent_opt: "h ⊢ get_parent node_ptr →r parent_opt" and
    h': "h ⊢ (case parent_opt of Some parent ⇒ remove_child parent node_ptr | None ⇒ return () ) →h
h'"
    using adopt_node
    by(auto simp add: adopt_node_def CD.adopt_node_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_ancestors_di_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_owner_document_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        split: if_splits)

  then have "h' ⊢ get_child_nodes ptr →r children"
    using adopt_node
    apply(auto simp add: adopt_node_def CD.adopt_node_def
        dest!: bind_returns_heap_E3[rotated, OF old_document, rotated]
        bind_returns_heap_E3[rotated, OF parent_opt, rotated]
        elim!: bind_returns_heap_E2[rotated, OF get_ancestors_di_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_owner_document_pure, rotated]
        bind_returns_heap_E4[rotated, OF h', rotated] split: if_splits)[1]
     apply(auto split: if_splits elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_ancestors_di_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated])[1]
     apply (simp add: set_disconnected_nodes_get_child_nodes children
        reads_writes_preserved[OF get_child_nodes_reads set_disconnected_nodes_writes])
    using children by blast
  show ?thesis
  proof(insert parent_opt h', induct parent_opt)
    case None
    then show ?case
      using child_parent_dual wellformed known_ptrs type_wf ‹h' ⊢ get_child_nodes ptr →r children›
        returns_result_eq by fastforce
  next
    case (Some option)
    then show ?case
      using remove_child_removes_child ‹h' ⊢ get_child_nodes ptr →r children› known_ptrs type_wf wellformed
      by auto
  qed
qed

lemma adopt_node_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ adopt_node document_ptr child →h h'"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
  shows "heap_is_wellformed h'"
proof -
  obtain old_document parent_opt h2 ancestors where
    "h ⊢ get_ancestors_di (cast document_ptr) →r ancestors" and
    "cast child ∉  set ancestors" and
    old_document: "h ⊢ get_owner_document (cast child) →r old_document" and
    parent_opt: "h ⊢ get_parent child →r parent_opt" and
    h2: "h ⊢ (case parent_opt of Some parent ⇒ remove_child parent child | None ⇒ return ()) →h h2"
and
    h': "h2 ⊢ (if document_ptr ≠ old_document then do {
        old_disc_nodes ← get_disconnected_nodes old_document;
        set_disconnected_nodes old_document (remove1 child old_disc_nodes);
        disc_nodes ← get_disconnected_nodes document_ptr;
```

```
        set_disconnected_nodes document_ptr (child # disc_nodes)
      } else do {
        return ()
      }) →h h'"
    using assms(2)
    apply(auto simp add:  adopt_node_def[unfolded CD.adopt_node_def] elim!: bind_returns_heap_E
        dest!: pure_returns_heap_eq[rotated, OF get_ancestors_di_pure])[1]
    apply(split if_splits)
    by(auto simp add:  elim!: bind_returns_heap_E
        dest!: pure_returns_heap_eq[rotated, OF get_owner_document_pure]
        pure_returns_heap_eq[rotated, OF get_parent_pure])

  have object_ptr_kinds_h_eq3: "object_ptr_kinds h = object_ptr_kinds h2"
    using h2 apply(simp split: option.splits)
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF remove_child_writes
    using remove_child_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have object_ptr_kinds_M_eq_h: "⋀ptrs. h ⊢ object_ptr_kinds_M →r ptrs = h2 ⊢ object_ptr_kinds_M
→r ptrs"
    unfolding object_ptr_kinds_M_defs by simp
  then have object_ptr_kinds_eq_h: "|h ⊢ object_ptr_kinds_M|r = |h2 ⊢ object_ptr_kinds_M|r"
    by simp
  then have node_ptr_kinds_eq_h: "|h ⊢ node_ptr_kinds_M|r = |h2 ⊢ node_ptr_kinds_M|r"
    using node_ptr_kinds_M_eq by blast

  have wellformed_h2: "heap_is_wellformed h2"
    using h2 remove_child_heap_is_wellformed_preserved known_ptrs type_wf
    by (metis (no_types, lifting) assms(1) option.case_eq_if pure_returns_heap_eq return_pure)
  then show "heap_is_wellformed h'"
  proof(cases "document_ptr = old_document")
    case True
    then show "heap_is_wellformed h'"
      using h' wellformed_h2 by auto
  next
    case False
    then obtain h3 old_disc_nodes disc_nodes_document_ptr_h3 where
      docs_neq: "document_ptr ≠ old_document" and
      old_disc_nodes: "h2 ⊢ get_disconnected_nodes old_document →r old_disc_nodes" and
      h3: "h2 ⊢ set_disconnected_nodes old_document (remove1 child old_disc_nodes) →h h3" and
      disc_nodes_document_ptr_h3: "h3 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_document_ptr_h3"
and
      h': "h3 ⊢ set_disconnected_nodes document_ptr (child # disc_nodes_document_ptr_h3) →h h'"
      using h'
      by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated]
)

    have object_ptr_kinds_h2_eq3: "object_ptr_kinds h2 = object_ptr_kinds h3"
      apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF set_disconnected_
h3])
      using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
      by (auto simp add: reflp_def transp_def)
    then have object_ptr_kinds_M_eq_h2: "⋀ptrs. h2 ⊢ object_ptr_kinds_M →r ptrs = h3 ⊢ object_ptr_kinds_M
→r ptrs"
      by(simp add: object_ptr_kinds_M_defs)
    then have object_ptr_kinds_eq_h2: "|h2 ⊢ object_ptr_kinds_M|r = |h3 ⊢ object_ptr_kinds_M|r"
      by(simp)
    then have node_ptr_kinds_eq_h2: "|h2 ⊢ node_ptr_kinds_M|r = |h3 ⊢ node_ptr_kinds_M|r"
      using node_ptr_kinds_M_eq by blast
    then have node_ptr_kinds_eq3_h2: "node_ptr_kinds h2 = node_ptr_kinds h3"
      by auto
    have document_ptr_kinds_eq2_h2: "|h2 ⊢ document_ptr_kinds_M|r = |h3 ⊢ document_ptr_kinds_M|r"
      using object_ptr_kinds_eq_h2 document_ptr_kinds_M_eq by auto
    then have document_ptr_kinds_eq3_h2: "document_ptr_kinds h2 = document_ptr_kinds h3"
```

```
      using object_ptr_kinds_eq_h2 document_ptr_kinds_M_eq by auto
    have children_eq_h2: "⋀ptr children. h2 ⊢ get_child_nodes ptr →r children = h3 ⊢ get_child_nodes
ptr →r children"
      using get_child_nodes_reads set_disconnected_nodes_writes h3
      apply(rule reads_writes_preserved)
      by (simp add: set_disconnected_nodes_get_child_nodes)
    then have children_eq2_h2: "⋀ptr. |h2 ⊢ get_child_nodes ptr|r = |h3 ⊢ get_child_nodes ptr|r"
      using select_result_eq by force

    have object_ptr_kinds_h3_eq3: "object_ptr_kinds h3 = object_ptr_kinds h'"
      apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF set_disconnected_
h'])
      using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
      by (auto simp add: reflp_def transp_def)
    then have object_ptr_kinds_M_eq_h3: "⋀ptrs. h3 ⊢ object_ptr_kinds_M →r ptrs = h' ⊢ object_ptr_kinds_M
→r ptrs"
      by(simp add: object_ptr_kinds_M_defs)
    then have object_ptr_kinds_eq_h3: "|h3 ⊢ object_ptr_kinds_M|r = |h' ⊢ object_ptr_kinds_M|r"
      by(simp)
    then have node_ptr_kinds_eq_h3: "|h3 ⊢ node_ptr_kinds_M|r = |h' ⊢ node_ptr_kinds_M|r"
      using node_ptr_kinds_M_eq by blast
    then have node_ptr_kinds_eq3_h3: "node_ptr_kinds h3 = node_ptr_kinds h'"
      by auto
    have document_ptr_kinds_eq2_h3: "|h3 ⊢ document_ptr_kinds_M|r = |h' ⊢ document_ptr_kinds_M|r"
      using object_ptr_kinds_eq_h3 document_ptr_kinds_M_eq by auto
    then have document_ptr_kinds_eq3_h3: "document_ptr_kinds h3 = document_ptr_kinds h'"
      using object_ptr_kinds_eq_h3 document_ptr_kinds_M_eq by auto
    have children_eq_h3: "⋀ptr children. h3 ⊢ get_child_nodes ptr →r children = h' ⊢ get_child_nodes
ptr →r children"
      using get_child_nodes_reads set_disconnected_nodes_writes h'
      apply(rule reads_writes_preserved)
      by (simp add: set_disconnected_nodes_get_child_nodes)
    then have children_eq2_h3: "⋀ptr. |h3 ⊢ get_child_nodes ptr|r = |h' ⊢ get_child_nodes ptr|r"
      using select_result_eq by force

    have disconnected_nodes_eq_h2:
      "⋀doc_ptr disc_nodes. old_document ≠ doc_ptr ⟹
h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes = h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
      using get_disconnected_nodes_reads set_disconnected_nodes_writes h3
      apply(rule reads_writes_preserved)
      by (simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
    then have disconnected_nodes_eq2_h2:
      "⋀doc_ptr. old_document ≠ doc_ptr ⟹
|h2 ⊢ get_disconnected_nodes doc_ptr|r = |h3 ⊢ get_disconnected_nodes doc_ptr|r"
      using select_result_eq by force
    obtain disc_nodes_old_document_h2 where disc_nodes_old_document_h2:
      "h2 ⊢ get_disconnected_nodes old_document →r disc_nodes_old_document_h2"
      using old_disc_nodes by blast
    then have disc_nodes_old_document_h3:
      "h3 ⊢ get_disconnected_nodes old_document →r remove1 child disc_nodes_old_document_h2"
      using h3 old_disc_nodes returns_result_eq set_disconnected_nodes_get_disconnected_nodes
      by fastforce
    have "distinct disc_nodes_old_document_h2"
      using disc_nodes_old_document_h2 local.heap_is_wellformed_disconnected_nodes_distinct wellformed_h2
      by blast


    have "type_wf h2"
    proof (insert h2, induct  parent_opt)
      case None
      then show ?case
        using type_wf by simp
    next
```

```
      case (Some option)
      then show ?case
        using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF remove_child_writes]
          type_wf remove_child_types_preserved
        by (simp add: reflp_def transp_def)
    qed
    then have "type_wf h3"
      using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h3]
      using  set_disconnected_nodes_types_preserved
      by(auto simp add: reflp_def transp_def)
    then have "type_wf h'"
      using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h']
      using  set_disconnected_nodes_types_preserved
      by(auto simp add: reflp_def transp_def)

    have disconnected_nodes_eq_h3:
      "⋀doc_ptr disc_nodes. document_ptr ≠ doc_ptr ⟹
h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
      using get_disconnected_nodes_reads set_disconnected_nodes_writes h'
      apply(rule reads_writes_preserved)
      by (simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
    then have disconnected_nodes_eq2_h3:
      "⋀doc_ptr. document_ptr ≠ doc_ptr ⟹
|h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
      using select_result_eq by force
    have disc_nodes_document_ptr_h2:
      "h2 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_document_ptr_h3"
      using disconnected_nodes_eq_h2 docs_neq disc_nodes_document_ptr_h3 by auto
    have disc_nodes_document_ptr_h':
      "h' ⊢ get_disconnected_nodes document_ptr →r child # disc_nodes_document_ptr_h3"
      using h' disc_nodes_document_ptr_h3
      using set_disconnected_nodes_get_disconnected_nodes by blast

    have document_ptr_in_heap: "document_ptr |∈| document_ptr_kinds h2"
      using disc_nodes_document_ptr_h3 document_ptr_kinds_eq2_h2 get_disconnected_nodes_ok assms(1)
      unfolding heap_is_wellformed_def
      using disc_nodes_document_ptr_h2 get_disconnected_nodes_ptr_in_heap by blast
    have old_document_in_heap: "old_document |∈| document_ptr_kinds h2"
      using disc_nodes_old_document_h3 document_ptr_kinds_eq2_h2 get_disconnected_nodes_ok assms(1)
      unfolding heap_is_wellformed_def
      using get_disconnected_nodes_ptr_in_heap old_disc_nodes by blast

    have "child ∈ set disc_nodes_old_document_h2"
    proof (insert parent_opt h2, induct parent_opt)
      case None
      then have "h = h2"
        by(auto)
      moreover have "CD.a_owner_document_valid h"
        using assms(1)  by(simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
      ultimately show ?case
        using old_document disc_nodes_old_document_h2 None(1) child_parent_dual[OF assms(1)]
          in_disconnected_nodes_no_parent assms(1) known_ptrs type_wf by blast
    next
      case (Some option)
      then show ?case
        apply(simp split: option.splits)
        using assms(1) disc_nodes_old_document_h2 old_document remove_child_in_disconnected_nodes known_ptrs
        by blast
    qed
    have "child ∉ set (remove1 child disc_nodes_old_document_h2)"
      using disc_nodes_old_document_h3 h3 known_ptrs wellformed_h2 ‹distinct disc_nodes_old_document_h2›
```

```
      by auto
    have "child ∉ set disc_nodes_document_ptr_h3"
    proof -
      have "CD.a_distinct_lists h2"
        using heap_is_wellformed_def CD.heap_is_wellformed_def wellformed_h2 by blast
      then have 0: "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
|h2 ⊢ document_ptr_kinds_M|_r))"
        by(simp add: CD.a_distinct_lists_def)
      show ?thesis
        using distinct_concat_map_E(1)[OF 0] ‹child ∈ set disc_nodes_old_document_h2›
          disc_nodes_old_document_h2 disc_nodes_document_ptr_h2
        by (meson ‹type_wf h2› docs_neq known_ptrs local.get_owner_document_disconnected_nodes
            local.known_ptrs_preserved object_ptr_kinds_h_eq3 returns_result_eq wellformed_h2)
    qed

    have child_in_heap: "child |∈| node_ptr_kinds h"
      using get_owner_document_ptr_in_heap[OF is_OK_returns_result_I[OF old_document]] node_ptr_kinds_commutes
      by blast
    have "CD.a_acyclic_heap h2"
      using wellformed_h2 by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
    have "parent_child_rel h' ⊆ parent_child_rel h2"
    proof
      fix x
      assume "x ∈ parent_child_rel h'"
      then show "x ∈ parent_child_rel h2"
        using object_ptr_kinds_h2_eq3  object_ptr_kinds_h3_eq3 children_eq2_h2 children_eq2_h3
          mem_Collect_eq object_ptr_kinds_M_eq_h3 select_result_eq split_cong
        unfolding CD.parent_child_rel_def
        by(simp)
    qed
    then have " CD.a_acyclic_heap h'"
      using ‹ CD.a_acyclic_heap h2›  CD.acyclic_heap_def acyclic_subset by blast

    moreover have " CD.a_all_ptrs_in_heap h2"
      using wellformed_h2 by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
    then have "CD.a_all_ptrs_in_heap h3"
      apply(auto simp add: CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq3_h2 children_eq_h2)[1]
       apply (metis CD.l_heap_is_wellformed_axioms ‹type_wf h2› children_eq2_h2 known_ptrs
          l_heap_is_wellformed.heap_is_wellformed_children_in_heap local.get_child_nodes_ok
          local.known_ptrs_known_ptr node_ptr_kinds_eq3_h2 object_ptr_kinds_h2_eq3 object_ptr_kinds_h_eq3
          returns_result_select_result wellformed_h2)
      by (metis (no_types, opaque_lifting) disc_nodes_old_document_h2 disc_nodes_old_document_h3
          disconnected_nodes_eq2_h2 document_ptr_kinds_eq3_h2 select_result_I2 set_remove1_subset
          subsetD)
    then have "CD.a_all_ptrs_in_heap h'"
      by (smt ‹child ∈ set disc_nodes_old_document_h2› children_eq2_h3 disc_nodes_document_ptr_h'
          disc_nodes_document_ptr_h2 disc_nodes_old_document_h2 disconnected_nodes_eq2_h3 document_ptr_kinds_eq3_h3
          local.CD.a_all_ptrs_in_heap_def local.heap_is_wellformed_disc_nodes_in_heap
          node_ptr_kinds_eq3_h2 node_ptr_kinds_eq3_h3 object_ptr_kinds_h3_eq3 select_result_I2 set_ConsD
          subset_code(1) wellformed_h2)

    moreover have "CD.a_owner_document_valid h2"
      using wellformed_h2 by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
    then have "CD.a_owner_document_valid h'"
      apply(simp add: CD.a_owner_document_valid_def node_ptr_kinds_eq_h2 node_ptr_kinds_eq3_h3
          object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 document_ptr_kinds_eq2_h2 document_ptr_kinds_eq2_h3
          children_eq2_h2 children_eq2_h3 )
      by (metis (no_types) disc_nodes_document_ptr_h' disc_nodes_document_ptr_h2
          disc_nodes_old_document_h2 disc_nodes_old_document_h3 disconnected_nodes_eq2_h2
          disconnected_nodes_eq2_h3 document_ptr_in_heap document_ptr_kinds_eq3_h2 document_ptr_kinds_eq3_h3
          in_set_remove1 list.set_intros(1) list.set_intros(2) node_ptr_kinds_eq3_h2 node_ptr_kinds_eq3_h3
          object_ptr_kinds_h2_eq3 object_ptr_kinds_h3_eq3 select_result_I2)
```

```
    have a_distinct_lists_h2: "CD.a_distinct_lists h2"
      using wellformed_h2 by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
    then have "CD.a_distinct_lists h'"
      apply(auto simp add: CD.a_distinct_lists_def object_ptr_kinds_eq_h3 object_ptr_kinds_eq_h2
          children_eq2_h2 children_eq2_h3)[1]
    proof -
      assume 1: "distinct (concat (map (λptr. |h' ⊢ get_child_nodes ptr|_r) (sorted_list_of_set
(fset (object_ptr_kinds h')))))"
        and 2: "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
(sorted_list_of_set (fset (document_ptr_kinds h2)))))"
        and 3: "(⋃x∈fset (object_ptr_kinds h'). set |h' ⊢ get_child_nodes x|_r) ∩
(⋃x∈fset (document_ptr_kinds h2). set |h2 ⊢ get_disconnected_nodes x|_r) = {}"
      show "distinct (concat (map (λdocument_ptr. |h' ⊢ get_disconnected_nodes document_ptr|_r)
(sorted_list_of_set (fset (document_ptr_kinds h')))))"
      proof(rule distinct_concat_map_I)
        show "distinct (sorted_list_of_set (fset (document_ptr_kinds h')))"
          by(auto simp add: document_ptr_kinds_M_def )
      next
        fix x
        assume a1: "x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h')))"
        have 4: "distinct |h2 ⊢ get_disconnected_nodes x|_r"
          using a_distinct_lists_h2 "2" a1 concat_map_all_distinct document_ptr_kinds_eq2_h2
            document_ptr_kinds_eq2_h3
          by fastforce
        then show "distinct |h' ⊢ get_disconnected_nodes x|_r"
        proof (cases "old_document ≠ x")
          case True
          then show ?thesis
          proof (cases "document_ptr ≠ x")
            case True
            then show ?thesis
              using disconnected_nodes_eq2_h2[OF ‹old_document ≠ x›]
                disconnected_nodes_eq2_h3[OF ‹document_ptr ≠ x›] 4
              by(auto)
          next
            case False
            then show ?thesis
              using  disc_nodes_document_ptr_h3 disc_nodes_document_ptr_h' 4
                ‹child ∉ set disc_nodes_document_ptr_h3›
              by(auto simp add: disconnected_nodes_eq2_h2[OF ‹old_document ≠ x›] )
          qed
        next
          case False
          then show ?thesis
            by (metis (no_types, opaque_lifting) ‹distinct disc_nodes_old_document_h2›
                disc_nodes_old_document_h3 disconnected_nodes_eq2_h3 distinct_remove1 docs_neq select_result_I2)
        qed
      next
        fix x y
        assume a0: "x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h')))"
          and a1: "y ∈ set (sorted_list_of_set (fset (document_ptr_kinds h')))"
          and a2: "x ≠ y"

        moreover have 5: "set |h2 ⊢ get_disconnected_nodes x|_r ∩ set |h2 ⊢ get_disconnected_nodes y|_r
= {}"
          using 2 calculation by (auto simp add: document_ptr_kinds_eq3_h2 document_ptr_kinds_eq3_h3
              dest: distinct_concat_map_E(1))
        ultimately show "set |h' ⊢ get_disconnected_nodes x|_r ∩ set |h' ⊢ get_disconnected_nodes y|_r =
{}"
        proof(cases "old_document = x")
          case True
          have "old_document ≠ y"
            using ‹x ≠ y› ‹old_document = x› by simp
```

```
          have "document_ptr ≠ x"
            using docs_neq ‹old_document = x› by auto
          show ?thesis
          proof(cases "document_ptr = y")
            case True
            then show ?thesis
              using 5 True select_result_I2[OF disc_nodes_document_ptr_h']
                select_result_I2[OF disc_nodes_document_ptr_h2]
                select_result_I2[OF disc_nodes_old_document_h2] select_result_I2[OF disc_nodes_old_document_h3]
                ‹old_document = x›
              by (metis (no_types, lifting) ‹child ∉ set (remove1 child disc_nodes_old_document_h2)›
                  ‹document_ptr ≠ x› disconnected_nodes_eq2_h3 disjoint_iff_not_equal notin_set_remove1
set_ConsD)
          next
            case False
            then show ?thesis
              using 5 select_result_I2[OF disc_nodes_document_ptr_h']
                select_result_I2[OF disc_nodes_document_ptr_h2]
                select_result_I2[OF disc_nodes_old_document_h2] select_result_I2[OF disc_nodes_old_document_h3]
                disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3 ‹old_document = x›
                docs_neq ‹old_document ≠ y›
              by (metis (no_types, lifting) disjoint_iff_not_equal notin_set_remove1)
          qed
        next
          case False
          then show ?thesis
          proof(cases "old_document = y")
            case True
            then show ?thesis
            proof(cases "document_ptr = x")
              case True
              show ?thesis
                using 5 select_result_I2[OF disc_nodes_document_ptr_h']
                  select_result_I2[OF disc_nodes_document_ptr_h2]
                  select_result_I2[OF disc_nodes_old_document_h2]
                  select_result_I2[OF disc_nodes_old_document_h3] ‹old_document ≠ x› ‹old_document = y›
‹document_ptr = x›
                apply(simp)
                by (metis (no_types, lifting) ‹child ∉ set (remove1 child disc_nodes_old_document_h2)›
                    disconnected_nodes_eq2_h3 disjoint_iff_not_equal notin_set_remove1)
            next
              case False
              then show ?thesis
                using 5 select_result_I2[OF disc_nodes_document_ptr_h']
                  select_result_I2[OF disc_nodes_document_ptr_h2]
                  select_result_I2[OF disc_nodes_old_document_h2]
                  select_result_I2[OF disc_nodes_old_document_h3] ‹old_document ≠ x› ‹old_document = y›
                  ‹document_ptr ≠ x›
                by (metis (no_types, lifting) disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3
                    disjoint_iff_not_equal docs_neq notin_set_remove1)
            qed
          next
            case False
            have "set |h2 ⊢ get_disconnected_nodes y|_r ∩ set disc_nodes_old_document_h2 = {}"
              by (metis DocumentMonad.ptr_kinds_M_ok DocumentMonad.ptr_kinds_M_ptr_kinds False
                  ‹type_wf h2› a1 disc_nodes_old_document_h2 document_ptr_kinds_M_def document_ptr_kinds_eq2_h2
                  document_ptr_kinds_eq2_h3 l_ptr_kinds_M.ptr_kinds_ptr_kinds_M local.get_disconnected_nodes_ok
                  local.heap_is_wellformed_one_disc_parent returns_result_select_result wellformed_h2)
            then show ?thesis
            proof(cases "document_ptr = x")
              case True
              then have "document_ptr ≠ y"
                using ‹x ≠ y› by auto
```

```
            have "set |h2 ⊢ get_disconnected_nodes y|_r ∩ set disc_nodes_old_document_h2 = {}"
              using ‹set |h2 ⊢ get_disconnected_nodes y|_r ∩ set disc_nodes_old_document_h2 = {}›
              by blast
            then show ?thesis
              using 5 select_result_I2[OF disc_nodes_document_ptr_h']
                select_result_I2[OF disc_nodes_document_ptr_h2]
                select_result_I2[OF disc_nodes_old_document_h2]
                select_result_I2[OF disc_nodes_old_document_h3] ‹old_document ≠ x› ‹old_document ≠ y›
                ‹document_ptr = x› ‹document_ptr ≠ y›
                ‹child ∈ set disc_nodes_old_document_h2› disconnected_nodes_eq2_h2
                disconnected_nodes_eq2_h3 ‹set |h2 ⊢ get_disconnected_nodes y|_r ∩ set disc_nodes_old_document_h2
= {}›
              by(auto)
          next
            case False
            then show ?thesis
            proof(cases "document_ptr = y")
              case True
              have f1: "set |h2 ⊢ get_disconnected_nodes x|_r ∩ set disc_nodes_document_ptr_h3 = {}"
                using 2 a1 document_ptr_in_heap document_ptr_kinds_eq2_h2 document_ptr_kinds_eq2_h3
                  ‹document_ptr ≠ x› select_result_I2[OF disc_nodes_document_ptr_h3, symmetric]
                  disconnected_nodes_eq2_h2[OF docs_neq[symmetric], symmetric]
                by (simp add: "5" True)
              moreover have f1: "set |h2 ⊢ get_disconnected_nodes x|_r ∩ set |h2 ⊢ get_disconnected_nodes
old_document|_r = {}"
                using 2 a1 old_document_in_heap document_ptr_kinds_eq2_h2 document_ptr_kinds_eq2_h3
                  ‹old_document ≠ x›
                by (metis (no_types, lifting) a0 distinct_concat_map_E(1) document_ptr_kinds_eq3_h2
                    document_ptr_kinds_eq3_h3 finite_fset set_sorted_list_of_set)
              ultimately show ?thesis
                using 5 select_result_I2[OF disc_nodes_document_ptr_h']
                  select_result_I2[OF disc_nodes_old_document_h2] ‹old_document ≠ x›
                  ‹document_ptr ≠ x› ‹document_ptr = y›
                  ‹child ∈ set disc_nodes_old_document_h2› disconnected_nodes_eq2_h2
                  disconnected_nodes_eq2_h3
                by auto
            next
              case False
              then show ?thesis
                using 5
                  select_result_I2[OF disc_nodes_old_document_h2] ‹old_document ≠ x›
                  ‹document_ptr ≠ x› ‹document_ptr ≠ y›
                  ‹child ∈ set disc_nodes_old_document_h2› disconnected_nodes_eq2_h2
                  disconnected_nodes_eq2_h3
                by (metis ‹set |h2 ⊢ get_disconnected_nodes y|_r ∩ set disc_nodes_old_document_h2 = {}›
                    empty_iff inf.idem)
            qed
          qed
        qed
      qed
    qed
  next
    fix x xa xb
    assume 0: "distinct (concat (map (λptr. |h' ⊢ get_child_nodes ptr|_r)
(sorted_list_of_set (fset (object_ptr_kinds h'))))"
      and 1: "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
(sorted_list_of_set (fset (document_ptr_kinds h2)))))"
      and 2: "(⋃x∈fset (object_ptr_kinds h'). set |h' ⊢ get_child_nodes x|_r) ∩
(⋃x∈fset (document_ptr_kinds h2). set |h2 ⊢ get_disconnected_nodes x|_r) = {}"
      and 3: "xa |∈| object_ptr_kinds h'"
      and 4: "x ∈ set |h' ⊢ get_child_nodes xa|_r"
      and 5: "xb |∈| document_ptr_kinds h'"
      and 6: "x ∈ set |h' ⊢ get_disconnected_nodes xb|_r"
```

```
    then show False
      using ‹child ∈ set disc_nodes_old_document_h2› disc_nodes_document_ptr_h'
        disc_nodes_document_ptr_h2 disc_nodes_old_document_h2 disc_nodes_old_document_h3
        disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3 document_ptr_kinds_eq2_h2 document_ptr_kinds_eq2_h3
        old_document_in_heap
      apply(auto)[1]
      apply(cases "xb = old_document")
    proof -
      assume a1: "xb = old_document"
      assume a2: "h2 ⊢ get_disconnected_nodes old_document →r disc_nodes_old_document_h2"
      assume a3: "h3 ⊢ get_disconnected_nodes old_document →r remove1 child disc_nodes_old_document_h2"
      assume a4: "x ∈ set |h' ⊢ get_child_nodes xa|r"
      assume "document_ptr_kinds h2 = document_ptr_kinds h'"
      assume a5: "(⋃x∈fset (object_ptr_kinds h'). set |h' ⊢ get_child_nodes x|r) ∩
(⋃x∈fset (document_ptr_kinds h'). set |h2 ⊢ get_disconnected_nodes x|r) = {}"
      have f6: "old_document |∈| document_ptr_kinds h'"
        using a1 ‹xb |∈| document_ptr_kinds h'› by blast
      have f7: "|h2 ⊢ get_disconnected_nodes old_document|r = disc_nodes_old_document_h2"
        using a2 by simp
      have "x ∈ set disc_nodes_old_document_h2"
        using f6 a3 a1 by (metis (no_types) ‹type_wf h'› ‹x ∈ set |h' ⊢ get_disconnected_nodes xb|r›
            disconnected_nodes_eq_h3 docs_neq get_disconnected_nodes_ok returns_result_eq
            returns_result_select_result set_remove1_subset subsetCE)
      then have "set |h' ⊢ get_child_nodes xa|r ∩  set |h2 ⊢ get_disconnected_nodes xb|r = {}"
        using f7 f6 a5 a4 ‹xa |∈| object_ptr_kinds h'›
        by fastforce
      then show ?thesis
        using ‹x ∈ set disc_nodes_old_document_h2› a1 a4 f7 by blast
    next
      assume a1: "xb ≠ old_document"
      assume a2: "h2 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_document_ptr_h3"
      assume a3: "h2 ⊢ get_disconnected_nodes old_document →r disc_nodes_old_document_h2"
      assume a4: "xa |∈| object_ptr_kinds h'"
      assume a5: "h' ⊢ get_disconnected_nodes document_ptr →r child # disc_nodes_document_ptr_h3"
      assume a6: "old_document |∈| document_ptr_kinds h'"
      assume a7: "x ∈ set |h' ⊢ get_disconnected_nodes xb|r"
      assume a8: "x ∈ set |h' ⊢ get_child_nodes xa|r"
      assume a9: "document_ptr_kinds h2 = document_ptr_kinds h'"
      assume a10: "⋀doc_ptr. old_document ≠ doc_ptr ⟹
|h2 ⊢ get_disconnected_nodes doc_ptr|r = |h3 ⊢ get_disconnected_nodes doc_ptr|r"
      assume a11: "⋀doc_ptr. document_ptr ≠ doc_ptr ⟹
|h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
      assume a12: "(⋃x∈fset (object_ptr_kinds h'). set |h' ⊢ get_child_nodes x|r) ∩
(⋃x∈fset (document_ptr_kinds h'). set |h2 ⊢ get_disconnected_nodes x|r) = {}"
      have f13: "⋀d. d ∉ set |h' ⊢ document_ptr_kinds_M|r ∨ h2 ⊢ ok get_disconnected_nodes d"
        using a9 ‹type_wf h2› get_disconnected_nodes_ok
        by simp
      then have f14: "|h2 ⊢ get_disconnected_nodes old_document|r = disc_nodes_old_document_h2"
        using a6 a3 by simp
      have "x ∉ set |h2 ⊢ get_disconnected_nodes xb|r"
        using a12 a8 a4 ‹xb |∈| document_ptr_kinds h'›
        by (meson UN_I disjoint_iff_not_equal)
      then have "x = child"
        using f13 a11 a10 a7 a5 a2 a1
        by (metis (no_types, lifting) select_result_I2 set_ConsD)
      then have "child ∉ set disc_nodes_old_document_h2"
        using f14 a12 a8 a6 a4
        by (metis ‹type_wf h'› adopt_node_removes_child assms(1) assms(2) type_wf
            get_child_nodes_ok known_ptrs local.known_ptrs_known_ptr object_ptr_kinds_h2_eq3
            object_ptr_kinds_h3_eq3 object_ptr_kinds_h_eq3 returns_result_select_result)
      then show ?thesis
        using ‹child ∈ set disc_nodes_old_document_h2› by fastforce
    qed
```

```
    qed
    ultimately have "heap_is_wellformed_Core_DOM h'"
      using ‹type_wf h'› ‹CD.a_owner_document_valid h'› CD.heap_is_wellformed_def by blast



    have shadow_root_eq_h2: "⋀ptr' shadow_root_ptr_opt. h2 ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt
=
h3 ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt"
      using get_shadow_root_reads set_disconnected_nodes_writes h3
      apply(rule reads_writes_preserved)
      by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_shadow_root
          set_disconnected_nodes_get_shadow_root)
    then
    have shadow_root_eq2_h2: "⋀ptr'. |h2 ⊢ get_shadow_root ptr'|r = |h3 ⊢ get_shadow_root ptr'|r"
      by (meson select_result_eq)

    have shadow_root_eq_h3: "⋀ptr' shadow_root_ptr_opt. h3 ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt
=
h' ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt"
      using get_shadow_root_reads set_disconnected_nodes_writes h'
      apply(rule reads_writes_preserved)
      by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_shadow_root
          set_disconnected_nodes_get_shadow_root)
    then
    have shadow_root_eq2_h3: "⋀ptr'. |h3 ⊢ get_shadow_root ptr'|r = |h' ⊢ get_shadow_root ptr'|r"
      by (meson select_result_eq)

    have tag_name_eq_h2: "⋀ptr' tag. h2 ⊢ get_tag_name ptr' →r tag = h3 ⊢ get_tag_name ptr' →r tag"
      using get_tag_name_reads set_disconnected_nodes_writes h3
      apply(rule reads_writes_preserved)
      by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_tag_name
          set_disconnected_nodes_get_tag_name)
    then
    have tag_name_eq2_h2: "⋀ptr'. |h2 ⊢ get_tag_name ptr'|r = |h3 ⊢ get_tag_name ptr'|r"
      by (meson select_result_eq)

    have tag_name_eq_h3: "⋀ptr' tag. h3 ⊢ get_tag_name ptr' →r tag = h' ⊢ get_tag_name ptr' →r tag"
      using get_tag_name_reads set_disconnected_nodes_writes h'
      apply(rule reads_writes_preserved)
      by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_tag_name
          set_disconnected_nodes_get_tag_name)
    then
    have tag_name_eq2_h3: "⋀ptr'. |h3 ⊢ get_tag_name ptr'|r = |h' ⊢ get_tag_name ptr'|r"
      by (meson select_result_eq)

    have object_ptr_kinds_eq_h2: "object_ptr_kinds h2 = object_ptr_kinds h3"
      apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
            OF set_disconnected_nodes_writes h3])
      unfolding adopt_node_locs_def remove_child_locs_def
      using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
      by (auto simp add: reflp_def transp_def  split: if_splits)

    have object_ptr_kinds_eq_h3: "object_ptr_kinds h3 = object_ptr_kinds h'"
      apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
            OF set_disconnected_nodes_writes h'])
      unfolding adopt_node_locs_def remove_child_locs_def
      using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
      by (auto simp add: reflp_def transp_def  split: if_splits)

    have document_ptr_kinds_eq_h2: "document_ptr_kinds h2 = document_ptr_kinds h3"
      using object_ptr_kinds_eq_h2
      by(auto simp add: document_ptr_kinds_def)
```

```
    have shadow_root_ptr_kinds_eq_h2: "shadow_root_ptr_kinds h2 = shadow_root_ptr_kinds h3"
      using object_ptr_kinds_eq_h2
      by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
    have element_ptr_kinds_eq_h2: "element_ptr_kinds h2 = element_ptr_kinds h3"
      using object_ptr_kinds_eq_h2
      by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)

    have document_ptr_kinds_eq_h3: "document_ptr_kinds h3 = document_ptr_kinds h'"
      using object_ptr_kinds_eq_h3
      by(auto simp add: document_ptr_kinds_def)
    have shadow_root_ptr_kinds_eq_h3: "shadow_root_ptr_kinds h3 = shadow_root_ptr_kinds h'"
      using object_ptr_kinds_eq_h3
      by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
    have element_ptr_kinds_eq_h3: "element_ptr_kinds h3 = element_ptr_kinds h'"
      using object_ptr_kinds_eq_h3
      by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)

    have "a_host_shadow_root_rel h' = a_host_shadow_root_rel h3" and
      "a_host_shadow_root_rel h3 = a_host_shadow_root_rel h2"
      by(auto simp add: a_host_shadow_root_rel_def shadow_root_eq2_h2 shadow_root_eq2_h3
          element_ptr_kinds_eq_h2 element_ptr_kinds_eq_h3)
    have "parent_child_rel h' = parent_child_rel h3" and "parent_child_rel h3 = parent_child_rel h2"
      by(auto simp add: CD.parent_child_rel_def children_eq2_h2 children_eq2_h3
          object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3)


    have "parent_child_rel h2 ⊆ parent_child_rel h"
      using h2 parent_opt
    proof (induct parent_opt)
      case None
      then show ?case
        by simp
    next
      case (Some parent)
      then
      have h2: "h ⊢ remove_child parent child →h h2"
        by auto
      have child_nodes_eq_h: "⋀ptr children. parent ≠ ptr ⟹
h ⊢ get_child_nodes ptr →r children = h2 ⊢ get_child_nodes ptr →r children"
        using get_child_nodes_reads remove_child_writes h2
        apply(rule reads_writes_preserved)
        apply(auto simp add: remove_child_locs_def)[1]
        by (simp add: set_child_nodes_get_child_nodes_different_pointers)
      moreover obtain children where children: "h ⊢ get_child_nodes parent →r children"
        using Some local.get_parent_child_dual by blast
      ultimately show ?thesis
        using object_ptr_kinds_eq_h h2
        apply(auto simp add: CD.parent_child_rel_def split: option.splits)[1]
        apply(case_tac "parent = a")
         apply (metis (no_types, lifting) ‹type_wf h3› children_eq2_h2 children_eq_h2 known_ptrs
            local.get_child_nodes_ok local.known_ptrs_known_ptr local.remove_child_children_subset
            object_ptr_kinds_h2_eq3 returns_result_select_result subset_code(1) type_wf)
        apply (metis (no_types, lifting) known_ptrs local.get_child_nodes_ok local.known_ptrs_known_ptr
            returns_result_select_result select_result_I2 type_wf)
        done
    qed

    have "a_host_shadow_root_rel h2 = a_host_shadow_root_rel h"
      using h2
    proof (induct parent_opt)
      case None
      then show ?case
```

```
        by simp
    next
      case (Some parent)
      then
      have h2: "h ⊢ remove_child parent child →h h2"
        by auto
      have "⋀ptr shadow_root. h ⊢ get_shadow_root ptr →r shadow_root = h2 ⊢ get_shadow_root ptr →r shadow_root'
        using get_shadow_root_reads remove_child_writes h2
        apply(rule reads_writes_preserved)
        apply(auto simp add: remove_child_locs_def)[1]
        by (auto simp add: set_disconnected_nodes_get_shadow_root set_child_nodes_get_shadow_root)
      then show ?case
        apply(auto simp add: a_host_shadow_root_rel_def)[1]
         apply (metis (mono_tags, lifting) Collect_cong ‹type_wf h2› case_prodE case_prodI
            l_heap_is_wellformed_Shadow_DOM_defs.a_host_shadow_root_rel_def local.get_shadow_root_ok
            local.a_host_shadow_root_rel_shadow_root returns_result_select_result)
        by (metis (no_types, lifting) Collect_cong case_prodE case_prodI local.get_shadow_root_ok
            local.a_host_shadow_root_rel_def local.a_host_shadow_root_rel_shadow_root returns_result_select_result
type_wf)
    qed

    have "a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h2 - {(cast old_document, cast child)}"
      apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h2 disconnected_nodes_eq2_h2)[1]
      using disconnected_nodes_eq2_h2  disc_nodes_old_document_h2 disc_nodes_old_document_h3
      using ‹distinct disc_nodes_old_document_h2›
        apply (metis (no_types, lifting) ‹child ∉ set (remove1 child disc_nodes_old_document_h2)›
          case_prodI in_set_remove1 mem_Collect_eq pair_imageI select_result_I2)
      using ‹child ∉ set (remove1 child disc_nodes_old_document_h2)› disc_nodes_old_document_h3
       apply auto[1]
      by (metis (no_types, lifting) case_prodI disc_nodes_old_document_h2 disc_nodes_old_document_h3
          disconnected_nodes_eq2_h2 in_set_remove1 mem_Collect_eq pair_imageI select_result_I2)

    have "a_ptr_disconnected_node_rel h3 ⊆ a_ptr_disconnected_node_rel h"
      using h2 parent_opt
    proof (induct parent_opt)
      case None
      then show ?case
        by(auto simp add: ‹a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h2 - {(cast old_document,
cast child)}›)
    next
      case (Some parent)
      then
      have h2: "h ⊢ remove_child parent child →h h2"
        by auto
      then
      obtain children_h h'2 disconnected_nodes_h where
        children_h: "h ⊢ get_child_nodes parent →r children_h" and
        child_in_children_h: "child ∈ set children_h" and
        disconnected_nodes_h: "h ⊢  get_disconnected_nodes old_document →r disconnected_nodes_h" and
        h'2: "h ⊢  set_disconnected_nodes old_document (child # disconnected_nodes_h) →h h'2" and
        h': "h'2 ⊢ set_child_nodes parent (remove1 child children_h) →h h2"
        using old_document
        apply(auto simp add: remove_child_def elim!: bind_returns_heap_E
            dest!: pure_returns_heap_eq[rotated, OF get_owner_document_pure]
            pure_returns_heap_eq[rotated, OF get_child_nodes_pure]
            pure_returns_heap_eq[rotated, OF get_disconnected_nodes_pure] split: if_splits)[1]
        using select_result_I2 by fastforce

      have "|h3 ⊢ document_ptr_kinds_M|r = |h ⊢ document_ptr_kinds_M|r"
        using object_ptr_kinds_eq_h object_ptr_kinds_eq_h2
        by(auto simp add: document_ptr_kinds_def)
      have disconnected_nodes_eq_h: "⋀ptr disc_nodes. old_document ≠ ptr ⟹
h ⊢ get_disconnected_nodes ptr →r disc_nodes = h2 ⊢ get_disconnected_nodes ptr →r disc_nodes"
```

