A Formalization of Safely Composable Web Components

Achim D. Brucker Michael Herzberg

September 9, 2026

*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

While the (safely composable) DOM with shadow trees provide the technical basis for defining web components, it does neither defines the concept of web components nor specifies the safety properties that web components should guarantee. Consequently, the standard also does not discuss how or even if the methods for modifying the DOM respect component boundaries. In AFP entry, we present a formally verified model of *safely composable* web components and define safety properties which ensure that different web components can only interact with each other using well-defined interfaces. Moreover, our verification of the application programming interface (API) of the DOM revealed numerous invariants that implementations of the DOM API need to preserve to ensure the integrity of components.

In comparison to the strict standard compliance formalization of Web Components in the AFP entry "DOM Components", the notion of components in this entry (based on "SC DOM" and "Shadow SC DOM") provides much stronger safety guarantees.

Keywords: Web Components, DOM

Contents

1 Introduction **7**

2 Safely Composable Web Components **11**

2.1 Core SC DOM Components (Core_DOM_DOM_Components) 11
2.2 Core SC DOM Components II (Core_DOM_SC_DOM_Components) 48
2.3 Scope Components (Core_DOM_SC_DOM_Components) 48
2.4 Shadow SC DOM Components (Shadow_DOM_DOM_Components) 108
2.5 Shadow root components (Shadow_DOM_DOM_Components) 108
2.6 Shadow SC DOM Components II (Shadow_DOM_SC_DOM_Components) 129
2.7 Shadow root scope components (Shadow_DOM_SC_DOM_Components) 129

1 Introduction

The trend towards ever more complex client-side web applications is unstoppable. Compared to traditional software development, client-side web development lacks a well-established component model which allows easily and safely reusing implementations. The Document Object Model (DOM) essentially defines a tree-like data structure (the *node tree*) for representing documents in general and HTML documents in particular.

Shadow trees are a recent addition to the DOM standard [10] to enable web developers to partition the node tree into "sub-trees." The vision of shadow trees is to enable web developers to provide a library of re-usable and customizable widgets. For example, let us consider a multi-tab view called *Fancy Tab*, which is a simplified version of [1].

The left-hand side of Figure 1.1 shows the rendered output of the widget in use while the right-hand side shows the HTML source code snippet. It provides a custom HTML tag `<fancy-tabs>` using an HTML template that developers can use to include the widget. Its children will be rendered inside the widget, more precisely, inside its *slots* (elements of type `slot`). It has a slot called "title" and a default slot, which receives all children that do not specify a "slot" attribute.

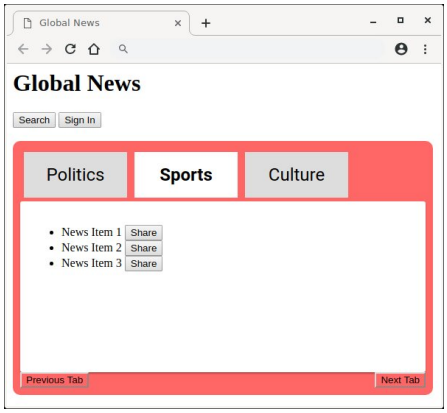
It is important to understand that slotting does *not change* the structure of the DOM (i. e., the underlying pointer graph): instead, slotting is implemented using special element attributes such as "slot," which control the final rendering. The DOM standard specifies methods that inspect the effect of these attributes such as `assigned_slot`, but the majority of DOM methods do not consider the semantics of these attributes and therefore do not traverse into shadow trees.

This provides an important boundary for client-side code. For example, a JavaScript program coming from the widget developer that changes the style attributes of the "Previous Tab" and "Next Tab" buttons in the lower corners of the widget will not affect buttons belonging to other parts coming from outside, i. e., the application of the widget consumer. Similarly, a JavaScript program that changes the styles of buttons outside of Fancy Tab, such as the navigation buttons, will not have any effect on them, even in the case of duplicate identifiers.

Sadly, the DOM standard neither defines the concept of web components nor specifies the safety properties that they should guarantee, not even informally. Consequently, the standard also does not discuss how or even if the methods for modifying the node tree respect component boundaries. Thus, shadow roots are only the very first step in defining a safe web component model.

Earlier [3, 4], we presented a formalization of the "flat" DOM (called Core DOM) without any support for shadow trees or components. We then extended this formalisation with support for shadow trees and slots [7].