```
        using get_disconnected_nodes_reads remove_child_writes h2
        apply(rule reads_writes_preserved)
        apply(auto simp add: remove_child_locs_def)[1]
        using old_document
        by (auto simp add:set_child_nodes_get_disconnected_nodes set_disconnected_nodes_get_disconnected_nodes_dif
      then
      have foo: "⋀ptr disc_nodes. old_document ≠ ptr ⟹
h ⊢ get_disconnected_nodes ptr →r disc_nodes = h3 ⊢ get_disconnected_nodes ptr →r disc_nodes"
        using disconnected_nodes_eq_h2 by simp
      then
      have foo2: "⋀ptr. old_document ≠ ptr ⟹ |h ⊢ get_disconnected_nodes ptr|r =
|h3 ⊢ get_disconnected_nodes ptr|r"
        by (meson select_result_eq)
      have "h'2 ⊢ get_disconnected_nodes old_document →r child # disconnected_nodes_h"
        using h'2
        using local.set_disconnected_nodes_get_disconnected_nodes by blast

      have "h2 ⊢ get_disconnected_nodes old_document →r child # disconnected_nodes_h"
        using get_disconnected_nodes_reads set_child_nodes_writes h'
          ‹h'2 ⊢ get_disconnected_nodes old_document →r child # disconnected_nodes_h›
        apply(rule reads_writes_separate_forwards)
        using local.set_child_nodes_get_disconnected_nodes by blast
      then have "h3 ⊢ get_disconnected_nodes old_document →r disconnected_nodes_h"
        using h3
        using disc_nodes_old_document_h2 disc_nodes_old_document_h3 returns_result_eq
        by fastforce
      have "a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h"
        using ‹|h3 ⊢ document_ptr_kinds_M|r = |h ⊢ document_ptr_kinds_M|r›
        apply(auto simp add: a_ptr_disconnected_node_rel_def )[1]
         apply(case_tac "old_document = aa")
        using disconnected_nodes_h ‹h3 ⊢ get_disconnected_nodes old_document →r disconnected_nodes_h›
        using foo2
          apply(auto)[1]
        using disconnected_nodes_h ‹h3 ⊢ get_disconnected_nodes old_document →r disconnected_nodes_h›
        using foo2
         apply(auto)[1]
        apply(case_tac "old_document = aa")
        using disconnected_nodes_h ‹h3 ⊢ get_disconnected_nodes old_document →r disconnected_nodes_h›
        using foo2
         apply(auto)[1]
        using disconnected_nodes_h ‹h3 ⊢ get_disconnected_nodes old_document →r disconnected_nodes_h›
        using foo2
        apply(auto)[1]
        done
      then show ?thesis
        by auto
    qed

    have "acyclic (parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2)"
      using local.heap_is_wellformed_def wellformed_h2 by blast
    then have "acyclic (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel
h3)"
      using ‹a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h2 - {(cast old_document, cast
child)}›
      by(auto simp add: ‹parent_child_rel h3 = parent_child_rel h2› ‹a_host_shadow_root_rel h3 = a_host_shadow_roo
h2› elim!: acyclic_subset)
    moreover
    have "a_ptr_disconnected_node_rel h' = insert (cast document_ptr, cast child) (a_ptr_disconnected_node_rel
h3)"
      using disconnected_nodes_eq2_h3[symmetric] disc_nodes_document_ptr_h3 disc_nodes_document_ptr_h' document_ptr
document_ptr_kinds_eq_h2]
      apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h3[symmetric])[1]
       apply(case_tac "document_ptr = aa")
```

```
        apply(auto)[1]
       apply(auto)[1]
      apply(case_tac "document_ptr = aa")
       apply(auto)[1]
      apply(auto)[1]
      done
    moreover have "(cast child, cast document_ptr) ∉ (parent_child_rel h ∪ a_host_shadow_root_rel h ∪
a_ptr_disconnected_node_rel h)*"
      using ‹h ⊢ get_ancestors_di (cast document_ptr) →r ancestors›
        ‹cast child ∉  set ancestors› get_ancestors_di_parent_child_a_host_shadow_root_rel
      using assms(1) known_ptrs type_wf by blast
    moreover have "(cast child, cast document_ptr) ∉ (parent_child_rel h3 ∪ a_host_shadow_root_rel h3
∪ a_ptr_disconnected_node_rel h3)*"
    proof -
      have "(parent_child_rel h3 ∪ local.a_host_shadow_root_rel h3 ∪ local.a_ptr_disconnected_node_rel
h3) ⊆ (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel h)"
        apply(simp add: ‹parent_child_rel h3 = parent_child_rel h2› ‹a_host_shadow_root_rel h3 = a_host_shadow_roo
h2› ‹a_host_shadow_root_rel h2 = a_host_shadow_root_rel h›)
        using ‹local.a_ptr_disconnected_node_rel h3 ⊆ local.a_ptr_disconnected_node_rel h› ‹parent_child_rel
h2 ⊆ parent_child_rel h›
        by blast
      then show ?thesis
        using calculation(3) rtrancl_mono by blast
    qed
    ultimately have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel
h')"
      by(auto simp add: ‹parent_child_rel h' = parent_child_rel h3› ‹a_host_shadow_root_rel h' = a_host_shadow_roo
h3›)

    show "heap_is_wellformed h'"
      using ‹heap_is_wellformed h2›
      using ‹heap_is_wellformed_Core_DOM h'›
      using ‹acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h')›
      apply(auto simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def
          a_all_ptrs_in_heap_def a_distinct_lists_def a_shadow_root_valid_def)[1]
      by(auto simp add: object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 element_ptr_kinds_eq_h2
          element_ptr_kinds_eq_h3 shadow_root_ptr_kinds_eq_h2 shadow_root_ptr_kinds_eq_h3  shadow_root_eq_h2
          shadow_root_eq_h3 shadow_root_eq2_h2 shadow_root_eq2_h3 tag_name_eq_h2 tag_name_eq_h3 tag_name_eq2_h2
          tag_name_eq2_h3)
  qed
qed


lemma adopt_node_node_in_disconnected_nodes:
  assumes wellformed: "heap_is_wellformed h"
    and adopt_node: "h ⊢ adopt_node owner_document node_ptr →h h'"
    and "h' ⊢ get_disconnected_nodes owner_document →r disc_nodes"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
  shows "node_ptr ∈ set disc_nodes"
proof -
  obtain old_document parent_opt h2 where
    old_document: "h ⊢ get_owner_document (cast node_ptr) →r old_document" and
    parent_opt: "h ⊢ get_parent node_ptr →r parent_opt" and
    h2: "h ⊢ (case parent_opt of Some parent ⇒ remove_child parent node_ptr | None ⇒ return ()) →h h2"
and
    h': "h2 ⊢ (if owner_document ≠ old_document then do {
        old_disc_nodes ← get_disconnected_nodes old_document;
        set_disconnected_nodes old_document (remove1 node_ptr old_disc_nodes);
        disc_nodes ← get_disconnected_nodes owner_document;
        set_disconnected_nodes owner_document (node_ptr # disc_nodes)
      } else do {
        return ()
```

```
      }) →h h'"
    using assms(2)[unfolded adopt_node_def CD.adopt_node_def]
    by(auto elim!: bind_returns_heap_E dest!: pure_returns_heap_eq[rotated, OF get_owner_document_pure]
        pure_returns_heap_eq[rotated, OF get_parent_pure] pure_returns_heap_eq[rotated, OF get_ancestors_di_pure]
        split: option.splits if_splits)

  show ?thesis
  proof (cases "owner_document = old_document")
    case True
    then show ?thesis
    proof (insert parent_opt h2, induct parent_opt)
      case None
      then have "h = h'"
        using h2 h' by(auto)
      then show ?case
        using in_disconnected_nodes_no_parent assms None old_document by blast
    next
      case (Some parent)
      then show ?case
        using remove_child_in_disconnected_nodes known_ptrs True h' assms(3) old_document
        by auto
    qed
  next
    case False
    then show ?thesis
      using assms(3)  h' list.set_intros(1) select_result_I2
        set_disconnected_nodes_get_disconnected_nodes
      apply(auto elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated])[1]
    proof -
      fix x  and h'a and xb
      assume a1: "h' ⊢ get_disconnected_nodes owner_document →r disc_nodes"
      assume a2: "⋀h document_ptr disc_nodes h'. h ⊢ set_disconnected_nodes document_ptr disc_nodes →h
h' ⟹
h' ⊢ get_disconnected_nodes document_ptr →r disc_nodes"
      assume "h'a ⊢ set_disconnected_nodes owner_document (node_ptr # xb) →h h'"
      then have "node_ptr # xb = disc_nodes"
        using a2 a1 by (meson returns_result_eq)
      then show ?thesis
        by (meson list.set_intros(1))
    qed
  qed
qed
end

interpretation i_adopt_node_wf_Core_DOM: l_adopt_node_wf_Core_DOM
  get_owner_document get_parent get_parent_locs remove_child remove_child_locs get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs adopt_node_Core_DOM
  adopt_node_locs_Core_DOM known_ptr type_wf get_child_nodes get_child_nodes_locs known_ptrs set_child_nodes
  set_child_nodes_locs remove heap_is_wellformed parent_child_rel
  by(auto simp add: l_adopt_node_wf_Core_DOM_def instances)
declare i_adopt_node_wf_Core_DOM.l_adopt_node_wf_Core_DOM_axioms [instances]


interpretation i_adopt_node_wf?: l_adopt_node_wf_Shadow_DOM
  get_owner_document get_parent get_parent_locs remove_child remove_child_locs get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs get_ancestors_di
  get_ancestors_di_locs adopt_node adopt_node_locs adopt_node_Core_DOM adopt_node_locs_Core_DOM known_ptr
type_wf
  get_child_nodes get_child_nodes_locs known_ptrs set_child_nodes set_child_nodes_locs get_host
  get_host_locs get_disconnected_document get_disconnected_document_locs remove heap_is_wellformed
  parent_child_rel
  by(auto simp add: l_adopt_node_wf_Shadow_DOM_def instances)
```

declare *l_adopt_node_wf$_{Shadow_DOM}$_axioms [instances]*

interpretation *i_adopt_node_wf2?: l_adopt_node_wf2$_{Shadow_DOM}$*
type_wf known_ptr get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
set_child_nodes set_child_nodes_locs get_shadow_root get_shadow_root_locs set_disconnected_nodes
set_disconnected_nodes_locs get_tag_name get_tag_name_locs get_owner_document get_parent get_parent_locs
remove_child remove_child_locs remove known_ptrs heap_is_wellformed parent_child_rel heap_is_wellformed$_{Core_DOM}$
get_host get_host_locs get_disconnected_document get_disconnected_document_locs get_root_node
get_root_node_locs get_ancestors_di get_ancestors_di_locs adopt_node adopt_node_locs adopt_node$_{Core_DOM}$
adopt_node_locs$_{Core_DOM}$
by(*auto simp add: l_adopt_node_wf2$_{Shadow_DOM}$_def instances*)
declare *l_adopt_node_wf2$_{Shadow_DOM}$_axioms [instances]*

lemma *adopt_node_wf_is_l_adopt_node_wf [instances]:*
"l_adopt_node_wf type_wf known_ptr heap_is_wellformed parent_child_rel get_child_nodes get_disconnected_nodes known_ptrs adopt_node"
apply(*auto simp add: l_adopt_node_wf_def l_adopt_node_wf_axioms_def instances*)*[1]*
using *adopt_node_preserves_wellformedness* **apply** *blast*
using *adopt_node_removes_child* **apply** *blast*
using *adopt_node_node_in_disconnected_nodes* **apply** *blast*
using *adopt_node_removes_first_child* **apply** *blast*
using *adopt_node_document_in_heap* **apply** *blast*
done

insert_before

locale *l_insert_before_wf2$_{Shadow_DOM}$ =*
l_get_child_nodes +
l_get_disconnected_nodes +
l_set_child_nodes_get_shadow_root +
l_set_disconnected_nodes_get_shadow_root +
l_set_child_nodes_get_tag_name +
l_set_disconnected_nodes_get_tag_name +
l_set_disconnected_nodes_get_disconnected_nodes +
l_set_child_nodes_get_disconnected_nodes +
l_set_disconnected_nodes_get_disconnected_nodes_wf +

l_insert_before$_{Core_DOM}$ _ _ _ _ _ _ get_ancestors_di get_ancestors_di_locs +

l_get_owner_document +
l_adopt_node$_{Shadow_DOM}$ +
l_adopt_node_wf +
l_heap_is_wellformed$_{Shadow_DOM}$ +
l_adopt_node_get_shadow_root +
l_get_ancestors_di_wf$_{Shadow_DOM}$ +
l_remove_child_wf2
begin
lemma *insert_before_child_preserves:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ insert_before ptr node child $\rightarrow_h$ h'"*
shows *"type_wf h'"* **and** *"known_ptrs h'"* **and** *"heap_is_wellformed h'"*
proof -
obtain *ancestors reference_child owner_document h2 h3 disconnected_nodes_h2* **where**
ancestors: "h ⊢ get_ancestors_di ptr $\rightarrow_r$ ancestors" **and**
node_not_in_ancestors: "cast node $\notin$ set ancestors" **and**
reference_child: "h ⊢ (if Some node = child then a_next_sibling node else return child) $\rightarrow_r$ reference_child"
and
owner_document: "h ⊢ get_owner_document ptr $\rightarrow_r$ owner_document" **and**
h2: "h ⊢ adopt_node owner_document node $\rightarrow_h$ h2" **and**
disconnected_nodes_h2: "h2 ⊢ get_disconnected_nodes owner_document $\rightarrow_r$ disconnected_nodes_h2" **and**
h3: "h2 ⊢ set_disconnected_nodes owner_document (remove1 node disconnected_nodes_h2) $\rightarrow_h$ h3" **and**

```
    h': "h3 ⊢ a_insert_node ptr node reference_child →h h'"
    using assms(4)
    by(auto simp add: insert_before_def a_ensure_pre_insertion_validity_def
        elim!: bind_returns_heap_E bind_returns_result_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_ancestors_pure, rotated]
        bind_returns_heap_E2[rotated, OF next_sibling_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_owner_document_pure, rotated]
        split: if_splits option.splits)

  obtain old_document parent_opt h'2  where

    old_document: "h ⊢ get_owner_document (cast node) →r old_document" and
    parent_opt: "h ⊢ get_parent node →r parent_opt" and
    h'2: "h ⊢ (case parent_opt of Some parent ⇒ remove_child parent node | None ⇒ return ()) →h h'2"
and
    h2': "h'2 ⊢ (if owner_document ≠ old_document then do {
        old_disc_nodes ← get_disconnected_nodes old_document;
        set_disconnected_nodes old_document (remove1 node old_disc_nodes);
        disc_nodes ← get_disconnected_nodes owner_document;
        set_disconnected_nodes owner_document (node # disc_nodes)
      } else do {
        return ()
      }) →h h2"
    using h2
    apply(auto simp add:  adopt_node_def[unfolded CD.adopt_node_def] elim!: bind_returns_heap_E
        dest!: pure_returns_heap_eq[rotated, OF get_ancestors_di_pure])[1]
    apply(split if_splits)
    by(auto simp add:  elim!: bind_returns_heap_E
        dest!: pure_returns_heap_eq[rotated, OF get_owner_document_pure] pure_returns_heap_eq[rotated, OF
get_parent_pure])

  have "type_wf h2"
    using ‹type_wf h›
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF adopt_node_writes h2]
    using adopt_node_types_preserved
    by(auto simp add: reflp_def transp_def)
  then have "type_wf h3"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h3]
    using set_disconnected_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)
  then show "type_wf h'"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF insert_node_writes h']
    using set_child_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  have "object_ptr_kinds h = object_ptr_kinds h2"
    using adopt_node_writes h2
    apply(rule writes_small_big)
    using adopt_node_pointers_preserved
    by(auto simp add: reflp_def transp_def)
  moreover have "... = object_ptr_kinds h3"
    using set_disconnected_nodes_writes h3
    apply(rule writes_small_big)
    using set_disconnected_nodes_pointers_preserved
    by(auto simp add: reflp_def transp_def)
  moreover have "... = object_ptr_kinds h'"
    using insert_node_writes h'
    apply(rule writes_small_big)
    using set_child_nodes_pointers_preserved
```

```
    by(auto simp add: reflp_def transp_def)

  ultimately
  show "known_ptrs h'"
    using ‹known_ptrs h› known_ptrs_preserved
    by blast

  have "known_ptrs h2"
    using ‹known_ptrs h› known_ptrs_preserved ‹object_ptr_kinds h = object_ptr_kinds h2›
    by blast
  then
  have "known_ptrs h3"
    using known_ptrs_preserved ‹object_ptr_kinds h2 = object_ptr_kinds h3›
    by blast

  have "known_ptr ptr"
    by (meson get_owner_document_ptr_in_heap is_OK_returns_result_I ‹known_ptrs h›
        l_known_ptrs.known_ptrs_known_ptr l_known_ptrs_axioms owner_document)

  have object_ptr_kinds_M_eq3_h: "object_ptr_kinds h = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF adopt_node_writes
h2])
    using adopt_node_pointers_preserved
      apply blast
    by (auto simp add: reflp_def transp_def)
  then have object_ptr_kinds_M_eq_h: "⋀ptrs. h ⊢ object_ptr_kinds_M →r ptrs = h2 ⊢ object_ptr_kinds_M
→r ptrs"
    by(simp add: object_ptr_kinds_M_defs )
  then have object_ptr_kinds_M_eq2_h: "|h ⊢ object_ptr_kinds_M|r = |h2 ⊢ object_ptr_kinds_M|r"
    by simp
  then have node_ptr_kinds_eq2_h: "|h ⊢ node_ptr_kinds_M|r = |h2 ⊢ node_ptr_kinds_M|r"
    using node_ptr_kinds_M_eq by blast

  have wellformed_h2: "heap_is_wellformed h2"
    using adopt_node_preserves_wellformedness[OF ‹heap_is_wellformed h› h2] ‹known_ptrs h› ‹type_wf h›
    .

  have object_ptr_kinds_M_eq3_h2: "object_ptr_kinds h2 = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
          OF set_disconnected_nodes_writes h3])
    unfolding a_remove_child_locs_def
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have object_ptr_kinds_M_eq_h2: "⋀ptrs. h2 ⊢ object_ptr_kinds_M →r ptrs = h3 ⊢ object_ptr_kinds_M
→r ptrs"
    by(simp add: object_ptr_kinds_M_defs)
  then have object_ptr_kinds_M_eq2_h2: "|h2 ⊢ object_ptr_kinds_M|r = |h3 ⊢ object_ptr_kinds_M|r"
    by simp
  then have node_ptr_kinds_eq2_h2: "|h2 ⊢ node_ptr_kinds_M|r = |h3 ⊢ node_ptr_kinds_M|r"
    using node_ptr_kinds_M_eq by blast
  have document_ptr_kinds_eq2_h2: "|h2 ⊢ document_ptr_kinds_M|r = |h3 ⊢ document_ptr_kinds_M|r"
    using object_ptr_kinds_M_eq2_h2 document_ptr_kinds_M_eq by auto

  have object_ptr_kinds_M_eq3_h': "object_ptr_kinds h3 = object_ptr_kinds h'"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF insert_node_writes
h'])
    unfolding a_remove_child_locs_def
    using set_child_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have object_ptr_kinds_M_eq_h3: "⋀ptrs. h3 ⊢ object_ptr_kinds_M →r ptrs = h' ⊢ object_ptr_kinds_M
→r ptrs"
    by(simp add: object_ptr_kinds_M_defs)
  then have object_ptr_kinds_M_eq2_h3: "|h3 ⊢ object_ptr_kinds_M|r = |h' ⊢ object_ptr_kinds_M|r"
```

```
    by simp
  then have node_ptr_kinds_eq2_h3: "|h3 ⊢ node_ptr_kinds_M|_r = |h' ⊢ node_ptr_kinds_M|_r"
    using node_ptr_kinds_M_eq by blast
  have document_ptr_kinds_eq2_h3: "|h3 ⊢ document_ptr_kinds_M|_r = |h' ⊢ document_ptr_kinds_M|_r"
    using object_ptr_kinds_M_eq2_h3 document_ptr_kinds_M_eq by auto


  have shadow_root_eq_h2: "⋀ptr' shadow_root. h ⊢ get_shadow_root ptr' →_r shadow_root =
h2 ⊢ get_shadow_root ptr' →_r shadow_root"
    using get_shadow_root_reads adopt_node_writes h2
    apply(rule reads_writes_preserved)
    using local.adopt_node_get_shadow_root by blast

  have disconnected_nodes_eq_h2: "⋀doc_ptr disc_nodes. owner_document ≠ doc_ptr ⟹
h2 ⊢ get_disconnected_nodes doc_ptr →_r disc_nodes = h3 ⊢ get_disconnected_nodes doc_ptr →_r disc_nodes"
    using get_disconnected_nodes_reads set_disconnected_nodes_writes h3
    apply(rule reads_writes_preserved)
    by (auto simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
  then have disconnected_nodes_eq2_h2: "⋀doc_ptr. doc_ptr ≠ owner_document ⟹
|h2 ⊢ get_disconnected_nodes doc_ptr|_r = |h3 ⊢ get_disconnected_nodes doc_ptr|_r"
    using select_result_eq by force
  have disconnected_nodes_h3: "h3 ⊢ get_disconnected_nodes owner_document →_r remove1 node disconnected_nodes_h2"
    using h3 set_disconnected_nodes_get_disconnected_nodes
    by blast

  have disconnected_nodes_eq_h3:
    "⋀doc_ptr disc_nodes. h3 ⊢ get_disconnected_nodes doc_ptr →_r disc_nodes = h' ⊢ get_disconnected_nodes
doc_ptr →_r disc_nodes"
    using get_disconnected_nodes_reads insert_node_writes  h'
    apply(rule reads_writes_preserved)
    using set_child_nodes_get_disconnected_nodes by fast
  then have disconnected_nodes_eq2_h3:
    "⋀doc_ptr. |h3 ⊢ get_disconnected_nodes doc_ptr|_r = |h' ⊢ get_disconnected_nodes doc_ptr|_r"
    using select_result_eq by force

  have children_eq_h2:
    "⋀ptr' children. h2 ⊢ get_child_nodes ptr' →_r children = h3 ⊢ get_child_nodes ptr' →_r children"
    using get_child_nodes_reads set_disconnected_nodes_writes h3
    apply(rule reads_writes_preserved)
    by (auto simp add: set_disconnected_nodes_get_child_nodes)
  then have children_eq2_h2: "⋀ptr'. |h2 ⊢ get_child_nodes ptr'|_r = |h3 ⊢ get_child_nodes ptr'|_r"
    using select_result_eq by force

  have children_eq_h3:
    "⋀ptr' children. ptr ≠ ptr' ⟹ h3 ⊢ get_child_nodes ptr' →_r children = h' ⊢ get_child_nodes ptr'
→_r children"
    using get_child_nodes_reads insert_node_writes h'
    apply(rule reads_writes_preserved)
    by (auto simp add: set_child_nodes_get_child_nodes_different_pointers)
  then have children_eq2_h3: "⋀ptr'. ptr ≠ ptr' ⟹ |h3 ⊢ get_child_nodes ptr'|_r = |h' ⊢ get_child_nodes
ptr'|_r"
    using select_result_eq by force
  obtain children_h3 where children_h3: "h3 ⊢ get_child_nodes ptr →_r children_h3"
    using h' a_insert_node_def by auto
  have children_h': "h' ⊢ get_child_nodes ptr →_r insert_before_list node reference_child children_h3"
    using h' ‹type_wf h3› ‹known_ptr ptr›
    by(auto simp add: a_insert_node_def elim!: bind_returns_heap_E2
        dest!: set_child_nodes_get_child_nodes returns_result_eq[OF children_h3])


  have shadow_root_eq_h: "⋀ptr' shadow_root_ptr_opt. h ⊢ get_shadow_root ptr' →_r shadow_root_ptr_opt =
h2 ⊢ get_shadow_root ptr' →_r shadow_root_ptr_opt"
    using get_shadow_root_reads adopt_node_writes h2
```

```
    apply(rule reads_writes_preserved)
    by(auto simp add: adopt_node_locs_def CD.adopt_node_locs_def CD.remove_child_locs_def
        set_child_nodes_get_shadow_root set_disconnected_nodes_get_shadow_root)
  then
  have shadow_root_eq2_h: "⋀ptr'. |h ⊢ get_shadow_root ptr'|r = |h2 ⊢ get_shadow_root ptr'|r"
    by (meson select_result_eq)

  have shadow_root_eq_h2: "⋀ptr' shadow_root_ptr_opt. h2 ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt
=
h3 ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt"
    using get_shadow_root_reads set_disconnected_nodes_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_shadow_root
        set_disconnected_nodes_get_shadow_root)
  then
  have shadow_root_eq2_h2: "⋀ptr'. |h2 ⊢ get_shadow_root ptr'|r = |h3 ⊢ get_shadow_root ptr'|r"
    by (meson select_result_eq)

  have shadow_root_eq_h3: "⋀ptr' shadow_root_ptr_opt. h3 ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt
=
h' ⊢ get_shadow_root ptr' →r shadow_root_ptr_opt"
    using get_shadow_root_reads insert_node_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_shadow_root
        set_disconnected_nodes_get_shadow_root)
  then
  have shadow_root_eq2_h3: "⋀ptr'. |h3 ⊢ get_shadow_root ptr'|r = |h' ⊢ get_shadow_root ptr'|r"
    by (meson select_result_eq)

  have tag_name_eq_h2: "⋀ptr' tag. h2 ⊢ get_tag_name ptr' →r tag = h3 ⊢ get_tag_name ptr' →r tag"
    using get_tag_name_reads set_disconnected_nodes_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_tag_name
        set_disconnected_nodes_get_tag_name)
  then
  have tag_name_eq2_h2: "⋀ptr'. |h2 ⊢ get_tag_name ptr'|r = |h3 ⊢ get_tag_name ptr'|r"
    by (meson select_result_eq)

  have tag_name_eq_h3: "⋀ptr' tag. h3 ⊢ get_tag_name ptr' →r tag = h' ⊢ get_tag_name ptr' →r tag"
    using get_tag_name_reads insert_node_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_get_tag_name
        set_disconnected_nodes_get_tag_name)
  then
  have tag_name_eq2_h3: "⋀ptr'. |h3 ⊢ get_tag_name ptr'|r = |h' ⊢ get_tag_name ptr'|r"
    by (meson select_result_eq)

  have object_ptr_kinds_eq_hx: "object_ptr_kinds h = object_ptr_kinds h'2"
    using h'2 apply(simp split: option.splits)
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF CD.remove_child_wri
    using CD.remove_child_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  have document_ptr_kinds_eq_hx: "document_ptr_kinds h = document_ptr_kinds h'2"
    using object_ptr_kinds_eq_hx
    by(auto simp add: document_ptr_kinds_def document_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_hx: "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'2"
    using object_ptr_kinds_eq_hx
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  have element_ptr_kinds_eq_hx: "element_ptr_kinds h = element_ptr_kinds h'2"
    using object_ptr_kinds_eq_hx
    by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)

  have object_ptr_kinds_eq_h: "object_ptr_kinds h = object_ptr_kinds h2"
```

```
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF adopt_node_writes
h2])
    unfolding adopt_node_locs_def CD.adopt_node_locs_def CD.remove_child_locs_def
    using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def  split: if_splits)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h = document_ptr_kinds h2"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def document_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_h: "shadow_root_ptr_kinds h = shadow_root_ptr_kinds h2"
    using object_ptr_kinds_eq_h
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  have element_ptr_kinds_eq_h: "element_ptr_kinds h = element_ptr_kinds h2"
    using object_ptr_kinds_eq_h
    by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h2 = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF set_disconnected_no
h3])
    unfolding adopt_node_locs_def remove_child_locs_def
    using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def  split: if_splits)
  have document_ptr_kinds_eq_h2: "document_ptr_kinds h2 = document_ptr_kinds h3"
    using object_ptr_kinds_eq_h2
    by(auto simp add: document_ptr_kinds_def document_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_h2: "shadow_root_ptr_kinds h2 = shadow_root_ptr_kinds h3"
    using object_ptr_kinds_eq_h2
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  have element_ptr_kinds_eq_h2: "element_ptr_kinds h2 = element_ptr_kinds h3"
    using object_ptr_kinds_eq_h2
    by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h3 = object_ptr_kinds h'"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'", OF insert_node_writes
h'])
    using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def  split: if_splits)
  have document_ptr_kinds_eq_h3: "document_ptr_kinds h3 = document_ptr_kinds h'"
    using object_ptr_kinds_eq_h3
    by(auto simp add: document_ptr_kinds_def document_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_h3: "shadow_root_ptr_kinds h3 = shadow_root_ptr_kinds h'"
    using object_ptr_kinds_eq_h3
    by(auto simp add: shadow_root_ptr_kinds_def document_ptr_kinds_def)
  have element_ptr_kinds_eq_h3: "element_ptr_kinds h3 = element_ptr_kinds h'"
    using object_ptr_kinds_eq_h3
    by(auto simp add: element_ptr_kinds_def node_ptr_kinds_def)


  have wellformed_h'2: "heap_is_wellformed h'2"
    using h'2 remove_child_heap_is_wellformed_preserved assms
    by (metis (no_types, lifting) assms(1) option.case_eq_if pure_returns_heap_eq return_pure)

  have "known_ptrs h'2"
    using ‹known_ptrs h› known_ptrs_preserved ‹object_ptr_kinds h = object_ptr_kinds h'2›
    by blast

  have "type_wf h'2"
    using ‹type_wf h› h'2
    apply(auto split: option.splits)[1]
    apply(drule writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF CD.remove_child_writes])
    using CD.remove_child_types_preserved
    by(auto simp add: reflp_def transp_def )
```

```
  have ptr_in_heap: "ptr |∈| object_ptr_kinds h3"
    using children_h3 get_child_nodes_ptr_in_heap by blast
  have node_in_heap: "node |∈| node_ptr_kinds h"
    using h2 adopt_node_child_in_heap by fast
  have child_not_in_any_children: "⋀p children. h2 ⊢ get_child_nodes p →r children ⟹ node ∉ set children"
    using ‹heap_is_wellformed h› h2 adopt_node_removes_child ‹type_wf h› ‹known_ptrs h› by auto
  have "node ∈ set disconnected_nodes_h2"
    using disconnected_nodes_h2 h2 adopt_node_node_in_disconnected_nodes assms(1) ‹type_wf h› ‹known_ptrs
h› by blast
  have node_not_in_disconnected_nodes: "⋀d. d |∈| document_ptr_kinds h3 ⟹ node ∉ set |h3 ⊢ get_disconnected_no
d|r"
  proof -
    fix d
    assume "d |∈| document_ptr_kinds h3"
    show "node ∉ set |h3 ⊢ get_disconnected_nodes d|r"
    proof (cases "d = owner_document")
      case True
      then show ?thesis
        using disconnected_nodes_h2 wellformed_h2 h3 remove_from_disconnected_nodes_removes wellformed_h2
          ‹d |∈| document_ptr_kinds h3› disconnected_nodes_h3
        by fastforce
    next
      case False
      then have "set |h2 ⊢ get_disconnected_nodes d|r ∩ set |h2 ⊢ get_disconnected_nodes owner_document|r
= {}"
        using distinct_concat_map_E(1) wellformed_h2
        by (metis (no_types, lifting) ‹d |∈| document_ptr_kinds h3› ‹type_wf h2› disconnected_nodes_h2
            document_ptr_kinds_M_def document_ptr_kinds_eq2_h2 l_ptr_kinds_M.ptr_kinds_ptr_kinds_M
            local.get_disconnected_nodes_ok local.heap_is_wellformed_one_disc_parent returns_result_select_result
            select_result_I2)
      then show ?thesis
        using disconnected_nodes_eq2_h2[OF False] ‹node ∈ set disconnected_nodes_h2› disconnected_nodes_h2
by fastforce
    qed
  qed

  have "cast node ≠ ptr"
    using ancestors node_not_in_ancestors get_ancestors_ptr
    by fast

  have "a_host_shadow_root_rel h = a_host_shadow_root_rel h2"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h shadow_root_eq2_h)
  have "a_host_shadow_root_rel h2 = a_host_shadow_root_rel h3"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h2 shadow_root_eq2_h2)
  have "a_host_shadow_root_rel h3 = a_host_shadow_root_rel h'"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h3 shadow_root_eq2_h3)

  have "parent_child_rel h2 ⊆ parent_child_rel h"
  proof -
    have "parent_child_rel h'2 ⊆ parent_child_rel h"
      using h'2 parent_opt
    proof (induct parent_opt)
      case None
      then show ?case
        by simp
    next
      case (Some parent)
      then
      have h'2: "h ⊢ remove_child parent node →h h'2"
        by auto
      then
      have "parent |∈| object_ptr_kinds h"
        using CD.remove_child_ptr_in_heap
```

```
        by blast
      have child_nodes_eq_h: "⋀ptr children. parent ≠ ptr ⟹ h ⊢ get_child_nodes ptr →r children =
h'2 ⊢ get_child_nodes ptr →r children"
        using get_child_nodes_reads CD.remove_child_writes h'2
        apply(rule reads_writes_preserved)
        apply(auto simp add: CD.remove_child_locs_def)[1]
        by (simp add: set_child_nodes_get_child_nodes_different_pointers)
      moreover obtain children where children: "h ⊢ get_child_nodes parent →r children"
        using Some local.get_parent_child_dual by blast
      moreover obtain children_h'2 where children_h'2: "h'2 ⊢ get_child_nodes parent →r children_h'2"
        using object_ptr_kinds_eq_hx calculation(2) ‹parent |∈| object_ptr_kinds h› get_child_nodes_ok
        by (metis ‹type_wf h'2› assms(3) is_OK_returns_result_E local.known_ptrs_known_ptr)
      ultimately show ?thesis
        using object_ptr_kinds_eq_h h2
        apply(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_hx split: option.splits)[1]
        apply(case_tac "parent = a")
        using CD.remove_child_children_subset
         apply (metis (no_types, lifting) assms(2) assms(3) contra_subsetD h'2 select_result_I2)
        by (metis select_result_eq)
    qed
    moreover have "parent_child_rel h2 = parent_child_rel h'2"
    proof(cases "owner_document = old_document")
      case True
      then show ?thesis
        using h2' by simp
    next
      case False
      then obtain h'3 old_disc_nodes disc_nodes_document_ptr_h'3 where
        docs_neq: "owner_document ≠ old_document" and
        old_disc_nodes: "h'2 ⊢ get_disconnected_nodes old_document →r old_disc_nodes" and
        h'3: "h'2 ⊢ set_disconnected_nodes old_document (remove1 node old_disc_nodes) →h h'3" and
        disc_nodes_document_ptr_h3: "h'3 ⊢ get_disconnected_nodes owner_document →r disc_nodes_document_ptr_h'3"
and
        h2': "h'3 ⊢ set_disconnected_nodes owner_document (node # disc_nodes_document_ptr_h'3) →h h2"
        using h2'
        by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure,
rotated] )

      have object_ptr_kinds_h'2_eq3: "object_ptr_kinds h'2 = object_ptr_kinds h'3"
        apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
              OF set_disconnected_nodes_writes h'3])
        using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
        by (auto simp add: reflp_def transp_def)
      then have object_ptr_kinds_M_eq_h'2: "⋀ptrs. h'2 ⊢ object_ptr_kinds_M →r ptrs = h'3 ⊢ object_ptr_kinds_M
→r ptrs"
        by(simp add: object_ptr_kinds_M_defs)
      then have object_ptr_kinds_eq_h'2: "|h'2 ⊢ object_ptr_kinds_M|r = |h'3 ⊢ object_ptr_kinds_M|r"
        by(simp)
      then have node_ptr_kinds_eq_h'2: "|h'2 ⊢ node_ptr_kinds_M|r = |h'3 ⊢ node_ptr_kinds_M|r"
        using node_ptr_kinds_M_eq by blast
      then have node_ptr_kinds_eq3_h'2: "node_ptr_kinds h'2 = node_ptr_kinds h'3"
        by auto
      have document_ptr_kinds_eq2_h'2: "|h'2 ⊢ document_ptr_kinds_M|r = |h'3 ⊢ document_ptr_kinds_M|r"
        using object_ptr_kinds_eq_h'2 document_ptr_kinds_M_eq by auto
      then have document_ptr_kinds_eq3_h'2: "document_ptr_kinds h'2 = document_ptr_kinds h'3"
        using object_ptr_kinds_eq_h'2 document_ptr_kinds_M_eq by auto
      have children_eq_h'2: "⋀ptr children. h'2 ⊢ get_child_nodes ptr →r children = h'3 ⊢ get_child_nodes
ptr →r children"
        using get_child_nodes_reads set_disconnected_nodes_writes h'3
        apply(rule reads_writes_preserved)
        by (simp add: set_disconnected_nodes_get_child_nodes)
      then have children_eq2_h'2: "⋀ptr. |h'2 ⊢ get_child_nodes ptr|r = |h'3 ⊢ get_child_nodes ptr|r"
        using select_result_eq by force
```

```
      have object_ptr_kinds_h'3_eq3: "object_ptr_kinds h'3 = object_ptr_kinds h2"
        apply(rule writes_small_big[where P="λh h2. object_ptr_kinds h = object_ptr_kinds h2",
              OF set_disconnected_nodes_writes h2'])
        using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
        by (auto simp add: reflp_def transp_def)
      then have object_ptr_kinds_M_eq_h'3: "⋀ptrs. h'3 ⊢ object_ptr_kinds_M →r ptrs = h2 ⊢ object_ptr_kinds_M
→r ptrs"
        by(simp add: object_ptr_kinds_M_defs)
      then have object_ptr_kinds_eq_h'3: "|h'3 ⊢ object_ptr_kinds_M|r = |h2 ⊢ object_ptr_kinds_M|r"
        by(simp)
      then have node_ptr_kinds_eq_h'3: "|h'3 ⊢ node_ptr_kinds_M|r = |h2 ⊢ node_ptr_kinds_M|r"
        using node_ptr_kinds_M_eq by blast
      then have node_ptr_kinds_eq3_h'3: "node_ptr_kinds h'3 = node_ptr_kinds h2"
        by auto
      have document_ptr_kinds_eq2_h'3: "|h'3 ⊢ document_ptr_kinds_M|r = |h2 ⊢ document_ptr_kinds_M|r"
        using object_ptr_kinds_eq_h'3 document_ptr_kinds_M_eq by auto
      then have document_ptr_kinds_eq3_h'3: "document_ptr_kinds h'3 = document_ptr_kinds h2"
        using object_ptr_kinds_eq_h'3 document_ptr_kinds_M_eq by auto
      have children_eq_h'3: "⋀ptr children. h'3 ⊢ get_child_nodes ptr →r children =
h2 ⊢ get_child_nodes ptr →r children"
        using get_child_nodes_reads set_disconnected_nodes_writes h2'
        apply(rule reads_writes_preserved)
        by (simp add: set_disconnected_nodes_get_child_nodes)
      then have children_eq2_h'3: "⋀ptr. |h'3 ⊢ get_child_nodes ptr|r = |h2 ⊢ get_child_nodes ptr|r"
        using select_result_eq by force

      show ?thesis
        by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_h'2_eq3 object_ptr_kinds_h'3_eq3
            children_eq2_h'3 children_eq2_h'2)
    qed
    ultimately
    show ?thesis
      by simp
  qed

  have "parent_child_rel h2 = parent_child_rel h3"
    by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_M_eq3_h2 children_eq2_h2)
  have "parent_child_rel h' = insert (ptr, cast node) ((parent_child_rel h3))"
    using children_h3 children_h' ptr_in_heap
    apply(auto simp add: CD.parent_child_rel_def object_ptr_kinds_M_eq3_h' children_eq2_h3
        insert_before_list_node_in_set)[1]
     apply (metis (no_types, lifting) children_eq2_h3 insert_before_list_in_set select_result_I2)
    by (metis (no_types, lifting) children_eq2_h3 imageI insert_before_list_in_set select_result_I2)

  have "a_ptr_disconnected_node_rel h3 ⊆ a_ptr_disconnected_node_rel h"
  proof -
    have "a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h2 - {(cast owner_document, cast
node)}"
      apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h2)[1]
         apply(case_tac "aa = owner_document")
      using disconnected_nodes_h2 disconnected_nodes_h3 notin_set_remove1 apply fastforce
      using disconnected_nodes_eq2_h2 apply auto[1]
      using node_not_in_disconnected_nodes apply blast
       apply(case_tac "aa = owner_document")
      using disconnected_nodes_h2 disconnected_nodes_h3 notin_set_remove1 apply fastforce
      using disconnected_nodes_eq2_h2 apply auto[1]
      apply(case_tac "aa = owner_document")
      using disconnected_nodes_h2 disconnected_nodes_h3 notin_set_remove1 apply fastforce
      using disconnected_nodes_eq2_h2 apply auto[1]
      done
    then have "a_ptr_disconnected_node_rel h'2 ⊆ a_ptr_disconnected_node_rel h ∪ {(cast old_document,
cast node)}"
```

```
      using h'2 parent_opt
    proof (induct parent_opt)
      case None
      then show ?case
        by auto
    next
      case (Some parent)
      then
      have h'2: "h ⊢ remove_child parent node →h h'2"
        by auto
      then
      have "parent |∈| object_ptr_kinds h"
        using CD.remove_child_ptr_in_heap
        by blast
      obtain children_h h''2 disconnected_nodes_h where
        owner_document: "h ⊢ get_owner_document (cast node) →r old_document" and
        children_h: "h ⊢ get_child_nodes parent →r children_h" and
        child_in_children_h: "node ∈ set children_h" and
        disconnected_nodes_h: "h ⊢  get_disconnected_nodes old_document →r disconnected_nodes_h" and
        h''2: "h ⊢  set_disconnected_nodes old_document (node # disconnected_nodes_h) →h h''2" and
        h'2: "h''2 ⊢ set_child_nodes parent (remove1 node children_h) →h h'2"
        using h'2 old_document
        apply(auto simp add: CD.remove_child_def elim!: bind_returns_heap_E
            dest!: pure_returns_heap_eq[rotated, OF get_owner_document_pure]
            pure_returns_heap_eq[rotated, OF get_child_nodes_pure] split: if_splits)[1]
        using pure_returns_heap_eq returns_result_eq by fastforce

      have disconnected_nodes_eq: "⋀ptr' disc_nodes. ptr' ≠ old_document ⟹
h ⊢ get_disconnected_nodes ptr' →r disc_nodes = h''2 ⊢ get_disconnected_nodes ptr' →r disc_nodes"
        using local.get_disconnected_nodes_reads set_disconnected_nodes_writes h''2
        apply(rule reads_writes_preserved)
        by (metis local.set_disconnected_nodes_get_disconnected_nodes_different_pointers)
      then
      have disconnected_nodes_eq2: "⋀ptr'. ptr' ≠ old_document ⟹
|h ⊢ get_disconnected_nodes ptr'|r = |h''2 ⊢ get_disconnected_nodes ptr'|r"
        by (meson select_result_eq)
      have "h''2 ⊢ get_disconnected_nodes old_document →r node # disconnected_nodes_h"
        using h''2 local.set_disconnected_nodes_get_disconnected_nodes
        by blast

      have disconnected_nodes_eq_h2:
        "⋀ptr' disc_nodes. h''2 ⊢ get_disconnected_nodes ptr' →r disc_nodes = h'2 ⊢ get_disconnected_nodes
ptr' →r disc_nodes"
        using local.get_disconnected_nodes_reads set_child_nodes_writes h'2
        apply(rule reads_writes_preserved)
        by (metis local.set_child_nodes_get_disconnected_nodes)
      then
      have disconnected_nodes_eq2_h2:
        "⋀ptr'. |h''2 ⊢ get_disconnected_nodes ptr'|r = |h'2 ⊢ get_disconnected_nodes ptr'|r"
        by (meson select_result_eq)
      show ?case
        apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_hx
            ‹⋀ptr'. |h''2 ⊢ get_disconnected_nodes ptr'|r = |h'2 ⊢ get_disconnected_nodes ptr'|r›[symmetric])[1]
         apply(case_tac "aa = old_document")
        using disconnected_nodes_h
          ‹h''2 ⊢ get_disconnected_nodes old_document →r node # disconnected_nodes_h›
          apply(auto)[1]
         apply(auto dest!: disconnected_nodes_eq2)[1]
        apply(case_tac "aa = old_document")
        using disconnected_nodes_h
          ‹h''2 ⊢ get_disconnected_nodes old_document →r node # disconnected_nodes_h›
         apply(auto)[1]
        apply(auto dest!: disconnected_nodes_eq2)[1]
```

```
        done
    qed
    show ?thesis
    proof(cases "owner_document = old_document")
      case True
      then have "a_ptr_disconnected_node_rel h'2 = a_ptr_disconnected_node_rel h2"
        using h2'
        by(auto simp add: a_ptr_disconnected_node_rel_def)
      then
      show ?thesis
        using ‹a_ptr_disconnected_node_rel h3 =
a_ptr_disconnected_node_rel h2 - {(cast owner_document, cast node)}›
        using True ‹local.a_ptr_disconnected_node_rel h'2 ⊆ local.a_ptr_disconnected_node_rel h ∪
{(cast_document_ptr2object_ptr old_document, cast_node_ptr2object_ptr node)}› by auto
    next
      case False
      then obtain h'3 old_disc_nodes disc_nodes_document_ptr_h'3 where
        docs_neq: "owner_document ≠ old_document" and
        old_disc_nodes: "h'2 ⊢ get_disconnected_nodes old_document →_r old_disc_nodes" and
        h'3: "h'2 ⊢ set_disconnected_nodes old_document (remove1 node old_disc_nodes) →_h h'3" and
        disc_nodes_document_ptr_h3: "h'3 ⊢ get_disconnected_nodes owner_document →_r disc_nodes_document_ptr_h'3"
and
        h2': "h'3 ⊢ set_disconnected_nodes owner_document (node # disc_nodes_document_ptr_h'3) →_h h2"
        using h2'
        by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure,
rotated] )

      have object_ptr_kinds_h'2_eq3: "object_ptr_kinds h'2 = object_ptr_kinds h'3"
        apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h = object_ptr_kinds h'",
              OF set_disconnected_nodes_writes h'3])
        using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
        by (auto simp add: reflp_def transp_def)
      then have object_ptr_kinds_M_eq_h'2:
        "⋀ptrs. h'2 ⊢ object_ptr_kinds_M →_r ptrs = h'3 ⊢ object_ptr_kinds_M →_r ptrs"
        by(simp add: object_ptr_kinds_M_defs)
      then have object_ptr_kinds_eq_h'2: "|h'2 ⊢ object_ptr_kinds_M|_r = |h'3 ⊢ object_ptr_kinds_M|_r"
        by(simp)
      then have node_ptr_kinds_eq_h'2: "|h'2 ⊢ node_ptr_kinds_M|_r = |h'3 ⊢ node_ptr_kinds_M|_r"
        using node_ptr_kinds_M_eq by blast
      then have node_ptr_kinds_eq3_h'2: "node_ptr_kinds h'2 = node_ptr_kinds h'3"
        by auto
      have document_ptr_kinds_eq2_h'2: "|h'2 ⊢ document_ptr_kinds_M|_r = |h'3 ⊢ document_ptr_kinds_M|_r"
        using object_ptr_kinds_eq_h'2 document_ptr_kinds_M_eq by auto
      then have document_ptr_kinds_eq3_h'2: "document_ptr_kinds h'2 = document_ptr_kinds h'3"
        using object_ptr_kinds_eq_h'2 document_ptr_kinds_M_eq by auto

      have disconnected_nodes_eq: "⋀ptr' disc_nodes. ptr' ≠ old_document ⟹
h'2 ⊢ get_disconnected_nodes ptr' →_r disc_nodes = h'3 ⊢ get_disconnected_nodes ptr' →_r disc_nodes"
        using local.get_disconnected_nodes_reads set_disconnected_nodes_writes h'3
        apply(rule reads_writes_preserved)
        by (metis local.set_disconnected_nodes_get_disconnected_nodes_different_pointers)
      then
      have disconnected_nodes_eq2: "⋀ptr'. ptr' ≠ old_document ⟹
|h'2 ⊢ get_disconnected_nodes ptr'|_r = |h'3 ⊢ get_disconnected_nodes ptr'|_r"
        by (meson select_result_eq)
      have "h'3 ⊢ get_disconnected_nodes old_document →_r (remove1 node old_disc_nodes)"
        using h'3 local.set_disconnected_nodes_get_disconnected_nodes
        by blast

      have object_ptr_kinds_h'3_eq3: "object_ptr_kinds h'3 = object_ptr_kinds h2"
        apply(rule writes_small_big[where P="λh h2. object_ptr_kinds h = object_ptr_kinds h2",
              OF set_disconnected_nodes_writes h2'])
        using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
```

```
        by (auto simp add: reflp_def transp_def)
      then have object_ptr_kinds_M_eq_h'3: "⋀ptrs. h'3 ⊢ object_ptr_kinds_M →r ptrs = h2 ⊢ object_ptr_kinds_M
→r ptrs"
        by(simp add: object_ptr_kinds_M_defs)
      then have object_ptr_kinds_eq_h'3: "|h'3 ⊢ object_ptr_kinds_M|r = |h2 ⊢ object_ptr_kinds_M|r"
        by(simp)
      then have node_ptr_kinds_eq_h'3: "|h'3 ⊢ node_ptr_kinds_M|r = |h2 ⊢ node_ptr_kinds_M|r"
        using node_ptr_kinds_M_eq by blast
      then have node_ptr_kinds_eq3_h'3: "node_ptr_kinds h'3 = node_ptr_kinds h2"
        by auto
      have document_ptr_kinds_eq2_h'3: "|h'3 ⊢ document_ptr_kinds_M|r = |h2 ⊢ document_ptr_kinds_M|r"
        using object_ptr_kinds_eq_h'3 document_ptr_kinds_M_eq by auto
      then have document_ptr_kinds_eq3_h'3: "document_ptr_kinds h'3 = document_ptr_kinds h2"
        using object_ptr_kinds_eq_h'3 document_ptr_kinds_M_eq by auto
      have disc_nodes_eq_h'3:
        "⋀ptr disc_nodes. h'3 ⊢ get_child_nodes ptr →r disc_nodes = h2 ⊢ get_child_nodes ptr →r disc_nodes"
        using get_child_nodes_reads set_disconnected_nodes_writes h2'
        apply(rule reads_writes_preserved)
        by (simp add: set_disconnected_nodes_get_child_nodes)
      then have disc_nodes_eq2_h'3: "⋀ptr. |h'3 ⊢ get_child_nodes ptr|r = |h2 ⊢ get_child_nodes ptr|r"
        using select_result_eq by force

      have disconnected_nodes_eq: "⋀ptr' disc_nodes. ptr' ≠ owner_document ⟹
h'3 ⊢ get_disconnected_nodes ptr' →r disc_nodes = h2 ⊢ get_disconnected_nodes ptr' →r disc_nodes"
        using local.get_disconnected_nodes_reads set_disconnected_nodes_writes h2'
        apply(rule reads_writes_preserved)
        by (metis local.set_disconnected_nodes_get_disconnected_nodes_different_pointers)
      then
      have disconnected_nodes_eq2': "⋀ptr'. ptr' ≠ owner_document ⟹
|h'3 ⊢ get_disconnected_nodes ptr'|r = |h2 ⊢ get_disconnected_nodes ptr'|r"
        by (meson select_result_eq)
      have "h2 ⊢ get_disconnected_nodes owner_document →r (node # disc_nodes_document_ptr_h'3)"
        using h2' local.set_disconnected_nodes_get_disconnected_nodes
        by blast

      have "a_ptr_disconnected_node_rel h'3 = a_ptr_disconnected_node_rel h'2 - {(cast old_document, cast
node)}"
        apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq2_h'2[simplified])[1]
           apply(case_tac "aa = old_document")
        using ‹h'3 ⊢ get_disconnected_nodes old_document →r (remove1 node old_disc_nodes)›
          notin_set_remove1 old_disc_nodes
            apply fastforce
           apply(auto dest!: disconnected_nodes_eq2)[1]
        using ‹h'3 ⊢ get_disconnected_nodes old_document →r remove1 node old_disc_nodes› h'3
          local.remove_from_disconnected_nodes_removes old_disc_nodes wellformed_h'2
          apply auto[1]
         defer
         apply(case_tac "aa = old_document")
        using ‹h'3 ⊢ get_disconnected_nodes old_document →r (remove1 node old_disc_nodes)›
          notin_set_remove1 old_disc_nodes
          apply fastforce
         apply(auto dest!: disconnected_nodes_eq2)[1]
        apply(case_tac "aa = old_document")
        using ‹h'3 ⊢ get_disconnected_nodes old_document →r (remove1 node old_disc_nodes)›
          notin_set_remove1 old_disc_nodes
         apply fastforce
        apply(auto dest!: disconnected_nodes_eq2)[1]
        done
      moreover
      have "a_ptr_disconnected_node_rel h2 = a_ptr_disconnected_node_rel h'3 ∪
{(cast owner_document, cast node)}"
        apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq2_h'3[simplified])[1]
           apply(case_tac "aa = owner_document")
```

```
            apply(simp)
           apply(auto dest!: disconnected_nodes_eq2')[1]
          apply(case_tac "aa = owner_document")
        using ‹h2 ⊢ get_disconnected_nodes owner_document →r node # disc_nodes_document_ptr_h'3›
          disc_nodes_document_ptr_h3 apply auto[1]
          apply(auto dest!: disconnected_nodes_eq2')[1]
        using ‹node ∈ set disconnected_nodes_h2› disconnected_nodes_h2 local.a_ptr_disconnected_node_rel_def
          local.a_ptr_disconnected_node_rel_disconnected_node apply blast
        defer
        apply(case_tac "aa = owner_document")
        using ‹h2 ⊢ get_disconnected_nodes owner_document →r node # disc_nodes_document_ptr_h'3›
          disc_nodes_document_ptr_h3 apply auto[1]
        apply(auto dest!: disconnected_nodes_eq2')[1]
        done
      ultimately show ?thesis
        using ‹a_ptr_disconnected_node_rel h3 =
a_ptr_disconnected_node_rel h2 - {(cast owner_document, cast node)}›
        using ‹a_ptr_disconnected_node_rel h'2 ⊆
a_ptr_disconnected_node_rel h ∪ {(cast old_document, cast node)}›
        by blast
    qed
  qed

  have "(cast node, ptr) ∉ (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel
h)*"
    using h2
    apply(auto simp add: adopt_node_def elim!: bind_returns_heap_E2 split: if_splits)[1]
    using ancestors assms(1) assms(2) assms(3) local.get_ancestors_di_parent_child_a_host_shadow_root_rel
node_not_in_ancestors
    by blast
  then
  have "(cast node, ptr) ∉ (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel
h3)*"
    apply(simp add: ‹a_host_shadow_root_rel h = a_host_shadow_root_rel h2› ‹a_host_shadow_root_rel h2 =
a_host_shadow_root_rel h3›)
    apply(simp add: ‹parent_child_rel h2 = parent_child_rel h3›[symmetric])
    using ‹parent_child_rel h2 ⊆ parent_child_rel h› ‹a_ptr_disconnected_node_rel h3 ⊆ a_ptr_disconnected_node_re
h›
    by (smt Un_assoc in_rtrancl_UnI sup.orderE sup_left_commute)

  have "CD.a_acyclic_heap h'"
  proof -
    have "acyclic (parent_child_rel h2)"
      using wellformed_h2  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)
    then have "acyclic (parent_child_rel h3)"
      by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_M_eq3_h2 children_eq2_h2)
    moreover have "cast node ∉ {x. (x, ptr) ∈ (parent_child_rel h3)*}"
      by (meson ‹(cast_node_ptr2object_ptr node, ptr) ∉ (parent_child_rel h3 ∪ local.a_host_shadow_root_rel
h3 ∪
local.a_ptr_disconnected_node_rel h3)*› in_rtrancl_UnI mem_Collect_eq)
    ultimately show ?thesis
      using ‹parent_child_rel h' = insert (ptr, cast node) ((parent_child_rel h3))›
      by(auto simp add: CD.acyclic_heap_def)
  qed


  moreover have "CD.a_all_ptrs_in_heap h2"
    using wellformed_h2  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  have "CD.a_all_ptrs_in_heap h'"
  proof -
    have "CD.a_all_ptrs_in_heap h3"
      using ‹CD.a_all_ptrs_in_heap h2›
      apply(auto simp add: CD.a_all_ptrs_in_heap_def object_ptr_kinds_M_eq2_h2 node_ptr_kinds_eq2_h2
```

```
        children_eq_h2)[1]
      apply (metis ‹known_ptrs h3› ‹type_wf h3› children_eq_h2
        l_heap_is_wellformed.heap_is_wellformed_children_in_heap local.get_child_nodes_ok
        local.known_ptrs_known_ptr local.l_heap_is_wellformed_axioms node_ptr_kinds_commutes
        object_ptr_kinds_eq_h2 returns_result_select_result wellformed_h2)
    by (metis (mono_tags, opaque_lifting) disconnected_nodes_eq2_h2 disconnected_nodes_h2
        disconnected_nodes_h3 document_ptr_kinds_eq_h2 node_ptr_kinds_commutes
        object_ptr_kinds_eq_h2 select_result_I2 set_remove1_subset subsetD)
  have "set children_h3 ⊆ set |h' ⊢ node_ptr_kinds_M|_r"
    using children_h3 ‹CD.a_all_ptrs_in_heap h3›
    apply(auto simp add: CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq2_h3)[1]
    using children_eq_h2 local.heap_is_wellformed_children_in_heap node_ptr_kinds_eq2_h2
      node_ptr_kinds_eq2_h3 wellformed_h2 by auto
  then have "set (insert_before_list node reference_child children_h3) ⊆ set |h' ⊢ node_ptr_kinds_M|_r"
    using node_in_heap
    apply(auto simp add: node_ptr_kinds_eq2_h node_ptr_kinds_eq2_h2 node_ptr_kinds_eq2_h3)[1]
    by (metis (no_types, opaque_lifting) contra_subsetD insert_before_list_in_set
        node_ptr_kinds_commutes object_ptr_kinds_M_eq3_h object_ptr_kinds_M_eq3_h' object_ptr_kinds_M_eq3_h2)
  then show ?thesis
    using ‹CD.a_all_ptrs_in_heap h3›
    apply(auto simp add: object_ptr_kinds_M_eq3_h' CD.a_all_ptrs_in_heap_def node_ptr_kinds_def
        node_ptr_kinds_eq2_h3 disconnected_nodes_eq_h3)[1]
      apply (metis (no_types, lifting) children_eq2_h3 children_h' select_result_I2 subsetD)
    by (metis (no_types, lifting) disconnected_nodes_eq2_h3 document_ptr_kinds_eq_h3 in_mono)
qed


moreover have "CD.a_distinct_lists h2"
  using wellformed_h2 by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def )
then have "CD.a_distinct_lists h3"
proof(auto simp add: CD.a_distinct_lists_def object_ptr_kinds_M_eq2_h2 document_ptr_kinds_eq2_h2
    children_eq2_h2 intro!: distinct_concat_map_I)
  fix x
  assume 1: "x |∈| document_ptr_kinds h3"
    and 2: "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
(sorted_list_of_set (fset (document_ptr_kinds h3)))))"
  show "distinct |h3 ⊢ get_disconnected_nodes x|_r"
    using distinct_concat_map_E(2)[OF 2] select_result_I2[OF disconnected_nodes_h3]
      disconnected_nodes_eq2_h2 select_result_I2[OF disconnected_nodes_h2] 1
    by (metis (full_types) distinct_remove1 finite_fset set_sorted_list_of_set)
next
  fix x y xa
  assume 1: "distinct (concat (map (λdocument_ptr. |h2 ⊢ get_disconnected_nodes document_ptr|_r)
(sorted_list_of_set (fset (document_ptr_kinds h3)))))"
    and 2: "x |∈| document_ptr_kinds h3"
    and 3: "y |∈| document_ptr_kinds h3"
    and 4: "x ≠ y"
    and 5: "xa ∈ set |h3 ⊢ get_disconnected_nodes x|_r"
    and 6: "xa ∈ set |h3 ⊢ get_disconnected_nodes y|_r"
  show False
  proof (cases "x = owner_document")
    case True
    then have "y ≠ owner_document"
      using 4 by simp
    show ?thesis
      using distinct_concat_map_E(1)[OF 1]
      using 2 3 4 5 6 select_result_I2[OF disconnected_nodes_h3]  select_result_I2[OF disconnected_nodes_h2]
      apply(auto simp add: True disconnected_nodes_eq2_h2[OF ‹y ≠ owner_document›])[1]
      by (metis (no_types, opaque_lifting) disconnected_nodes_eq2_h2 disjoint_iff_not_equal notin_set_remove1)
  next
    case False
    then show ?thesis
    proof (cases "y = owner_document")
```

```
        case True
        then show ?thesis
          using distinct_concat_map_E(1)[OF 1]
          using 2 3 4 5 6 select_result_I2[OF disconnected_nodes_h3]  select_result_I2[OF disconnected_nodes_h2]
          apply(auto simp add: True disconnected_nodes_eq2_h2[OF ‹x ≠ owner_document›])[1]
          by (metis (no_types, opaque_lifting) disconnected_nodes_eq2_h2 disjoint_iff_not_equal notin_set_remove1)
      next
        case False
        then show ?thesis
          using distinct_concat_map_E(1)[OF 1, simplified, OF 2 3 4] 5 6
          using disconnected_nodes_eq2_h2 disconnected_nodes_h2 disconnected_nodes_h3
            disjoint_iff_not_equal finite_fset notin_set_remove1 select_result_I2
            set_sorted_list_of_set
          by (metis (no_types, lifting))
      qed
    qed
  next
    fix x xa xb
    assume 1: "(⋃x∈fset (object_ptr_kinds h3). set |h3 ⊢ get_child_nodes x|r) ∩
(⋃x∈fset (document_ptr_kinds h3). set |h2 ⊢ get_disconnected_nodes x|r) = {}"
      and 2: "xa |∈| object_ptr_kinds h3"
      and 3: "x ∈ set |h3 ⊢ get_child_nodes xa|r"
      and 4: "xb |∈| document_ptr_kinds h3"
      and 5: "x ∈ set |h3 ⊢ get_disconnected_nodes xb|r"
    have 6: "set |h3 ⊢ get_child_nodes xa|r ∩ set |h2 ⊢ get_disconnected_nodes xb|r = {}"
      using 1 2 4
      by (metis ‹type_wf h2› children_eq2_h2 document_ptr_kinds_commutes ‹known_ptrs h›
          local.get_child_nodes_ok local.get_disconnected_nodes_ok local.heap_is_wellformed_children_disc_nodes_dif
          local.known_ptrs_known_ptr object_ptr_kinds_M_eq3_h object_ptr_kinds_M_eq3_h2 returns_result_select_resul
wellformed_h2)
    show False
    proof (cases "xb = owner_document")
      case True
      then show ?thesis
        using select_result_I2[OF disconnected_nodes_h3,folded select_result_I2[OF disconnected_nodes_h2]]
        by (metis (no_types, lifting) "3" "5" "6" disjoint_iff_not_equal notin_set_remove1)
    next
      case False
      show ?thesis
        using 2 3 4 5 6 unfolding disconnected_nodes_eq2_h2[OF False] by auto
    qed
  qed
  then have "CD.a_distinct_lists h'"
  proof(auto simp add: CD.a_distinct_lists_def document_ptr_kinds_eq2_h3 object_ptr_kinds_M_eq2_h3
      disconnected_nodes_eq2_h3 intro!: distinct_concat_map_I)
    fix x
    assume 1: "distinct (concat (map (λptr. |h3 ⊢ get_child_nodes ptr|r) (sorted_list_of_set (fset (object_ptr_ki
h'))))" and
      2: "x |∈| object_ptr_kinds h'"
    have 3: "⋀p. p |∈| object_ptr_kinds h' ⟹ distinct |h3 ⊢ get_child_nodes p|r"
      using 1  by (auto elim: distinct_concat_map_E)
    show "distinct |h' ⊢ get_child_nodes x|r"
    proof(cases "ptr = x")
      case True
      show ?thesis
        using 3[OF 2] children_h3 children_h'
        by(auto simp add: True  insert_before_list_distinct dest: child_not_in_any_children[unfolded children_eq_h
    next
      case False
      show ?thesis
        using children_eq2_h3[OF False] 3[OF 2] by auto
    qed
  next
```

```
    fix x y xa
    assume 1: "distinct (concat (map (λptr. |h3 ⊢ get_child_nodes ptr|_r) (sorted_list_of_set (fset (object_ptr_ki
h'))))"
      and 2: "x |∈| object_ptr_kinds h'"
      and 3: "y |∈| object_ptr_kinds h'"
      and 4: "x ≠ y"
      and 5: "xa ∈ set |h' ⊢ get_child_nodes x|_r"
      and 6: "xa ∈ set |h' ⊢ get_child_nodes y|_r"
    have 7:"set |h3 ⊢ get_child_nodes x|_r ∩ set |h3 ⊢ get_child_nodes y|_r = {}"
      using distinct_concat_map_E(1)[OF 1] 2 3 4 by auto
    show False
    proof (cases "ptr = x")
      case True
      then have "ptr ≠ y"
        using 4 by simp
      then show ?thesis
        using  children_h3 children_h' child_not_in_any_children[unfolded children_eq_h2] 5 6
        apply(auto simp add:  True children_eq2_h3[OF ‹ptr ≠ y›])[1]
        by (metis (no_types, opaque_lifting) "3" "7" ‹type_wf h3› children_eq2_h3 disjoint_iff_not_equal
            get_child_nodes_ok insert_before_list_in_set ‹known_ptrs h› local.known_ptrs_known_ptr
            object_ptr_kinds_M_eq3_h object_ptr_kinds_M_eq3_h' object_ptr_kinds_M_eq3_h2
            returns_result_select_result select_result_I2)
    next
      case False
      then show ?thesis
      proof (cases "ptr = y")
        case True
        then show ?thesis
          using  children_h3 children_h' child_not_in_any_children[unfolded children_eq_h2] 5 6
          apply(auto simp add:  True children_eq2_h3[OF ‹ptr ≠ x›])[1]
          by (metis (no_types, opaque_lifting) "2" "4" "7" IntI ‹known_ptrs h3› ‹type_wf h'›
              children_eq_h3 empty_iff insert_before_list_in_set local.get_child_nodes_ok local.known_ptrs_known_pt
              object_ptr_kinds_M_eq3_h' returns_result_select_result select_result_I2)
      next
        case False
        then show ?thesis
          using children_eq2_h3[OF ‹ptr ≠ x›] children_eq2_h3[OF ‹ptr ≠ y›] 5 6 7 by auto
      qed
    qed
  next
    fix x xa xb
    assume 1: " (⋃x∈fset (object_ptr_kinds h'). set |h3 ⊢ get_child_nodes x|_r) ∩ (⋃x∈fset (document_ptr_kinds
h'). set |h' ⊢ get_disconnected_nodes x|_r) = {} "
      and 2: "xa |∈| object_ptr_kinds h'"
      and 3: "x ∈ set |h' ⊢ get_child_nodes xa|_r"
      and 4: "xb |∈| document_ptr_kinds h'"
      and 5: "x ∈ set |h' ⊢ get_disconnected_nodes xb|_r"
    have 6: "set |h3 ⊢ get_child_nodes xa|_r ∩ set |h' ⊢ get_disconnected_nodes xb|_r = {}"
      using 1 2 3 4 5
    proof -
      have "∀h d. ¬ type_wf h ∨ d |∉| document_ptr_kinds h ∨ h ⊢ ok get_disconnected_nodes d"
        using local.get_disconnected_nodes_ok by satx
      then have "h' ⊢ ok get_disconnected_nodes xb"
        using "4" ‹type_wf h'› by fastforce
      then have f1: "h3 ⊢ get_disconnected_nodes xb →_r |h' ⊢ get_disconnected_nodes xb|_r"
        by (simp add: disconnected_nodes_eq_h3)
      have "xa |∈| object_ptr_kinds h3"
        using "2" object_ptr_kinds_M_eq3_h' by blast
      then show ?thesis
        using f1 ‹local.CD.a_distinct_lists h3› CD.distinct_lists_no_parent by fastforce
    qed
    show False
    proof (cases "ptr = xa")
```

```
      case True
      show ?thesis
        using 6 node_not_in_disconnected_nodes  3 4 5  select_result_I2[OF children_h']
          select_result_I2[OF children_h3] True disconnected_nodes_eq2_h3
        by (metis (no_types, lifting) "2" DocumentMonad.ptr_kinds_ptr_kinds_M ‹CD.a_distinct_lists h3›
            ‹type_wf h'› disconnected_nodes_eq_h3 CD.distinct_lists_no_parent document_ptr_kinds_eq2_h3
            get_disconnected_nodes_ok insert_before_list_in_set object_ptr_kinds_M_eq3_h' returns_result_select_res

    next
      case False
      then show ?thesis
        using 1 2 3 4 5 children_eq2_h3[OF False] by fastforce
    qed
  qed

  moreover have "CD.a_owner_document_valid h2"
    using wellformed_h2  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_owner_document_valid h'"
    apply(auto simp add: CD.a_owner_document_valid_def object_ptr_kinds_M_eq2_h2 object_ptr_kinds_M_eq2_h3
        node_ptr_kinds_eq2_h2 node_ptr_kinds_eq2_h3 document_ptr_kinds_eq2_h2 document_ptr_kinds_eq2_h3
children_eq2_h2 )[1]
    apply(auto simp add: document_ptr_kinds_eq2_h2[simplified]  document_ptr_kinds_eq2_h3[simplified]
        object_ptr_kinds_M_eq2_h2[simplified] object_ptr_kinds_M_eq2_h3[simplified] node_ptr_kinds_eq2_h2[simplifie
        node_ptr_kinds_eq2_h3[simplified])[1]
    apply(auto simp add: disconnected_nodes_eq2_h3[symmetric])[1]
    by (smt Core_DOM_Functions.i_insert_before.insert_before_list_in_set children_eq2_h3 children_h'
        children_h3 disconnected_nodes_eq2_h2 disconnected_nodes_h2 disconnected_nodes_h3 in_set_remove1
        object_ptr_kinds_eq_h3 ptr_in_heap select_result_I2)

  ultimately have "heap_is_wellformed_Core_DOM h'"
    by (simp add: CD.heap_is_wellformed_def)

  have "a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h2 - {(cast owner_document, cast node)}"
    apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h2 disconnected_nodes_eq2_h2)[1]
      apply(case_tac "aa = owner_document")
       apply (metis (no_types, lifting) case_prodI disconnected_nodes_h2 disconnected_nodes_h3
        in_set_remove1 mem_Collect_eq node_not_in_disconnected_nodes pair_imageI select_result_I2)
    using disconnected_nodes_eq2_h2 apply auto[1]
    using node_not_in_disconnected_nodes apply blast
    by (metis (no_types, lifting) case_prodI disconnected_nodes_eq2_h2 disconnected_nodes_h2
        disconnected_nodes_h3 in_set_remove1 mem_Collect_eq pair_imageI select_result_I2)
  have "a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h'"
    by(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h3 disconnected_nodes_eq2_h3)

  have "acyclic (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)"
    using ‹heap_is_wellformed h2›
    by(auto simp add: ‹a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h2 - {(cast owner_document,
cast node)}›
        heap_is_wellformed_def ‹parent_child_rel h2 = parent_child_rel h3› ‹a_host_shadow_root_rel h2 =
a_host_shadow_root_rel h3› elim!: acyclic_subset)
  then
  have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ local.a_ptr_disconnected_node_rel h')"
    using ‹(cast node, ptr) ∉ (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel
h3)*›
    by(auto simp add: ‹a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h'› ‹a_host_shadow_root_rel
h3 =
a_host_shadow_root_rel h'› ‹parent_child_rel h' = insert (ptr, cast node) ((parent_child_rel h3))›)
  then
  show "heap_is_wellformed h'"
    using ‹heap_is_wellformed h2›
    using ‹heap_is_wellformed_Core_DOM h'›
    apply(auto simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def
        a_all_ptrs_in_heap_def a_distinct_lists_def a_shadow_root_valid_def)[1]
```

```
    by(auto simp add: object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 element_ptr_kinds_eq_h2
        element_ptr_kinds_eq_h3 shadow_root_ptr_kinds_eq_h2 shadow_root_ptr_kinds_eq_h3  shadow_root_eq_h2
        shadow_root_eq_h3 shadow_root_eq2_h2 shadow_root_eq2_h3 tag_name_eq_h2 tag_name_eq_h3 tag_name_eq2_h2
        tag_name_eq2_h3)

qed
end

interpretation i_insert_before_wf?: l_insert_before_wf_Core_DOM get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs set_child_nodes set_child_nodes_locs get_ancestors_di get_ancestors_di_locs adopt_node
  adopt_node_locs set_disconnected_nodes set_disconnected_nodes_locs get_disconnected_nodes get_disconnected_nodes_
  get_owner_document insert_before insert_before_locs append_child type_wf known_ptr known_ptrs heap_is_wellformed
  parent_child_rel
  by(simp add: l_insert_before_wf_Core_DOM_def instances)
declare l_insert_before_wf_Core_DOM_axioms [instances]

lemma insert_before_wf_is_l_insert_before_wf [instances]: "l_insert_before_wf Shadow_DOM.heap_is_wellformed
ShadowRootClass.type_wf ShadowRootClass.known_ptr ShadowRootClass.known_ptrs
     Shadow_DOM.insert_before Shadow_DOM.get_child_nodes"
  apply(auto simp add: l_insert_before_wf_def l_insert_before_wf_axioms_def instances)[1]
  using insert_before_removes_child apply fast
  done

lemma l_set_disconnected_nodes_get_disconnected_nodes_wf [instances]: "l_set_disconnected_nodes_get_disconnected_
ShadowRootClass.type_wf
ShadowRootClass.known_ptr Shadow_DOM.heap_is_wellformed Shadow_DOM.parent_child_rel Shadow_DOM.get_child_nodes
     get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs"
  apply(auto simp add: l_set_disconnected_nodes_get_disconnected_nodes_wf_def l_set_disconnected_nodes_get_disconn
instances)[1]
  by (metis Diff_iff Shadow_DOM.i_heap_is_wellformed.heap_is_wellformed_disconnected_nodes_distinct Shadow_DOM.i_r
insert_iff returns_result_eq set_remove1_eq)

interpretation i_insert_before_wf2?: l_insert_before_wf2_Shadow_DOM
  type_wf known_ptr get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  set_child_nodes set_child_nodes_locs get_shadow_root get_shadow_root_locs set_disconnected_nodes
  set_disconnected_nodes_locs get_tag_name get_tag_name_locs heap_is_wellformed parent_child_rel get_parent
  get_parent_locs adopt_node adopt_node_locs get_owner_document insert_before insert_before_locs append_child
  known_ptrs remove_child remove_child_locs get_ancestors_di get_ancestors_di_locs adopt_node_Core_DOM
  adopt_node_locs_Core_DOM get_host get_host_locs get_disconnected_document get_disconnected_document_locs
  remove heap_is_wellformed_Core_DOM
  by(auto simp add: l_insert_before_wf2_Shadow_DOM_def instances)
declare l_insert_before_wf2_Shadow_DOM_axioms [instances]


lemma insert_before_wf2_is_l_insert_before_wf2 [instances]:
  "l_insert_before_wf2 ShadowRootClass.type_wf ShadowRootClass.known_ptr ShadowRootClass.known_ptrs Shadow_DOM.inse
     Shadow_DOM.heap_is_wellformed"
  apply(auto simp add: l_insert_before_wf2_def l_insert_before_wf2_axioms_def instances)[1]
  using insert_before_child_preserves apply(fast, fast, fast)
  done
```

append_child

```
locale l_append_child_wf_Shadow_DOM =
  l_insert_before_wf2_Shadow_DOM +
  l_append_child_Core_DOM
begin

lemma append_child_heap_is_wellformed_preserved:
  assumes wellformed: "heap_is_wellformed h"
    and append_child: "h ⊢ append_child ptr node →_h h'"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
```

```
  shows "heap_is_wellformed h'" and "type_wf h'" and "known_ptrs h'"
  using assms
  by(auto simp add: append_child_def intro: insert_before_child_preserves)

lemma append_child_children:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_child_nodes ptr →r xs"
  assumes "h ⊢ append_child ptr node →h h'"
  assumes "node ∉ set xs"
  shows "h' ⊢ get_child_nodes ptr →r xs @ [node]"
proof -

  obtain ancestors owner_document h2 h3 disconnected_nodes_h2 where
    ancestors: "h ⊢ get_ancestors_di ptr →r ancestors" and
    node_not_in_ancestors: "cast node ∉ set ancestors" and
    owner_document: "h ⊢ get_owner_document ptr →r owner_document" and
    h2: "h ⊢ adopt_node owner_document node →h h2" and
    disconnected_nodes_h2: "h2 ⊢ get_disconnected_nodes owner_document →r disconnected_nodes_h2" and
    h3: "h2 ⊢ set_disconnected_nodes owner_document (remove1 node disconnected_nodes_h2) →h h3" and
    h': "h3 ⊢ a_insert_node ptr node None →h h'"
    using assms(5)
    by(auto simp add: append_child_def insert_before_def a_ensure_pre_insertion_validity_def
        elim!: bind_returns_heap_E bind_returns_result_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_ancestors_pure, rotated]
        bind_returns_heap_E2[rotated, OF next_sibling_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_owner_document_pure, rotated]
        split: if_splits option.splits)

  have "⋀parent. |h ⊢ get_parent node|r = Some parent ⟹ parent ≠ ptr"
    using assms(1) assms(4) assms(6)
    by (metis (no_types, lifting) assms(2) assms(3) h2 is_OK_returns_heap_I is_OK_returns_result_E
        local.adopt_node_child_in_heap local.get_parent_child_dual local.get_parent_ok
        select_result_I2)
  have "h2 ⊢ get_child_nodes ptr →r xs"
    using get_child_nodes_reads adopt_node_writes h2 assms(4)
    apply(rule reads_writes_separate_forwards)
    using ‹⋀parent. |h ⊢ get_parent node|r = Some parent ⟹ parent ≠ ptr›
    apply(auto simp add: adopt_node_locs_def CD.adopt_node_locs_def CD.remove_child_locs_def)[1]
    by (meson local.set_child_nodes_get_child_nodes_different_pointers)

  have "h3 ⊢ get_child_nodes ptr →r xs"
    using get_child_nodes_reads set_disconnected_nodes_writes h3 ‹h2 ⊢ get_child_nodes ptr →r xs›
    apply(rule reads_writes_separate_forwards)
    by(auto)

  have "ptr |∈| object_ptr_kinds h"
    by (meson ancestors is_OK_returns_result_I local.get_ancestors_ptr_in_heap)
  then
  have "known_ptr ptr"
    using assms(3)
    using local.known_ptrs_known_ptr by blast

  have "type_wf h2"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF adopt_node_writes h2]
    using adopt_node_types_preserved ‹type_wf h›
    by(auto simp add: adopt_node_locs_def remove_child_locs_def reflp_def transp_def split: if_splits)
  then
  have "type_wf h3"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h3]
```

```
    using  set_disconnected_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  show "h' ⊢ get_child_nodes ptr →r xs@[node]"
    using h'
    apply(auto simp add: a_insert_node_def
        dest!: bind_returns_heap_E3[rotated, OF ‹h3 ⊢ get_child_nodes ptr →r xs›
          get_child_nodes_pure, rotated])[1]
    using ‹type_wf h3› set_child_nodes_get_child_nodes ‹known_ptr ptr›
    by metis
qed

lemma append_child_for_all_on_children:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_child_nodes ptr →r xs"
  assumes "h ⊢ forall_M (append_child ptr) nodes →h h'"
  assumes "set nodes ∩ set xs = {}"
  assumes "distinct nodes"
  shows "h' ⊢ get_child_nodes ptr →r xs@nodes"
  using assms
  apply(induct nodes arbitrary: h xs)
   apply(simp)
proof(auto elim!: bind_returns_heap_E)[1]fix a nodes h xs h'a
  assume 0: "(⋀h xs. heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h
              ⟹ h ⊢ get_child_nodes ptr →r xs ⟹ h ⊢ forall_M (append_child ptr) nodes →h h'
              ⟹ set nodes ∩ set xs = {} ⟹ h' ⊢ get_child_nodes ptr →r xs @ nodes)"
    and 1: "heap_is_wellformed h"
    and 2: "type_wf h"
    and 3: "known_ptrs h"
    and 4: "h ⊢ get_child_nodes ptr →r xs"
    and 5: "h ⊢ append_child ptr a →r ()"
    and 6: "h ⊢ append_child ptr a →h h'a"
    and 7: "h'a ⊢ forall_M (append_child ptr) nodes →h h'"
    and 8: "a ∉ set xs"
    and 9: "set nodes ∩ set xs = {}"
    and 10: "a ∉ set nodes"
    and 11: "distinct nodes"
  then have "h'a ⊢ get_child_nodes ptr →r xs @ [a]"
    using append_child_children 6
    using "1" "2" "3" "4" "8" by blast

  moreover have "heap_is_wellformed h'a" and "type_wf h'a" and "known_ptrs h'a"
    using insert_before_child_preserves 1 2 3 6 append_child_def
    by metis+
  moreover have "set nodes ∩ set (xs @ [a]) = {}"
    using 9 10
    by auto
  ultimately show "h' ⊢ get_child_nodes ptr →r xs @ a # nodes"
    using 0 7
    by fastforce
qed


lemma append_child_for_all_on_no_children:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_child_nodes ptr →r []"
  assumes "h ⊢ forall_M (append_child ptr) nodes →h h'"
  assumes "distinct nodes"
  shows "h' ⊢ get_child_nodes ptr →r nodes"
  using assms append_child_for_all_on_children
  by force
end
```

interpretation *i_append_child_wf?: l_append_child_wf*$_{Shadow_DOM}$
type_wf known_ptr get_child_nodes get_child_nodes_locs get_disconnected_nodes
get_disconnected_nodes_locs set_child_nodes set_child_nodes_locs get_shadow_root get_shadow_root_locs
set_disconnected_nodes set_disconnected_nodes_locs get_tag_name get_tag_name_locs heap_is_wellformed
parent_child_rel get_parent get_parent_locs adopt_node adopt_node_locs get_owner_document insert_before
insert_before_locs append_child known_ptrs remove_child remove_child_locs get_ancestors_di
get_ancestors_di_locs adopt_node$_{Core_DOM}$ *adopt_node_locs*$_{Core_DOM}$ *get_host get_host_locs get_disconnected_docum*
get_disconnected_document_locs remove heap_is_wellformed$_{Core_DOM}$
by(*auto simp add: l_append_child_wf*$_{Shadow_DOM}$*_def instances*)
declare *l_append_child_wf*$_{Shadow_DOM}$*_axioms [instances]*

lemma *append_child_wf_is_l_append_child_wf [instances]:*
"l_append_child_wf type_wf known_ptr known_ptrs append_child heap_is_wellformed"
apply(*auto simp add: l_append_child_wf_def l_append_child_wf_axioms_def instances*)*[1]*
using *append_child_heap_is_wellformed_preserved* **by** *fast+*

to_tree_order

interpretation *i_to_tree_order_wf?: l_to_tree_order_wf*$_{Core_DOM}$ *known_ptr type_wf get_child_nodes*
get_child_nodes_locs to_tree_order known_ptrs get_parent get_parent_locs heap_is_wellformed
parent_child_rel get_disconnected_nodes get_disconnected_nodes_locs
apply(*auto simp add: l_to_tree_order_wf*$_{Core_DOM}$*_def instances*)*[1]*
done
declare *l_to_tree_order_wf*$_{Core_DOM}$*_axioms [instances]*

lemma *to_tree_order_wf_is_l_to_tree_order_wf [instances]:*
"l_to_tree_order_wf heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
to_tree_order get_parent get_child_nodes"
apply(*auto simp add: l_to_tree_order_wf_def l_to_tree_order_wf_axioms_def instances*)*[1]*
using *to_tree_order_ok* **apply** *fast*
using *to_tree_order_ptrs_in_heap* **apply** *fast*
using *to_tree_order_parent_child_rel* **apply**(*fast, fast*)
using *to_tree_order_child2* **apply** *blast*
using *to_tree_order_node_ptrs* **apply** *fast*
using *to_tree_order_child* **apply** *fast*
using *to_tree_order_ptr_in_result* **apply** *fast*
using *to_tree_order_parent* **apply** *fast*
using *to_tree_order_subset* **apply** *fast*
done

get_root_node

interpretation *i_to_tree_order_wf_get_root_node_wf?: l_to_tree_order_wf_get_root_node_wf*$_{Core_DOM}$
known_ptr type_wf known_ptrs heap_is_wellformed parent_child_rel get_child_nodes get_child_nodes_locs
get_disconnected_nodes get_disconnected_nodes_locs get_parent get_parent_locs get_ancestors
get_ancestors_locs get_root_node get_root_node_locs to_tree_order
by(*auto simp add: l_to_tree_order_wf_get_root_node_wf*$_{Core_DOM}$*_def instances*)
declare *l_to_tree_order_wf_get_root_node_wf*$_{Core_DOM}$*_axioms [instances]*

lemma *to_tree_order_wf_get_root_node_wf_is_l_to_tree_order_wf_get_root_node_wf [instances]:*
"l_to_tree_order_wf_get_root_node_wf ShadowRootClass.type_wf ShadowRootClass.known_ptr
ShadowRootClass.known_ptrs to_tree_order Shadow_DOM.get_root_node
Shadow_DOM.heap_is_wellformed"
apply(*auto simp add: l_to_tree_order_wf_get_root_node_wf_def*
l_to_tree_order_wf_get_root_node_wf_axioms_def instances)*[1]*
using *to_tree_order_get_root_node* **apply** *fast*
using *to_tree_order_same_root* **apply** *fast*
done

to_tree_order_si

locale *l_to_tree_order_si_wf*$_{Shadow_DOM}$ =
l_get_child_nodes +
l_get_parent_get_host_get_disconnected_document_wf$_{Shadow_DOM}$ *+*
l_to_tree_order_si$_{Shadow_DOM}$

```
begin
lemma to_tree_order_si_ok:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
    and "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (to_tree_order_si ptr)"
proof(insert assms(1) assms(4), induct rule: heap_wellformed_induct_si)
  case (step parent)
  have "known_ptr parent"
    using assms(2) local.known_ptrs_known_ptr step.prems
    by blast
  then show ?case
    using step
    using assms(1) assms(2) assms(3)
    using local.heap_is_wellformed_children_in_heap local.get_shadow_root_shadow_root_ptr_in_heap
    by(auto simp add: to_tree_order_si_def[of parent] intro: get_child_nodes_ok get_shadow_root_ok
        intro!: bind_is_OK_pure_I map_M_pure_I bind_pure_I map_M_ok_I split: option.splits)
qed
end
interpretation i_to_tree_order_si_wf?: l_to_tree_order_si_wf_Shadow_DOM
  type_wf known_ptr get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs heap_is_wellforme
parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document get_disconnected_doc
known_ptrs get_parent get_parent_locs to_tree_order_si
  by(auto simp add: l_to_tree_order_si_wf_Shadow_DOM_def instances)
declare l_to_tree_order_si_wf_Shadow_DOM_axioms [instances]
```

get_assigned_nodes

```
lemma forall_M_small_big: "h ⊢ forall_M f xs →h h' ⟹ P h ⟹
(⋀h h' x. x ∈ set xs ⟹ h ⊢ f x →h h' ⟹ P h ⟹ P h') ⟹ P h'"
  by(induct xs arbitrary: h) (auto elim!: bind_returns_heap_E)

locale l_assigned_nodes_wf_Shadow_DOM =
  l_assigned_nodes_Shadow_DOM +
  l_heap_is_wellformed +
  l_remove_child_wf2 +
  l_append_child_wf +
  l_remove_shadow_root_wf_Shadow_DOM
begin

lemma assigned_nodes_distinct:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ assigned_nodes slot →r nodes"
  shows "distinct nodes"
proof -
  have "⋀ptr children. h ⊢ get_child_nodes ptr →r children ⟹ distinct children"
    using assms(1) local.heap_is_wellformed_children_distinct by blast
  then show ?thesis
    using assms
    apply(auto simp add: assigned_nodes_def elim!: bind_returns_result_E2 split: if_splits)[1]
    by (simp add: filter_M_distinct)
qed


lemma flatten_dom_preserves:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
  assumes "h ⊢ flatten_dom →h h'"
  shows "heap_is_wellformed h'" and "known_ptrs h'" and "type_wf h'"
proof -
  obtain tups h2 element_ptrs shadow_root_ptrs where
    "h ⊢ element_ptr_kinds_M →r element_ptrs" and
    tups: "h ⊢ map_filter_M2 (λelement_ptr. do {
      tag ← get_tag_name element_ptr;
```