In this AFP entries, we use the basis provided by our earlier work for defining a *formally verified model of web components* in general and, in particular, the notion of *weak* and *strong component safety*. For all methods that query, modify, or transform the DOM, we formally analyze their level of component safety. In more detail,



(a) User view

```
<fancy-tabs>
  <button slot="title">Politics</button>
  <button slot="title" selected>Sports</button>
  <button slot="title">Culture</button>
  <section>content panel 1</section>
  <ul>
    <li>News Item 1 <button>Share</button></li>
    <li>News Item 2 <button>Share</button></li>
    <li>News Item 3 <button>Share</button></li>
  </ul>
  <section>content panel 3</section>
</fancy-tabs>
```

(b) Consumer view

Figure 1.1: A simple example: a fancy tab component.

the contribution of this AFP entry is four-fold:

1. We provide a formal model of web components and their safety guarantees to web developers, enabling a compositional development of web applications,
2. for each method, we formally verify that it is either weakly or strongly component safe, or we provide a proof showing that it is not component safe,
3. we fill the gaps in the standard by explicitly formalizing invariants that are left out in the standard. These invariants are required to ensure that methods in the standard preserve a valid node tree. Finally,
4. we present a formal model of the DOM with shadow roots including the methods for querying, modifying, and transforming DOM instances with shadow roots.

Overall, our work gives web developers the guarantee that their code will respect the component boundaries as long as they abstain from or are careful when using certain DOM methods such as `appendChild` or `ownerDocument`.

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle (we refer readers interested in a more high-level presentation of the work to [8, 9]. The structure follows the theory dependencies (see Figure 1.2).

Important Note: This document describes the formalization of the *Safely Composable Web Components* (based on the SC DOM), 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” [2], “Shadow_DOM” [6], and “COM_Components” [5].

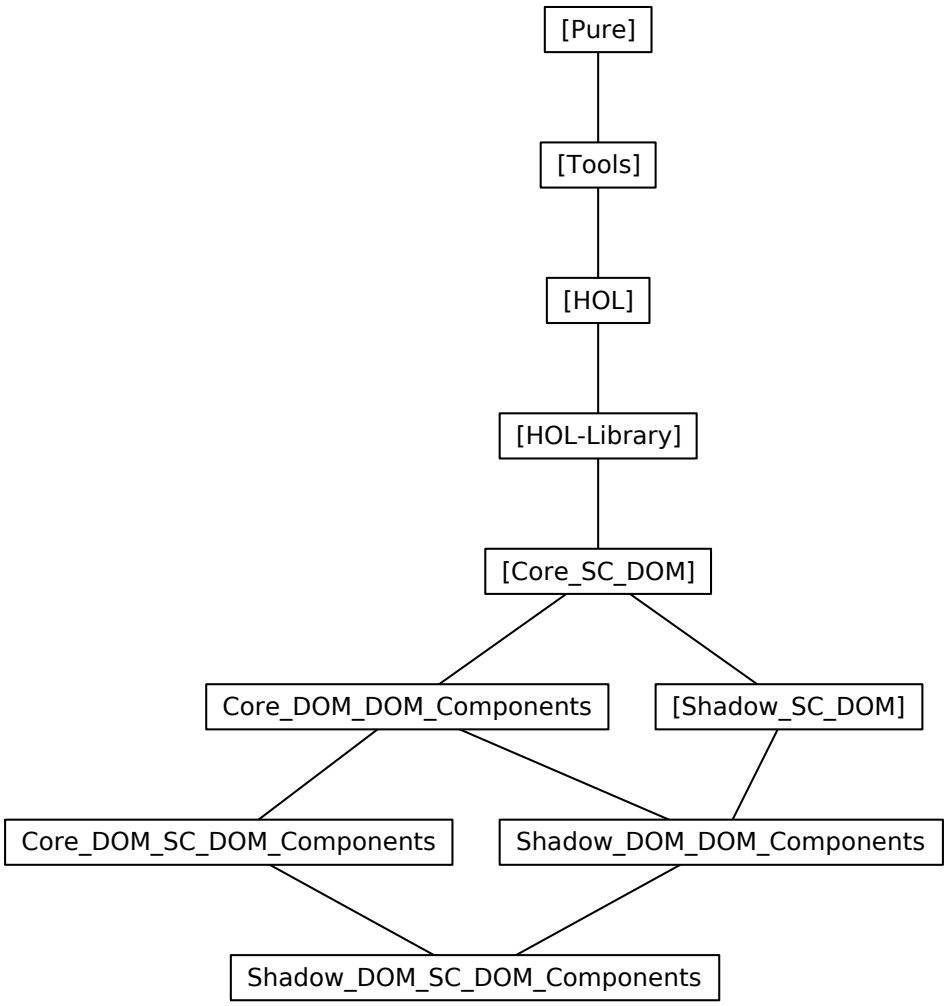


Figure 1.2: The Dependency Graph of the Isabelle Theories.