```
      assigned_nodes ← assigned_nodes element_ptr;
      (if tag = ''slot'' ∧ assigned_nodes ≠ []
then return (Some (element_ptr, assigned_nodes)) else return None)}) element_ptrs →r tups"
    (is "h ⊢ map_filter_M2 ?f element_ptrs →r tups") and
    h2: "h ⊢ forall_M (λ(slot, assigned_nodes). do {
      get_child_nodes (cast slot) ≫= forall_M remove;
      forall_M (append_child (cast slot)) assigned_nodes
    }) tups →h h2" and
    "h2 ⊢ shadow_root_ptr_kinds_M →r shadow_root_ptrs" and
    h': "h2 ⊢ forall_M (λshadow_root_ptr. do {
      host ← get_host shadow_root_ptr;
      get_child_nodes (cast host) ≫= forall_M remove;
      get_child_nodes (cast shadow_root_ptr) ≫= forall_M (append_child (cast host));
      remove_shadow_root host
    }) shadow_root_ptrs →h h'"
    using ‹h ⊢ flatten_dom →h h'›
    apply(auto simp add: flatten_dom_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF ElementMonad.ptr_kinds_M_pure, rotated]
        bind_returns_heap_E2[rotated, OF ShadowRootMonad.ptr_kinds_M_pure, rotated])[1]
    apply(drule pure_returns_heap_eq)
    by(auto intro!: map_filter_M2_pure bind_pure_I)
  have "heap_is_wellformed h2 ∧ known_ptrs h2 ∧ type_wf h2"
    using h2 ‹heap_is_wellformed h› ‹known_ptrs h› ‹type_wf h›
    by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated]
        elim!: forall_M_small_big[where P = "λh. heap_is_wellformed h ∧ known_ptrs h ∧ type_wf h", simplified]
        intro: remove_preserves_known_ptrs remove_heap_is_wellformed_preserved remove_preserves_type_wf
        append_child_preserves_known_ptrs append_child_heap_is_wellformed_preserved append_child_preserves_type_wf)
  then
  show "heap_is_wellformed h'" and "known_ptrs h'" and "type_wf h'"
    using h'
    by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_host_pure, rotated] bind_returns_heap_
OF get_child_nodes_pure, rotated]
        dest!: forall_M_small_big[where P = "λh. heap_is_wellformed h ∧ known_ptrs h ∧ type_wf h", simplified]
        intro: remove_preserves_known_ptrs remove_heap_is_wellformed_preserved remove_preserves_type_wf
        append_child_preserves_known_ptrs append_child_heap_is_wellformed_preserved append_child_preserves_type_wf
        remove_shadow_root_preserves
        )
qed
end

interpretation i_assigned_nodes_wf?: l_assigned_nodes_wf_Shadow_DOM
  known_ptr assigned_nodes assigned_nodes_flatten flatten_dom get_child_nodes get_child_nodes_locs
  get_tag_name get_tag_name_locs get_root_node get_root_node_locs get_host get_host_locs find_slot
  assigned_slot remove insert_before insert_before_locs append_child remove_shadow_root
  remove_shadow_root_locs type_wf get_shadow_root get_shadow_root_locs set_shadow_root
  set_shadow_root_locs get_parent get_parent_locs to_tree_order heap_is_wellformed parent_child_rel
  get_disconnected_nodes get_disconnected_nodes_locs known_ptrs remove_child remove_child_locs
  heap_is_wellformed_Core_DOM get_disconnected_document get_disconnected_document_locs
  by(auto simp add: l_assigned_nodes_wf_Shadow_DOM_def instances)
declare l_assigned_nodes_wf_Shadow_DOM_axioms [instances]
```

get_shadow_root_safe

```
locale l_get_shadow_root_safe_wf_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM  get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs
  known_ptr type_wf heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs
+
  l_type_wf type_wf +
  l_get_shadow_root_safe_Shadow_DOM type_wf get_shadow_root_safe get_shadow_root_safe_locs get_shadow_root
  get_shadow_root_locs get_mode get_mode_locs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
```

```
    and type_wf :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root_safe :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_safe_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒
bool) set"
    and get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
begin


end
```

create_element

```
locale l_create_element_wf_Shadow_DOM =
  l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_tag_name type_wf set_tag_name set_tag_name_locs +
  l_create_element_defs create_element +
  l_heap_is_wellformed_Shadow_DOM  get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf
  heap_is_wellformed parent_child_rel
  heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs +
  l_new_element_get_disconnected_nodes get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_tag_name_get_disconnected_nodes type_wf set_tag_name set_tag_name_locs
  get_disconnected_nodes get_disconnected_nodes_locs +
  l_create_element_Shadow_DOM get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs set_tag_name set_tag_name_locs type_wf
  create_element known_ptr type_wf_Core_DOM known_ptr_Core_DOM +
  l_new_element_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs +
  l_set_tag_name_get_child_nodes type_wf set_tag_name set_tag_name_locs known_ptr
  get_child_nodes get_child_nodes_locs +
  l_set_tag_name_get_tag_name type_wf get_tag_name get_tag_name_locs set_tag_name set_tag_name_locs +
  l_new_element_get_tag_name type_wf get_tag_name get_tag_name_locs +
  l_set_disconnected_nodes_get_child_nodes set_disconnected_nodes set_disconnected_nodes_locs
  get_child_nodes get_child_nodes_locs +
  l_set_disconnected_nodes_get_shadow_root set_disconnected_nodes set_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs +
  l_set_disconnected_nodes_get_tag_name type_wf set_disconnected_nodes set_disconnected_nodes_locs
  get_tag_name get_tag_name_locs +
  l_set_disconnected_nodes type_wf set_disconnected_nodes set_disconnected_nodes_locs +
  l_set_disconnected_nodes_get_disconnected_nodes  type_wf get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs +
  l_new_element_get_shadow_root type_wf get_shadow_root get_shadow_root_locs +
  l_set_tag_name_get_shadow_root type_wf set_tag_name set_tag_name_locs get_shadow_root get_shadow_root_locs
+
  l_new_element type_wf +
  l_known_ptrs known_ptr known_ptrs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
```

```
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and set_tag_name :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
    and set_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and create_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
begin
lemma create_element_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ create_element document_ptr tag →_h h'"
    and "type_wf h"
    and "known_ptrs h"
  shows "heap_is_wellformed h'" and "type_wf h'" and "known_ptrs h'"
proof -
  obtain new_element_ptr h2 h3 disc_nodes_h3 where
    new_element_ptr: "h ⊢ new_element →_r new_element_ptr" and
    h2: "h ⊢ new_element →_h h2" and
    h3: "h2 ⊢ set_tag_name new_element_ptr tag →_h h3" and
    disc_nodes_h3: "h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes_h3" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_element_ptr # disc_nodes_h3) →_h h'"
    using assms(2)
    by(auto simp add: create_element_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF CD.get_disconnected_nodes_pure, rotated] )
  then have "h ⊢ create_element document_ptr tag →_r new_element_ptr"
    apply(auto simp add: create_element_def intro!: bind_returns_result_I)[1]
      apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
     apply (metis is_OK_returns_heap_E is_OK_returns_result_I CD.get_disconnected_nodes_pure
        pure_returns_heap_eq)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)

  have "new_element_ptr ∉ set |h ⊢ element_ptr_kinds_M|_r"
    using new_element_ptr ElementMonad.ptr_kinds_ptr_kinds_M h2
    using new_element_ptr_not_in_heap by blast
  then have "cast new_element_ptr ∉ set |h ⊢ node_ptr_kinds_M|_r"
    by simp
  then have "cast new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r"
    by simp

  have object_ptr_kinds_eq_h: "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_element_ptr|}"
    using new_element_new_ptr h2 new_element_ptr by blast
  then have node_ptr_kinds_eq_h: "node_ptr_kinds h2 = node_ptr_kinds h |∪| {|cast new_element_ptr|}"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h |∪| {|new_element_ptr|}"
    apply(simp add: element_ptr_kinds_def)
    by force
```

```
  have character_data_ptr_kinds_eq_h: "character_data_ptr_kinds h2 = character_data_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def character_data_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_tag_name_writes h3])
    using set_tag_name_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)
  then have element_ptr_kinds_eq_h2: "element_ptr_kinds h3 = element_ptr_kinds h2"
    by(simp add: element_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)
  then have element_ptr_kinds_eq_h3: "element_ptr_kinds h' = element_ptr_kinds h3"
    by(simp add: element_ptr_kinds_def)

  have "known_ptr (cast new_element_ptr)"
    using ‹h ⊢ create_element document_ptr tag →r new_element_ptr› local.create_element_known_ptr
    by blast
  then
  have "known_ptrs h2"
    using known_ptrs_new_ptr object_ptr_kinds_eq_h ‹known_ptrs h› h2
    by blast
  then
  have "known_ptrs h3"
    using known_ptrs_preserved object_ptr_kinds_eq_h2 by blast
  then
  show "known_ptrs h'"
    using known_ptrs_preserved object_ptr_kinds_eq_h3 by blast


  have "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      CD.get_disconnected_nodes_ptr_in_heap ‹type_wf h› document_ptr_kinds_def
    by (metis is_OK_returns_result_I)

  have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_element_ptr
              ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads h2 get_child_nodes_new_element[rotated, OF new_element_ptr h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h: "⋀ptr'. ptr' ≠ cast new_element_ptr
                                  ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have "h2 ⊢ get_child_nodes (cast new_element_ptr) →r []"
    using new_element_ptr h2 new_element_ptr_in_heap[OF h2 new_element_ptr]
```

```
    new_element_is_element_ptr[OF new_element_ptr] new_element_no_child_nodes
  by blast
have tag_name_eq_h:
  "⋀ptr' disc_nodes. ptr' ≠  new_element_ptr
          ⟹ h ⊢ get_tag_name ptr' →r disc_nodes
                                        = h2 ⊢ get_tag_name ptr' →r disc_nodes"
  using get_tag_name_reads h2 get_tag_name_new_element[rotated, OF new_element_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by(blast)+
then have tag_name_eq2_h: "⋀ptr'. ptr' ≠ new_element_ptr
                                ⟹ |h ⊢ get_tag_name ptr'|r = |h2 ⊢ get_tag_name ptr'|r"
  using select_result_eq by force
have "h2 ⊢ get_tag_name new_element_ptr →r ''''"
  using new_element_ptr h2 new_element_ptr_in_heap[OF h2 new_element_ptr]
    new_element_is_element_ptr[OF new_element_ptr] new_element_empty_tag_name
  by blast


have disconnected_nodes_eq_h:
  "⋀doc_ptr disc_nodes. h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                        = h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
  using CD.get_disconnected_nodes_reads h2 get_disconnected_nodes_new_element[OF new_element_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by blast+
then have disconnected_nodes_eq2_h:
  "⋀doc_ptr. |h ⊢ get_disconnected_nodes doc_ptr|r = |h2 ⊢ get_disconnected_nodes doc_ptr|r"
  using select_result_eq by force

have children_eq_h2:
  "⋀ptr' children. h2 ⊢ get_child_nodes ptr' →r children = h3 ⊢ get_child_nodes ptr' →r children"
  using CD.get_child_nodes_reads set_tag_name_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_tag_name_get_child_nodes)
then have children_eq2_h2: "⋀ptr'. |h2 ⊢ get_child_nodes ptr'|r = |h3 ⊢ get_child_nodes ptr'|r"
  using select_result_eq by force
have disconnected_nodes_eq_h2:
  "⋀doc_ptr disc_nodes. h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                          = h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
  using CD.get_disconnected_nodes_reads set_tag_name_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_tag_name_get_disconnected_nodes)
then have disconnected_nodes_eq2_h2:
  "⋀doc_ptr. |h2 ⊢ get_disconnected_nodes doc_ptr|r = |h3 ⊢ get_disconnected_nodes doc_ptr|r"
  using select_result_eq by force
have tag_name_eq_h2:
  "⋀ptr' disc_nodes. ptr' ≠  new_element_ptr
          ⟹ h2 ⊢ get_tag_name ptr' →r disc_nodes
                                        = h3 ⊢ get_tag_name ptr' →r disc_nodes"
  apply(rule reads_writes_preserved[OF get_tag_name_reads set_tag_name_writes h3])
  by (metis local.set_tag_name_get_tag_name_different_pointers)
then have tag_name_eq2_h2: "⋀ptr'. ptr' ≠ new_element_ptr
                                ⟹ |h2 ⊢ get_tag_name ptr'|r = |h3 ⊢ get_tag_name ptr'|r"
  using select_result_eq by force
have "h2 ⊢ get_tag_name new_element_ptr →r ''''"
  using new_element_ptr h2 new_element_ptr_in_heap[OF h2 new_element_ptr]
    new_element_is_element_ptr[OF new_element_ptr] new_element_empty_tag_name
  by blast

have "type_wf h2"
  using ‹type_wf h› new_element_types_preserved h2 by blast
then have "type_wf h3"
  using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_tag_name_writes h3]
  using  set_tag_name_types_preserved
```

```
    by(auto simp add: reflp_def transp_def)
  then show "type_wf h'"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h']
    using  set_disconnected_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  have children_eq_h3:
    "⋀ptr' children. h3 ⊢ get_child_nodes ptr' →r children = h' ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_child_nodes)
  then have children_eq2_h3: "⋀ptr'. |h3 ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h3:
    "⋀doc_ptr disc_nodes. document_ptr ≠ doc_ptr
      ⟹ h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                          = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using CD.get_disconnected_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
  then have disconnected_nodes_eq2_h3:
    "⋀doc_ptr. document_ptr ≠ doc_ptr
              ⟹ |h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force
  have tag_name_eq_h2:
    "⋀ptr' disc_nodes. ptr' ≠  new_element_ptr
             ⟹ h2 ⊢ get_tag_name ptr' →r disc_nodes
                                        = h3 ⊢ get_tag_name ptr' →r disc_nodes"
    apply(rule reads_writes_preserved[OF get_tag_name_reads set_tag_name_writes h3])
    by (metis local.set_tag_name_get_tag_name_different_pointers)
  then have tag_name_eq2_h2: "⋀ptr'. ptr' ≠ new_element_ptr
                                  ⟹ |h2 ⊢ get_tag_name ptr'|r = |h3 ⊢ get_tag_name ptr'|r"
    using select_result_eq by force

  have disc_nodes_document_ptr_h2: "h2 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h2 disc_nodes_h3 by auto
  then have disc_nodes_document_ptr_h: "h ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h by auto
  then have "cast new_element_ptr ∉ set disc_nodes_h3"
    using ‹heap_is_wellformed h›
    using ‹castelement_ptr2node_ptr new_element_ptr ∉ set |h ⊢ node_ptr_kinds_M|r›
      a_all_ptrs_in_heap_def heap_is_wellformed_def
    using NodeMonad.ptr_kinds_ptr_kinds_M local.heap_is_wellformed_disc_nodes_in_heap by blast

  have tag_name_eq_h3:
    "⋀ptr' disc_nodes. h3 ⊢ get_tag_name ptr' →r disc_nodes
                                        = h' ⊢ get_tag_name ptr' →r disc_nodes"
    apply(rule reads_writes_preserved[OF get_tag_name_reads set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_get_tag_name
    by blast
  then have tag_name_eq2_h3: "⋀ptr'. |h3 ⊢ get_tag_name ptr'|r = |h' ⊢ get_tag_name ptr'|r"
    using select_result_eq by force


  have "acyclic (parent_child_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)
  also have "parent_child_rel h = parent_child_rel h2"
  proof(auto simp add: CD.parent_child_rel_def)[1]
    fix a x
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
```

```
  then show "a |∈| object_ptr_kinds h2"
    by (simp add: object_ptr_kinds_eq_h)
next
  fix a x
  assume 0: "a |∈| object_ptr_kinds h"
    and 1: "x ∈ set |h ⊢ get_child_nodes a|_r"
  then show "x ∈ set |h2 ⊢ get_child_nodes a|_r"
    by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹cast_element_ptr2object_ptr new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r› children_eq2_h)
next
  fix a x
  assume 0: "a |∈| object_ptr_kinds h2"
    and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
  then show "a |∈| object_ptr_kinds h"
    using object_ptr_kinds_eq_h ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []›
    by(auto)
next
  fix a x
  assume 0: "a |∈| object_ptr_kinds h2"
    and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
  then show "x ∈ set |h ⊢ get_child_nodes a|_r"
    by (metis (no_types, lifting)
        ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []›
        children_eq2_h empty_iff empty_set image_eqI select_result_I2)
qed
also have "... = parent_child_rel h3"
  by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
also have "... = parent_child_rel h'"
  by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
finally have "CD.a_acyclic_heap h'"
  by (simp add: CD.acyclic_heap_def)

have "CD.a_all_ptrs_in_heap h"
  using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
then have "CD.a_all_ptrs_in_heap h2"
  apply(auto simp add: CD.a_all_ptrs_in_heap_def)[1]
   apply (metis ‹known_ptrs h2› ‹parent_child_rel h = parent_child_rel h2› ‹type_wf h2› assms(1)
      assms(3) funion_iff CD.get_child_nodes_ok local.known_ptrs_known_ptr
      local.parent_child_rel_child_in_heap CD.parent_child_rel_child_nodes2 node_ptr_kinds_commutes
      node_ptr_kinds_eq_h returns_result_select_result)
  by (metis (no_types, opaque_lifting) CD.get_child_nodes_ok CD.get_child_nodes_ptr_in_heap
      ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []› assms(3) assms(4) children_eq_h
      disconnected_nodes_eq2_h document_ptr_kinds_eq_h is_OK_returns_result_I
      local.known_ptrs_known_ptr node_ptr_kinds_commutes returns_result_select_result subsetD)
then have "CD.a_all_ptrs_in_heap h3"
  by (simp add: children_eq2_h2 disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
      CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h2 object_ptr_kinds_eq_h2)
then have "CD.a_all_ptrs_in_heap h'"
  by (smt (verit) children_eq2_h3 disc_nodes_h3 disconnected_nodes_eq2_h3 document_ptr_kinds_eq_h3
      element_ptr_kinds_commutes h' h2 local.CD.a_all_ptrs_in_heap_def
      local.set_disconnected_nodes_get_disconnected_nodes new_element_ptr new_element_ptr_in_heap
      node_ptr_kinds_eq_h2 node_ptr_kinds_eq_h3 object_ptr_kinds_eq_h3 select_result_I2
      set_ConsD subset_code(1))

have "⋀p. p |∈| object_ptr_kinds h ⟹ cast new_element_ptr ∉ set |h ⊢ get_child_nodes p|_r"
  using ‹heap_is_wellformed h› ‹cast_element_ptr2node_ptr new_element_ptr ∉ set |h ⊢ node_ptr_kinds_M|_r›
    heap_is_wellformed_children_in_heap
  by (meson NodeMonad.ptr_kinds_ptr_kinds_M CD.a_all_ptrs_in_heap_def assms(3) assms(4) fset_mp
      fset_of_list_elem CD.get_child_nodes_ok known_ptrs_known_ptr returns_result_select_result)
then have "⋀p. p |∈| object_ptr_kinds h2 ⟹ cast new_element_ptr ∉ set |h2 ⊢ get_child_nodes p|_r"
  using children_eq2_h
  apply(auto simp add: object_ptr_kinds_eq_h)[1]
  using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []› apply auto[1]
```

```
    by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹cast_element_ptr2object_ptr new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r›)
  then have "⋀p. p |∈| object_ptr_kinds h3 ⟹ cast new_element_ptr ∉ set |h3 ⊢ get_child_nodes p|_r"
    using object_ptr_kinds_eq_h2 children_eq2_h2 by auto
  then have new_element_ptr_not_in_any_children:
    "⋀p. p |∈| object_ptr_kinds h' ⟹ cast new_element_ptr ∉ set |h' ⊢ get_child_nodes p|_r"
    using object_ptr_kinds_eq_h3 children_eq2_h3 by auto

  have "CD.a_distinct_lists h"
    using ‹heap_is_wellformed h›
    by (simp add: CD.heap_is_wellformed_def heap_is_wellformed_def)
  then have "CD.a_distinct_lists h2"
    using ‹h2 ⊢ get_child_nodes (cast new_element_ptr) →_r []›
    apply(auto simp add: CD.a_distinct_lists_def object_ptr_kinds_eq_h document_ptr_kinds_eq_h
        disconnected_nodes_eq2_h intro!: distinct_concat_map_I)[1]
      apply (metis distinct_sorted_list_of_set finite_fset sorted_list_of_set_insert_remove)
     apply(case_tac "x=cast new_element_ptr")
      apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
     apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
    apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
    apply (metis IntI assms(1) assms(3) assms(4) empty_iff  CD.get_child_nodes_ok
        local.heap_is_wellformed_one_parent local.known_ptrs_known_ptr returns_result_select_result)
    apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
    by (metis ‹ CD.a_distinct_lists h› ‹type_wf h2› disconnected_nodes_eq_h document_ptr_kinds_eq_h
        CD.distinct_lists_no_parent  get_disconnected_nodes_ok returns_result_select_result)

  then have " CD.a_distinct_lists h3"
    by(auto simp add:  CD.a_distinct_lists_def disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
        children_eq2_h2 object_ptr_kinds_eq_h2)
  then have " CD.a_distinct_lists h'"
  proof(auto simp add:  CD.a_distinct_lists_def disconnected_nodes_eq2_h3 children_eq2_h3
      object_ptr_kinds_eq_h3 document_ptr_kinds_eq_h3
      intro!: distinct_concat_map_I)[1]
    fix x
    assume "distinct (concat (map (λdocument_ptr. |h3 ⊢ get_disconnected_nodes document_ptr|_r)
                                        (sorted_list_of_set (fset (document_ptr_kinds h3)))))"
      and "x |∈| document_ptr_kinds h3"
    then show "distinct |h' ⊢ get_disconnected_nodes x|_r"
      using document_ptr_kinds_eq_h3 disconnected_nodes_eq_h3 h' set_disconnected_nodes_get_disconnected_nodes
      by (metis (no_types, lifting) ‹cast_element_ptr2node_ptr new_element_ptr ∉ set disc_nodes_h3›
          ‹ CD.a_distinct_lists h3› ‹type_wf h'› disc_nodes_h3 distinct.simps(2)
          CD.distinct_lists_disconnected_nodes  get_disconnected_nodes_ok returns_result_eq
          returns_result_select_result)
  next
    fix x y xa
    assume "distinct (concat (map (λdocument_ptr. |h3 ⊢ get_disconnected_nodes document_ptr|_r)
                                       (sorted_list_of_set (fset (document_ptr_kinds h3)))))"
      and "x |∈| document_ptr_kinds h3"
      and "y |∈| document_ptr_kinds h3"
      and "x ≠ y"
      and "xa ∈ set |h' ⊢ get_disconnected_nodes x|_r"
      and "xa ∈ set |h' ⊢ get_disconnected_nodes y|_r"
    moreover have "set |h3 ⊢ get_disconnected_nodes x|_r ∩ set |h3 ⊢ get_disconnected_nodes y|_r = {}"
      using calculation by(auto dest: distinct_concat_map_E(1))
    ultimately show "False"
      apply(-)
      apply(cases "x = document_ptr")
       apply(smt (verit) NodeMonad.ptr_kinds_ptr_kinds_M ‹cast_element_ptr2node_ptr new_element_ptr ∉ set
|h ⊢ node_ptr_kinds_M|_r› ‹CD.a_all_ptrs_in_heap h›
          disc_nodes_h3 disconnected_nodes_eq2_h disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3
          disjoint_iff_not_equal document_ptr_kinds_eq_h document_ptr_kinds_eq_h2 h'
          set_disconnected_nodes_get_disconnected_nodes
          CD.a_all_ptrs_in_heap_def
```

```
        select_result_I2 set_ConsD subsetD)
    by (smt (verit) NodeMonad.ptr_kinds_ptr_kinds_M ‹cast_element_ptr2node_ptr new_element_ptr ∉ set |h
⊢ node_ptr_kinds_M|_r› ‹CD.a_all_ptrs_in_heap h›
        disc_nodes_document_ptr_h2 disconnected_nodes_eq2_h disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3
        disjoint_iff_not_equal document_ptr_kinds_eq_h document_ptr_kinds_eq_h2 h'
        l_set_disconnected_nodes_get_disconnected_nodes.set_disconnected_nodes_get_disconnected_nodes
        CD.a_all_ptrs_in_heap_def local.l_set_disconnected_nodes_get_disconnected_nodes_axioms
        select_result_I2 set_ConsD subsetD)
  next
    fix x xa xb
    assume 2: "(⋃x∈fset (object_ptr_kinds h3). set |h' ⊢ get_child_nodes x|_r)
                ∩ (⋃x∈fset (document_ptr_kinds h3). set |h3 ⊢ get_disconnected_nodes x|_r) = {}"
      and 3: "xa |∈| object_ptr_kinds h3"
      and 4: "x ∈ set |h' ⊢ get_child_nodes xa|_r"
      and 5: "xb |∈| document_ptr_kinds h3"
      and 6: "x ∈ set |h' ⊢ get_disconnected_nodes xb|_r"
    show "False"
      using disc_nodes_document_ptr_h disconnected_nodes_eq2_h3
      apply -
      apply(cases "xb = document_ptr")
       apply (metis (no_types, opaque_lifting) "3" "4" "6"
        ‹⋀p. p |∈| object_ptr_kinds h3
                  ⟹ cast_element_ptr2node_ptr new_element_ptr ∉ set |h3 ⊢ get_child_nodes p|_r›
        ‹ CD.a_distinct_lists h3› children_eq2_h3 disc_nodes_h3 CD.distinct_lists_no_parent h'
        select_result_I2 set_ConsD  set_disconnected_nodes_get_disconnected_nodes)
      by (metis "3" "4" "5" "6" ‹ CD.a_distinct_lists h3› ‹type_wf h3› children_eq2_h3
        CD.distinct_lists_no_parent  get_disconnected_nodes_ok returns_result_select_result)
  qed

  have "CD.a_owner_document_valid h"
    using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_owner_document_valid h'"
    using disc_nodes_h3 ‹document_ptr |∈| document_ptr_kinds h›
    apply(auto simp add: CD.a_owner_document_valid_def)[1]
    apply(auto simp add:  object_ptr_kinds_eq_h object_ptr_kinds_eq_h3 )[1]
    apply(auto simp add: object_ptr_kinds_eq_h2)[1]
    apply(auto simp add:  document_ptr_kinds_eq_h document_ptr_kinds_eq_h3 )[1]
    apply(auto simp add: document_ptr_kinds_eq_h2)[1]
    apply(auto simp add:  node_ptr_kinds_eq_h node_ptr_kinds_eq_h3 )[1]
    apply(auto simp add: node_ptr_kinds_eq_h2 node_ptr_kinds_eq_h )[1]
     apply(auto simp add: children_eq2_h2[symmetric] children_eq2_h3[symmetric]
        disconnected_nodes_eq2_h disconnected_nodes_eq2_h2
        disconnected_nodes_eq2_h3)[1]
     apply (metis (no_types, lifting) document_ptr_kinds_eq_h h' list.set_intros(1)
        local.set_disconnected_nodes_get_disconnected_nodes select_result_I2)
    apply(simp add: object_ptr_kinds_eq_h)
    by(metis (no_types, opaque_lifting) NodeMonad.ptr_kinds_ptr_kinds_M
        ‹cast_element_ptr2node_ptr new_element_ptr ∉ set |h ⊢ node_ptr_kinds_M|_r› children_eq2_h children_eq2_h2
        children_eq2_h3 disconnected_nodes_eq2_h disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3
        document_ptr_kinds_eq_h h'
        l_set_disconnected_nodes_get_disconnected_nodes.set_disconnected_nodes_get_disconnected_nodes
        list.set_intros(2) local.l_set_disconnected_nodes_get_disconnected_nodes_axioms node_ptr_kinds_commutes
        select_result_I2)

  have "CD.a_heap_is_wellformed h'"
    using ‹CD.a_acyclic_heap h'› ‹CD.a_all_ptrs_in_heap h'› ‹CD.a_distinct_lists h'› ‹CD.a_owner_document_valid
h'›
    by(simp add: CD.a_heap_is_wellformed_def)
```

```
  have shadow_root_ptr_kinds_eq_h: "shadow_root_ptr_kinds h2 = shadow_root_ptr_kinds h"
    using document_ptr_kinds_eq_h
    by(auto simp add: shadow_root_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_h2: "shadow_root_ptr_kinds h3 = shadow_root_ptr_kinds h2"
    using document_ptr_kinds_eq_h2
    by(auto simp add: shadow_root_ptr_kinds_def)
  have shadow_root_ptr_kinds_eq_h3: "shadow_root_ptr_kinds h' = shadow_root_ptr_kinds h3"
    using document_ptr_kinds_eq_h3
    by(auto simp add: shadow_root_ptr_kinds_def)



  have shadow_root_eq_h: "⋀element_ptr shadow_root_opt. element_ptr ≠ new_element_ptr
              ⟹ h ⊢ get_shadow_root element_ptr →r shadow_root_opt = h2 ⊢ get_shadow_root element_ptr
→r shadow_root_opt"
  proof -
    fix element_ptr shadow_root_opt
    assume "element_ptr ≠ new_element_ptr "
    have "∀P ∈ get_shadow_root_locs element_ptr. P h h2"
      using get_shadow_root_new_element new_element_ptr h2
      using ‹element_ptr ≠ new_element_ptr› by blast
    then
    have "preserved (get_shadow_root element_ptr) h h2"
      using get_shadow_root_new_element[rotated, OF new_element_ptr h2]
      using get_shadow_root_reads
      by(simp add: reads_def)
    then show "h ⊢ get_shadow_root element_ptr →r shadow_root_opt = h2 ⊢ get_shadow_root element_ptr
→r shadow_root_opt"
      by (simp add: preserved_def)
  qed
  have shadow_root_none: "h2 ⊢ get_shadow_root (new_element_ptr) →r None"
    using new_element_ptr h2 new_element_ptr_in_heap[OF h2 new_element_ptr]
      new_element_is_element_ptr[OF new_element_ptr] new_element_no_shadow_root
    by blast

  have shadow_root_eq_h2:
    "⋀ptr' children. h2 ⊢ get_shadow_root ptr' →r children = h3 ⊢ get_shadow_root ptr' →r children"
    using get_shadow_root_reads set_tag_name_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_tag_name_get_shadow_root)
  have shadow_root_eq_h3:
    "⋀ptr' children. h3 ⊢ get_shadow_root ptr' →r children = h' ⊢ get_shadow_root ptr' →r children"
    using get_shadow_root_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    using set_disconnected_nodes_get_shadow_root
    by(auto simp add: set_disconnected_nodes_get_shadow_root)


  have "a_all_ptrs_in_heap h"
    by (simp add: assms(1) local.a_all_ptrs_in_heap_def local.get_shadow_root_shadow_root_ptr_in_heap)
  then have "a_all_ptrs_in_heap h2"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h)[1]
    using returns_result_eq shadow_root_eq_h shadow_root_none by fastforce
  then have "a_all_ptrs_in_heap h3"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h2)[1]
    using shadow_root_eq_h2 by blast
  then have "a_all_ptrs_in_heap h'"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h3)[1]
    by (simp add: shadow_root_eq_h3)

  have "a_distinct_lists h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
```

```
  then have "a_distinct_lists h2"
    apply(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h)[1]
    apply(auto simp add: distinct_insort intro!: distinct_concat_map_I split: option.splits)[1]
    apply(case_tac "x = new_element_ptr")
    using shadow_root_none apply auto[1]
    using shadow_root_eq_h
    by (smt (verit) Diff_empty Diff_insert0 ElementMonad.ptr_kinds_M_ptr_kinds
        ElementMonad.ptr_kinds_ptr_kinds_M assms(1) assms(3) h2 insort_split
        local.get_shadow_root_ok local.shadow_root_same_host new_element_ptr new_element_ptr_not_in_heap
        option.distinct(1) returns_result_select_result select_result_I2 shadow_root_none)
  then have "a_distinct_lists h3"
    by(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h2 select_result_eq[OF shadow_root_eq_h2])
  then have "a_distinct_lists h'"
    by(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h3 select_result_eq[OF shadow_root_eq_h3])


  have "a_shadow_root_valid h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then have "a_shadow_root_valid h2"
  proof (unfold a_shadow_root_valid_def; safe)
    fix shadow_root_ptr
    assume "∀shadow_root_ptr∈fset (shadow_root_ptr_kinds h). ∃host∈fset (element_ptr_kinds h).
|h ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types ∧ |h ⊢ get_shadow_root host|_r = Some shadow_root_ptr"
    assume "shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h2)"

    obtain previous_host where
      "previous_host ∈ fset (element_ptr_kinds h)" and
      "|h ⊢ get_tag_name previous_host|_r ∈ safe_shadow_root_element_types" and
      "|h ⊢ get_shadow_root previous_host|_r = Some shadow_root_ptr"
      by (metis ‹local.a_shadow_root_valid h› ‹shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h2)›
          local.a_shadow_root_valid_def shadow_root_ptr_kinds_eq_h)
    moreover have "previous_host ≠ new_element_ptr"
      using calculation(1) h2 new_element_ptr new_element_ptr_not_in_heap by auto
    ultimately have "|h2 ⊢ get_tag_name previous_host|_r ∈ safe_shadow_root_element_types" and
      "|h2 ⊢ get_shadow_root previous_host|_r = Some shadow_root_ptr"
      using shadow_root_eq_h
       apply (simp add: tag_name_eq2_h)
      by (metis ‹previous_host ≠ new_element_ptr› ‹|h ⊢ get_shadow_root previous_host|_r = Some shadow_root_ptr›
          select_result_eq shadow_root_eq_h)
    then
    show "∃host∈fset (element_ptr_kinds h2). |h2 ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types
∧
|h2 ⊢ get_shadow_root host|_r = Some shadow_root_ptr"
      by (meson ‹previous_host ∈ fset (element_ptr_kinds h)› ‹previous_host ≠ new_element_ptr› assms(3)
          local.get_shadow_root_ok local.get_shadow_root_ptr_in_heap returns_result_select_result shadow_root_eq_h)
  qed
  then have "a_shadow_root_valid h3"
  proof (unfold a_shadow_root_valid_def; safe)
    fix shadow_root_ptr
    assume "∀shadow_root_ptr∈fset (shadow_root_ptr_kinds h2). ∃host∈fset (element_ptr_kinds h2).
|h2 ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types ∧ |h2 ⊢ get_shadow_root host|_r = Some shadow_root_ptr"
    assume "shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h3)"

    obtain previous_host where
      "previous_host ∈ fset (element_ptr_kinds h2)" and
      "|h2 ⊢ get_tag_name previous_host|_r ∈ safe_shadow_root_element_types" and
      "|h2 ⊢ get_shadow_root previous_host|_r = Some shadow_root_ptr"
      by (metis ‹local.a_shadow_root_valid h2› ‹shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h3)›
          local.a_shadow_root_valid_def shadow_root_ptr_kinds_eq_h2)
    moreover have "previous_host ≠ new_element_ptr"
      using calculation(1) h3 new_element_ptr new_element_ptr_not_in_heap
      using calculation(3) shadow_root_none by auto
```

```
    ultimately have "|h2 ⊢ get_tag_name previous_host|r ∈ safe_shadow_root_element_types" and
      "|h2 ⊢ get_shadow_root previous_host|r = Some shadow_root_ptr"
      using shadow_root_eq_h2
       apply (simp add: tag_name_eq2_h2)
      by (metis ‹previous_host ≠ new_element_ptr› ‹|h2 ⊢ get_shadow_root previous_host|r = Some shadow_root_ptr›
          select_result_eq shadow_root_eq_h)
    then
    show "∃host∈fset (element_ptr_kinds h3). |h3 ⊢ get_tag_name host|r ∈ safe_shadow_root_element_types
∧
|h3 ⊢ get_shadow_root host|r = Some shadow_root_ptr"
      by (smt (verit) ‹previous_host ∈ fset (element_ptr_kinds h2)› ‹previous_host ≠ new_element_ptr›
‹type_wf h2›
          ‹type_wf h3› element_ptr_kinds_eq_h2 local.get_shadow_root_ok returns_result_eq
          returns_result_select_result shadow_root_eq_h2 tag_name_eq2_h2)
  qed
  then have "a_shadow_root_valid h'"
    apply(auto simp add: a_shadow_root_valid_def element_ptr_kinds_eq_h3 shadow_root_eq_h3
        shadow_root_ptr_kinds_eq_h3 tag_name_eq2_h3)[1]
    by (smt (z3) ‹type_wf h3› local.get_shadow_root_ok returns_result_select_result
        select_result_I2 shadow_root_eq_h3)



  have "a_host_shadow_root_rel h = a_host_shadow_root_rel h2"
    apply(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h shadow_root_eq_h)[1]
      apply (smt assms(3) case_prod_conv h2 image_iff local.get_shadow_root_ok mem_Collect_eq new_element_ptr
        new_element_ptr_not_in_heap returns_result_select_result select_result_I2 shadow_root_eq_h)
    using shadow_root_none apply auto[1]
    apply (metis (no_types, lifting) Collect_cong assms(3) case_prodE case_prodI h2 host_shadow_root_rel_def
        i_get_parent_get_host_get_disconnected_document_wf.a_host_shadow_root_rel_shadow_root
        local.a_host_shadow_root_rel_def local.get_shadow_root_impl local.get_shadow_root_ok new_element_ptr
        new_element_ptr_not_in_heap returns_result_select_result select_result_I2 shadow_root_eq_h)
    done
  have "a_host_shadow_root_rel h2 = a_host_shadow_root_rel h3"
    apply(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h2 shadow_root_eq_h2)[1]
     apply (smt Collect_cong ‹type_wf h2› case_prodE case_prodI element_ptr_kinds_eq_h2 host_shadow_root_rel_def
        i_get_root_node_si_wf.a_host_shadow_root_rel_shadow_root local.a_host_shadow_root_rel_def local.get_shadow_
        local.get_shadow_root_ok returns_result_select_result shadow_root_eq_h2)
    by (metis (no_types, lifting) Collect_cong ‹type_wf h3› case_prodI2 case_prod_conv element_ptr_kinds_eq_h2
        host_shadow_root_rel_def i_get_root_node_si_wf.a_host_shadow_root_rel_shadow_root local.a_host_shadow_root_
        local.get_shadow_root_impl local.get_shadow_root_ok returns_result_select_result shadow_root_eq_h2)
  have "a_host_shadow_root_rel h3 = a_host_shadow_root_rel h'"
    apply(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h2 shadow_root_eq_h2)[1]
     apply (smt Collect_cong Shadow_DOM.a_host_shadow_root_rel_def ‹type_wf h3› case_prodD case_prodI2
        element_ptr_kinds_eq_h2 i_get_root_node_si_wf.a_host_shadow_root_rel_shadow_root local.get_shadow_root_impl
        local.get_shadow_root_ok returns_result_select_result shadow_root_eq_h3)
    apply (smt Collect_cong ‹type_wf h'› case_prodE case_prodI element_ptr_kinds_eq_h2 host_shadow_root_rel_def
        i_get_root_node_si_wf.a_host_shadow_root_rel_shadow_root l_heap_is_wellformed_Shadow_DOM_defs.a_host_shado
        local.get_shadow_root_impl local.get_shadow_root_ok returns_result_select_result shadow_root_eq_h3)
    done


  have "a_ptr_disconnected_node_rel h = a_ptr_disconnected_node_rel h2"
    by(simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h disconnected_nodes_eq2_h)
  have "a_ptr_disconnected_node_rel h2 = a_ptr_disconnected_node_rel h3"
    by(simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h2 disconnected_nodes_eq2_h2)

  have "h' ⊢ get_disconnected_nodes document_ptr →r cast new_element_ptr # disc_nodes_h3"
    using h' local.set_disconnected_nodes_get_disconnected_nodes by auto
  have " document_ptr |∈| document_ptr_kinds h3"
    by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h document_ptr_kinds_eq_h2)
  have "cast new_element_ptr ∈ set |h' ⊢ get_disconnected_nodes document_ptr|r"
    using ‹h' ⊢ get_disconnected_nodes document_ptr →r cast_element_ptr2node_ptr new_element_ptr # disc_nodes_h3›
```

```
    by auto

  have "a_ptr_disconnected_node_rel h' = {(cast document_ptr, cast new_element_ptr)} ∪ a_ptr_disconnected_node_rel
h3"
    apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h3 disconnected_nodes_eq2_h3)[1]
      apply(case_tac "aa = document_ptr")
    using disc_nodes_h3 h' ‹h' ⊢ get_disconnected_nodes document_ptr →r cast new_element_ptr # disc_nodes_h3›
       apply(auto)[1]
    using disconnected_nodes_eq2_h3 apply auto[1]
    using ‹h' ⊢ get_disconnected_nodes document_ptr →r cast new_element_ptr # disc_nodes_h3›
    using ‹cast new_element_ptr ∈ set |h' ⊢ get_disconnected_nodes document_ptr|r›
    using ‹document_ptr |∈| document_ptr_kinds h3› apply auto[1]
    apply(case_tac "document_ptr = aa")
    using ‹h' ⊢ get_disconnected_nodes document_ptr →r castelement_ptr2node_ptr new_element_ptr # disc_nodes_h3›
disc_nodes_h3
     apply auto[1]
    using disconnected_nodes_eq_h3[THEN select_result_eq, simplified] by auto

  have "acyclic (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def)
  have "parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h =
parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2"
    using ‹local.a_host_shadow_root_rel h = local.a_host_shadow_root_rel h2›
      ‹local.a_ptr_disconnected_node_rel h = local.a_ptr_disconnected_node_rel h2› ‹parent_child_rel h =
parent_child_rel h2›
    by auto
  have "parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2 =
parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3"
    using ‹local.a_host_shadow_root_rel h2 = local.a_host_shadow_root_rel h3›
      ‹local.a_ptr_disconnected_node_rel h2 = local.a_ptr_disconnected_node_rel h3› ‹parent_child_rel h2
= parent_child_rel h3›
    by auto
  have "parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h' =
{(cast document_ptr, cast new_element_ptr)} ∪ parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected
h3"
    by (simp add: ‹local.a_host_shadow_root_rel h3 = local.a_host_shadow_root_rel h'›
        ‹local.a_ptr_disconnected_node_rel h' = {(castdocument_ptr2object_ptr document_ptr, castelement_ptr2object_ptr
new_element_ptr)} ∪
local.a_ptr_disconnected_node_rel h3› ‹parent_child_rel h3 = parent_child_rel h'›)

  have "⋀a b. (a, b) ∈ parent_child_rel h3 ⟹ a ≠ cast new_element_ptr"
    using CD.parent_child_rel_parent_in_heap ‹parent_child_rel h = parent_child_rel h2›
      ‹parent_child_rel h2 = parent_child_rel h3› element_ptr_kinds_commutes h2 new_element_ptr
      new_element_ptr_not_in_heap node_ptr_kinds_commutes
    by blast
  moreover
  have "⋀a b. (a, b) ∈ a_host_shadow_root_rel h3 ⟹ a ≠ cast new_element_ptr"
    using shadow_root_eq_h2 shadow_root_none
    by(auto simp add: a_host_shadow_root_rel_def)
  moreover
  have "⋀a b. (a, b) ∈ a_ptr_disconnected_node_rel h3 ⟹ a ≠ cast new_element_ptr"
    by(auto simp add: a_ptr_disconnected_node_rel_def)
  moreover
  have "cast new_element_ptr ∉ {x. (x, cast document_ptr) ∈
(parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)*}"
    by (smt Un_iff ‹⋀b a. (a, b) ∈ local.a_host_shadow_root_rel h3 ⟹
a ≠ castelement_ptr2object_ptr new_element_ptr› ‹⋀b a. (a, b) ∈ local.a_ptr_disconnected_node_rel h3 ⟹
a ≠ castelement_ptr2object_ptr new_element_ptr› ‹⋀b a. (a, b) ∈ parent_child_rel h3 ⟹
a ≠ castelement_ptr2object_ptr new_element_ptr› cast_document_ptr_not_node_ptr(1) converse_rtranclE mem_Collect_eq)
  moreover
  have "acyclic (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)"
    using ‹acyclic (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel
```

```
h)›
      ‹parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel h =
parent_child_rel h2 ∪ local.a_host_shadow_root_rel h2 ∪ local.a_ptr_disconnected_node_rel h2›
      ‹parent_child_rel h2 ∪ local.a_host_shadow_root_rel h2 ∪ local.a_ptr_disconnected_node_rel h2 =
parent_child_rel h3 ∪ local.a_host_shadow_root_rel h3 ∪ local.a_ptr_disconnected_node_rel h3›
    by auto
  ultimately have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel
h')"
    by(simp add: ‹parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h' =
{(cast document_ptr, cast new_element_ptr)} ∪ parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected
h3›)


  show " heap_is_wellformed h' "
    using ‹acyclic (parent_child_rel h' ∪ local.a_host_shadow_root_rel h' ∪ local.a_ptr_disconnected_node_rel
h')›
    by(simp add: heap_is_wellformed_def CD.heap_is_wellformed_impl ‹local.CD.a_heap_is_wellformed h'›
        ‹local.a_all_ptrs_in_heap h'› ‹local.a_distinct_lists h'› ‹local.a_shadow_root_valid h'›)
qed
end

interpretation i_create_element_wf?: l_create_element_wf_Shadow_DOM known_ptr known_ptrs type_wf
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs heap_is_wellformed
  parent_child_rel set_tag_name set_tag_name_locs set_disconnected_nodes
  set_disconnected_nodes_locs create_element get_shadow_root get_shadow_root_locs get_tag_name
  get_tag_name_locs heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs DocumentClass.known_ptr DocumentClass.type_wf
  by(auto simp add: l_create_element_wf_Shadow_DOM_def instances)
declare l_create_element_wf_Core_DOM_axioms [instances]
```

create_character_data

```
locale l_create_character_data_wf_Shadow_DOM =
  l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_heap_is_wellformed_Shadow_DOM get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nod
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf
  heap_is_wellformed parent_child_rel
  heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs +
  l_create_character_data_Shadow_DOM get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs set_val set_val_locs create_character_data known_ptr
  type_wf_Core_DOM known_ptr_Core_DOM
  + l_new_character_data_get_disconnected_nodes
  get_disconnected_nodes get_disconnected_nodes_locs

+ l_set_val_get_disconnected_nodes
type_wf set_val set_val_locs get_disconnected_nodes get_disconnected_nodes_locs
+ l_new_character_data_get_child_nodes
type_wf known_ptr get_child_nodes get_child_nodes_locs
+ l_set_val_get_child_nodes
type_wf set_val set_val_locs known_ptr get_child_nodes get_child_nodes_locs
+ l_set_disconnected_nodes_get_child_nodes
set_disconnected_nodes set_disconnected_nodes_locs get_child_nodes get_child_nodes_locs
+ l_set_disconnected_nodes
type_wf set_disconnected_nodes set_disconnected_nodes_locs
+ l_set_disconnected_nodes_get_disconnected_nodes
type_wf get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
set_disconnected_nodes_locs
+ l_set_val_get_shadow_root type_wf set_val set_val_locs get_shadow_root get_shadow_root_locs
+ l_set_disconnected_nodes_get_shadow_root set_disconnected_nodes set_disconnected_nodes_locs
get_shadow_root get_shadow_root_locs
+ l_new_character_data_get_tag_name
get_tag_name get_tag_name_locs
```

```
+ l_set_val_get_tag_name type_wf set_val set_val_locs get_tag_name get_tag_name_locs
+ l_get_tag_name type_wf get_tag_name get_tag_name_locs
+ l_set_disconnected_nodes_get_tag_name type_wf set_disconnected_nodes set_disconnected_nodes_locs
get_tag_name get_tag_name_locs
+ l_new_character_data
type_wf
+ l_known_ptrs
known_ptr known_ptrs
for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
  and known_ptrs :: "(_) heap ⇒ bool"
  and type_wf :: "(_) heap ⇒ bool"
  and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
  and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
  and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed :: "(_) heap ⇒ bool"
  and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
  and set_tag_name :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
  and set_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
  and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
  and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
  and create_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
  and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
  and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
  and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
  and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  and set_val :: "(_) character_data_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
  and set_val_locs :: "(_) character_data_ptr ⇒ ((_) heap, exception, unit) prog set"
  and create_character_data ::
  "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) character_data_ptr) prog"
  and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
  and type_wf_Core_DOM :: "(_) heap ⇒ bool"
begin
lemma create_character_data_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ create_character_data document_ptr text →_h h'"
    and "type_wf h"
    and "known_ptrs h"
  shows "heap_is_wellformed h'" and "type_wf h'" and "known_ptrs h'"
proof -
  obtain new_character_data_ptr h2 h3 disc_nodes_h3 where
    new_character_data_ptr: "h ⊢ new_character_data →_r new_character_data_ptr" and
    h2: "h ⊢ new_character_data →_h h2" and
    h3: "h2 ⊢ set_val new_character_data_ptr text →_h h3" and
    disc_nodes_h3: "h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes_h3" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_character_data_ptr # disc_nodes_h3) →_h h'"
    using assms(2)
    by(auto simp add: CD.create_character_data_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF CD.get_disconnected_nodes_pure, rotated] )
  then have "h ⊢ create_character_data document_ptr text →_r new_character_data_ptr"
    apply(auto simp add: CD.create_character_data_def intro!: bind_returns_result_I)[1]
      apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
     apply (metis is_OK_returns_heap_E is_OK_returns_result_I local.CD.get_disconnected_nodes_pure
        pure_returns_heap_eq)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
```

```
  have "new_character_data_ptr ∉ set |h ⊢ character_data_ptr_kinds_M|_r"
    using new_character_data_ptr CharacterDataMonad.ptr_kinds_ptr_kinds_M h2
    using new_character_data_ptr_not_in_heap by blast
  then have "cast new_character_data_ptr ∉ set |h ⊢ node_ptr_kinds_M|_r"
    by simp
  then have "cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r"
    by simp



  have object_ptr_kinds_eq_h:
    "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    using new_character_data_new_ptr h2 new_character_data_ptr by blast
  then have node_ptr_kinds_eq_h:
    "node_ptr_kinds h2 = node_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have character_data_ptr_kinds_eq_h:
    "character_data_ptr_kinds h2 = character_data_ptr_kinds h |∪| {|new_character_data_ptr|}"
    apply(simp add: character_data_ptr_kinds_def)
    by force
  have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF CD.set_val_writes h3])
    using CD.set_val_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)

  have "known_ptr (cast new_character_data_ptr)"
    using ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr›
      local.create_character_data_known_ptr by blast
  then
  have "known_ptrs h2"
    using known_ptrs_new_ptr object_ptr_kinds_eq_h ‹known_ptrs h› h2
    by blast
  then
  have "known_ptrs h3"
    using known_ptrs_preserved object_ptr_kinds_eq_h2 by blast
  then
  show "known_ptrs h'"
    using known_ptrs_preserved object_ptr_kinds_eq_h3 by blast
```

```
  have "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      CD.get_disconnected_nodes_ptr_in_heap ‹type_wf h› document_ptr_kinds_def
    by (metis is_OK_returns_result_I)

  have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_character_data_ptr
                 ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads h2 get_child_nodes_new_character_data[rotated, OF new_character_data_ptr
h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h:
    "⋀ptr'. ptr' ≠ cast new_character_data_ptr
      ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have object_ptr_kinds_eq_h:
    "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    using new_character_data_new_ptr h2 new_character_data_ptr by blast
  then have node_ptr_kinds_eq_h:
    "node_ptr_kinds h2 = node_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have character_data_ptr_kinds_eq_h:
    "character_data_ptr_kinds h2 = character_data_ptr_kinds h |∪| {|new_character_data_ptr|}"
    apply(simp add: character_data_ptr_kinds_def)
    by force
  have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF CD.set_val_writes h3])
    using CD.set_val_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)
  then have character_data_ptr_kinds_eq_h2: "character_data_ptr_kinds h3 = character_data_ptr_kinds h2"
    by(simp add: character_data_ptr_kinds_def)
  have element_ptr_kinds_eq_h2: "element_ptr_kinds h3 = element_ptr_kinds h2"
    using node_ptr_kinds_eq_h2
    by(simp add: element_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)
  then have character_data_ptr_kinds_eq_h3: "character_data_ptr_kinds h' = character_data_ptr_kinds h3"
    by(simp add: character_data_ptr_kinds_def)
  have element_ptr_kinds_eq_h3: "element_ptr_kinds h' = element_ptr_kinds h3"
```

```
    using node_ptr_kinds_eq_h3
    by(simp add: element_ptr_kinds_def)


  have "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      CD.get_disconnected_nodes_ptr_in_heap ‹type_wf h› document_ptr_kinds_def
    by (metis is_OK_returns_result_I)

  have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_character_data_ptr
              ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads h2 get_child_nodes_new_character_data[rotated, OF new_character_data_ptr
h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h: "⋀ptr'. ptr' ≠ cast new_character_data_ptr
                              ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force

  have "h2 ⊢ get_child_nodes (cast new_character_data_ptr) →r []"
    using new_character_data_ptr h2 new_character_data_ptr_in_heap[OF h2 new_character_data_ptr]
      new_character_data_is_character_data_ptr[OF new_character_data_ptr]
      new_character_data_no_child_nodes
    by blast
  have disconnected_nodes_eq_h:
    "⋀doc_ptr disc_nodes. h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                        = h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using CD.get_disconnected_nodes_reads h2
      get_disconnected_nodes_new_character_data[OF new_character_data_ptr h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have disconnected_nodes_eq2_h:
    "⋀doc_ptr. |h ⊢ get_disconnected_nodes doc_ptr|r = |h2 ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force
  have tag_name_eq_h:
    "⋀ptr' disc_nodes. h ⊢ get_tag_name ptr' →r disc_nodes
                                        = h2 ⊢ get_tag_name ptr' →r disc_nodes"
    using get_tag_name_reads h2
      get_tag_name_new_character_data[OF new_character_data_ptr h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have tag_name_eq2_h: "⋀ptr'. |h ⊢ get_tag_name ptr'|r = |h2 ⊢ get_tag_name ptr'|r"
    using select_result_eq by force

  have children_eq_h2:
    "⋀ptr' children. h2 ⊢ get_child_nodes ptr' →r children = h3 ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads CD.set_val_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_val_get_child_nodes)
  then have children_eq2_h2:
    "⋀ptr'. |h2 ⊢ get_child_nodes ptr'|r = |h3 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h2:
    "⋀doc_ptr disc_nodes. h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                         = h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using CD.get_disconnected_nodes_reads CD.set_val_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_val_get_disconnected_nodes)
  then have disconnected_nodes_eq2_h2:
    "⋀doc_ptr. |h2 ⊢ get_disconnected_nodes doc_ptr|r = |h3 ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force
  have tag_name_eq_h2:
    "⋀ptr' disc_nodes. h2 ⊢ get_tag_name ptr' →r disc_nodes
```

```
                                        = h3 ⊢ get_tag_name ptr' →r disc_nodes"
    using get_tag_name_reads CD.set_val_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_val_get_tag_name)
  then have tag_name_eq2_h2: "⋀ptr'. |h2 ⊢ get_tag_name ptr'|r = |h3 ⊢ get_tag_name ptr'|r"
    using select_result_eq by force

  have "type_wf h2"
    using ‹type_wf h› new_character_data_types_preserved h2 by blast
  then have "type_wf h3"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF CD.set_val_writes h3]
    using  set_val_types_preserved
    by(auto simp add: reflp_def transp_def)
  then show "type_wf h'"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h']
    using  set_disconnected_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  have children_eq_h3:
    "⋀ptr' children. h3 ⊢ get_child_nodes ptr' →r children = h' ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_child_nodes)
  then have children_eq2_h3:
    " ⋀ptr'. |h3 ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h3: "⋀doc_ptr disc_nodes. document_ptr ≠ doc_ptr
    ⟹ h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                          = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using CD.get_disconnected_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
  then have disconnected_nodes_eq2_h3: "⋀doc_ptr. document_ptr ≠ doc_ptr
            ⟹ |h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force
  have tag_name_eq_h3:
    "⋀ptr' disc_nodes. h3 ⊢ get_tag_name ptr' →r disc_nodes
                                        = h' ⊢ get_tag_name ptr' →r disc_nodes"
    using get_tag_name_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_tag_name)
  then have tag_name_eq2_h3: "⋀ptr'. |h3 ⊢ get_tag_name ptr'|r = |h' ⊢ get_tag_name ptr'|r"
    using select_result_eq by force

  have disc_nodes_document_ptr_h2: "h2 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h2 disc_nodes_h3 by auto
  then have disc_nodes_document_ptr_h: "h ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h by auto
  then have "cast new_character_data_ptr ∉ set disc_nodes_h3"
    using ‹heap_is_wellformed h› using ‹cast new_character_data_ptr ∉ set |h ⊢ node_ptr_kinds_M|r›
      a_all_ptrs_in_heap_def heap_is_wellformed_def
    using NodeMonad.ptr_kinds_ptr_kinds_M local.heap_is_wellformed_disc_nodes_in_heap by blast

  have "acyclic (parent_child_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)
  also have "parent_child_rel h = parent_child_rel h2"
  proof(auto simp add: CD.parent_child_rel_def)[1]
    fix a x
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
    then show "a |∈| object_ptr_kinds h2"
```

```
      by (simp add: object_ptr_kinds_eq_h)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|_r"
    then show "x ∈ set |h2 ⊢ get_child_nodes a|_r"
      by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
          ‹cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r› children_eq2_h)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h2"
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
    then show "a |∈| object_ptr_kinds h"
      using object_ptr_kinds_eq_h ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →_r []›
      by(auto)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h2"
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
    then show "x ∈ set |h ⊢ get_child_nodes a|_r"
      by (metis (no_types, lifting) ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →_r []›
          children_eq2_h empty_iff empty_set image_eqI select_result_I2)
  qed
  also have "... = parent_child_rel h3"
    by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
  also have "... = parent_child_rel h'"
    by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
  finally have "CD.a_acyclic_heap h'"
    by (simp add: CD.acyclic_heap_def)

  have "CD.a_all_ptrs_in_heap h"
    using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_all_ptrs_in_heap h2"
    apply(auto simp add: CD.a_all_ptrs_in_heap_def)[1]
    using node_ptr_kinds_eq_h ‹cast new_character_data_ptr ∉ set |h ⊢ node_ptr_kinds_M|_r›
      ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →_r []›
     apply (metis (no_types, opaque_lifting) NodeMonad.ptr_kinds_ptr_kinds_M ‹parent_child_rel h = parent_child_re
h2›
        children_eq2_h finsert_iff funion_finsert_right CD.parent_child_rel_child
        CD.parent_child_rel_parent_in_heap node_ptr_kinds_commutes object_ptr_kinds_eq_h
        select_result_I2 subsetD sup_bot.right_neutral)
    by (metis (no_types, opaque_lifting) CD.get_child_nodes_ok CD.get_child_nodes_ptr_in_heap
        ‹h2 ⊢ get_child_nodes (cast_character_data_ptr2object_ptr new_character_data_ptr) →_r []› assms(3) assms(4)
        children_eq_h disconnected_nodes_eq2_h document_ptr_kinds_eq_h is_OK_returns_result_I
        local.known_ptrs_known_ptr node_ptr_kinds_commutes returns_result_select_result subset_code(1))
  then have "CD.a_all_ptrs_in_heap h3"
    by (simp add: children_eq2_h2 disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
        CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h2 object_ptr_kinds_eq_h2)
  then have "CD.a_all_ptrs_in_heap h'"
    by (smt (verit) character_data_ptr_kinds_commutes character_data_ptr_kinds_eq_h2 children_eq2_h3
        disc_nodes_h3 disconnected_nodes_eq2_h3 document_ptr_kinds_eq_h3 h' h2 local.CD.a_all_ptrs_in_heap_def
        local.set_disconnected_nodes_get_disconnected_nodes new_character_data_ptr new_character_data_ptr_in_heap
        node_ptr_kinds_eq_h3 object_ptr_kinds_eq_h3 select_result_I2 set_ConsD subset_code(1))

  have "⋀p. p |∈| object_ptr_kinds h ⟹ cast new_character_data_ptr ∉ set |h ⊢ get_child_nodes p|_r"
    using ‹heap_is_wellformed h› ‹cast new_character_data_ptr ∉ set |h ⊢ node_ptr_kinds_M|_r›
      heap_is_wellformed_children_in_heap
    by (meson NodeMonad.ptr_kinds_ptr_kinds_M CD.a_all_ptrs_in_heap_def assms(3) assms(4) fset_mp
        fset_of_list_elem CD.get_child_nodes_ok known_ptrs_known_ptr returns_result_select_result)
  then have "⋀p. p |∈| object_ptr_kinds h2 ⟹ cast new_character_data_ptr ∉ set |h2 ⊢ get_child_nodes
p|_r"
    using children_eq2_h
    apply(auto simp add: object_ptr_kinds_eq_h)[1]
```

```
    using ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →r []› apply auto[1]
    by (metis ObjectMonad.ptr_kinds_ptr_kinds_M ‹cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|r›)
  then have "⋀p. p |∈| object_ptr_kinds h3 ⟹ cast new_character_data_ptr ∉ set |h3 ⊢ get_child_nodes
p|r"
    using object_ptr_kinds_eq_h2 children_eq2_h2 by auto
  then have new_character_data_ptr_not_in_any_children:
    "⋀p. p |∈| object_ptr_kinds h' ⟹ cast new_character_data_ptr ∉ set |h' ⊢ get_child_nodes p|r"
    using object_ptr_kinds_eq_h3 children_eq2_h3 by auto

  have "CD.a_distinct_lists h"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_distinct_lists h2"
    using ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →r []›
    apply(auto simp add: CD.a_distinct_lists_def object_ptr_kinds_eq_h document_ptr_kinds_eq_h
        disconnected_nodes_eq2_h intro!: distinct_concat_map_I)[1]
      apply (metis distinct_sorted_list_of_set finite_fset sorted_list_of_set_insert_remove)
     apply(case_tac "x=cast new_character_data_ptr")
      apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
     apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
    apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
    apply (metis IntI assms(1) assms(3) assms(4) empty_iff CD.get_child_nodes_ok
        local.heap_is_wellformed_one_parent local.known_ptrs_known_ptr
        returns_result_select_result)
    apply(auto simp add: children_eq2_h[symmetric] insort_split dest: distinct_concat_map_E(2))[1]
    thm children_eq2_h

    using ‹CD.a_distinct_lists h› ‹type_wf h2› disconnected_nodes_eq_h document_ptr_kinds_eq_h
      CD.distinct_lists_no_parent get_disconnected_nodes_ok returns_result_select_result
    by metis
  then have "CD.a_distinct_lists h3"
    by(auto simp add: CD.a_distinct_lists_def disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
        children_eq2_h2 object_ptr_kinds_eq_h2)[1]
  then have "CD.a_distinct_lists h'"
  proof(auto simp add: CD.a_distinct_lists_def disconnected_nodes_eq2_h3 children_eq2_h3
      object_ptr_kinds_eq_h3 document_ptr_kinds_eq_h3 intro!: distinct_concat_map_I)[1]
    fix x
    assume "distinct (concat (map (λdocument_ptr. |h3 ⊢ get_disconnected_nodes document_ptr|r)
                    (sorted_list_of_set (fset (document_ptr_kinds h3)))))"
      and "x |∈| document_ptr_kinds h3"
    then show "distinct |h' ⊢ get_disconnected_nodes x|r"
      using document_ptr_kinds_eq_h3 disconnected_nodes_eq_h3 h' set_disconnected_nodes_get_disconnected_nodes
      by (metis (no_types, opaque_lifting) ‹castcharacter_data_ptr2node_ptr new_character_data_ptr ∉ set disc_nodes_h
          ‹type_wf h2› assms(1) disc_nodes_document_ptr_h disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3
          disconnected_nodes_eq_h distinct.simps(2) document_ptr_kinds_eq_h2 local.get_disconnected_nodes_ok
          local.heap_is_wellformed_disconnected_nodes_distinct returns_result_select_result select_result_I2)
  next
    fix x y xa
    assume "distinct (concat (map (λdocument_ptr. |h3 ⊢ get_disconnected_nodes document_ptr|r)
                                    (sorted_list_of_set (fset (document_ptr_kinds h3)))))"
      and "x |∈| document_ptr_kinds h3"
      and "y |∈| document_ptr_kinds h3"
      and "x ≠ y"
      and "xa ∈ set |h' ⊢ get_disconnected_nodes x|r"
      and "xa ∈ set |h' ⊢ get_disconnected_nodes y|r"
    moreover have "set |h3 ⊢ get_disconnected_nodes x|r ∩ set |h3 ⊢ get_disconnected_nodes y|r = {}"
      using calculation by(auto dest: distinct_concat_map_E(1))
    ultimately show "False"
      using NodeMonad.ptr_kinds_ptr_kinds_M ‹castcharacter_data_ptr2node_ptr new_character_data_ptr ∉ set
|h ⊢ node_ptr_kinds_M|r›

      by (smt (verit) local.CD.a_all_ptrs_in_heap_def ‹CD.a_all_ptrs_in_heap h› disc_nodes_document_ptr_h2
          disconnected_nodes_eq2_h
```

```
        disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3 disjoint_iff_not_equal
        document_ptr_kinds_eq_h document_ptr_kinds_eq_h2 h'
        l_set_disconnected_nodes_get_disconnected_nodes.set_disconnected_nodes_get_disconnected_nodes
        local.a_all_ptrs_in_heap_def local.l_set_disconnected_nodes_get_disconnected_nodes_axioms
        select_result_I2 set_ConsD subsetD)
next
  fix x xa xb
  assume 2: "(⋃x∈fset (object_ptr_kinds h3). set |h' ⊢ get_child_nodes x|r)
              ∩ (⋃x∈fset (document_ptr_kinds h3). set |h3 ⊢ get_disconnected_nodes x|r) = {}"
    and 3: "xa |∈| object_ptr_kinds h3"
    and 4: "x ∈ set |h' ⊢ get_child_nodes xa|r"
    and 5: "xb |∈| document_ptr_kinds h3"
    and 6: "x ∈ set |h' ⊢ get_disconnected_nodes xb|r"
  show "False"
    using disc_nodes_document_ptr_h disconnected_nodes_eq2_h3
    apply(cases "document_ptr = xb")
     apply (metis (no_types, lifting) "3" "4" "5" "6" CD.distinct_lists_no_parent
        ‹local.CD.a_distinct_lists h2› ‹type_wf h'› children_eq2_h2 children_eq2_h3 disc_nodes_document_ptr_h2
        document_ptr_kinds_eq_h3 h' local.get_disconnected_nodes_ok local.set_disconnected_nodes_get_disconnected
        new_character_data_ptr_not_in_any_children object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 returns_result_e
        returns_result_select_result set_ConsD)
    by (metis "3" "4" "5" "6" CD.distinct_lists_no_parent ‹local.CD.a_distinct_lists h3› ‹type_wf h3›
        children_eq2_h3 local.get_disconnected_nodes_ok returns_result_select_result)
qed

have "CD.a_owner_document_valid h"
  using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
then have "CD.a_owner_document_valid h'"
  using disc_nodes_h3 ‹document_ptr |∈| document_ptr_kinds h›
  apply(simp add: CD.a_owner_document_valid_def)
  apply(simp add: object_ptr_kinds_eq_h object_ptr_kinds_eq_h3 )
  apply(simp add: object_ptr_kinds_eq_h2)
  apply(simp add: document_ptr_kinds_eq_h document_ptr_kinds_eq_h3 )
  apply(simp add: document_ptr_kinds_eq_h2)
  apply(simp add: node_ptr_kinds_eq_h node_ptr_kinds_eq_h3 )
  apply(simp add: node_ptr_kinds_eq_h2 node_ptr_kinds_eq_h )
  apply(auto simp add: children_eq2_h2[symmetric] children_eq2_h3[symmetric] disconnected_nodes_eq2_h
      disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3)[1]
   apply (metis (no_types, lifting) document_ptr_kinds_eq_h h' list.set_intros(1)
      local.set_disconnected_nodes_get_disconnected_nodes select_result_I2)
  apply(simp add: object_ptr_kinds_eq_h)
  by (metis (mono_tags, opaque_lifting) ‹cast_character_data_ptr2object_ptr new_character_data_ptr ∉ set |h
⊢ object_ptr_kinds_M|r›
      children_eq2_h disconnected_nodes_eq2_h3 document_ptr_kinds_eq_h h'
      l_ptr_kinds_M.ptr_kinds_ptr_kinds_M
      l_set_disconnected_nodes_get_disconnected_nodes.set_disconnected_nodes_get_disconnected_nodes
      list.set_intros(2) local.l_set_disconnected_nodes_get_disconnected_nodes_axioms
      object_ptr_kinds_M_def
      select_result_I2)



have shadow_root_ptr_kinds_eq_h: "shadow_root_ptr_kinds h2 = shadow_root_ptr_kinds h"
  using document_ptr_kinds_eq_h
  by(auto simp add: shadow_root_ptr_kinds_def)
have shadow_root_ptr_kinds_eq_h2: "shadow_root_ptr_kinds h3 = shadow_root_ptr_kinds h2"
  using document_ptr_kinds_eq_h2
  by(auto simp add: shadow_root_ptr_kinds_def)
have shadow_root_ptr_kinds_eq_h3: "shadow_root_ptr_kinds h' = shadow_root_ptr_kinds h3"
  using document_ptr_kinds_eq_h3
```

```
    by(auto simp add: shadow_root_ptr_kinds_def)


  have shadow_root_eq_h: "⋀character_data_ptr shadow_root_opt.
h ⊢ get_shadow_root character_data_ptr →r shadow_root_opt =
h2 ⊢ get_shadow_root character_data_ptr →r shadow_root_opt"
    using get_shadow_root_reads h2 get_shadow_root_new_character_data[rotated, OF h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    using local.get_shadow_root_locs_impl new_character_data_ptr apply blast
    using local.get_shadow_root_locs_impl new_character_data_ptr by blast


  have shadow_root_eq_h2:
    "⋀ptr' children. h2 ⊢ get_shadow_root ptr' →r children = h3 ⊢ get_shadow_root ptr' →r children"
    using get_shadow_root_reads set_val_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_val_get_shadow_root)
  have shadow_root_eq_h3:
    "⋀ptr' children. h3 ⊢ get_shadow_root ptr' →r children = h' ⊢ get_shadow_root ptr' →r children"
    using get_shadow_root_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    using set_disconnected_nodes_get_shadow_root
    by(auto simp add: set_disconnected_nodes_get_shadow_root)


  have "a_all_ptrs_in_heap h"
    by (simp add: assms(1) local.a_all_ptrs_in_heap_def local.get_shadow_root_shadow_root_ptr_in_heap)
  then have "a_all_ptrs_in_heap h2"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h)[1]
    using returns_result_eq shadow_root_eq_h by fastforce
  then have "a_all_ptrs_in_heap h3"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h2)[1]
    using shadow_root_eq_h2 by blast
  then have "a_all_ptrs_in_heap h'"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h3)[1]
    by (simp add: shadow_root_eq_h3)

  have "a_distinct_lists h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then have "a_distinct_lists h2"
    apply(auto simp add: a_distinct_lists_def character_data_ptr_kinds_eq_h)[1]
    apply(auto simp add: distinct_insort intro!: distinct_concat_map_I split: option.splits)[1]
    by (metis ‹type_wf h2› assms(1) assms(3) local.get_shadow_root_ok local.shadow_root_same_host
        returns_result_select_result shadow_root_eq_h)
  then have "a_distinct_lists h3"
    by(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h2 select_result_eq[OF shadow_root_eq_h2])
  then have "a_distinct_lists h'"
    by(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h3 select_result_eq[OF shadow_root_eq_h3])


  have "a_shadow_root_valid h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then have "a_shadow_root_valid h2"
    by(auto simp add: a_shadow_root_valid_def shadow_root_ptr_kinds_eq_h element_ptr_kinds_eq_h
        select_result_eq[OF shadow_root_eq_h] tag_name_eq2_h)
  then have "a_shadow_root_valid h3"
    by(auto simp add: a_shadow_root_valid_def shadow_root_ptr_kinds_eq_h2 element_ptr_kinds_eq_h2
        select_result_eq[OF shadow_root_eq_h2] tag_name_eq2_h2)
  then have "a_shadow_root_valid h'"
    by(auto simp add: a_shadow_root_valid_def shadow_root_ptr_kinds_eq_h3 element_ptr_kinds_eq_h3
```

```
        select_result_eq[OF shadow_root_eq_h3] tag_name_eq2_h3)


  have "a_host_shadow_root_rel h = a_host_shadow_root_rel h2"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h select_result_eq[OF shadow_root_eq_h])
  have "a_host_shadow_root_rel h2 = a_host_shadow_root_rel h3"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h2 select_result_eq[OF shadow_root_eq_h2])
  have "a_host_shadow_root_rel h3 = a_host_shadow_root_rel h'"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h3 select_result_eq[OF shadow_root_eq_h3])

  have "a_ptr_disconnected_node_rel h = a_ptr_disconnected_node_rel h2"
    by(simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h disconnected_nodes_eq2_h)
  have "a_ptr_disconnected_node_rel h2 = a_ptr_disconnected_node_rel h3"
    by(simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h2 disconnected_nodes_eq2_h2)

  have "h' ⊢ get_disconnected_nodes document_ptr →r cast new_character_data_ptr # disc_nodes_h3"
    using h' local.set_disconnected_nodes_get_disconnected_nodes by auto
  have " document_ptr |∈| document_ptr_kinds h3"
    by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h document_ptr_kinds_eq_h2)
  have "cast new_character_data_ptr ∈ set |h' ⊢ get_disconnected_nodes document_ptr|r"
    using ‹h' ⊢ get_disconnected_nodes document_ptr →r cast new_character_data_ptr # disc_nodes_h3› by
auto

  have "a_ptr_disconnected_node_rel h' = {(cast document_ptr, cast new_character_data_ptr)} ∪ a_ptr_disconnected_n
h3"
    apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h3 disconnected_nodes_eq2_h3)[1]
      apply(case_tac "aa = document_ptr")
    using disc_nodes_h3 h' ‹h' ⊢ get_disconnected_nodes document_ptr →r cast new_character_data_ptr # disc_nodes_
      apply(auto)[1]
    using disconnected_nodes_eq2_h3 apply auto[1]
    using ‹h' ⊢ get_disconnected_nodes document_ptr →r cast new_character_data_ptr # disc_nodes_h3›
    using ‹cast new_character_data_ptr ∈ set |h' ⊢ get_disconnected_nodes document_ptr|r›
    using ‹document_ptr |∈| document_ptr_kinds h3› apply auto[1]
    apply(case_tac "document_ptr = aa")
    using ‹h' ⊢ get_disconnected_nodes document_ptr →r cast new_character_data_ptr # disc_nodes_h3› disc_nodes_h3
apply auto[1]
    using disconnected_nodes_eq_h3[THEN select_result_eq, simplified] by auto

  have "acyclic (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def)
  have "parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h =
parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2"
    using ‹local.a_host_shadow_root_rel h = local.a_host_shadow_root_rel h2›
      ‹local.a_ptr_disconnected_node_rel h = local.a_ptr_disconnected_node_rel h2› ‹parent_child_rel h =
parent_child_rel h2› by auto
  have "parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2 =
parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3"
    using ‹local.a_host_shadow_root_rel h2 = local.a_host_shadow_root_rel h3›
      ‹local.a_ptr_disconnected_node_rel h2 = local.a_ptr_disconnected_node_rel h3› ‹parent_child_rel h2
= parent_child_rel h3› by auto
  have "parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h' =
{(cast document_ptr, cast new_character_data_ptr)} ∪ parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪
a_ptr_disconnected_node_rel h3"
    by (simp add: ‹local.a_host_shadow_root_rel h3 = local.a_host_shadow_root_rel h'›
        ‹local.a_ptr_disconnected_node_rel h' = {(cast_document_ptr2object_ptr document_ptr, cast new_character_data_p
∪
local.a_ptr_disconnected_node_rel h3› ‹parent_child_rel h3 = parent_child_rel h'›)

  have "⋀a b. (a, b) ∈ parent_child_rel h3 ⟹ a ≠ cast new_character_data_ptr"
    using CD.parent_child_rel_parent_in_heap ‹parent_child_rel h = parent_child_rel h2›
      ‹parent_child_rel h2 = parent_child_rel h3› character_data_ptr_kinds_commutes h2 new_character_data_ptr
```

```
      new_character_data_ptr_not_in_heap node_ptr_kinds_commutes by blast
  moreover
  have "⋀a b. (a, b) ∈ a_host_shadow_root_rel h3 ⟹ a ≠ cast new_character_data_ptr"
    using shadow_root_eq_h2
    by(auto simp add: a_host_shadow_root_rel_def)
  moreover
  have "⋀a b. (a, b) ∈ a_ptr_disconnected_node_rel h3 ⟹ a ≠ cast new_character_data_ptr"
    by(auto simp add: a_ptr_disconnected_node_rel_def)
  moreover
  have "cast new_character_data_ptr ∉ {x. (x, cast document_ptr) ∈
(parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)*}"
    by (smt Un_iff calculation(1) calculation(2) calculation(3) cast_document_ptr_not_node_ptr(2)
        converse_rtranclE mem_Collect_eq)
  moreover
  have "acyclic (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)"
    using ‹acyclic (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel
h)›
      ‹parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel h =
parent_child_rel h2 ∪ local.a_host_shadow_root_rel h2 ∪ local.a_ptr_disconnected_node_rel h2›
      ‹parent_child_rel h2 ∪ local.a_host_shadow_root_rel h2 ∪ local.a_ptr_disconnected_node_rel h2 =
parent_child_rel h3 ∪ local.a_host_shadow_root_rel h3 ∪ local.a_ptr_disconnected_node_rel h3›
    by auto
  ultimately have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel
h')"
    by(simp add: ‹parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h' =
{(cast document_ptr, cast new_character_data_ptr)} ∪ parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪
a_ptr_disconnected_node_rel h3›)


  have "CD.a_heap_is_wellformed h'"
    apply(simp add: CD.a_heap_is_wellformed_def)
    by (simp add: ‹local.CD.a_acyclic_heap h'› ‹local.CD.a_all_ptrs_in_heap h'›
        ‹local.CD.a_distinct_lists h'› ‹local.CD.a_owner_document_valid h'›)

  show " heap_is_wellformed h' "
    using ‹acyclic (parent_child_rel h' ∪ local.a_host_shadow_root_rel h' ∪ local.a_ptr_disconnected_node_rel
h')›
    by(simp add: heap_is_wellformed_def CD.heap_is_wellformed_impl ‹local.CD.a_heap_is_wellformed h'›
        ‹local.a_all_ptrs_in_heap h'› ‹local.a_distinct_lists h'› ‹local.a_shadow_root_valid h'›)
qed
end
```

create_document

```
locale l_create_document_wf_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf
  heap_is_wellformed parent_child_rel
  heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document get_disconnected_document_locs
  + l_new_document_get_disconnected_nodes
  get_disconnected_nodes get_disconnected_nodes_locs
  + l_create_document_Core_DOM
  create_document
  + l_new_document_get_child_nodes
  type_wf known_ptr get_child_nodes get_child_nodes_locs
  + l_get_tag_name type_wf get_tag_name get_tag_name_locs
  + l_new_document_get_tag_name get_tag_name get_tag_name_locs
  + l_get_disconnected_nodes_Shadow_DOM type_wf type_wf_Core_DOM get_disconnected_nodes get_disconnected_nodes_loc
  + l_new_document
  type_wf
  + l_known_ptrs
  known_ptr known_ptrs
```

```
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and set_val :: "(_) character_data_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
    and set_val_locs :: "(_) character_data_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and create_document :: "((_) heap, exception, (_) document_ptr) prog"
    and known_ptrs :: "(_) heap ⇒ bool"
begin

lemma create_document_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ create_document →_h h'"
    and "type_wf h"
    and "known_ptrs h"
  shows "heap_is_wellformed h'"
proof -
  obtain new_document_ptr where
    new_document_ptr: "h ⊢ new_document →_r new_document_ptr" and
    h': "h ⊢ new_document →_h h'"
    using assms(2)
    apply(simp add: create_document_def)
    using new_document_ok by blast

  have "new_document_ptr ∉ set |h ⊢ document_ptr_kinds_M|_r"
    using new_document_ptr DocumentMonad.ptr_kinds_ptr_kinds_M
    using new_document_ptr_not_in_heap h' by blast
  then have "cast new_document_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r"
    by simp

  have "new_document_ptr |∉| document_ptr_kinds h"
    using new_document_ptr DocumentMonad.ptr_kinds_ptr_kinds_M
    using new_document_ptr_not_in_heap h' by blast
  then have "cast new_document_ptr |∉| object_ptr_kinds h"
    by simp

  have object_ptr_kinds_eq: "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_document_ptr|}"
    using new_document_new_ptr h' new_document_ptr by blast
  then have node_ptr_kinds_eq: "node_ptr_kinds h' = node_ptr_kinds h"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have character_data_ptr_kinds_eq_h: "character_data_ptr_kinds h' = character_data_ptr_kinds h"
    by(simp add: character_data_ptr_kinds_def)
  have element_ptr_kinds_eq_h: "element_ptr_kinds h' = element_ptr_kinds h"
    using object_ptr_kinds_eq
    by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)
```

```
have document_ptr_kinds_eq_h: "document_ptr_kinds h' = document_ptr_kinds h |∪| {|new_document_ptr|}"
  using object_ptr_kinds_eq
  by (auto simp add: document_ptr_kinds_def)


have children_eq:
  "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_document_ptr
          ⟹ h ⊢ get_child_nodes ptr' →r children = h' ⊢ get_child_nodes ptr' →r children"
  using CD.get_child_nodes_reads h' get_child_nodes_new_document[rotated, OF new_document_ptr h']
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by blast+
then have children_eq2: "⋀ptr'. ptr' ≠ cast new_document_ptr
                                ⟹ |h ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
  using select_result_eq by force


have "h' ⊢ get_child_nodes (cast new_document_ptr) →r []"
  using new_document_ptr h' new_document_ptr_in_heap[OF h' new_document_ptr]
    new_document_is_document_ptr[OF new_document_ptr] new_document_no_child_nodes
  by blast
have disconnected_nodes_eq_h:
  "⋀doc_ptr disc_nodes. doc_ptr ≠ new_document_ptr
  ⟹ h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
  using CD.get_disconnected_nodes_reads h' get_disconnected_nodes_new_document_different_pointers new_document_pt
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by (metis(full_types) ‹⋀thesis. (⋀new_document_ptr.
          ⟦h ⊢ new_document →r new_document_ptr; h ⊢ new_document →h h'⟧ ⟹ thesis) ⟹ thesis›
      local.get_disconnected_nodes_new_document_different_pointers new_document_ptr)+
then have disconnected_nodes_eq2_h: "⋀doc_ptr. doc_ptr ≠ new_document_ptr
              ⟹ |h ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
  using select_result_eq by force
have "h' ⊢ get_disconnected_nodes new_document_ptr →r []"
  using h' local.new_document_no_disconnected_nodes new_document_ptr by blast

have "type_wf h'"
  using ‹type_wf h› new_document_types_preserved h' by blast

have "acyclic (parent_child_rel h)"
  using ‹heap_is_wellformed h›
  by (auto simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)
also have "parent_child_rel h = parent_child_rel h'"
proof(auto simp add: CD.parent_child_rel_def)[1]
  fix a x
  assume 0: "a |∈| object_ptr_kinds h"
    and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
  then show "a |∈| object_ptr_kinds h'"
    by (simp add: object_ptr_kinds_eq)
next
  fix a x
  assume 0: "a |∈| object_ptr_kinds h"
    and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
  then show "x ∈ set |h' ⊢ get_child_nodes a|r"
    by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹cast new_document_ptr ∉ set |h ⊢ object_ptr_kinds_M|r› children_eq2)
next
  fix a x
  assume 0: "a |∈| object_ptr_kinds h'"
    and 1: "x ∈ set |h' ⊢ get_child_nodes a|r"
  then show "a |∈| object_ptr_kinds h"
    using object_ptr_kinds_eq ‹h' ⊢ get_child_nodes (cast new_document_ptr) →r []›
    by(auto)
next
  fix a x
```

```
    assume 0: "a |∈| object_ptr_kinds h'"
      and 1: "x ∈ set |h' ⊢ get_child_nodes a|_r"
    then show "x ∈ set |h ⊢ get_child_nodes a|_r"
      by (metis (no_types, lifting) ‹h' ⊢ get_child_nodes (cast new_document_ptr) →_r []›
          children_eq2 empty_iff empty_set image_eqI select_result_I2)
  qed
  finally have "CD.a_acyclic_heap h'"
    by (simp add: CD.acyclic_heap_def)

  have "CD.a_all_ptrs_in_heap h"
    using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def )
  then have "CD.a_all_ptrs_in_heap h'"
    apply(auto simp add: CD.a_all_ptrs_in_heap_def)[1]
    using  ObjectMonad.ptr_kinds_ptr_kinds_M
      ‹cast_document_ptr2object_ptr new_document_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r›
      ‹parent_child_rel h = parent_child_rel h'› assms(1) children_eq fset_of_list_elem
      local.heap_is_wellformed_children_in_heap CD.parent_child_rel_child
      CD.parent_child_rel_parent_in_heap node_ptr_kinds_eq
     apply (metis (no_types, opaque_lifting) ‹h' ⊢ get_child_nodes (cast_document_ptr2object_ptr new_document_ptr)
→_r []›
        children_eq2 finsert_iff funion_finsert_right object_ptr_kinds_eq
        select_result_I2 subsetD sup_bot.right_neutral)
    by (metis (no_types, lifting) ‹h' ⊢ get_disconnected_nodes new_document_ptr →_r []› ‹type_wf h'›
        assms(1) disconnected_nodes_eq_h empty_iff empty_set local.get_disconnected_nodes_ok
        local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_eq returns_result_select_result select_result_I2

  have "CD.a_distinct_lists h"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_distinct_lists h'"
    using ‹h' ⊢ get_disconnected_nodes new_document_ptr →_r []›
      ‹h' ⊢ get_child_nodes (cast new_document_ptr) →_r []›

    apply(auto simp add: children_eq2[symmetric] CD.a_distinct_lists_def insort_split object_ptr_kinds_eq
        document_ptr_kinds_eq_h  disconnected_nodes_eq2_h intro!: distinct_concat_map_I)[1]
          apply (metis distinct_sorted_list_of_set finite_fset sorted_list_of_set_insert_remove)

         apply(auto simp add:  dest: distinct_concat_map_E)[1]
        apply(auto simp add:  dest: distinct_concat_map_E)[1]
    using ‹new_document_ptr |∉| document_ptr_kinds h›
       apply(auto simp add: distinct_insort dest: distinct_concat_map_E)[1]
      apply (metis assms(1) assms(3) disconnected_nodes_eq2_h get_disconnected_nodes_ok
        local.heap_is_wellformed_disconnected_nodes_distinct
        returns_result_select_result)
  proof -
    fix x :: "(_) document_ptr" and y :: "(_) document_ptr" and xa :: "(_) node_ptr"
    assume a1: "x ≠ y"
    assume a2: "x |∈| document_ptr_kinds h"
    assume a3: "x ≠ new_document_ptr"
    assume a4: "y |∈| document_ptr_kinds h"
    assume a5: "y ≠ new_document_ptr"
    assume a6: "distinct (concat (map (λdocument_ptr. |h ⊢ get_disconnected_nodes document_ptr|_r)
                                        (sorted_list_of_set (fset (document_ptr_kinds h)))))"
    assume a7: "xa ∈ set |h' ⊢ get_disconnected_nodes x|_r"
    assume a8: "xa ∈ set |h' ⊢ get_disconnected_nodes y|_r"
    have f9: "xa ∈ set |h ⊢ get_disconnected_nodes x|_r"
      using a7 a3 disconnected_nodes_eq2_h by presburger
    have f10: "xa ∈ set |h ⊢ get_disconnected_nodes y|_r"
      using a8 a5 disconnected_nodes_eq2_h by presburger
    have f11: "y ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))"
      using a4 by simp
    have "x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))"
      using a2 by simp
```