2 Safely Composable Web Components

2.1 Core SC DOM Components (Core_DOM_DOM_Components)

theory *Core_DOM_DOM_Components*
  imports *Core_SC_DOM.Core_DOM*
begin

2.1.1 Components

locale *l_get_dom_component*$_{Core_DOM}$*_defs =*
  l_get_root_node_defs get_root_node get_root_node_locs +
  l_to_tree_order_defs to_tree_order
  for *get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"*
    and *get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"*
    and *to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"*
begin

definition *a_get_dom_component :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"*
  where
    "a_get_dom_component ptr = do {
      root ← get_root_node ptr;
      to_tree_order root
    }"

definition *a_is_strongly_dom_component_safe ::*
  "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  where
    "a_is_strongly_dom_component_safe S_{arg} S_{result} *h h' = (*
      let removed_pointers = fset (object_ptr_kinds h) - fset (object_ptr_kinds h') in
      let added_pointers = fset (object_ptr_kinds h') - fset (object_ptr_kinds h) in
      let arg_components =
        (⋃ptr ∈ (⋃ptr ∈ S_{arg}*. set |h ⊢ a_get_dom_component ptr|*$_r$*) ∩*
        fset (object_ptr_kinds h). set |h ⊢ a_get_dom_component ptr|$_r$*) in*
      let arg_components' =
        (⋃ptr ∈ (⋃ptr ∈ S_{arg}*. set |h ⊢ a_get_dom_component ptr|*$_r$*) ∩*
        fset (object_ptr_kinds h'). set |h' ⊢ a_get_dom_component ptr|$_r$*) in*
      removed_pointers ⊆ arg_components ∧
      added_pointers ⊆ arg_components' ∧
      S_{result} *⊆ arg_components' ∧*
      (∀ outside_ptr ∈ fset (object_ptr_kinds h) ∩ fset (object_ptr_kinds h') -
        (⋃ptr ∈ S_{arg}*. set |h ⊢ a_get_dom_component ptr|*$_r$*). preserved (get_M outside_ptr id) h h'))"*