```
    then show False
      using f11 f10 f9 a6 a1 by (meson disjoint_iff_not_equal distinct_concat_map_E(1))
  next
    fix x xa xb
    assume 0: "h' ⊢ get_disconnected_nodes new_document_ptr →r []"
      and 1: "h' ⊢ get_child_nodes (cast_document_ptr2object_ptr new_document_ptr) →r []"
      and 2: "distinct (concat (map (λptr. |h ⊢ get_child_nodes ptr|r)
                                          (sorted_list_of_set (fset (object_ptr_kinds h)))))"
      and 3: "distinct (concat (map (λdocument_ptr. |h ⊢ get_disconnected_nodes document_ptr|r)
                                     (sorted_list_of_set (fset (document_ptr_kinds h)))))"
      and 4: "(⋃x∈fset (object_ptr_kinds h). set |h ⊢ get_child_nodes x|r)
                  ∩ (⋃x∈fset (document_ptr_kinds h). set |h ⊢ get_disconnected_nodes x|r) = {}"
      and 5: "x ∈ set |h ⊢ get_child_nodes xa|r"
      and 6: "x ∈ set |h' ⊢ get_disconnected_nodes xb|r"
      and 7: "xa |∈| object_ptr_kinds h"
      and 8: "xa ≠ cast_document_ptr2object_ptr new_document_ptr"
      and 9: "xb |∈| document_ptr_kinds h"
      and 10: "xb ≠ new_document_ptr"
    then show "False"

      by (metis ‹CD.a_distinct_lists h› assms(3) disconnected_nodes_eq2_h
          CD.distinct_lists_no_parent get_disconnected_nodes_ok
          returns_result_select_result)
  qed

  have "CD.a_owner_document_valid h"
    using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_owner_document_valid h'"
    apply(auto simp add: CD.a_owner_document_valid_def)[1]
    by (metis ‹cast_document_ptr2object_ptr new_document_ptr |∉| object_ptr_kinds h›
        children_eq2 disconnected_nodes_eq2_h document_ptr_kinds_commutes funion_iff
        node_ptr_kinds_eq object_ptr_kinds_eq)

  have shadow_root_eq_h: "⋀character_data_ptr shadow_root_opt. h ⊢ get_shadow_root character_data_ptr →r
shadow_root_opt =
h' ⊢ get_shadow_root character_data_ptr →r shadow_root_opt"
    using get_shadow_root_reads assms(2) get_shadow_root_new_document[rotated, OF h']
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    using local.get_shadow_root_locs_impl new_document_ptr apply blast
    using local.get_shadow_root_locs_impl new_document_ptr by blast


  have "a_all_ptrs_in_heap h"
    by (simp add: assms(1) local.a_all_ptrs_in_heap_def local.get_shadow_root_shadow_root_ptr_in_heap)
  then have "a_all_ptrs_in_heap h'"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_def document_ptr_kinds_eq_h)[1]
    using shadow_root_eq_h by fastforce

  have "a_distinct_lists h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then have "a_distinct_lists h'"
    apply(auto simp add: a_distinct_lists_def character_data_ptr_kinds_eq_h)[1]
    apply(auto simp add: distinct_insort intro!: distinct_concat_map_I split: option.splits)[1]
    by (metis ‹type_wf h'› assms(1) assms(3) local.get_shadow_root_ok local.shadow_root_same_host
        returns_result_select_result shadow_root_eq_h)


  have tag_name_eq_h:
    "⋀ptr' disc_nodes. h ⊢ get_tag_name ptr' →r disc_nodes
                                        = h' ⊢ get_tag_name ptr' →r disc_nodes"
    using get_tag_name_reads h'
      get_tag_name_new_document[OF new_document_ptr h']
```

```
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+

  have "a_shadow_root_valid h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then have "a_shadow_root_valid h'"
    using new_document_is_document_ptr[OF new_document_ptr]
    by(auto simp add: a_shadow_root_valid_def element_ptr_kinds_eq_h document_ptr_kinds_eq_h
        shadow_root_ptr_kinds_def select_result_eq[OF shadow_root_eq_h] select_result_eq[OF tag_name_eq_h]
        is_shadow_root_ptr_kind_document_ptr_def is_document_ptr_document_ptr_def cast_shadow_root_ptr2document_ptr_def
        split: option.splits)


  have "a_host_shadow_root_rel h = a_host_shadow_root_rel h'"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h select_result_eq[OF shadow_root_eq_h])

  have "a_ptr_disconnected_node_rel h = a_ptr_disconnected_node_rel h'"
    apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h disconnected_nodes_eq2_h)[1]
    using ‹new_document_ptr |∉| document_ptr_kinds h› disconnected_nodes_eq2_h apply fastforce
    using new_document_disconnected_nodes[OF h' new_document_ptr]
     apply(simp add: CD.get_disconnected_nodes_impl CD.a_get_disconnected_nodes_def)
    using ‹new_document_ptr |∉| document_ptr_kinds h› disconnected_nodes_eq2_h apply fastforce
    done

  have "acyclic (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def)
  moreover
  have "parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h =
parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h'"
    by (simp add: ‹local.a_host_shadow_root_rel h = local.a_host_shadow_root_rel h'›
        ‹local.a_ptr_disconnected_node_rel h = local.a_ptr_disconnected_node_rel h'›
        ‹parent_child_rel h = parent_child_rel h'›)
  ultimately have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel
h')"
    by simp

  have "CD.a_heap_is_wellformed h'"
    apply(simp add: CD.a_heap_is_wellformed_def)
    by (simp add: ‹local.CD.a_acyclic_heap h'› ‹local.CD.a_all_ptrs_in_heap h'›
        ‹local.CD.a_distinct_lists h'› ‹local.CD.a_owner_document_valid h'›)

  show "heap_is_wellformed h'"
    using CD.heap_is_wellformed_impl ‹acyclic (parent_child_rel h' ∪ local.a_host_shadow_root_rel h' ∪
local.a_ptr_disconnected_node_rel h')› ‹local.CD.a_heap_is_wellformed h'› ‹local.a_all_ptrs_in_heap h'›
      ‹local.a_distinct_lists h'› ‹local.a_shadow_root_valid h'› local.heap_is_wellformed_def by auto
qed
end

interpretation l_create_document_wf_Shadow_DOM known_ptr type_wf DocumentClass.type_wf get_child_nodes
  get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root
  get_shadow_root_locs get_tag_name get_tag_name_locs
  heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document get_disconnected_document_locs
  heap_is_wellformed parent_child_rel set_val set_val_locs set_disconnected_nodes set_disconnected_nodes_locs
  create_document known_ptrs
  by(auto simp add: l_create_document_wf_Shadow_DOM_def instances)
```

attach_shadow_root

```
locale l_attach_shadow_root_wf_Shadow_DOM =
  l_get_disconnected_nodes
  type_wf get_disconnected_nodes get_disconnected_nodes_locs
```

```
  + l_heap_is_wellformed_Shadow_DOM
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf
  heap_is_wellformed parent_child_rel
  heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document get_disconnected_document_locs
  + l_attach_shadow_root_Shadow_DOM known_ptr set_shadow_root set_shadow_root_locs set_mode set_mode_locs
  attach_shadow_root type_wf get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_locs
  + l_new_shadow_root_get_disconnected_nodes
  get_disconnected_nodes get_disconnected_nodes_locs

+ l_set_mode_get_disconnected_nodes
type_wf set_mode set_mode_locs get_disconnected_nodes get_disconnected_nodes_locs
+ l_new_shadow_root_get_child_nodes
type_wf known_ptr get_child_nodes get_child_nodes_locs
+ l_new_shadow_root_get_tag_name
type_wf get_tag_name get_tag_name_locs
+ l_set_mode_get_child_nodes
type_wf set_mode set_mode_locs known_ptr get_child_nodes get_child_nodes_locs
+ l_set_shadow_root_get_child_nodes
type_wf set_shadow_root set_shadow_root_locs known_ptr get_child_nodes get_child_nodes_locs
+ l_set_shadow_root
type_wf set_shadow_root set_shadow_root_locs
+ l_set_shadow_root_get_disconnected_nodes
set_shadow_root set_shadow_root_locs get_disconnected_nodes get_disconnected_nodes_locs
+ l_set_mode_get_shadow_root type_wf set_mode set_mode_locs get_shadow_root get_shadow_root_locs
+ l_set_shadow_root_get_shadow_root type_wf set_shadow_root set_shadow_root_locs
get_shadow_root get_shadow_root_locs
+ l_new_character_data_get_tag_name
get_tag_name get_tag_name_locs
+ l_set_mode_get_tag_name type_wf set_mode set_mode_locs get_tag_name get_tag_name_locs
+ l_get_tag_name type_wf get_tag_name get_tag_name_locs
+ l_set_shadow_root_get_tag_name set_shadow_root set_shadow_root_locs get_tag_name get_tag_name_locs
+ l_new_shadow_root
type_wf
+ l_known_ptrs
known_ptr known_ptrs
for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
  and known_ptrs :: "(_) heap ⇒ bool"
  and type_wf :: "(_) heap ⇒ bool"
  and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
  and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
  and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed :: "(_) heap ⇒ bool"
  and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
  and set_tag_name :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
  and set_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
  and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
  and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
  and create_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
  and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
  and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
  and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
  and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  and set_val :: "(_) character_data_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
  and set_val_locs :: "(_) character_data_ptr ⇒ ((_) heap, exception, unit) prog set"
  and create_character_data ::
  "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) character_data_ptr) prog"
```

```
  and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
  and type_wf_Core_DOM :: "(_) heap ⇒ bool"
  and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
  and set_shadow_root_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set"
  and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"
  and set_mode_locs :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog set"
  and attach_shadow_root :: "(_) element_ptr ⇒ shadow_root_mode ⇒ (_, (_) shadow_root_ptr) dom_prog"
begin
lemma attach_shadow_root_child_preserves:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ attach_shadow_root element_ptr new_mode →_h h'"
  shows "type_wf h'" and "known_ptrs h'" and "heap_is_wellformed h'"
proof -
  obtain h2 h3 new_shadow_root_ptr element_tag_name where
    element_tag_name: "h ⊢ get_tag_name element_ptr →_r element_tag_name" and
    "element_tag_name ∈ safe_shadow_root_element_types" and
    prev_shadow_root: "h ⊢ get_shadow_root element_ptr →_r None" and
    h2: "h ⊢ new_ShadowRoot_M →_h h2" and
    new_shadow_root_ptr: "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr" and
    h3: "h2 ⊢ set_mode new_shadow_root_ptr new_mode →_h h3" and
    h': "h3 ⊢ set_shadow_root element_ptr (Some new_shadow_root_ptr) →_h h'"
    using assms(4)
    by(auto simp add: attach_shadow_root_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_tag_name_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_shadow_root_pure, rotated] split: if_splits)

  have "h ⊢ attach_shadow_root element_ptr new_mode →_r new_shadow_root_ptr"
    thm  bind_pure_returns_result_I[OF get_tag_name_pure]
    apply(unfold attach_shadow_root_def)[1]
    using element_tag_name
    apply(rule bind_pure_returns_result_I[OF get_tag_name_pure])
    apply(rule bind_pure_returns_result_I)
    using ‹element_tag_name ∈ safe_shadow_root_element_types› apply(simp)
    using ‹element_tag_name ∈ safe_shadow_root_element_types› apply(simp)
    using prev_shadow_root
    apply(rule bind_pure_returns_result_I[OF get_shadow_root_pure])
    apply(rule bind_pure_returns_result_I)
      apply(simp)
     apply(simp)
    using h2 new_shadow_root_ptr h3 h'
    by(auto intro!: bind_returns_result_I intro: is_OK_returns_result_E[OF is_OK_returns_heap_I[OF h3]]
        is_OK_returns_result_E[OF is_OK_returns_heap_I[OF h']])

  have "new_shadow_root_ptr ∉ set |h ⊢ shadow_root_ptr_kinds_M|_r"
    using new_shadow_root_ptr ShadowRootMonad.ptr_kinds_ptr_kinds_M h2
    using new_ShadowRoot_M_ptr_not_in_heap by blast
  then have "cast new_shadow_root_ptr ∉ set |h ⊢ document_ptr_kinds_M|_r"
    by simp
  then have "cast new_shadow_root_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r"
    by simp



  have object_ptr_kinds_eq_h:
    "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
    using new_ShadowRoot_M_new_ptr h2 new_shadow_root_ptr by blast
  then have document_ptr_kinds_eq_h:
    "document_ptr_kinds h2 = document_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
    apply(simp add: document_ptr_kinds_def)
    by force
  then have shadow_root_ptr_kinds_eq_h:
    "shadow_root_ptr_kinds h2 = shadow_root_ptr_kinds h |∪| {|new_shadow_root_ptr|}"
    apply(simp add: shadow_root_ptr_kinds_def)
```

```
  by force
have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h"
  using object_ptr_kinds_eq_h
  by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)

have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
  apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
        OF set_mode_writes h3])
  using set_mode_pointers_preserved
  by (auto simp add: reflp_def transp_def)
then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
  by (auto simp add: document_ptr_kinds_def)
then have shadow_root_ptr_kinds_eq_h2: "shadow_root_ptr_kinds h3 = shadow_root_ptr_kinds h2"
  by (auto simp add: shadow_root_ptr_kinds_def)
have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
  using object_ptr_kinds_eq_h2
  by(auto simp add: node_ptr_kinds_def)

have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
  apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
        OF set_shadow_root_writes h'])
  using set_shadow_root_pointers_preserved
  by (auto simp add: reflp_def transp_def)
then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
  by (auto simp add: document_ptr_kinds_def)
then have shadow_root_ptr_kinds_eq_h3: "shadow_root_ptr_kinds h' = shadow_root_ptr_kinds h3"
  by (auto simp add: shadow_root_ptr_kinds_def)
have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
  using object_ptr_kinds_eq_h3
  by(auto simp add: node_ptr_kinds_def)

have "known_ptr (cast new_shadow_root_ptr)"
  using ‹h ⊢ attach_shadow_root element_ptr new_mode →r new_shadow_root_ptr› create_shadow_root_known_ptr
  by blast
then
have "known_ptrs h2"
  using known_ptrs_new_ptr object_ptr_kinds_eq_h ‹known_ptrs h› h2
  by blast
then
have "known_ptrs h3"
  using known_ptrs_preserved object_ptr_kinds_eq_h2 by blast
then
show "known_ptrs h'"
  using known_ptrs_preserved object_ptr_kinds_eq_h3 by blast

have "element_ptr |∈| element_ptr_kinds h"
  by (meson ‹h ⊢ attach_shadow_root element_ptr new_mode →r new_shadow_root_ptr› is_OK_returns_result_I
      local.attach_shadow_root_element_ptr_in_heap)


have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_shadow_root_ptr
                 ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
  using CD.get_child_nodes_reads h2 get_child_nodes_new_shadow_root[rotated, OF new_shadow_root_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by blast+
then have children_eq2_h:
  "⋀ptr'. ptr' ≠ cast new_shadow_root_ptr
    ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
  using select_result_eq by force
have object_ptr_kinds_eq_h:
  "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
  using new_ShadowRoot_new_ptr h2 new_shadow_root_ptr object_ptr_kinds_eq_h by blast
then have node_ptr_kinds_eq_h:
```

```
    "node_ptr_kinds h2 = node_ptr_kinds h"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have character_data_ptr_kinds_eq_h:
    "character_data_ptr_kinds h2 = character_data_ptr_kinds h"
    apply(simp add: character_data_ptr_kinds_def)
    done
  have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
    using object_ptr_kinds_eq_h
    by (auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_mode_writes h3])
    using set_mode_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)
  then have character_data_ptr_kinds_eq_h2: "character_data_ptr_kinds h3 = character_data_ptr_kinds h2"
    by(simp add: character_data_ptr_kinds_def)
  have element_ptr_kinds_eq_h2: "element_ptr_kinds h3 = element_ptr_kinds h2"
    using node_ptr_kinds_eq_h2
    by(simp add: element_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_shadow_root_writes h'])
    using set_shadow_root_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)
  then have character_data_ptr_kinds_eq_h3: "character_data_ptr_kinds h' = character_data_ptr_kinds h3"
    by(simp add: character_data_ptr_kinds_def)
  have element_ptr_kinds_eq_h3: "element_ptr_kinds h' = element_ptr_kinds h3"
    using node_ptr_kinds_eq_h3
    by(simp add: element_ptr_kinds_def)


  have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_shadow_root_ptr
              ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using CD.get_child_nodes_reads h2 get_child_nodes_new_shadow_root[rotated, OF new_shadow_root_ptr h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h: "⋀ptr'. ptr' ≠ cast new_shadow_root_ptr
                              ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force

  have "h2 ⊢ get_child_nodes (cast new_shadow_root_ptr) →r []"
    using h2 local.new_shadow_root_no_child_nodes new_shadow_root_ptr by auto

  have disconnected_nodes_eq_h:
    "⋀doc_ptr disc_nodes. doc_ptr ≠ cast new_shadow_root_ptr
              ⟹ h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                        = h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
```

```
  using get_disconnected_nodes_reads h2
    get_disconnected_nodes_new_shadow_root_different_pointers[rotated, OF new_shadow_root_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by (metis (no_types, lifting))+
then have disconnected_nodes_eq2_h:
  "⋀doc_ptr. doc_ptr ≠ cast new_shadow_root_ptr
            ⟹ |h ⊢ get_disconnected_nodes doc_ptr|_r = |h2 ⊢ get_disconnected_nodes doc_ptr|_r"
  using select_result_eq by force

have "h2 ⊢ get_disconnected_nodes (cast new_shadow_root_ptr) →_r []"
  using h2 new_shadow_root_no_disconnected_nodes new_shadow_root_ptr by auto

have tag_name_eq_h:
  "⋀ptr' disc_nodes. h ⊢ get_tag_name ptr' →_r disc_nodes
                                        = h2 ⊢ get_tag_name ptr' →_r disc_nodes"
  using get_tag_name_reads h2
    get_tag_name_new_shadow_root[OF new_shadow_root_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by blast+
then have tag_name_eq2_h: "⋀ptr'. |h ⊢ get_tag_name ptr'|_r = |h2 ⊢ get_tag_name ptr'|_r"
  using select_result_eq by force

have children_eq_h2:
  "⋀ptr' children. h2 ⊢ get_child_nodes ptr' →_r children = h3 ⊢ get_child_nodes ptr' →_r children"
  using CD.get_child_nodes_reads set_mode_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_mode_get_child_nodes)
then have children_eq2_h2:
  "⋀ptr'. |h2 ⊢ get_child_nodes ptr'|_r = |h3 ⊢ get_child_nodes ptr'|_r"
  using select_result_eq by force
have disconnected_nodes_eq_h2:
  "⋀doc_ptr disc_nodes. h2 ⊢ get_disconnected_nodes doc_ptr →_r disc_nodes
                                         = h3 ⊢ get_disconnected_nodes doc_ptr →_r disc_nodes"
  using get_disconnected_nodes_reads set_mode_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_mode_get_disconnected_nodes)
then have disconnected_nodes_eq2_h2:
  "⋀doc_ptr. |h2 ⊢ get_disconnected_nodes doc_ptr|_r = |h3 ⊢ get_disconnected_nodes doc_ptr|_r"
  using select_result_eq by force
have tag_name_eq_h2:
  "⋀ptr' disc_nodes. h2 ⊢ get_tag_name ptr' →_r disc_nodes
                                        = h3 ⊢ get_tag_name ptr' →_r disc_nodes"
  using get_tag_name_reads set_mode_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_mode_get_tag_name)
then have tag_name_eq2_h2: "⋀ptr'. |h2 ⊢ get_tag_name ptr'|_r = |h3 ⊢ get_tag_name ptr'|_r"
  using select_result_eq by force

have "type_wf h2"
  using ‹type_wf h› new_shadow_root_types_preserved h2 by blast
then have "type_wf h3"
  using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_mode_writes h3]
  using  set_mode_types_preserved
  by(auto simp add: reflp_def transp_def)
then show "type_wf h'"
  using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_shadow_root_writes h']
  using  set_shadow_root_types_preserved
  by(auto simp add: reflp_def transp_def)

have children_eq_h3:
  "⋀ptr' children. h3 ⊢ get_child_nodes ptr' →_r children = h' ⊢ get_child_nodes ptr' →_r children"
  using CD.get_child_nodes_reads set_shadow_root_writes h'
  apply(rule reads_writes_preserved)
```

```
    by(auto simp add: set_shadow_root_get_child_nodes)
  then have children_eq2_h3:
    " ⋀ptr'. |h3 ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h3: "⋀doc_ptr disc_nodes. h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads set_shadow_root_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_shadow_root_get_disconnected_nodes)
  then have disconnected_nodes_eq2_h3: "⋀doc_ptr. |h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected
doc_ptr|r"
    using select_result_eq by force
  have tag_name_eq_h3:
    "⋀ptr' disc_nodes. h3 ⊢ get_tag_name ptr' →r disc_nodes
                                          = h' ⊢ get_tag_name ptr' →r disc_nodes"
    using get_tag_name_reads set_shadow_root_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_shadow_root_get_tag_name)
  then have tag_name_eq2_h3: "⋀ptr'. |h3 ⊢ get_tag_name ptr'|r = |h' ⊢ get_tag_name ptr'|r"
    using select_result_eq by force

  have "acyclic (parent_child_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def CD.acyclic_heap_def)
  also have "parent_child_rel h = parent_child_rel h2"
  proof(auto simp add: CD.parent_child_rel_def)[1]
    fix a x
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
    then show "a |∈| object_ptr_kinds h2"
      by (simp add: object_ptr_kinds_eq_h)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
    then show "x ∈ set |h2 ⊢ get_child_nodes a|r"
      by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
          ‹cast new_shadow_root_ptr ∉ set |h ⊢ object_ptr_kinds_M|r› children_eq2_h)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h2"
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|r"
    then show "a |∈| object_ptr_kinds h"
      using object_ptr_kinds_eq_h ‹h2 ⊢ get_child_nodes (cast new_shadow_root_ptr) →r []›
      by(auto)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h2"
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|r"
    then show "x ∈ set |h ⊢ get_child_nodes a|r"
      by (metis (no_types, lifting) ‹h2 ⊢ get_child_nodes (cast new_shadow_root_ptr) →r []›
          children_eq2_h empty_iff empty_set image_eqI select_result_I2)
  qed
  also have "... = parent_child_rel h3"
    by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
  also have "... = parent_child_rel h'"
    by(auto simp add: CD.parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
  finally have "CD.a_acyclic_heap h'"
    by (simp add: CD.acyclic_heap_def)

  have "CD.a_all_ptrs_in_heap h"
    using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_all_ptrs_in_heap h2"
```

```
    apply(auto simp add: CD.a_all_ptrs_in_heap_def)[1]
    using node_ptr_kinds_eq_h
      ‹h2 ⊢ get_child_nodes (cast new_shadow_root_ptr) →r []›
     apply (metis (no_types, lifting) CD.get_child_nodes_ok CD.l_heap_is_wellformedCore_DOM_axioms ‹known_ptrs
h2›
        ‹parent_child_rel h = parent_child_rel h2› ‹type_wf h2› assms(1) assms(2) l_heap_is_wellformedCore_DOM.pa
        local.known_ptrs_known_ptr local.parent_child_rel_child_in_heap node_ptr_kinds_commutes returns_result_sele
    by (metis (no_types, opaque_lifting) ‹h2 ⊢ get_disconnected_nodes (castshadow_root_ptr2document_ptr new_shadow_r
→r []›
        ‹type_wf h2› disconnected_nodes_eq_h empty_iff is_OK_returns_result_E is_OK_returns_result_I
        local.get_disconnected_nodes_ok local.get_disconnected_nodes_ptr_in_heap node_ptr_kinds_eq_h select_result_
        set_empty subset_code(1))
  then have "CD.a_all_ptrs_in_heap h3"
    by (simp add: children_eq2_h2 disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
        CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h2 object_ptr_kinds_eq_h2)
  then have "CD.a_all_ptrs_in_heap h'"
    by (simp add: children_eq2_h3 disconnected_nodes_eq2_h3 document_ptr_kinds_eq_h3
        CD.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h3 object_ptr_kinds_eq_h3)

  have "cast new_shadow_root_ptr |∉| document_ptr_kinds h"
    using h2 newShadowRoot_M_ptr_not_in_heap new_shadow_root_ptr shadow_root_ptr_kinds_commutes by blast

  have "CD.a_distinct_lists h"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
  then have "CD.a_distinct_lists h2"
    using ‹h2 ⊢ get_disconnected_nodes (cast new_shadow_root_ptr) →r []›
      ‹h2 ⊢ get_child_nodes (cast new_shadow_root_ptr) →r []›

    apply(auto simp add: children_eq2_h[symmetric] CD.a_distinct_lists_def insort_split object_ptr_kinds_eq_h
        document_ptr_kinds_eq_h  disconnected_nodes_eq2_h intro!: distinct_concat_map_I)[1]
          apply (metis distinct_sorted_list_of_set finite_fset sorted_list_of_set_insert_remove)

         apply(auto simp add:  dest: distinct_concat_map_E)[1]
        apply(auto simp add:  dest: distinct_concat_map_E)[1]
    using ‹cast new_shadow_root_ptr |∉| document_ptr_kinds h›
       apply(auto simp add: distinct_insort dest: distinct_concat_map_E)[1]

      apply (metis (no_types) DocumentMonad.ptr_kinds_M_ptr_kinds DocumentMonad.ptr_kinds_ptr_kinds_M
        concat_map_all_distinct disconnected_nodes_eq2_h select_result_I2)
  proof -
    fix x :: "(_) document_ptr" and y :: "(_) document_ptr" and xa :: "(_) node_ptr"
    assume a1: "x ≠ y"
    assume a2: "x |∈| document_ptr_kinds h"
    assume a3: "x ≠ cast new_shadow_root_ptr"
    assume a4: "y |∈| document_ptr_kinds h"
    assume a5: "y ≠ cast new_shadow_root_ptr"
    assume a6: "distinct (concat (map (λdocument_ptr. |h ⊢ get_disconnected_nodes document_ptr|r)
                                        (sorted_list_of_set (fset (document_ptr_kinds h)))))"
    assume a7: "xa ∈ set |h2 ⊢ get_disconnected_nodes x|r"
    assume a8: "xa ∈ set |h2 ⊢ get_disconnected_nodes y|r"
    have f9: "xa ∈ set |h ⊢ get_disconnected_nodes x|r"
      using a7 a3 disconnected_nodes_eq2_h
      by (simp add: disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3)
    have f10: "xa ∈ set |h ⊢ get_disconnected_nodes y|r"
      using a8 a5 disconnected_nodes_eq2_h
      by (simp add: disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3)
    have f11: "y ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))"
      using a4 by simp
    have "x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h)))"
      using a2 by simp
    then show False
      using f11 f10 f9 a6 a1 by (meson disjoint_iff_not_equal distinct_concat_map_E(1))
```

```
next
  fix x xa xb
  assume 0: "h2 ⊢ get_disconnected_nodes (cast new_shadow_root_ptr) →r []"
    and 1: "h2 ⊢ get_child_nodes (cast new_shadow_root_ptr) →r []"
    and 2: "distinct (concat (map (λptr. |h ⊢ get_child_nodes ptr|r)
                                        (sorted_list_of_set (fset (object_ptr_kinds h)))))"
    and 3: "distinct (concat (map (λdocument_ptr. |h ⊢ get_disconnected_nodes document_ptr|r)
                                    (sorted_list_of_set (fset (document_ptr_kinds h)))))"
    and 4: "(⋃x∈fset (object_ptr_kinds h). set |h ⊢ get_child_nodes x|r)
                ∩ (⋃x∈fset (document_ptr_kinds h). set |h ⊢ get_disconnected_nodes x|r) = {}"
    and 5: "x ∈ set |h ⊢ get_child_nodes xa|r"
    and 6: "x ∈ set |h2 ⊢ get_disconnected_nodes xb|r"
    and 7: "xa |∈| object_ptr_kinds h"
    and 8: "xa ≠ cast new_shadow_root_ptr"
    and 9: "xb |∈| document_ptr_kinds h"
    and 10: "xb ≠ cast new_shadow_root_ptr"
  then show "False"
    by (metis CD.distinct_lists_no_parent ‹local.CD.a_distinct_lists h› assms(2) disconnected_nodes_eq2_h
        local.get_disconnected_nodes_ok returns_result_select_result)
qed
then have "CD.a_distinct_lists h3"
  by(auto simp add: CD.a_distinct_lists_def disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
      children_eq2_h2 object_ptr_kinds_eq_h2)[1]
then have "CD.a_distinct_lists h'"
  by(auto simp add: CD.a_distinct_lists_def disconnected_nodes_eq2_h3 children_eq2_h3
      object_ptr_kinds_eq_h3 document_ptr_kinds_eq_h3 intro!: distinct_concat_map_I)

have "CD.a_owner_document_valid h"
  using ‹heap_is_wellformed h›  by (simp add: heap_is_wellformed_def CD.heap_is_wellformed_def)
then have "CD.a_owner_document_valid h'"

  apply(simp add: CD.a_owner_document_valid_def)
  apply(simp add: object_ptr_kinds_eq_h object_ptr_kinds_eq_h3 )
  apply(simp add: object_ptr_kinds_eq_h2)
  apply(simp add: document_ptr_kinds_eq_h document_ptr_kinds_eq_h3 )
  apply(simp add: document_ptr_kinds_eq_h2)
  apply(simp add: node_ptr_kinds_eq_h node_ptr_kinds_eq_h3 )
  apply(simp add: node_ptr_kinds_eq_h2 node_ptr_kinds_eq_h )
  apply(auto simp add: children_eq2_h2[symmetric] children_eq2_h3[symmetric] disconnected_nodes_eq2_h
      disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3)[1]
  by (metis CD.get_child_nodes_ok CD.get_child_nodes_ptr_in_heap
      ‹cast_shadow_root_ptr2document_ptr new_shadow_root_ptr |∉| document_ptr_kinds h› assms(2) assms(3)
children_eq2_h
      children_eq_h disconnected_nodes_eq2_h disconnected_nodes_eq2_h2 disconnected_nodes_eq2_h3
      document_ptr_kinds_commutes is_OK_returns_result_I local.known_ptrs_known_ptr
      returns_result_select_result)
```

```
have shadow_root_eq_h: "⋀character_data_ptr shadow_root_opt. h ⊢ get_shadow_root character_data_ptr →r
shadow_root_opt =
h2 ⊢ get_shadow_root character_data_ptr →r shadow_root_opt"
  using get_shadow_root_reads h2 get_shadow_root_new_shadow_root[rotated, OF h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  using local.get_shadow_root_locs_impl new_shadow_root_ptr apply blast
  using local.get_shadow_root_locs_impl new_shadow_root_ptr by blast
```

```
  have shadow_root_eq_h2:
    "⋀ptr' children. h2 ⊢ get_shadow_root ptr' →r children = h3 ⊢ get_shadow_root ptr' →r children"
    using get_shadow_root_reads set_mode_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_mode_get_shadow_root)
  have shadow_root_eq_h3:
    "⋀ptr' children. element_ptr ≠ ptr' ⟹ h3 ⊢ get_shadow_root ptr' →r children = h' ⊢ get_shadow_root
ptr' →r children"
    using get_shadow_root_reads set_shadow_root_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_shadow_root_get_shadow_root_different_pointers)
  have shadow_root_h3: "h' ⊢ get_shadow_root element_ptr →r Some new_shadow_root_ptr"
    using ‹type_wf h3› h' local.set_shadow_root_get_shadow_root by blast


  have "a_all_ptrs_in_heap h"
    by (simp add: assms(1) local.a_all_ptrs_in_heap_def local.get_shadow_root_shadow_root_ptr_in_heap)
  then have "a_all_ptrs_in_heap h2"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h)[1]
    using returns_result_eq shadow_root_eq_h by fastforce
  then have "a_all_ptrs_in_heap h3"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h2)[1]
    using shadow_root_eq_h2 by blast
  then have "a_all_ptrs_in_heap h'"
    apply(auto simp add: a_all_ptrs_in_heap_def shadow_root_ptr_kinds_eq_h3)[1]
    apply(case_tac "shadow_root_ptr = new_shadow_root_ptr")
    using h2 new_ShadowRoot_M_ptr_in_heap new_shadow_root_ptr shadow_root_ptr_kinds_eq_h2
     apply blast
    using ‹type_wf h3› h' local.set_shadow_root_get_shadow_root returns_result_eq shadow_root_eq_h3
    apply fastforce
    done

  have "a_distinct_lists h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then have "a_distinct_lists h2"
    apply(auto simp add: a_distinct_lists_def character_data_ptr_kinds_eq_h)[1]
    apply(auto simp add: distinct_insort intro!: distinct_concat_map_I split: option.splits)[1]
    by (metis ‹type_wf h2› assms(1) assms(2) local.get_shadow_root_ok local.shadow_root_same_host
        returns_result_select_result shadow_root_eq_h)
  then have "a_distinct_lists h3"
    by(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h2 select_result_eq[OF shadow_root_eq_h2])
  then have "a_distinct_lists h'"
    apply(auto simp add: a_distinct_lists_def element_ptr_kinds_eq_h3 select_result_eq[OF shadow_root_eq_h3])[1]
    apply(auto simp add: distinct_insort intro!: distinct_concat_map_I split: option.splits)[1]
    by (smt ‹type_wf h3› assms(1) assms(2) h' h2 local.get_shadow_root_ok
        local.get_shadow_root_shadow_root_ptr_in_heap local.set_shadow_root_get_shadow_root local.shadow_root_same_
        new_ShadowRoot_M_ptr_not_in_heap new_shadow_root_ptr returns_result_select_result select_result_I2
shadow_root_eq_h
        shadow_root_eq_h2 shadow_root_eq_h3)


  have "a_shadow_root_valid h"
    using assms(1)
    by (simp add: heap_is_wellformed_def)
  then
  have "a_shadow_root_valid h'"
  proof(unfold a_shadow_root_valid_def; safe)
    fix shadow_root_ptr
    assume "∀shadow_root_ptr∈fset (shadow_root_ptr_kinds h). ∃host∈fset (element_ptr_kinds h).
|h ⊢ get_tag_name host|r ∈ safe_shadow_root_element_types ∧ |h ⊢ get_shadow_root host|r = Some shadow_root_ptr"
    assume "a_shadow_root_valid h"
```

```
    assume "shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h')"
    show "∃host∈fset (element_ptr_kinds h'). |h' ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types
∧
|h' ⊢ get_shadow_root host|_r = Some shadow_root_ptr"
    proof (cases "shadow_root_ptr = new_shadow_root_ptr")
      case True
      have "element_ptr ∈ fset (element_ptr_kinds h')"
        by (simp add: ‹element_ptr |∈| element_ptr_kinds h› element_ptr_kinds_eq_h element_ptr_kinds_eq_h2
            element_ptr_kinds_eq_h3)
      moreover have "|h' ⊢ get_tag_name element_ptr|_r ∈ safe_shadow_root_element_types"
        by (smt ‹⋀thesis. (⋀h2 h3 new_shadow_root_ptr element_tag_name. ⟦h ⊢ get_tag_name element_ptr
→_r element_tag_name;
element_tag_name ∈ safe_shadow_root_element_types; h ⊢ get_shadow_root element_ptr →_r None; h ⊢ new_ShadowRoot_M
→_h h2;
h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr; h2 ⊢ set_mode new_shadow_root_ptr new_mode →_h h3;
h3 ⊢ set_shadow_root element_ptr (Some new_shadow_root_ptr) →_h h'⟧ ⟹ thesis) ⟹ thesis›
            select_result_I2 tag_name_eq2_h tag_name_eq2_h2 tag_name_eq2_h3)
      moreover have "|h' ⊢ get_shadow_root element_ptr|_r = Some shadow_root_ptr"
        using shadow_root_h3
        by (simp add: True)
      ultimately
      show ?thesis
        by meson
    next
      case False
      then obtain host where host: "host ∈ fset (element_ptr_kinds h)" and
        "|h ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types" and
        "|h ⊢ get_shadow_root host|_r = Some shadow_root_ptr"
        using ‹shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h')›
        using ‹∀shadow_root_ptr∈fset (shadow_root_ptr_kinds h). ∃host∈fset (element_ptr_kinds h).
|h ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types ∧ |h ⊢ get_shadow_root host|_r = Some shadow_root_ptr›
        by (auto simp add: shadow_root_ptr_kinds_eq_h3 shadow_root_ptr_kinds_eq_h2 shadow_root_ptr_kinds_eq_h)
      moreover have "host ≠ element_ptr"
        using calculation(3) prev_shadow_root by auto
      ultimately show ?thesis
        using element_ptr_kinds_eq_h3 element_ptr_kinds_eq_h2 element_ptr_kinds_eq_h
        by (smt (verit) ‹type_wf h'› assms(2) local.get_shadow_root_ok returns_result_eq
            returns_result_select_result shadow_root_eq_h shadow_root_eq_h2 shadow_root_eq_h3 tag_name_eq2_h
            tag_name_eq2_h2 tag_name_eq2_h3)
    qed
  qed


  have "a_host_shadow_root_rel h = a_host_shadow_root_rel h2"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h select_result_eq[OF shadow_root_eq_h])
  have "a_host_shadow_root_rel h2 = a_host_shadow_root_rel h3"
    by(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h2 select_result_eq[OF shadow_root_eq_h2])
  have "a_host_shadow_root_rel h' = {(cast element_ptr, cast new_shadow_root_ptr)} ∪ a_host_shadow_root_rel
h3"
    apply(auto simp add: a_host_shadow_root_rel_def element_ptr_kinds_eq_h3 )[1]
       apply(case_tac "element_ptr ≠ aa")
    using select_result_eq[OF shadow_root_eq_h3] apply (simp add: image_iff)
    using select_result_eq[OF shadow_root_eq_h3]
       apply (metis (no_types, lifting) ‹local.a_host_shadow_root_rel h = local.a_host_shadow_root_rel h2›
        ‹local.a_host_shadow_root_rel h2 = local.a_host_shadow_root_rel h3› ‹type_wf h3› host_shadow_root_rel_def
        i_get_parent_get_host_get_disconnected_document_wf.a_host_shadow_root_rel_shadow_root local.get_shadow_root
        local.get_shadow_root_ok option.distinct(1) prev_shadow_root returns_result_select_result)
      apply (metis (mono_tags, lifting) ‹⋀ptr'. (⋀x. element_ptr ≠ ptr') ⟹
|h3 ⊢ get_shadow_root ptr'|_r = |h' ⊢ get_shadow_root ptr'|_r› case_prod_conv image_iff
        is_OK_returns_result_I mem_Collect_eq option.inject returns_result_eq returns_result_select_result
        shadow_root_h3)
    using element_ptr_kinds_eq_h3 local.get_shadow_root_ptr_in_heap shadow_root_h3 apply fastforce
    apply (smt Shadow_DOM.a_host_shadow_root_rel_def ‹⋀ptr'. (⋀x. element_ptr ≠ ptr') ⟹
```

```
|h3 ⊢ get_shadow_root ptr'|_r = |h' ⊢ get_shadow_root ptr'|_r› ‹type_wf h3› case_prodE case_prodI
        i_get_root_node_si_wf.a_host_shadow_root_rel_shadow_root image_iff local.get_shadow_root_impl
        local.get_shadow_root_ok mem_Collect_eq option.distinct(1) prev_shadow_root returns_result_eq
        returns_result_select_result shadow_root_eq_h shadow_root_eq_h2)
    done

  have "a_ptr_disconnected_node_rel h = a_ptr_disconnected_node_rel h2"
    apply(auto simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h)[1]
      apply (metis (no_types, lifting) ‹cast_shadow_root_ptr2document_ptr new_shadow_root_ptr |∉| document_ptr_kinds
h›
        case_prodI disconnected_nodes_eq2_h mem_Collect_eq pair_imageI)
    using ‹h2 ⊢ get_disconnected_nodes (cast_shadow_root_ptr2document_ptr new_shadow_root_ptr) →_r []›
     apply auto[1]
    apply(case_tac "cast new_shadow_root_ptr ≠ aa")
     apply (simp add: disconnected_nodes_eq2_h image_iff)
    using ‹cast_shadow_root_ptr2document_ptr new_shadow_root_ptr |∉| document_ptr_kinds h›
    apply blast
    done
  have "a_ptr_disconnected_node_rel h2 = a_ptr_disconnected_node_rel h3"
    by(simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h2 disconnected_nodes_eq2_h2)
  have "a_ptr_disconnected_node_rel h3 = a_ptr_disconnected_node_rel h'"
    by(simp add: a_ptr_disconnected_node_rel_def document_ptr_kinds_eq_h3 disconnected_nodes_eq2_h3)

  have "acyclic (parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
    using ‹heap_is_wellformed h›
    by (simp add: heap_is_wellformed_def)
  have "parent_child_rel h ∪ a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h =
parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2"
    using ‹local.a_host_shadow_root_rel h = local.a_host_shadow_root_rel h2›
      ‹local.a_ptr_disconnected_node_rel h = local.a_ptr_disconnected_node_rel h2› ‹parent_child_rel h =
parent_child_rel h2›
    by auto
  have "parent_child_rel h2 ∪ a_host_shadow_root_rel h2 ∪ a_ptr_disconnected_node_rel h2 =
parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3"
    using ‹local.a_host_shadow_root_rel h2 = local.a_host_shadow_root_rel h3›
      ‹local.a_ptr_disconnected_node_rel h2 = local.a_ptr_disconnected_node_rel h3› ‹parent_child_rel h2
= parent_child_rel h3›
    by auto
  have "parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h' =
{(cast element_ptr, cast new_shadow_root_ptr)} ∪ parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnec
h3"
    by (simp add: ‹local.a_host_shadow_root_rel h' =
{(cast_element_ptr2object_ptr element_ptr, cast_shadow_root_ptr2object_ptr new_shadow_root_ptr)} ∪ local.a_host_shadow_root
h3›
        ‹local.a_ptr_disconnected_node_rel h3 = local.a_ptr_disconnected_node_rel h'› ‹parent_child_rel
h3 = parent_child_rel h'›)

  have "⋀a b. (a, b) ∈ parent_child_rel h3 ⟹ a ≠ cast new_shadow_root_ptr"
    using CD.parent_child_rel_parent_in_heap ‹cast_shadow_root_ptr2document_ptr new_shadow_root_ptr |∉| document_ptr_k
h›
      ‹parent_child_rel h = parent_child_rel h2› ‹parent_child_rel h2 = parent_child_rel h3› document_ptr_kinds_co
    by blast
  moreover
  have "⋀a b. (a, b) ∈ a_host_shadow_root_rel h3 ⟹ a ≠ cast new_shadow_root_ptr"
    using shadow_root_eq_h2
    by(auto simp add: a_host_shadow_root_rel_def)
  moreover
  have "⋀a b. (a, b) ∈ a_ptr_disconnected_node_rel h3 ⟹ a ≠ cast new_shadow_root_ptr"
    using ‹h2 ⊢ get_disconnected_nodes (cast new_shadow_root_ptr) →_r []›
    by(auto simp add: a_ptr_disconnected_node_rel_def disconnected_nodes_eq_h2)
  moreover
  have "cast new_shadow_root_ptr ∉ {x. (x, cast element_ptr) ∈
(parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)*}"
```

```
    by (smt Un_iff calculation(1) calculation(2) calculation(3) cast_document_ptr_not_node_ptr(2) converse_rtrancl
mem_Collect_eq)
  moreover
  have "acyclic (parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3)"
    using ‹acyclic (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel
h)›
      ‹parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ local.a_ptr_disconnected_node_rel h =
parent_child_rel h2 ∪ local.a_host_shadow_root_rel h2 ∪ local.a_ptr_disconnected_node_rel h2›
      ‹parent_child_rel h2 ∪ local.a_host_shadow_root_rel h2 ∪ local.a_ptr_disconnected_node_rel h2 =
parent_child_rel h3 ∪ local.a_host_shadow_root_rel h3 ∪ local.a_ptr_disconnected_node_rel h3› by auto
  ultimately have "acyclic (parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel
h')"
    by(simp add: ‹parent_child_rel h' ∪ a_host_shadow_root_rel h' ∪ a_ptr_disconnected_node_rel h' =
{(cast element_ptr, cast new_shadow_root_ptr)} ∪
parent_child_rel h3 ∪ a_host_shadow_root_rel h3 ∪ a_ptr_disconnected_node_rel h3›)

  have "CD.a_heap_is_wellformed h'"
    apply(simp add: CD.a_heap_is_wellformed_def)
    by (simp add: ‹local.CD.a_acyclic_heap h'› ‹local.CD.a_all_ptrs_in_heap h'› ‹local.CD.a_distinct_lists
h'›
        ‹local.CD.a_owner_document_valid h'›)

  show "heap_is_wellformed h' "
    using ‹acyclic (parent_child_rel h' ∪ local.a_host_shadow_root_rel h' ∪ local.a_ptr_disconnected_node_rel
h')›
    by(simp add: heap_is_wellformed_def CD.heap_is_wellformed_impl ‹local.CD.a_heap_is_wellformed h'›
        ‹local.a_all_ptrs_in_heap h'› ‹local.a_distinct_lists h'› ‹local.a_shadow_root_valid h'›)
qed
end

interpretation l_attach_shadow_root_wf?: l_attach_shadow_root_wf_Shadow_DOM known_ptr known_ptrs type_wf
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  heap_is_wellformed parent_child_rel set_tag_name set_tag_name_locs set_disconnected_nodes
  set_disconnected_nodes_locs create_element get_shadow_root get_shadow_root_locs get_tag_name
  get_tag_name_locs heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs set_val set_val_locs create_character_data DocumentClass.known_ptr
  DocumentClass.type_wf set_shadow_root set_shadow_root_locs set_mode set_mode_locs attach_shadow_root
  by(auto simp add: l_attach_shadow_root_wf_Shadow_DOM_def instances)
declare l_attach_shadow_root_wf_Shadow_DOM_axioms [instances]

end
```

3 Test Suite

In this chapter, we present the formalized compliance test cases for the core DOM. As our formalization is executable, we can (symbolically) execute the test cases on top of our model. Executing these test cases successfully shows that our model is compliant to the official DOM standard. As future work, we plan to generate test cases from our formal model (e.g., using [9, 11]) to improve the quality of the official compliance test suite. For more details on the relation of test and proof in the context of web standards, we refer the reader to [4].

3.1 Shadow DOM Base Tests (Shadow_DOM_BaseTest)

```
theory Shadow_DOM_BaseTest
  imports
    "../Shadow_DOM"
begin


definition "assert_throws e p = do {
  h ← get_heap;
  (if (h ⊢ p →e e) then return () else error AssertException)
}"
notation assert_throws (‹assert'_throws'(_, _')›)

definition "test p h ⟷ h ⊢ ok p"


definition field_access :: "(string ⇒ (_, (_) object_ptr option) dom_prog) ⇒ string ⇒
(_, (_) object_ptr option) dom_prog"  (infix ‹.› 80)
  where
    "field_access m field = m field"

definition assert_equals :: "'a ⇒ 'a ⇒ (_, unit) dom_prog"
  where
    "assert_equals l r = (if l = r then return () else error AssertException)"
definition assert_equals_with_message :: "'a ⇒ 'a ⇒ 'b ⇒ (_, unit) dom_prog"
  where
    "assert_equals_with_message l r _ = (if l = r then return () else error AssertException)"
notation assert_equals (‹assert'_equals'(_, _')›)
notation assert_equals_with_message (‹assert'_equals'(_, _, _')›)
notation assert_equals (‹assert'_array'_equals'(_, _')›)
notation assert_equals_with_message (‹assert'_array'_equals'(_, _, _')›)

definition assert_not_equals :: "'a ⇒ 'a ⇒ (_, unit) dom_prog"
  where
    "assert_not_equals l r = (if l ≠ r then return () else error AssertException)"
definition assert_not_equals_with_message :: "'a ⇒ 'a ⇒ 'b ⇒ (_, unit) dom_prog"
  where
    "assert_not_equals_with_message l r _ = (if l ≠ r then return () else error AssertException)"
notation assert_not_equals (‹assert'_not'_equals'(_, _')›)
notation assert_not_equals_with_message (‹assert'_not'_equals'(_, _, _')›)
notation assert_not_equals (‹assert'_array'_not'_equals'(_, _')›)
notation assert_not_equals_with_message (‹assert'_array'_not'_equals'(_, _, _')›)


definition removeWhiteSpaceOnlyTextNodes :: "((_) object_ptr option) ⇒ (_, unit) dom_prog"
  where
```

```
    "removeWhiteSpaceOnlyTextNodes _ = return ()"

partial_function (dom_prog) assert_equal_subtrees :: "(_::linorder) object_ptr option ⇒
(_::linorder) object_ptr option ⇒ (_, unit) dom_prog"
  where
    [code]: "assert_equal_subtrees ptr ptr' = (case cast (the ptr) of
      None ⇒ (case cast (the ptr) of
        None ⇒ (case cast (the ptr) of
          None ⇒ (case cast (the ptr) of
            None ⇒ error AssertException |
            Some shadow_root_ptr ⇒ (case cast (the ptr') of
              None ⇒ error AssertException |
              Some shadow_root_ptr' ⇒ do {
                mode_val ← get_M shadow_root_ptr mode;
                mode_val' ← get_M shadow_root_ptr' mode;
                (if mode_val = mode_val' then return () else error AssertException);
                child_nodes_val ← get_M shadow_root_ptr RShadowRoot.child_nodes;
                child_nodes_val' ← get_M shadow_root_ptr' RShadowRoot.child_nodes;
                (if length child_nodes_val = length child_nodes_val'
                then return () else error AssertException);
                map_M (λ(ptr, ptr'). assert_equal_subtrees (Some (cast ptr)) (Some (cast ptr')))
                  (zip child_nodes_val child_nodes_val');
                document_ptr ← return (cast shadow_root_ptr);
                document_ptr' ← return (cast shadow_root_ptr');
                document_element_val ← get_M_Document document_ptr document_element;
                document_element_val' ← get_M_Document document_ptr' document_element;
                (if (document_element_val = None) ∧ (document_element_val' = None)
                then return () else assert_equal_subtrees (Some (cast (the document_element_val)))
                  (Some (cast (the document_element_val'))));
                disconnected_nodes_val ← get_M document_ptr disconnected_nodes;
                disconnected_nodes_val' ← get_M document_ptr' disconnected_nodes;
                (if length disconnected_nodes_val = length disconnected_nodes_val'
                then return () else error AssertException);
                map_M (λ(ptr, ptr'). assert_equal_subtrees (Some (cast ptr)) (Some (cast ptr')))
                  (zip disconnected_nodes_val disconnected_nodes_val');
                doctype_val ← get_M document_ptr doctype;
                doctype_val' ← get_M document_ptr' doctype;
                (if doctype_val = doctype_val' then return () else error AssertException);
                return ()
              }) |
          Some document_ptr ⇒ (case cast (the ptr') of
            None ⇒ error AssertException |
            Some document_ptr' ⇒ do {
              document_element_val ← get_M_Document document_ptr document_element;
              document_element_val' ← get_M_Document document_ptr' document_element;
              (if (document_element_val = None) ∧ (document_element_val' = None)
              then return () else assert_equal_subtrees (Some (cast (the document_element_val)))
                (Some (cast (the document_element_val'))));
              disconnected_nodes_val ← get_M document_ptr disconnected_nodes;
              disconnected_nodes_val' ← get_M document_ptr' disconnected_nodes;
              (if length disconnected_nodes_val = length disconnected_nodes_val'
              then return () else error AssertException);
              map_M (λ(ptr, ptr'). assert_equal_subtrees (Some (cast ptr)) (Some (cast ptr')))
                (zip disconnected_nodes_val disconnected_nodes_val');
              doctype_val ← get_M document_ptr doctype;
              doctype_val' ← get_M document_ptr' doctype;
              (if doctype_val = doctype_val' then return () else error AssertException);
              return ()
            }) |
        Some character_data_ptr ⇒ (case cast (the ptr') of
          None ⇒ error AssertException |
          Some character_data_ptr' ⇒ do {
            val_val ← get_M character_data_ptr val;
```

```
          val_val' ← get_M character_data_ptr' val;
          (if val_val = val_val' then return () else error AssertException)
        })) |
      Some element_ptr ⇒ (case cast (the ptr') of
        None ⇒ error AssertException |
        Some element_ptr' ⇒ do {
          tag_name_val ← get_M element_ptr tag_name;
          tag_name_val' ← get_M element_ptr' tag_name;
          (if tag_name_val = tag_name_val' then return () else error AssertException);
          child_nodes_val ← get_M element_ptr RElement.child_nodes;
          child_nodes_val' ← get_M element_ptr' RElement.child_nodes;
          (if length child_nodes_val = length child_nodes_val' then return () else error AssertException);
          map_M (λ(ptr, ptr'). assert_equal_subtrees (Some (cast ptr)) (Some (cast ptr')))
            (zip child_nodes_val child_nodes_val');
          attrs_val ← get_M element_ptr attrs;
          attrs_val' ← get_M element_ptr' attrs;
          (if attrs_val = attrs_val' then return () else error AssertException);
          shadow_root_opt_val ← get_M element_ptr shadow_root_opt;
          shadow_root_opt_val' ← get_M element_ptr' shadow_root_opt;
          (if (shadow_root_opt_val = None) ∧ (shadow_root_opt_val' = None)
          then return () else assert_equal_subtrees (Some (cast (the shadow_root_opt_val)))
            (Some (cast (the shadow_root_opt_val'))));
          return ()
        }))"
notation assert_equal_subtrees (‹assert'_equal'_subtrees'(_, _')›)
```

3.1.1 Making the functions under test compatible with untyped languages such as JavaScript

```
fun set_attribute_with_null :: "((_) object_ptr option) ⇒ attr_key ⇒ attr_value ⇒ (_, unit) dom_prog"
  where
    "set_attribute_with_null (Some ptr) k v = (case cast ptr of
      Some element_ptr ⇒ set_attribute element_ptr k (Some v))"
fun set_attribute_with_null2 :: "((_) object_ptr option) ⇒ attr_key ⇒ attr_value option ⇒ (_, unit) dom_prog"
  where
    "set_attribute_with_null2 (Some ptr) k v = (case cast ptr of
      Some element_ptr ⇒ set_attribute element_ptr k v)"
notation set_attribute_with_null (‹_ . setAttribute'(_, _')›)
notation set_attribute_with_null2 (‹_ . setAttribute'(_, _')›)

fun get_child_nodes_Core_DOM_with_null :: "((_) object_ptr option) ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "get_child_nodes_Core_DOM_with_null (Some ptr) = do {
      children ← get_child_nodes ptr;
      return (map (Some ∘ cast) children)
    }"
notation get_child_nodes_Core_DOM_with_null (‹_ . childNodes›)

fun create_element_with_null :: "((_) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "create_element_with_null (Some owner_document_obj) tag = (case cast owner_document_obj of
      Some owner_document ⇒ do {
        element_ptr ← create_element owner_document tag;
        return (Some (cast element_ptr))})"
notation create_element_with_null (‹_ . createElement'(_')›)

fun create_character_data_with_null :: "((_) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option))
dom_prog"
  where
    "create_character_data_with_null (Some owner_document_obj) tag = (case cast owner_document_obj of
      Some owner_document ⇒ do {
        character_data_ptr ← create_character_data owner_document tag;
        return (Some (cast character_data_ptr))})"
```

```
notation create_character_data_with_null (‹_ . createTextNode'(_')›)

definition create_document_with_null :: "string ⇒ (_, ((_::linorder) object_ptr option)) dom_prog"
  where
    "create_document_with_null title = do {
      new_document_ptr ← create_document;
      html ← create_element new_document_ptr ''html'';
      append_child (cast new_document_ptr) (cast html);
      heap ← create_element new_document_ptr ''heap'';
      append_child (cast html) (cast heap);
      body ← create_element new_document_ptr ''body'';
      append_child (cast html) (cast body);
      return (Some (cast new_document_ptr))
    }"
abbreviation "create_document_with_null2 _ _ _ ≡ create_document_with_null ''''"
notation create_document_with_null (‹createDocument'(_')›)
notation create_document_with_null2 (‹createDocument'(_, _, _')›)

fun get_element_by_id_with_null ::
  "((_::linorder) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_element_by_id_with_null (Some ptr) id' = do {
        element_ptr_opt ← get_element_by_id ptr id';
        (case element_ptr_opt of
          Some element_ptr ⇒ return (Some (cast_element_ptr2object_ptr element_ptr))
        | None ⇒ return None)}"
  | "get_element_by_id_with_null _ _ = error SegmentationFault"
notation get_element_by_id_with_null (‹_ . getElementById'(_')›)

fun get_elements_by_class_name_with_null ::
  "((_::linorder) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option) list) dom_prog"
  where
    "get_elements_by_class_name_with_null (Some ptr) class_name =
      get_elements_by_class_name ptr class_name ≫= map_M (return ∘ Some ∘ cast_element_ptr2object_ptr)"
notation get_elements_by_class_name_with_null (‹_ . getElementsByClassName'(_')›)

fun get_elements_by_tag_name_with_null ::
  "((_::linorder) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option) list) dom_prog"
  where
    "get_elements_by_tag_name_with_null (Some ptr) tag =
      get_elements_by_tag_name ptr tag ≫= map_M (return ∘ Some ∘ cast_element_ptr2object_ptr)"
notation get_elements_by_tag_name_with_null (‹_ . getElementsByTagName'(_')›)

fun insert_before_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ ((_) object_ptr option) ⇒
(_, ((_) object_ptr option)) dom_prog"
  where
    "insert_before_with_null (Some ptr) (Some child_obj) ref_child_obj_opt = (case cast child_obj of
      Some child ⇒ do {
        (case ref_child_obj_opt of
          Some ref_child_obj ⇒ insert_before ptr child (cast ref_child_obj)
        | None ⇒ insert_before ptr child None);
        return (Some child_obj)}
    | None ⇒ error HierarchyRequestError)"
notation insert_before_with_null (‹_ . insertBefore'(_, _')›)

fun append_child_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ (_, unit) dom_prog"
  where
    "append_child_with_null (Some ptr) (Some child_obj) = (case cast child_obj of
      Some child ⇒ append_child ptr child
    | None ⇒ error SegmentationFault)"
notation append_child_with_null (‹_ . appendChild'(_')›)
```

```
code_thms append_child_with_null
fun get_body :: "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_body ptr = do {
       ptrs ← ptr . getElementsByTagName(''body'');
       return (hd ptrs)
    }"
notation get_body (<_ . body>)

fun get_document_element_with_null ::
  "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_document_element_with_null (Some ptr) = (case cast_object_ptr2document_ptr ptr of
    Some document_ptr ⇒ do {
      element_ptr_opt ← get_M document_ptr document_element;
      return (case element_ptr_opt of
        Some element_ptr ⇒ Some (cast_element_ptr2object_ptr element_ptr)
      | None ⇒ None)})"
notation get_document_element_with_null (<_ . documentElement>)

fun get_owner_document_with_null :: "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_owner_document_with_null (Some ptr) = (do {
        document_ptr ← get_owner_document ptr;
        return (Some (cast_document_ptr2object_ptr document_ptr))})"
notation get_owner_document_with_null (<_ . ownerDocument>)

fun remove_with_null :: "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "remove_with_null (Some child) = (case cast child of
      Some child_node ⇒ do {
        remove child_node;
        return (Some child)}
    | None ⇒ error NotFoundError)"
  | "remove_with_null None = error TypeError"
notation remove_with_null (<_ . remove'(')>)

fun remove_child_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "remove_child_with_null (Some ptr) (Some child) = (case cast child of
      Some child_node ⇒ do {
        remove_child ptr child_node;
        return (Some child)}
    | None ⇒ error NotFoundError)"
  | "remove_child_with_null None _ = error TypeError"
  | "remove_child_with_null _ None = error TypeError"
notation remove_child_with_null (<_ . removeChild>)

fun get_tag_name_with_null :: "((_) object_ptr option) ⇒ (_, attr_value) dom_prog"
  where
    "get_tag_name_with_null (Some ptr) = (case cast ptr of
      Some element_ptr ⇒ get_M element_ptr tag_name)"
notation get_tag_name_with_null (<_ . tagName>)

abbreviation "remove_attribute_with_null ptr k ≡ set_attribute_with_null2 ptr k None"
notation remove_attribute_with_null (<_ . removeAttribute'(_')>)

fun get_attribute_with_null :: "((_) object_ptr option) ⇒ attr_key ⇒ (_, attr_value option) dom_prog"
  where
    "get_attribute_with_null (Some ptr) k = (case cast ptr of
      Some element_ptr ⇒ get_attribute element_ptr k)"
fun get_attribute_with_null2 :: "((_) object_ptr option) ⇒ attr_key ⇒ (_, attr_value) dom_prog"
```

```
  where
    "get_attribute_with_null2 (Some ptr) k = (case cast ptr of
      Some element_ptr ⇒ do {
        a ← get_attribute element_ptr k;
        return (the a)})"
notation get_attribute_with_null (‹_ . getAttribute'(_')›)
notation get_attribute_with_null2 (‹_ . getAttribute'(_')›)

fun get_parent_with_null :: "((_::linorder) object_ptr option) ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "get_parent_with_null (Some ptr) = (case cast ptr of
      Some node_ptr ⇒ get_parent node_ptr)"
notation get_parent_with_null (‹_ . parentNode›)

fun first_child_with_null :: "((_) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "first_child_with_null (Some ptr) = do {
      child_opt ← first_child ptr;
      return (case child_opt of
        Some child ⇒ Some (cast child)
      | None ⇒ None)}"
notation first_child_with_null (‹_ . firstChild›)

fun adopt_node_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "adopt_node_with_null (Some ptr) (Some child) = (case cast ptr of
      Some document_ptr ⇒ (case cast child of
        Some child_node ⇒ do {
          adopt_node document_ptr child_node;
          return (Some child)}))"
notation adopt_node_with_null (‹_ . adoptNode'(_')›)


fun get_shadow_root_with_null :: "((_) object_ptr option) ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "get_shadow_root_with_null (Some ptr) = (case cast ptr of
      Some element_ptr ⇒ do {
        shadow_root ← get_shadow_root element_ptr;
        (case shadow_root of Some sr ⇒ return (Some (cast sr))
                             | None ⇒ return None)})"
notation get_shadow_root_with_null (‹_ . shadowRoot›)
```

3.1.2 Making the functions under test compatible with untyped languages such as JavaScript

```
fun get_element_by_id_si_with_null ::
  "(_::linorder) object_ptr option ⇒ string ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "get_element_by_id_si_with_null (Some ptr) id' = do {
        element_ptr_opt ← get_element_by_id_si ptr id';
        (case element_ptr_opt of
          Some element_ptr ⇒ return (Some (cast_element_ptr2object_ptr element_ptr))
        | None ⇒ return None)}"
  | "get_element_by_id_si_with_null _ _ = error SegmentationFault"

fun find_slot_closed_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "find_slot_closed_with_null (Some ptr) = (case cast_object_ptr2node_ptr ptr of
      Some node_ptr ⇒ do {
        element_ptr_opt ← find_slot True node_ptr;
        (case element_ptr_opt of
```

```
        Some element_ptr ⇒ return (Some (cast_element_ptr2object_ptr element_ptr))
      | None ⇒ return None)}
    | None ⇒ error SegmentationFault)"
  | "find_slot_closed_with_null None = error SegmentationFault"
notation find_slot_closed_with_null (‹_ . assignedSlot›)

fun assigned_nodes_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "assigned_nodes_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes element_ptr;
        return (map Some (map cast_node_ptr2object_ptr l))}
    | None ⇒ error SegmentationFault)"
  | "assigned_nodes_with_null None = error SegmentationFault"
notation assigned_nodes_with_null (‹_ . assignedNodes'(')›)

fun assigned_nodes_flatten_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "assigned_nodes_flatten_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes_flatten element_ptr;
        return (map Some (map cast_node_ptr2object_ptr l))}
    | None ⇒ error SegmentationFault)"
  | "assigned_nodes_flatten_with_null None = error SegmentationFault"
notation assigned_nodes_flatten_with_null (‹_ . assignedNodes'(True')›)

fun get_assigned_elements_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "get_assigned_elements_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes element_ptr;
        l ← map_filter_M (return ∘ cast_node_ptr2element_ptr) l;
        return (map Some (map cast l))}
    | None ⇒ error SegmentationFault)"
  | "get_assigned_elements_with_null None = error SegmentationFault"
notation get_assigned_elements_with_null (‹_ . assignedElements'(')›)

fun get_assigned_elements_flatten_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "get_assigned_elements_flatten_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes_flatten element_ptr;
        return (map Some (map cast_node_ptr2object_ptr l))}
    | None ⇒ error SegmentationFault)"
  | "get_assigned_elements_flatten_with_null None = error SegmentationFault"
notation get_assigned_elements_flatten_with_null (‹_ . assignedElements'(True')›)

fun createTestTree ::
  "(_::linorder) object_ptr option ⇒ (_, string ⇒ (_, (_) object_ptr option) dom_prog) dom_prog"
  where
    "createTestTree (Some ref) = do {
      tups ← to_tree_order_si ref ≫= map_filter_M (λptr. do {
        (case cast ptr of
          Some element_ptr ⇒ do {
            iden_opt ← get_attribute element_ptr ''id'';
            (case iden_opt of
              Some iden ⇒ return (Some (iden, ptr))
            | None ⇒ return None)
          }
```

```
        | None ⇒ return None)});
      return (return ∘ map_of tups)
    }"
  | "createTestTree None = error SegmentationFault"
```

end

3.2 Testing slots (slots)

This theory contains the test cases for slots.

theory *slots*
imports
  "Shadow_DOM_BaseTest"
begin

definition *slots_heap :: "heap$_{final}$"* **where**

```
  "slots_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some (cast (element_ptr.Re
1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Shadow%20DOM%3A%20Slots%20and%20assignments
    (cast (element_ptr.Ref 4), cast (create_element_obj ''meta'' [] (fmap_of_list [(''name'', ''author''),
(''title'', ''Hayato Ito''), (''href'', ''mailto:hayato@google.com'')]) None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''resources/shadow-d
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''body'' [cast (element_ptr.Ref 9), cast (element_ptr.Ref
13), cast (element_ptr.Ref 14), cast (element_ptr.Ref 18), cast (element_ptr.Ref 19), cast (element_ptr.Ref
21), cast (element_ptr.Ref 22), cast (element_ptr.Ref 30), cast (element_ptr.Ref 31), cast (element_ptr.Ref
39), cast (element_ptr.Ref 40), cast (element_ptr.Ref 48), cast (element_ptr.Ref 49), cast (element_ptr.Ref
57), cast (element_ptr.Ref 58), cast (element_ptr.Ref 64), cast (element_ptr.Ref 65), cast (element_ptr.Ref
71), cast (element_ptr.Ref 72), cast (element_ptr.Ref 78), cast (element_ptr.Ref 79), cast (element_ptr.Ref
85), cast (element_ptr.Ref 86), cast (element_ptr.Ref 92), cast (element_ptr.Ref 93), cast (element_ptr.Ref
112)] fmempty None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''div'' [cast (element_ptr.Ref 10)] (fmap_of_list
[(''id'', ''test_basic'')]) None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''div'' [cast (element_ptr.Ref 11)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 1))))),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 1), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 12)])),
    (cast (element_ptr.Ref 12), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 13), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 14), cast (create_element_obj ''div'' [cast (element_ptr.Ref 15)] (fmap_of_list
[(''id'', ''test_basic_closed'')]) None)),
    (cast (element_ptr.Ref 15), cast (create_element_obj ''div'' [cast (element_ptr.Ref 16)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 2))))),
    (cast (element_ptr.Ref 16), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 2), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 17)])),
```

```
    (cast (element_ptr.Ref 17), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 18), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''div'' [cast (element_ptr.Ref 20)] (fmap_of_list
[(''id'', ''test_slot_not_in_shadow'')]) None)),
    (cast (element_ptr.Ref 20), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1'')]) None)),
    (cast (element_ptr.Ref 21), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 4)] fmempty
None)),
    (cast (character_data_ptr.Ref 4), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 22), cast (create_element_obj ''div'' [cast (element_ptr.Ref 23), cast (element_ptr.Ref
25)] (fmap_of_list [(''id'', ''test_slot_not_in_shadow_2'')]) None)),
    (cast (element_ptr.Ref 23), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 24)] (fmap_of_list
[(''id'', ''s1'')]) None)),
    (cast (element_ptr.Ref 24), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1'')]) None)),
    (cast (element_ptr.Ref 25), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 26), cast (element_ptr.Ref
27)] (fmap_of_list [(''id'', ''s2'')]) None)),
    (cast (element_ptr.Ref 26), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2'')]) None)),
    (cast (element_ptr.Ref 27), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 28), cast (element_ptr.Ref
29)] (fmap_of_list [(''id'', ''s3'')]) None)),
    (cast (element_ptr.Ref 28), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3_1'')])
None)),
    (cast (element_ptr.Ref 29), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3_2'')])
None)),
    (cast (element_ptr.Ref 30), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 5)] fmempty
None)),
    (cast (character_data_ptr.Ref 5), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 31), cast (create_element_obj ''div'' [cast (element_ptr.Ref 32)] (fmap_of_list
[(''id'', ''test_slot_name_matching'')]) None)),
    (cast (element_ptr.Ref 32), cast (create_element_obj ''div'' [cast (element_ptr.Ref 33), cast (element_ptr.Ref
34), cast (element_ptr.Ref 35)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 3))))),
    (cast (element_ptr.Ref 33), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 34), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 35), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3''), (''slot'',
''yyy'')]) None)),
    (cast (shadow_root_ptr.Ref 3), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 36), cast (element_ptr.
37), cast (element_ptr.Ref 38)])),
    (cast (element_ptr.Ref 36), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 37), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 38), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s3''), (''name'',
''xxx'')]) None)),
    (cast (element_ptr.Ref 39), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 6)] fmempty
None)),
    (cast (character_data_ptr.Ref 6), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 40), cast (create_element_obj ''div'' [cast (element_ptr.Ref 41)] (fmap_of_list
[(''id'', ''test_no_direct_host_child'')]) None)),
    (cast (element_ptr.Ref 41), cast (create_element_obj ''div'' [cast (element_ptr.Ref 42), cast (element_ptr.Ref
43), cast (element_ptr.Ref 44)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 4))))),
    (cast (element_ptr.Ref 42), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 43), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 44), cast (create_element_obj ''div'' [cast (element_ptr.Ref 45)] fmempty None)),
    (cast (element_ptr.Ref 45), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 4), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 46), cast (element_ptr.
47)])),
    (cast (element_ptr.Ref 46), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
```

```
''slot1'')]) None)),
    (cast (element_ptr.Ref 47), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 48), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 7)] fmempty
None)),
    (cast (character_data_ptr.Ref 7), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 49), cast (create_element_obj ''div'' [cast (element_ptr.Ref 50)] (fmap_of_list
[(''id'', ''test_default_slot'')]) None)),
    (cast (element_ptr.Ref 50), cast (create_element_obj ''div'' [cast (element_ptr.Ref 51), cast (element_ptr.Ref
52), cast (element_ptr.Ref 53)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 5))))),
    (cast (element_ptr.Ref 51), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1'')]) None)),
    (cast (element_ptr.Ref 52), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
'''')]) None)),
    (cast (element_ptr.Ref 53), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3''), (''slot'',
''foo'')]) None)),
    (cast (shadow_root_ptr.Ref 5), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 54), cast (element_ptr.
55), cast (element_ptr.Ref 56)])),
    (cast (element_ptr.Ref 54), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 55), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2'')]) None)),
    (cast (element_ptr.Ref 56), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s3'')]) None)),
    (cast (element_ptr.Ref 57), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 8)] fmempty
None)),
    (cast (character_data_ptr.Ref 8), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 58), cast (create_element_obj ''div'' [cast (element_ptr.Ref 59)] (fmap_of_list
[(''id'', ''test_slot_in_slot'')]) None)),
    (cast (element_ptr.Ref 59), cast (create_element_obj ''div'' [cast (element_ptr.Ref 60), cast (element_ptr.Ref
61)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 6))))),
    (cast (element_ptr.Ref 60), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 61), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 6), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 62)])),
    (cast (element_ptr.Ref 62), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 63)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 63), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 64), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 9)] fmempty
None)),
    (cast (character_data_ptr.Ref 9), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 65), cast (create_element_obj ''div'' [cast (element_ptr.Ref 66)] (fmap_of_list
[(''id'', ''test_slot_is_assigned_to_slot'')]) None)),
    (cast (element_ptr.Ref 66), cast (create_element_obj ''div'' [cast (element_ptr.Ref 67)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 7))))),
    (cast (element_ptr.Ref 67), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 7), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 68)])),
    (cast (element_ptr.Ref 68), cast (create_element_obj ''div'' [cast (element_ptr.Ref 69)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 8))))),
    (cast (element_ptr.Ref 69), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 8), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 70)])),
    (cast (element_ptr.Ref 70), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 71), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 10)] fmempty
None)),
    (cast (character_data_ptr.Ref 10), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 72), cast (create_element_obj ''div'' [cast (element_ptr.Ref 73)] (fmap_of_list
[(''id'', ''test_open_closed'')]) None)),
    (cast (element_ptr.Ref 73), cast (create_element_obj ''div'' [cast (element_ptr.Ref 74)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 9))))),
    (cast (element_ptr.Ref 74), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
```

```
    (cast (shadow_root_ptr.Ref 9), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 75)])),
    (cast (element_ptr.Ref 75), cast (create_element_obj ''div'' [cast (element_ptr.Ref 76)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 10))))),
    (cast (element_ptr.Ref 76), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 10), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 77)])),
    (cast (element_ptr.Ref 77), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 78), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 11)] fmempty
None)),
    (cast (character_data_ptr.Ref 11), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 79), cast (create_element_obj ''div'' [cast (element_ptr.Ref 80)] (fmap_of_list
[(''id'', ''test_closed_closed'')]) None)),
    (cast (element_ptr.Ref 80), cast (create_element_obj ''div'' [cast (element_ptr.Ref 81)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 11))))),
    (cast (element_ptr.Ref 81), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 11), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 82)])),
    (cast (element_ptr.Ref 82), cast (create_element_obj ''div'' [cast (element_ptr.Ref 83)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 12))))),
    (cast (element_ptr.Ref 83), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 12), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 84)])),
    (cast (element_ptr.Ref 84), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 85), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 12)] fmempty
None)),
    (cast (character_data_ptr.Ref 12), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 86), cast (create_element_obj ''div'' [cast (element_ptr.Ref 87)] (fmap_of_list
[(''id'', ''test_closed_open'')]) None)),
    (cast (element_ptr.Ref 87), cast (create_element_obj ''div'' [cast (element_ptr.Ref 88)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 13))))),
    (cast (element_ptr.Ref 88), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 13), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 89)])),
    (cast (element_ptr.Ref 89), cast (create_element_obj ''div'' [cast (element_ptr.Ref 90)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 14))))),
    (cast (element_ptr.Ref 90), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 14), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 91)])),
    (cast (element_ptr.Ref 91), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 92), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 13)] fmempty
None)),
    (cast (character_data_ptr.Ref 13), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 93), cast (create_element_obj ''div'' [cast (element_ptr.Ref 94)] (fmap_of_list
[(''id'', ''test_complex'')]) None)),
    (cast (element_ptr.Ref 94), cast (create_element_obj ''div'' [cast (element_ptr.Ref 95), cast (element_ptr.Ref
96), cast (element_ptr.Ref 97), cast (element_ptr.Ref 98)] (fmap_of_list [(''id'', ''host1'')]) (Some (cast
(shadow_root_ptr.Ref 15))))),
    (cast (element_ptr.Ref 95), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 96), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 97), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3'')]) None)),
    (cast (element_ptr.Ref 98), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c4''), (''slot'',
''slot-none'')]) None)),
    (cast (shadow_root_ptr.Ref 15), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 99)])),
    (cast (element_ptr.Ref 99), cast (create_element_obj ''div'' [cast (element_ptr.Ref 100), cast (element_ptr.Ref
101), cast (element_ptr.Ref 102), cast (element_ptr.Ref 103), cast (element_ptr.Ref 104), cast (element_ptr.Ref
105), cast (element_ptr.Ref 106), cast (element_ptr.Ref 107)] (fmap_of_list [(''id'', ''host2'')]) (Some
(cast (shadow_root_ptr.Ref 16))))),
    (cast (element_ptr.Ref 100), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
```

```
''slot1''), (''slot'', ''slot5'')]) None)),
    (cast (element_ptr.Ref 101), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2''), (''slot'', ''slot6'')]) None)),
    (cast (element_ptr.Ref 102), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s3'')])
None)),
    (cast (element_ptr.Ref 103), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s4''), (''name'',
''slot4''), (''slot'', ''slot-none'')]) None)),
    (cast (element_ptr.Ref 104), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c5''), (''slot'',
''slot5'')]) None)),
    (cast (element_ptr.Ref 105), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c6''), (''slot'',
''slot6'')]) None)),
    (cast (element_ptr.Ref 106), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c7'')]) None)),
    (cast (element_ptr.Ref 107), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c8''), (''slot'',
''slot-none'')]) None)),
    (cast (shadow_root_ptr.Ref 16), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 108), cast
(element_ptr.Ref 109), cast (element_ptr.Ref 110), cast (element_ptr.Ref 111)])),
    (cast (element_ptr.Ref 108), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s5''), (''name'',
''slot5'')]) None)),
    (cast (element_ptr.Ref 109), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s6''), (''name'',
''slot6'')]) None)),
    (cast (element_ptr.Ref 110), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s7'')])
None)),
    (cast (element_ptr.Ref 111), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s8''), (''name'',
''slot8'')]) None)),
    (cast (element_ptr.Ref 112), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 14)]
fmempty None)),
    (cast (character_data_ptr.Ref 14), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

definition *slots_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option"* **where** *"slots_document = Some (cast (document_ptr.Ref 1))"*

'Slots: Basic.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_basic'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_basic'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedNodes();
  tmp7 ← n . ''c1'';
  assert_array_equals(tmp6, [tmp7])
}) slots_heap"
  by eval
```

'Slots: Basic, elements only.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_basic'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  tmp3 ← n . ''c1'';
  assert_array_equals(tmp2, [tmp3])
}) slots_heap"
  by eval
```

'Slots: Slots in closed.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_basic_closed'');
  n ← createTestTree(tmp0);
```

```
  tmp1 ← n . ''test_basic_closed'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedNodes();
  tmp6 ← n . ''c1'';
  assert_array_equals(tmp5, [tmp6])
}) slots_heap"
  by eval
```

'Slots: Slots in closed, elements only.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_basic_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  tmp3 ← n . ''c1'';
  assert_array_equals(tmp2, [tmp3])
}) slots_heap"
  by eval
```

'Slots: Slots not in a shadow tree.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_slot_not_in_shadow'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_not_in_shadow'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp3 ← tmp2 . assignedNodes();
  assert_array_equals(tmp3, [])
}) slots_heap"
  by eval
```

'Slots: Slots not in a shadow tree, elements only.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_slot_not_in_shadow'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  assert_array_equals(tmp2, [])
}) slots_heap"
  by eval
```

'Slots: Distributed nodes for Slots not in a shadow tree.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_slot_not_in_shadow_2'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_not_in_shadow_2'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''c2'';
  tmp5 ← tmp4 . assignedSlot;
  assert_equals(tmp5, None);
  tmp6 ← n . ''c3_1'';
  tmp7 ← tmp6 . assignedSlot;
  assert_equals(tmp7, None);
  tmp8 ← n . ''c3_2'';
  tmp9 ← tmp8 . assignedSlot;
```

```
  assert_equals(tmp9, None);
  tmp10 ← n . ''s1'';
  tmp11 ← tmp10 . assignedNodes();
  assert_array_equals(tmp11, []);
  tmp12 ← n . ''s2'';
  tmp13 ← tmp12 . assignedNodes();
  assert_array_equals(tmp13, []);
  tmp14 ← n . ''s3'';
  tmp15 ← tmp14 . assignedNodes();
  assert_array_equals(tmp15, []);
  tmp16 ← n . ''s1'';
  tmp17 ← tmp16 . assignedNodes(True);
  assert_array_equals(tmp17, []);
  tmp18 ← n . ''s2'';
  tmp19 ← tmp18 . assignedNodes(True);
  assert_array_equals(tmp19, []);
  tmp20 ← n . ''s3'';
  tmp21 ← tmp20 . assignedNodes(True);
  assert_array_equals(tmp21, [])
}) slots_heap"
  by eval
```

'Slots: Name matching'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_slot_name_matching'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_name_matching'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None)
}) slots_heap"
  by eval
```

'Slots: No direct host child.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_no_direct_host_child'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_no_direct_host_child'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s1'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None);
  tmp10 ← n . ''s1'';
  tmp11 ← tmp10 . assignedNodes();
  tmp12 ← n . ''c1'';
  tmp13 ← n . ''c2'';
```

```
  assert_array_equals(tmp11, [tmp12, tmp13])
}) slots_heap"
  by eval
```

'Slots: Default Slot.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_default_slot'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_default_slot'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s2'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None)
}) slots_heap"
  by eval
```

'Slots: Slot in Slot does not matter in assignment.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_slot_in_slot'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_in_slot'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s2'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s1'';
  assert_equals(tmp6, tmp7)
}) slots_heap"
  by eval
```

'Slots: Slot is assigned to another slot'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_slot_is_assigned_to_slot'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_is_assigned_to_slot'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''s1'';
  tmp9 ← tmp8 . assignedNodes();
  tmp10 ← n . ''c1'';
  assert_array_equals(tmp9, [tmp10]);
  tmp11 ← n . ''s2'';
  tmp12 ← tmp11 . assignedNodes();
  tmp13 ← n . ''s1'';
  assert_array_equals(tmp12, [tmp13]);
```

```
  tmp14 ← n . ''s1'';
  tmp15 ← tmp14 . assignedNodes(True);
  tmp16 ← n . ''c1'';
  assert_array_equals(tmp15, [tmp16]);
  tmp17 ← n . ''s2'';
  tmp18 ← tmp17 . assignedNodes(True);
  tmp19 ← n . ''c1'';
  assert_array_equals(tmp18, [tmp19])
}) slots_heap"
  by eval
```

'Slots: Open > Closed.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_open_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_open_closed'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''c1'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s2'';
  tmp11 ← tmp10 . assignedNodes();
  tmp12 ← n . ''s1'';
  assert_array_equals(tmp11, [tmp12]);
  tmp13 ← n . ''s1'';
  tmp14 ← tmp13 . assignedNodes(True);
  tmp15 ← n . ''c1'';
  assert_array_equals(tmp14, [tmp15]);
  tmp16 ← n . ''s2'';
  tmp17 ← tmp16 . assignedNodes(True);
  tmp18 ← n . ''c1'';
  assert_array_equals(tmp17, [tmp18])
}) slots_heap"
  by eval
```

'Slots: Closed > Closed.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_closed_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_closed_closed'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedSlot;
  assert_equals(tmp5, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp6 ← n . ''s1'';
  tmp7 ← tmp6 . assignedNodes();
  tmp8 ← n . ''c1'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s2'';
  tmp10 ← tmp9 . assignedNodes();
  tmp11 ← n . ''s1'';
  assert_array_equals(tmp10, [tmp11]);
```

```
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  assert_array_equals(tmp16, [tmp17])
}) slots_heap"
  by eval
```

'Slots: Closed > Open.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_closed_open'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_closed_open'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedSlot;
  tmp6 ← n . ''s2'';
  assert_equals(tmp5, tmp6);
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''c1'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s2'';
  tmp11 ← tmp10 . assignedNodes();
  tmp12 ← n . ''s1'';
  assert_array_equals(tmp11, [tmp12]);
  tmp13 ← n . ''s1'';
  tmp14 ← tmp13 . assignedNodes(True);
  tmp15 ← n . ''c1'';
  assert_array_equals(tmp14, [tmp15]);
  tmp16 ← n . ''s2'';
  tmp17 ← tmp16 . assignedNodes(True);
  tmp18 ← n . ''c1'';
  assert_array_equals(tmp17, [tmp18])
}) slots_heap"
  by eval
```

'Slots: Complex case: Basi line.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  tmp10 ← n . ''s3'';
  assert_equals(tmp9, tmp10);
  tmp11 ← n . ''c4'';
  tmp12 ← tmp11 . assignedSlot;
```

```
assert_equals(tmp12, None);
tmp13 ← n . ''s1'';
tmp14 ← tmp13 . assignedSlot;
tmp15 ← n . ''s5'';
assert_equals(tmp14, tmp15);
tmp16 ← n . ''s2'';
tmp17 ← tmp16 . assignedSlot;
tmp18 ← n . ''s6'';
assert_equals(tmp17, tmp18);
tmp19 ← n . ''s3'';
tmp20 ← tmp19 . assignedSlot;
tmp21 ← n . ''s7'';
assert_equals(tmp20, tmp21);
tmp22 ← n . ''s4'';
tmp23 ← tmp22 . assignedSlot;
assert_equals(tmp23, None);
tmp24 ← n . ''c5'';
tmp25 ← tmp24 . assignedSlot;
tmp26 ← n . ''s5'';
assert_equals(tmp25, tmp26);
tmp27 ← n . ''c6'';
tmp28 ← tmp27 . assignedSlot;
tmp29 ← n . ''s6'';
assert_equals(tmp28, tmp29);
tmp30 ← n . ''c7'';
tmp31 ← tmp30 . assignedSlot;
tmp32 ← n . ''s7'';
assert_equals(tmp31, tmp32);
tmp33 ← n . ''c8'';
tmp34 ← tmp33 . assignedSlot;
assert_equals(tmp34, None);
tmp35 ← n . ''s1'';
tmp36 ← tmp35 . assignedNodes();
tmp37 ← n . ''c1'';
assert_array_equals(tmp36, [tmp37]);
tmp38 ← n . ''s2'';
tmp39 ← tmp38 . assignedNodes();
tmp40 ← n . ''c2'';
assert_array_equals(tmp39, [tmp40]);
tmp41 ← n . ''s3'';
tmp42 ← tmp41 . assignedNodes();
tmp43 ← n . ''c3'';
assert_array_equals(tmp42, [tmp43]);
tmp44 ← n . ''s4'';
tmp45 ← tmp44 . assignedNodes();
assert_array_equals(tmp45, []);
tmp46 ← n . ''s5'';
tmp47 ← tmp46 . assignedNodes();
tmp48 ← n . ''s1'';
tmp49 ← n . ''c5'';
assert_array_equals(tmp47, [tmp48, tmp49]);
tmp50 ← n . ''s6'';
tmp51 ← tmp50 . assignedNodes();
tmp52 ← n . ''s2'';
tmp53 ← n . ''c6'';
assert_array_equals(tmp51, [tmp52, tmp53]);
tmp54 ← n . ''s7'';
tmp55 ← tmp54 . assignedNodes();
tmp56 ← n . ''s3'';
tmp57 ← n . ''c7'';
assert_array_equals(tmp55, [tmp56, tmp57]);
tmp58 ← n . ''s8'';
tmp59 ← tmp58 . assignedNodes();
```

```
  assert_array_equals(tmp59, []);
  tmp60 ← n . ''s1'';
  tmp61 ← tmp60 . assignedNodes(True);
  tmp62 ← n . ''c1'';
  assert_array_equals(tmp61, [tmp62]);
  tmp63 ← n . ''s2'';
  tmp64 ← tmp63 . assignedNodes(True);
  tmp65 ← n . ''c2'';
  assert_array_equals(tmp64, [tmp65]);
  tmp66 ← n . ''s3'';
  tmp67 ← tmp66 . assignedNodes(True);
  tmp68 ← n . ''c3'';
  assert_array_equals(tmp67, [tmp68]);
  tmp69 ← n . ''s4'';
  tmp70 ← tmp69 . assignedNodes(True);
  assert_array_equals(tmp70, []);
  tmp71 ← n . ''s5'';
  tmp72 ← tmp71 . assignedNodes(True);
  tmp73 ← n . ''c1'';
  tmp74 ← n . ''c5'';
  assert_array_equals(tmp72, [tmp73, tmp74]);
  tmp75 ← n . ''s6'';
  tmp76 ← tmp75 . assignedNodes(True);
  tmp77 ← n . ''c2'';
  tmp78 ← n . ''c6'';
  assert_array_equals(tmp76, [tmp77, tmp78]);
  tmp79 ← n . ''s7'';
  tmp80 ← tmp79 . assignedNodes(True);
  tmp81 ← n . ''c3'';
  tmp82 ← n . ''c7'';
  assert_array_equals(tmp80, [tmp81, tmp82]);
  tmp83 ← n . ''s8'';
  tmp84 ← tmp83 . assignedNodes(True);
  assert_array_equals(tmp84, [])
}) slots_heap"
  by eval
```

'Slots: Mutation: appendChild.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  d1 ← slots_document . createElement(''div'');
  d1 . setAttribute(''slot'', ''slot1'');
  tmp2 ← n . ''host1'';
  tmp2 . appendChild(d1);
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5, d1]);
  tmp6 ← d1 . assignedSlot;
  tmp7 ← n . ''s1'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''s5'';
  tmp9 ← tmp8 . assignedNodes(True);
  tmp10 ← n . ''c1'';
  tmp11 ← n . ''c5'';
  assert_array_equals(tmp9, [tmp10, d1, tmp11])
}) slots_heap"
  by eval
```

'Slots: Mutation: Change slot= attribute 1.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . setAttribute(''slot'', ''slot-none'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''c1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s5'';
  tmp8 ← tmp7 . assignedNodes(True);
  tmp9 ← n . ''c5'';
  assert_array_equals(tmp8, [tmp9])
}) slots_heap"
  by eval
```

'Slots: Mutation: Change slot= attribute 2.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . setAttribute(''slot'', ''slot2'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s2'';
  tmp6 ← tmp5 . assignedNodes();
  tmp7 ← n . ''c1'';
  tmp8 ← n . ''c2'';
  assert_array_equals(tmp6, [tmp7, tmp8]);
  tmp9 ← n . ''c1'';
  tmp10 ← tmp9 . assignedSlot;
  tmp11 ← n . ''s2'';
  assert_equals(tmp10, tmp11);
  tmp12 ← n . ''s5'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c5'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s6'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  tmp18 ← n . ''c2'';
  tmp19 ← n . ''c6'';
  assert_array_equals(tmp16, [tmp17, tmp18, tmp19])
}) slots_heap"
  by eval
```

'Slots: Mutation: Change slot= attribute 3.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c4'';
  tmp2 . setAttribute(''slot'', ''slot1'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
```

```
  tmp5 ← n . ''c1'';
  tmp6 ← n . ''c4'';
  assert_array_equals(tmp4, [tmp5, tmp6]);
  tmp7 ← n . ''c4'';
  tmp8 ← tmp7 . assignedSlot;
  tmp9 ← n . ''s1'';
  assert_equals(tmp8, tmp9);
  tmp10 ← n . ''s5'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''c1'';
  tmp13 ← n . ''c4'';
  tmp14 ← n . ''c5'';
  assert_array_equals(tmp11, [tmp12, tmp13, tmp14])
}) slots_heap"
  by eval
```

'Slots: Mutation: Remove a child.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''c1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s5'';
  tmp8 ← tmp7 . assignedNodes(True);
  tmp9 ← n . ''c5'';
  assert_array_equals(tmp8, [tmp9])
}) slots_heap"
  by eval
```

'Slots: Mutation: Add a slot: after.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  slot ← slots_document . createElement(''slot'');
  slot . setAttribute(''name'', ''slot1'');
  tmp2 ← n . ''host2'';
  tmp2 . appendChild(slot);
  tmp3 ← slot . assignedNodes();
  assert_array_equals(tmp3, [])
}) slots_heap"
  by eval
```

'Slots: Mutation: Add a slot: before.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  slot ← slots_document . createElement(''slot'');
  slot . setAttribute(''name'', ''slot1'');
  tmp3 ← n . ''s1'';
  tmp2 ← n . ''host2'';
  tmp2 . insertBefore(slot, tmp3);
```

```
  tmp4 ← slot . assignedNodes();
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''c1'';
  tmp7 ← tmp6 . assignedSlot;
  assert_equals(tmp7, slot);
  tmp8 ← n . ''s7'';
  tmp9 ← tmp8 . assignedNodes();
  tmp10 ← n . ''s3'';
  tmp11 ← n . ''c7'';
  assert_array_equals(tmp9, [slot, tmp10, tmp11]);
  tmp12 ← n . ''s7'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  tmp15 ← n . ''c3'';
  tmp16 ← n . ''c7'';
  assert_array_equals(tmp13, [tmp14, tmp15, tmp16])
}) slots_heap"
  by eval
```

'Slots: Mutation: Remove a slot.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''c1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s5'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''c5'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s5'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''c5'';
  assert_array_equals(tmp11, [tmp12])
}) slots_heap"
  by eval
```

'Slots: Mutation: Change slot name= attribute.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . setAttribute(''name'', ''slot2'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c2'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''c1'';
  tmp7 ← tmp6 . assignedSlot;
  assert_equals(tmp7, None);
  tmp8 ← n . ''c2'';
  tmp9 ← tmp8 . assignedSlot;
  tmp10 ← n . ''s1'';
```

```
  assert_equals(tmp9, tmp10);
  tmp11 ← n . ''s5'';
  tmp12 ← tmp11 . assignedNodes();
  tmp13 ← n . ''s1'';
  tmp14 ← n . ''c5'';
  assert_array_equals(tmp12, [tmp13, tmp14]);
  tmp15 ← n . ''s5'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c2'';
  tmp18 ← n . ''c5'';
  assert_array_equals(tmp16, [tmp17, tmp18])
}) slots_heap"
  by eval
```

'Slots: Mutation: Change slot slot= attribute.'

```
lemma "test (do {
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . setAttribute(''slot'', ''slot6'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''s5'';
  tmp7 ← tmp6 . assignedNodes();
  tmp8 ← n . ''c5'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s6'';
  tmp10 ← tmp9 . assignedNodes();
  tmp11 ← n . ''s1'';
  tmp12 ← n . ''s2'';
  tmp13 ← n . ''c6'';
  assert_array_equals(tmp10, [tmp11, tmp12, tmp13]);
  tmp14 ← n . ''s6'';
  tmp15 ← tmp14 . assignedNodes(True);
  tmp16 ← n . ''c1'';
  tmp17 ← n . ''c2'';
  tmp18 ← n . ''c6'';
  assert_array_equals(tmp15, [tmp16, tmp17, tmp18])
}) slots_heap"
  by eval
```

end

3.3 Testing slots_fallback (slots_fallback)

This theory contains the test cases for slots_fallback.

```
theory slots_fallback
imports
  "Shadow_DOM_BaseTest"
begin

definition slots_fallback_heap :: "heap_final" where
  "slots_fallback_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some (cast
(element_ptr.Ref 1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
8)] fmempty None)),
```

```
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Shadow%20DOM%3A%20Slots%20and%20fallback%20
    (cast (element_ptr.Ref 4), cast (create_element_obj ''meta'' [] (fmap_of_list [(''name'', ''author''),
(''title'', ''Hayato Ito''), (''href'', ''mailto:hayato@google.com'')]) None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''resources/shadow-d
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''body'' [cast (element_ptr.Ref 9), cast (element_ptr.Ref
13), cast (element_ptr.Ref 14), cast (element_ptr.Ref 19), cast (element_ptr.Ref 20), cast (element_ptr.Ref
26), cast (element_ptr.Ref 27), cast (element_ptr.Ref 33), cast (element_ptr.Ref 34), cast (element_ptr.Ref
46)] fmempty None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''div'' [cast (element_ptr.Ref 10)] (fmap_of_list
[(''id'', ''test1'')]) None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''host'')])
(Some (cast (shadow_root_ptr.Ref 1))))),
    (cast (shadow_root_ptr.Ref 1), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 11)])),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 12)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 12), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 13), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 14), cast (create_element_obj ''div'' [cast (element_ptr.Ref 15)] (fmap_of_list
[(''id'', ''test2'')]) None)),
    (cast (element_ptr.Ref 15), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''host'')])
(Some (cast (shadow_root_ptr.Ref 2))))),
    (cast (shadow_root_ptr.Ref 2), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 16)])),
    (cast (element_ptr.Ref 16), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 17)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 17), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 18)] (fmap_of_list
[(''id'', ''s2''), (''name'', ''slot2'')]) None)),
    (cast (element_ptr.Ref 18), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 20), cast (create_element_obj ''div'' [cast (element_ptr.Ref 21)] (fmap_of_list
[(''id'', ''test3'')]) None)),
    (cast (element_ptr.Ref 21), cast (create_element_obj ''div'' [cast (element_ptr.Ref 22)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 3))))),
    (cast (element_ptr.Ref 22), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 3), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 23)])),
    (cast (element_ptr.Ref 23), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 24)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 24), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 25)] (fmap_of_list
[(''id'', ''s2''), (''name'', ''slot2'')]) None)),
    (cast (element_ptr.Ref 25), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 26), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 4)] fmempty
None)),
    (cast (character_data_ptr.Ref 4), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 27), cast (create_element_obj ''div'' [cast (element_ptr.Ref 28)] (fmap_of_list
[(''id'', ''test4'')]) None)),
    (cast (element_ptr.Ref 28), cast (create_element_obj ''div'' [cast (element_ptr.Ref 29)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 4))))),
    (cast (element_ptr.Ref 29), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 4), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 30)])),
```

```
    (cast (element_ptr.Ref 30), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 31)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 31), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 32)] (fmap_of_list
[(''id'', ''s2''), (''name'', ''slot2'')]) None)),
    (cast (element_ptr.Ref 32), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 33), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 5)] fmempty
None)),
    (cast (character_data_ptr.Ref 5), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 34), cast (create_element_obj ''div'' [cast (element_ptr.Ref 35)] (fmap_of_list
[(''id'', ''test5'')]) None)),
    (cast (element_ptr.Ref 35), cast (create_element_obj ''div'' [cast (element_ptr.Ref 36)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 5))))),
    (cast (element_ptr.Ref 36), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 5), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 37)])),
    (cast (element_ptr.Ref 37), cast (create_element_obj ''div'' [cast (element_ptr.Ref 38)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 6))))),
    (cast (element_ptr.Ref 38), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 39), cast (element_ptr.Ref
41)] (fmap_of_list [(''id'', ''s2''), (''name'', ''slot2''), (''slot'', ''slot3'')]) None)),
    (cast (element_ptr.Ref 39), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 40)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 40), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 41), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f2'')]) None)),
    (cast (shadow_root_ptr.Ref 6), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 42)])),
    (cast (element_ptr.Ref 42), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 43), cast (element_ptr.Ref
45)] (fmap_of_list [(''id'', ''s4''), (''name'', ''slot4'')]) None)),
    (cast (element_ptr.Ref 43), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 44)] (fmap_of_list
[(''id'', ''s3''), (''name'', ''slot3'')]) None)),
    (cast (element_ptr.Ref 44), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f3'')]) None)),
    (cast (element_ptr.Ref 45), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f4'')]) None)),
    (cast (element_ptr.Ref 46), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 6)] fmempty
None)),
    (cast (character_data_ptr.Ref 6), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

definition *slots_fallback_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option"* **where** *"slots_fallba*
= Some (cast (document_ptr.Ref 1))"

'Slots fallback: Basic.'

lemma *"test (do {*

```
  tmp0 ← slots_fallback_document . getElementById(''test1'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test1'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''f1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedNodes();
  assert_array_equals(tmp5, []);
  tmp6 ← n . ''s1'';
  tmp7 ← tmp6 . assignedNodes(True);
  tmp8 ← n . ''f1'';
  assert_array_equals(tmp7, [tmp8])
}) slots_fallback_heap"
```

by *eval*

'Slots fallback: Basic, elements only.'

lemma *"test (do {*

```
  tmp0 ← slots_fallback_document . getElementById(''test1'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  assert_array_equals(tmp2, []);
```

```
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedElements(True);
  tmp5 ← n . ''f1'';
  assert_array_equals(tmp4, [tmp5])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Slots in Slots.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test2'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test2'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''f1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedNodes();
  assert_array_equals(tmp5, []);
  tmp6 ← n . ''s2'';
  tmp7 ← tmp6 . assignedNodes();
  assert_array_equals(tmp7, []);
  tmp8 ← n . ''s1'';
  tmp9 ← tmp8 . assignedNodes(True);
  tmp10 ← n . ''f1'';
  assert_array_equals(tmp9, [tmp10]);
  tmp11 ← n . ''s2'';
  tmp12 ← tmp11 . assignedNodes(True);
  tmp13 ← n . ''f1'';
  assert_array_equals(tmp12, [tmp13])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Slots in Slots, elements only.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test2'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  assert_array_equals(tmp2, []);
  tmp3 ← n . ''s2'';
  tmp4 ← tmp3 . assignedElements();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedElements(True);
  tmp7 ← n . ''f1'';
  assert_array_equals(tmp6, [tmp7]);
  tmp8 ← n . ''s2'';
  tmp9 ← tmp8 . assignedElements(True);
  tmp10 ← n . ''f1'';
  assert_array_equals(tmp9, [tmp10])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Fallback contents should not be used if a node is assigned.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test3'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test3'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
```

```
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''f1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''c1'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s2'';
  tmp11 ← tmp10 . assignedNodes();
  assert_array_equals(tmp11, []);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''f1'';
  assert_array_equals(tmp16, [tmp17])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Slots in Slots: Assigned nodes should be used as fallback contents of another slot'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test4'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test4'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s2'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''f1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
  assert_array_equals(tmp8, []);
  tmp9 ← n . ''s2'';
  tmp10 ← tmp9 . assignedNodes();
  tmp11 ← n . ''c1'';
  assert_array_equals(tmp10, [tmp11]);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  assert_array_equals(tmp16, [tmp17])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Complex case.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp3 ← tmp2 . assignedNodes();
  tmp4 ← n . ''c1'';
  assert_array_equals(tmp3, [tmp4]);
```

```
  tmp5 ← n . ''s2'';
  tmp6 ← tmp5 . assignedNodes();
  assert_array_equals(tmp6, []);
  tmp7 ← n . ''s3'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''s2'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s4'';
  tmp11 ← tmp10 . assignedNodes();
  assert_array_equals(tmp11, []);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  tmp18 ← n . ''f2'';
  assert_array_equals(tmp16, [tmp17, tmp18]);
  tmp19 ← n . ''s3'';
  tmp20 ← tmp19 . assignedNodes(True);
  tmp21 ← n . ''c1'';
  tmp22 ← n . ''f2'';
  assert_array_equals(tmp20, [tmp21, tmp22]);
  tmp23 ← n . ''s4'';
  tmp24 ← tmp23 . assignedNodes(True);
  tmp25 ← n . ''c1'';
  tmp26 ← n . ''f2'';
  tmp27 ← n . ''f4'';
  assert_array_equals(tmp24, [tmp25, tmp26, tmp27])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Complex case, elements only.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  tmp3 ← n . ''c1'';
  assert_array_equals(tmp2, [tmp3]);
  tmp4 ← n . ''s2'';
  tmp5 ← tmp4 . assignedElements();
  assert_array_equals(tmp5, []);
  tmp6 ← n . ''s3'';
  tmp7 ← tmp6 . assignedElements();
  tmp8 ← n . ''s2'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s4'';
  tmp10 ← tmp9 . assignedElements();
  assert_array_equals(tmp10, []);
  tmp11 ← n . ''s1'';
  tmp12 ← tmp11 . assignedElements(True);
  tmp13 ← n . ''c1'';
  assert_array_equals(tmp12, [tmp13]);
  tmp14 ← n . ''s2'';
  tmp15 ← tmp14 . assignedElements(True);
  tmp16 ← n . ''c1'';
  tmp17 ← n . ''f2'';
  assert_array_equals(tmp15, [tmp16, tmp17]);
  tmp18 ← n . ''s3'';
  tmp19 ← tmp18 . assignedElements(True);
  tmp20 ← n . ''c1'';
```

```
  tmp21 ← n . ''f2'';
  assert_array_equals(tmp19, [tmp20, tmp21]);
  tmp22 ← n . ''s4'';
  tmp23 ← tmp22 . assignedElements(True);
  tmp24 ← n . ''c1'';
  tmp25 ← n . ''f2'';
  tmp26 ← n . ''f4'';
  assert_array_equals(tmp23, [tmp24, tmp25, tmp26])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Mutation. Append fallback contents.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  d1 ← slots_fallback_document . createElement(''div'');
  tmp2 ← n . ''s2'';
  tmp2 . appendChild(d1);
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes(True);
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''s2'';
  tmp7 ← tmp6 . assignedNodes(True);
  tmp8 ← n . ''c1'';
  tmp9 ← n . ''f2'';
  assert_array_equals(tmp7, [tmp8, tmp9, d1]);
  tmp10 ← n . ''s3'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''c1'';
  tmp13 ← n . ''f2'';
  assert_array_equals(tmp11, [tmp12, tmp13, d1]);
  tmp14 ← n . ''s4'';
  tmp15 ← tmp14 . assignedNodes(True);
  tmp16 ← n . ''c1'';
  tmp17 ← n . ''f2'';
  tmp18 ← n . ''f4'';
  assert_array_equals(tmp15, [tmp16, tmp17, d1, tmp18])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Mutation. Remove fallback contents.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''f2'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes(True);
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''s2'';
  tmp7 ← tmp6 . assignedNodes(True);
  tmp8 ← n . ''c1'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s3'';
  tmp10 ← tmp9 . assignedNodes(True);
  tmp11 ← n . ''c1'';
  assert_array_equals(tmp10, [tmp11]);
```

```
  tmp12 ← n . ''s4'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  tmp15 ← n . ''f4'';
  assert_array_equals(tmp13, [tmp14, tmp15])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Mutation. Assign a node to a slot so that fallback contens are no longer used.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  d2 ← slots_fallback_document . createElement(''div'');
  d2 . setAttribute(''slot'', ''slot2'');
  tmp2 ← n . ''host1'';
  tmp2 . appendChild(d2);
  tmp3 ← n . ''s2'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, [d2]);
  tmp5 ← n . ''s2'';
  tmp6 ← tmp5 . assignedNodes(True);
  assert_array_equals(tmp6, [d2]);
  tmp7 ← n . ''s3'';
  tmp8 ← tmp7 . assignedNodes(True);
  assert_array_equals(tmp8, [d2]);
  tmp9 ← n . ''s4'';
  tmp10 ← tmp9 . assignedNodes(True);
  tmp11 ← n . ''f4'';
  assert_array_equals(tmp10, [d2, tmp11])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Mutation. Remove an assigned node from a slot so that fallback contens will be used.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedNodes(True);
  tmp7 ← n . ''f1'';
  assert_array_equals(tmp6, [tmp7]);
  tmp8 ← n . ''s2'';
  tmp9 ← tmp8 . assignedNodes(True);
  tmp10 ← n . ''f1'';
  tmp11 ← n . ''f2'';
  assert_array_equals(tmp9, [tmp10, tmp11]);
  tmp12 ← n . ''s3'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''f1'';
  tmp15 ← n . ''f2'';
  assert_array_equals(tmp13, [tmp14, tmp15]);
  tmp16 ← n . ''s4'';
  tmp17 ← tmp16 . assignedNodes(True);
  tmp18 ← n . ''f1'';
  tmp19 ← n . ''f2'';
```

```
  tmp20 ← n . ''f4'';
  assert_array_equals(tmp17, [tmp18, tmp19, tmp20])
}) slots_fallback_heap"
  by eval
```

'Slots fallback: Mutation. Remove a slot which is a fallback content of another slot.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedNodes(True);
  assert_array_equals(tmp6, [], ''fall back contents should be empty because s1 is not in a shadow tree.'');
  tmp7 ← n . ''s2'';
  tmp8 ← tmp7 . assignedNodes(True);
  tmp9 ← n . ''f2'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s3'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''f2'';
  assert_array_equals(tmp11, [tmp12]);
  tmp13 ← n . ''s4'';
  tmp14 ← tmp13 . assignedNodes(True);
  tmp15 ← n . ''f2'';
  tmp16 ← n . ''f4'';
  assert_array_equals(tmp14, [tmp15, tmp16])
}) slots_fallback_heap"
  by eval
```

end

3.4 Testing Document_adoptNode (Shadow_DOM_Document_adoptNode)

This theory contains the test cases for Document_adoptNode.

```
theory Shadow_DOM_Document_adoptNode
imports
  "Shadow_DOM_BaseTest"
begin

definition Document_adoptNode_heap :: heap_final where
  "Document_adoptNode_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some
(cast (element_ptr.Ref 1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''meta'' [] (fmap_of_list [(''charset'', ''utf-8'')])
None)),
    (cast (element_ptr.Ref 4), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Document.adoptNode'')),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''link'' [] (fmap_of_list [(''rel'', ''help''),
(''href'', ''https://dom.spec.whatwg.org/#dom-document-adoptnode'')]) None)),
```

```
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''body'' [cast (element_ptr.Ref 9), cast (element_ptr.Ref
10), cast (element_ptr.Ref 11)] fmempty None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''x<'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''x'')),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

definition *Document_adoptNode_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option"* **where** *"Document_adoptNode_document = Some (cast (document_ptr.Ref 1))"*

"Adopting an Element called 'x<' should work."

lemma *"test (do {*

```
  tmp0 ← Document_adoptNode_document . getElementsByTagName(''x<'');
  y ← return (tmp0 ! 0);
  child ← y . firstChild;
  tmp1 ← y . parentNode;
  tmp2 ← Document_adoptNode_document . body;
  assert_equals(tmp1, tmp2);
  tmp3 ← y . ownerDocument;
  assert_equals(tmp3, Document_adoptNode_document);
  tmp4 ← Document_adoptNode_document . adoptNode(y);
  assert_equals(tmp4, y);
  tmp5 ← y . parentNode;
  assert_equals(tmp5, None);
  tmp6 ← y . firstChild;
  assert_equals(tmp6, child);
  tmp7 ← y . ownerDocument;
  assert_equals(tmp7, Document_adoptNode_document);
  tmp8 ← child . ownerDocument;
  assert_equals(tmp8, Document_adoptNode_document);
  doc ← createDocument(None, None, None);
  tmp9 ← doc . adoptNode(y);
  assert_equals(tmp9, y);
  tmp10 ← y . parentNode;
  assert_equals(tmp10, None);
  tmp11 ← y . firstChild;
  assert_equals(tmp11, child);
  tmp12 ← y . ownerDocument;
  assert_equals(tmp12, doc);
  tmp13 ← child . ownerDocument;
  assert_equals(tmp13, doc)
}) Document_adoptNode_heap"
```

by *eval*

"Adopting an Element called ':good:times:' should work."

lemma *"test (do {*

```
  x ← Document_adoptNode_document . createElement(''':good:times:'');
  tmp0 ← Document_adoptNode_document . adoptNode(x);
  assert_equals(tmp0, x);
  doc ← createDocument(None, None, None);
  tmp1 ← doc . adoptNode(x);
  assert_equals(tmp1, x);
  tmp2 ← x . parentNode;
  assert_equals(tmp2, None);
  tmp3 ← x . ownerDocument;
  assert_equals(tmp3, doc)
```

```
}) Document_adoptNode_heap"
  by eval
```

end

3.5 Testing Document_getElementById (Shadow_DOM_Document_getElementById)

This theory contains the test cases for Document_getElementById.

```
theory Shadow_DOM_Document_getElementById
imports
  "Shadow_DOM_BaseTest"
begin

definition Document_getElementById_heap :: heap_final where
  "Document_getElementById_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html
(Some (cast (element_ptr.Ref 1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
9)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''meta'' [] (fmap_of_list [(''charset'', ''utf-8'')])
None)),
    (cast (element_ptr.Ref 4), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Document.getElementById'')),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''link'' [] (fmap_of_list [(''rel'', ''author''),
(''title'', ''Tetsuharu OHZEKI''), (''href'', ''mailto:saneyuki.snyk@gmail.com'')]) None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''link'' [] (fmap_of_list [(''rel'', ''help''),
(''href'', ''https://dom.spec.whatwg.org/#dom-document-getelementbyid'')]) None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''body'' [cast (element_ptr.Ref 10), cast (element_ptr.Ref
11), cast (element_ptr.Ref 12), cast (element_ptr.Ref 13), cast (element_ptr.Ref 16), cast (element_ptr.Ref
19)] fmempty None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', '''')]) None)),
    (cast (element_ptr.Ref 12), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''test1'')])
None)),
    (cast (element_ptr.Ref 13), cast (create_element_obj ''div'' [cast (element_ptr.Ref 14), cast (element_ptr.Ref
15)] (fmap_of_list [(''id'', ''test5''), (''data-name'', ''1st'')]) None)),
    (cast (element_ptr.Ref 14), cast (create_element_obj ''p'' [cast (character_data_ptr.Ref 2)] (fmap_of_list
[(''id'', ''test5''), (''data-name'', ''2nd'')]) None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''P'')),
    (cast (element_ptr.Ref 15), cast (create_element_obj ''input'' [] (fmap_of_list [(''id'', ''test5''),
(''type'', ''submit''), (''value'', ''Submit''), (''data-name'', ''3rd'')]) None)),
    (cast (element_ptr.Ref 16), cast (create_element_obj ''div'' [cast (element_ptr.Ref 17)] (fmap_of_list
[(''id'', ''outer'')]) None)),
    (cast (element_ptr.Ref 17), cast (create_element_obj ''div'' [cast (element_ptr.Ref 18)] (fmap_of_list
[(''id'', ''middle'')]) None)),
    (cast (element_ptr.Ref 18), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''inner'')])
None)),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"

definition Document_getElementById_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option" where
```

```
"Document_getElementById_document = Some (cast (document_ptr.Ref 1))"
```

"Document.getElementById with a script-inserted element"

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test2'';
  test ← Document_getElementById_document . createElement(''div'');
  test . setAttribute(''id'', TEST_ID);
  gBody . appendChild(test);
  result ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(result, None, ''should not be null.'');
  tmp0 ← result . tagName;
  assert_equals(tmp0, ''div'', ''should have appended element's tag name'');
  gBody . removeChild(test);
  removed ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(removed, None, ''should not get removed element.'')
}) Document_getElementById_heap"
  by eval
```

"update ‘id‘ attribute via setAttribute/removeAttribute"

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test3'';
  test ← Document_getElementById_document . createElement(''div'');
  test . setAttribute(''id'', TEST_ID);
  gBody . appendChild(test);
  UPDATED_ID ← return ''test3-updated'';
  test . setAttribute(''id'', UPDATED_ID);
  e ← Document_getElementById_document . getElementById(UPDATED_ID);
  assert_equals(e, test, ''should get the element with id.'');
  old ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(old, None, ''shouldn't get the element by the old id.'');
  test . removeAttribute(''id'');
  e2 ← Document_getElementById_document . getElementById(UPDATED_ID);
  assert_equals(e2, None, ''should return null when the passed id is none in document.'')
}) Document_getElementById_heap"
  by eval
```

"Ensure that the id attribute only affects elements present in a document"

```
lemma "test (do {
  TEST_ID ← return ''test4-should-not-exist'';
  e ← Document_getElementById_document . createElement(''div'');
  e . setAttribute(''id'', TEST_ID);
  tmp0 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp0, None, ''should be null'');
  tmp1 ← Document_getElementById_document . body;
  tmp1 . appendChild(e);
  tmp2 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp2, e, ''should be the appended element'')
}) Document_getElementById_heap"
  by eval
```

"in tree order, within the context object's tree"

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test5'';
  target ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(target, None, ''should not be null'');
  tmp0 ← target . getAttribute(''data-name'');
  assert_equals(tmp0, ''1st'', ''should return the 1st'');
  element4 ← Document_getElementById_document . createElement(''div'');
  element4 . setAttribute(''id'', TEST_ID);
```

```
  element4 . setAttribute(''data-name'', ''4th'');
  gBody . appendChild(element4);
  target2 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(target2, None, ''should not be null'');
  tmp1 ← target2 . getAttribute(''data-name'');
  assert_equals(tmp1, ''1st'', ''should be the 1st'');
  tmp2 ← target2 . parentNode;
  tmp2 . removeChild(target2);
  target3 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(target3, None, ''should not be null'');
  tmp3 ← target3 . getAttribute(''data-name'');
  assert_equals(tmp3, ''4th'', ''should be the 4th'')
}) Document_getElementById_heap"
  by eval
```

"Modern browsers optimize this method with using internal id cache. This test checks that their optimization should effect only append to ‘Document‘, not append to ‘Node‘."

```
lemma "test (do {
  TEST_ID ← return ''test6'';
  s ← Document_getElementById_document . createElement(''div'');
  s . setAttribute(''id'', TEST_ID);
  tmp0 ← Document_getElementById_document . createElement(''div'');
  tmp0 . appendChild(s);
  tmp1 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp1, None, ''should be null'')
}) Document_getElementById_heap"
  by eval
```

"changing attribute’s value via ‘Attr‘ gotten from ‘Element.attribute‘."

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test7'';
  element ← Document_getElementById_document . createElement(''div'');
  element . setAttribute(''id'', TEST_ID);
  gBody . appendChild(element);
  target ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(target, element, ''should return the element before changing the value'');
  element . setAttribute(''id'', (TEST_ID @ ''-updated''));
  target2 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(target2, None, ''should return null after updated id via Attr.value'');
  target3 ← Document_getElementById_document . getElementById((TEST_ID @ ''-updated''));
  assert_equals(target3, element, ''should be equal to the updated element.'')
}) Document_getElementById_heap"
  by eval
```

"update ‘id‘ attribute via element.id"

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test12'';
  test ← Document_getElementById_document . createElement(''div'');
  test . setAttribute(''id'', TEST_ID);
  gBody . appendChild(test);
  UPDATED_ID ← return (TEST_ID @ ''-updated'');
  test . setAttribute(''id'', UPDATED_ID);
  e ← Document_getElementById_document . getElementById(UPDATED_ID);
  assert_equals(e, test, ''should get the element with id.'');
  old ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(old, None, ''shouldn't get the element by the old id.'');
  test . setAttribute(''id'', '''');
  e2 ← Document_getElementById_document . getElementById(UPDATED_ID);
  assert_equals(e2, None, ''should return null when the passed id is none in document.'')
}) Document_getElementById_heap"
  by eval
```

"where insertion order and tree order don't match"

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test13'';
  container ← Document_getElementById_document . createElement(''div'');
  container . setAttribute(''id'', (TEST_ID @ ''-fixture''));
  gBody . appendChild(container);
  element1 ← Document_getElementById_document . createElement(''div'');
  element1 . setAttribute(''id'', TEST_ID);
  element2 ← Document_getElementById_document . createElement(''div'');
  element2 . setAttribute(''id'', TEST_ID);
  element3 ← Document_getElementById_document . createElement(''div'');
  element3 . setAttribute(''id'', TEST_ID);
  element4 ← Document_getElementById_document . createElement(''div'');
  element4 . setAttribute(''id'', TEST_ID);
  container . appendChild(element2);
  container . appendChild(element4);
  container . insertBefore(element3, element4);
  container . insertBefore(element1, element2);
  test ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(test, element1, ''should return 1st element'');
  container . removeChild(element1);
  test ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(test, element2, ''should return 2nd element'');
  container . removeChild(element2);
  test ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(test, element3, ''should return 3rd element'');
  container . removeChild(element3);
  test ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(test, element4, ''should return 4th element'');
  container . removeChild(element4)
}) Document_getElementById_heap"
  by eval
```

"Inserting an id by inserting its parent node"

```
lemma "test (do {
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test14'';
  a ← Document_getElementById_document . createElement(''a'');
  b ← Document_getElementById_document . createElement(''b'');
  a . appendChild(b);
  b . setAttribute(''id'', TEST_ID);
  tmp0 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp0, None);
  gBody . appendChild(a);
  tmp1 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp1, b)
}) Document_getElementById_heap"
  by eval
```

"Document.getElementById must not return nodes not present in document"

```
lemma "test (do {
  TEST_ID ← return ''test15'';
  outer ← Document_getElementById_document . getElementById(''outer'');
  middle ← Document_getElementById_document . getElementById(''middle'');
  inner ← Document_getElementById_document . getElementById(''inner'');
  tmp0 ← Document_getElementById_document . getElementById(''middle'');
  outer . removeChild(tmp0);
  new_el ← Document_getElementById_document . createElement(''h1'');
  new_el . setAttribute(''id'', ''heading'');
  inner . appendChild(new_el);
  tmp1 ← Document_getElementById_document . getElementById(''heading'');
```

```
  assert_equals(tmp1, None)
}) Document_getElementById_heap"
  by eval
```

end

3.6 Testing Node_insertBefore (Shadow_DOM_Node_insertBefore)

This theory contains the test cases for Node_insertBefore.

```
theory Shadow_DOM_Node_insertBefore
imports
  "Shadow_DOM_BaseTest"
begin

definition Node_insertBefore_heap :: heap_final where
  "Node_insertBefore_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some
(cast (element_ptr.Ref 1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
6)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Node.insertBefore'')),
    (cast (element_ptr.Ref 4), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''body'' [cast (element_ptr.Ref 7), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"

definition Node_insertBefore_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option" where
"Node_insertBefore_document = Some (cast (document_ptr.Ref 1))"
```

"Calling insertBefore an a leaf node Text must throw HIERARCHY_REQUEST_ERR."

```
lemma "test (do {
  node ← Node_insertBefore_document . createTextNode(''Foo'');
  tmp0 ← Node_insertBefore_document . createTextNode(''fail'');
  assert_throws(HierarchyRequestError, node . insertBefore(tmp0, None))
}) Node_insertBefore_heap"
  by eval
```

"Calling insertBefore with an inclusive ancestor of the context object must throw HIERARCHY_REQUEST_ERR."

```
lemma "test (do {
  tmp1 ← Node_insertBefore_document . body;
  tmp2 ← Node_insertBefore_document . getElementById(''log'');
  tmp0 ← Node_insertBefore_document . body;
  assert_throws(HierarchyRequestError, tmp0 . insertBefore(tmp1, tmp2));
  tmp4 ← Node_insertBefore_document . documentElement;
  tmp5 ← Node_insertBefore_document . getElementById(''log'');
  tmp3 ← Node_insertBefore_document . body;
  assert_throws(HierarchyRequestError, tmp3 . insertBefore(tmp4, tmp5))
}) Node_insertBefore_heap"
  by eval
```

"Calling insertBefore with a reference child whose parent is not the context node must throw a NotFoundError."

```
lemma "test (do {
```

```
  a ← Node_insertBefore_document . createElement(''div'');
  b ← Node_insertBefore_document . createElement(''div'');
  c ← Node_insertBefore_document . createElement(''div'');
  assert_throws(NotFoundError, a . insertBefore(b, c))
}) Node_insertBefore_heap"
  by eval
```

"If the context node is a document, inserting a document or text node should throw a HierarchyRequestError."

```
lemma "test (do {
  doc ← createDocument(''title'');
  doc2 ← createDocument(''title2'');
  tmp0 ← doc . documentElement;
  assert_throws(HierarchyRequestError, doc . insertBefore(doc2, tmp0));
  tmp1 ← doc . createTextNode(''text'');
  tmp2 ← doc . documentElement;
  assert_throws(HierarchyRequestError, doc . insertBefore(tmp1, tmp2))
}) Node_insertBefore_heap"
  by eval
```

"Inserting a node before itself should not move the node"

```
lemma "test (do {
  a ← Node_insertBefore_document . createElement(''div'');
  b ← Node_insertBefore_document . createElement(''div'');
  c ← Node_insertBefore_document . createElement(''div'');
  a . appendChild(b);
  a . appendChild(c);
  tmp0 ← a . childNodes;
  assert_array_equals(tmp0, [b, c]);
  tmp1 ← a . insertBefore(b, b);
  assert_equals(tmp1, b);
  tmp2 ← a . childNodes;
  assert_array_equals(tmp2, [b, c]);
  tmp3 ← a . insertBefore(c, c);
  assert_equals(tmp3, c);
  tmp4 ← a . childNodes;
  assert_array_equals(tmp4, [b, c])
}) Node_insertBefore_heap"
  by eval
```

end

3.7 Testing Node_removeChild (Shadow_DOM_Node_removeChild)

This theory contains the test cases for Node_removeChild.

```
theory Shadow_DOM_Node_removeChild
imports
  "Shadow_DOM_BaseTest"
begin

definition Node_removeChild_heap :: heap_final where
  "Node_removeChild_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some
(cast (element_ptr.Ref 1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
7)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Node.removeChild'')),
    (cast (element_ptr.Ref 4), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
```

```
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''creators.js'')])
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''body'' [cast (element_ptr.Ref 8), cast (element_ptr.Ref
9), cast (element_ptr.Ref 10)] fmempty None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''iframe'' [] (fmap_of_list [(''src'', ''about:blank'')])
None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

definition *Node_removeChild_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option"* **where** *"Node_remov*
= Some (cast (document_ptr.Ref 1))"

"Passing a detached Element to removeChild should not affect it."

```
lemma "test (do {
  doc ← return Node_removeChild_document;
  s ← doc . createElement(''div'');
  tmp0 ← s . ownerDocument;
  assert_equals(tmp0, doc);
  tmp1 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp1 . removeChild(s));
  tmp2 ← s . ownerDocument;
  assert_equals(tmp2, doc)
}) Node_removeChild_heap"
  by eval
```

"Passing a non-detached Element to removeChild should not affect it."

```
lemma "test (do {
  doc ← return Node_removeChild_document;
  s ← doc . createElement(''div'');
  tmp0 ← doc . documentElement;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  tmp2 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp2 . removeChild(s));
  tmp3 ← s . ownerDocument;
  assert_equals(tmp3, doc)
}) Node_removeChild_heap"
  by eval
```

"Calling removeChild on an Element with no children should throw NOT_FOUND_ERR."

```
lemma "test (do {
  doc ← return Node_removeChild_document;
  s ← doc . createElement(''div'');
  tmp0 ← doc . body;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  assert_throws(NotFoundError, s . removeChild(doc))
}) Node_removeChild_heap"
  by eval
```

"Passing a detached Element to removeChild should not affect it."

```
lemma "test (do {
  doc ← createDocument('''');
  s ← doc . createElement(''div'');
  tmp0 ← s . ownerDocument;
  assert_equals(tmp0, doc);
```

```
  tmp1 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp1 . removeChild(s));
  tmp2 ← s . ownerDocument;
  assert_equals(tmp2, doc)
}) Node_removeChild_heap"
  by eval
```

"Passing a non-detached Element to removeChild should not affect it."

```
lemma "test (do {
  doc ← createDocument(’’’’);
  s ← doc . createElement(’’div’’);
  tmp0 ← doc . documentElement;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  tmp2 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp2 . removeChild(s));
  tmp3 ← s . ownerDocument;
  assert_equals(tmp3, doc)
}) Node_removeChild_heap"
  by eval
```

"Calling removeChild on an Element with no children should throw NOT_FOUND_ERR."

```
lemma "test (do {
  doc ← createDocument(’’’’);
  s ← doc . createElement(’’div’’);
  tmp0 ← doc . body;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  assert_throws(NotFoundError, s . removeChild(doc))
}) Node_removeChild_heap"
  by eval
```

"Passing a value that is not a Node reference to removeChild should throw TypeError."

```
lemma "test (do {
  tmp0 ← Node_removeChild_document . body;
  assert_throws(TypeError, tmp0 . removeChild(None))
}) Node_removeChild_heap"
  by eval
```

end

3.8 Shadow DOM Tests (Shadow_DOM_Tests)

```
theory Shadow_DOM_Tests
  imports
  "tests/slots"
  "tests/slots_fallback"
  "tests/Shadow_DOM_Document_adoptNode"
  "tests/Shadow_DOM_Document_getElementById"
  "tests/Shadow_DOM_Node_insertBefore"
  "tests/Shadow_DOM_Node_removeChild"
begin
end
```

Bibliography

[1] A. Bohannon and B. C. Pierce. Featherweight Firefox: Formalizing the core of a web browser. In *Usenix Conference on Web Application Development (WebApps)*, June 2010. URL `http://www.cis.upenn.edu/~bohannon/browser-model/`.

[2] A. D. Brucker. *An Interactive Proof Environment for Object-oriented Specifications*. PhD thesis, ETH Zurich, mar 2007. URL `https://www.brucker.ch/bibliography/abstract/brucker-interactive-2007`. ETH Dissertation No. 17097.

[3] A. D. Brucker and M. Herzberg. The core DOM. *Archive of Formal Proofs*, dec 2018. ISSN 2150-914x. URL `https://www.brucker.ch/bibliography/abstract/brucker.ea-afp-core-dom-2018-a`. `http://www.isa-afp.org/entries/Core_DOM.html`, Formal proof development.

[4] A. D. Brucker and M. Herzberg. Formalizing (web) standards: An application of test and proof. In C. Dubois and B. Wolff, editors, *TAP 2018: Tests And Proofs*, number 10889 in Lecture Notes in Computer Science, pages 159–166. Springer-Verlag, Heidelberg, 2018. ISBN 978-3-642-38915-3. doi: 10.1007/978-3-319-92994-1_9. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-standard-compliance-testing-2018`.

[5] A. D. Brucker and M. Herzberg. The safely composable DOM. *Archive of Formal Proofs*, Sept. 2020. ISSN 2150-914x. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-afp-core-sc-dom-2020`. `http://www.isa-afp.org/entries/Core_SC_DOM.html`, Formal proof development.

[6] A. D. Brucker and M. Herzberg. A formalization of safely composable web components. *Archive of Formal Proofs*, Sept. 2020. ISSN 2150-914x. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-afp-sc-dom-components-2020`. `http://www.isa-afp.org/entries/SC_DOM_Components.html`, Formal proof development.

[7] A. D. Brucker and M. Herzberg. Shadow dom: A formal model of the document object model *with Shadow Roots*. *Archive of Formal Proofs*, Sept. 2020. ISSN 2150-914x. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-afp-shadow-dom-2020`. `http://www.isa-afp.org/entries/Shadow_DOM.html`, Formal proof development.

[8] A. D. Brucker and M. Herzberg. A formally verified model of web components. In S.-S. Jongmans and F. Arbab, editors, *Formal Aspects of Component Software (FACS)*, number 12018 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2020. ISBN 3-540-25109-X. doi: 10.1007/978-3-030-40914-2_3. URL `https://www.brucker.ch/bibliography/abstract/brucker.ea-web-components-2019`.

[9] A. D. Brucker and B. Wolff. Interactive testing using HOL-TestGen. In W. Grieskamp and C. Weise, editors, *Formal Approaches to Testing of Software*, number 3997 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2005. ISBN 3-540-25109-X. doi: 10.1007/11759744_7. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-interactive-2005`.

[10] A. D. Brucker and B. Wolff. An extensible encoding of object-oriented data models in hol. *Journal of Automated Reasoning*, 41:219–249, 2008. ISSN 0168-7433. doi: 10.1007/s10817-008-9108-3. URL `https://www.brucker.ch/bibliography/abstract/brucker.ea-extensible-2008-b`.

[11] A. D. Brucker and B. Wolff. On theorem prover-based testing. *Formal Aspects of Computing*, 25(5):683–721, 2013. ISSN 0934-5043. doi: 10.1007/s00165-012-0222-y. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-theorem-prover-2012`.

[12] P. Gardner, G. Smith, M. J. Wheelhouse, and U. Zarfaty. DOM: towards a formal specification. In *PLAN-X 2008, Programming Language Technologies for XML, An ACM SIGPLAN Workshop colocated with POPL 2008, San Francisco, California, USA, January 9, 2008*, 2008. URL `http://gemo.futurs.inria.fr/events/PLANX2008/papers/p18.pdf`.

[13] M. Herzberg. *Formal Foundations for Provably Safe Web Components*. PhD thesis, The University of Sheffield, 2020.

[14] D. Jang, Z. Tatlock, and S. Lerner. Establishing browser security guarantees through formal shim verification. In T. Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 113–128. USENIX Association, 2012. URL `https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/jang`.

[15] G. Klein. Operating system verification — an overview. *Sādhanā*, 34(1):27–69, Feb. 2009.

[16] A. Raad, J. F. Santos, and P. Gardner. DOM: specification and client reasoning. In A. Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 401–422, 2016. ISBN 978-3-319-47957-6. doi: 10.1007/978-3-319-47958-3_21.

[17] W3C. W3C DOM4, Nov. 2015. URL `https://www.w3.org/TR/dom/`.

[18] WHATWG. DOM – living standard, Mar. 2017. URL `https://dom.spec.whatwg.org/commit-snapshots/6253e53af2fbfaa6d25ad09fd54280d8083b2a97/`. Last Updated 24 March 2017.