definition *a_is_weakly_dom_component_safe ::*
  "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  where
    "a_is_weakly_dom_component_safe S_{arg} S_{result} *h h' = (*
      let removed_pointers = fset (object_ptr_kinds h) - fset (object_ptr_kinds h') in
      let added_pointers = fset (object_ptr_kinds h') - fset (object_ptr_kinds h) in
      let arg_components =
        (⋃ptr ∈ (⋃ptr ∈ S_{arg}*. set |h ⊢ a_get_dom_component ptr|*$_r$*) ∩*
        fset (object_ptr_kinds h). set |h ⊢ a_get_dom_component ptr|$_r$*) in*
      let arg_components' =
        (⋃ptr ∈ (⋃ptr ∈ S_{arg}*. set |h ⊢ a_get_dom_component ptr|*$_r$*) ∩*
        fset (object_ptr_kinds h'). set |h' ⊢ a_get_dom_component ptr|$_r$*) in*
      removed_pointers ⊆ arg_components ∧
      S_{result} *⊆ arg_components' ∪ added_pointers ∧*

```
      (∀ outside_ptr ∈ fset (object_ptr_kinds h) ∩ fset (object_ptr_kinds h') -
        (⋃ptr ∈ S_arg. set |h ⊢ a_get_dom_component ptr|_r). preserved (get_M outside_ptr id) h h'))"

lemma "a_is_strongly_dom_component_safe S_arg S_result h h' ⟹ a_is_weakly_dom_component_safe S_arg S_result
h h'"
  by(auto simp add: a_is_strongly_dom_component_safe_def a_is_weakly_dom_component_safe_def Let_def)

definition is_document_component :: "(_) object_ptr list ⇒ bool"
  where
    "is_document_component c = is_document_ptr_kind (hd c)"

definition is_disconnected_component :: "(_) object_ptr list ⇒ bool"
  where
    "is_disconnected_component c = is_node_ptr_kind (hd c)"
end

global_interpretation l_get_dom_component_Core_DOM_defs get_root_node get_root_node_locs to_tree_order
  defines get_dom_component = a_get_dom_component
    and is_strongly_dom_component_safe = a_is_strongly_dom_component_safe
    and is_weakly_dom_component_safe = a_is_weakly_dom_component_safe
  .


locale l_get_dom_component_defs =
  fixes get_dom_component :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  fixes is_strongly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  fixes is_weakly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"

locale l_get_dom_component_Core_DOM =
  l_to_tree_order_wf +
  l_get_dom_component_defs +
  l_get_dom_component_Core_DOM_defs +
  l_get_ancestors +
  l_get_ancestors_wf +
  l_get_root_node +
  l_get_root_node_wf_Core_DOM +
  l_get_parent +
  l_get_parent_wf +
  l_get_element_by +
  l_to_tree_order_wf_get_root_node_wf +

  assumes get_dom_component_impl: "get_dom_component = a_get_dom_component"
  assumes is_strongly_dom_component_safe_impl:
    "is_strongly_dom_component_safe = a_is_strongly_dom_component_safe"
  assumes is_weakly_dom_component_safe_impl:
    "is_weakly_dom_component_safe = a_is_weakly_dom_component_safe"
begin
lemmas get_dom_component_def = a_get_dom_component_def[folded get_dom_component_impl]
lemmas is_strongly_dom_component_safe_def =
  a_is_strongly_dom_component_safe_def[folded is_strongly_dom_component_safe_impl]
lemmas is_weakly_dom_component_safe_def =
  a_is_weakly_dom_component_safe_def[folded is_weakly_dom_component_safe_impl]

lemma get_dom_component_ptr_in_heap:
  assumes "h ⊢ ok (get_dom_component ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms get_root_node_ptr_in_heap
  by(auto simp add: get_dom_component_def)

lemma get_dom_component_ok:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
```

```
  assumes "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_dom_component ptr)"
  using assms
  apply(auto simp add: get_dom_component_def a_get_root_node_def intro!: bind_is_OK_pure_I)[1]
  using get_root_node_ok  to_tree_order_ok get_root_node_ptr_in_heap
   apply blast
  by (simp add: local.get_root_node_root_in_heap local.to_tree_order_ok)

lemma get_dom_component_ptr:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  shows "ptr ∈ set c"
proof(insert assms(1) assms(4), induct ptr rule: heap_wellformed_induct_rev )
  case (step child)
  then show ?case
  proof (cases "is_node_ptr_kind child")
    case True
    obtain node_ptr where
      node_ptr: "cast_node_ptr2object_ptr node_ptr = child"
      using ‹is_node_ptr_kind child› node_ptr_casts_commute3 by blast
    have "child |∈| object_ptr_kinds h"
      using ‹h ⊢ get_dom_component child →r c› get_dom_component_ptr_in_heap by fast
    with node_ptr have "node_ptr |∈| node_ptr_kinds h"
      by auto
    then obtain parent_opt where
      parent: "h ⊢ get_parent node_ptr →r parent_opt"
      using get_parent_ok ‹type_wf h› ‹known_ptrs h›
      by fast
    then show ?thesis
    proof (induct parent_opt)
      case None
      then have "h ⊢ get_root_node (cast node_ptr) →r cast node_ptr"
        by (simp add: local.get_root_node_no_parent)
      then show ?case
        using ‹type_wf h› ‹known_ptrs h› node_ptr step(2)
        apply(auto simp add: get_dom_component_def a_get_root_node_def elim!: bind_returns_result_E2)[1]
        using to_tree_order_ptr_in_result returns_result_eq by fastforce
    next
      case (Some parent_ptr)
      then have "h ⊢ get_dom_component parent_ptr →r c"
        using step(2) node_ptr ‹type_wf h› ‹known_ptrs h› ‹heap_is_wellformed h›
          get_root_node_parent_same
        by(auto simp add: get_dom_component_def elim!: bind_returns_result_E2
            intro!: bind_pure_returns_result_I)
      then have "parent_ptr ∈ set c"
        using step node_ptr Some by blast
      then show ?case
        using  ‹type_wf h› ‹known_ptrs h› ‹heap_is_wellformed h› step(2) node_ptr Some
        apply(auto simp add: get_dom_component_def elim!: bind_returns_result_E2)[1]
        using to_tree_order_parent by blast
    qed
  next
    case False
    then show ?thesis
      using ‹type_wf h› ‹known_ptrs h› step(2)
      apply(auto simp add: get_dom_component_def elim!: bind_returns_result_E2)[1]
      by (metis is_OK_returns_result_I local.get_root_node_not_node_same
          local.get_root_node_ptr_in_heap local.to_tree_order_ptr_in_result returns_result_eq)
  qed
qed

lemma get_dom_component_parent_inside:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
```

```
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "cast node_ptr ∈ set c"
  assumes "h ⊢ get_parent node_ptr →r Some parent"
  shows "parent ∈ set c"
proof -
  have "parent |∈| object_ptr_kinds h"
    using assms(6) get_parent_parent_in_heap by blast

  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr" and c: "h ⊢ to_tree_order root_ptr
→r c"
    using assms(4)
    by (metis (no_types, opaque_lifting) bind_returns_result_E2 get_dom_component_def get_root_node_pure)
  then have "h ⊢ get_root_node (cast node_ptr) →r root_ptr"
    using assms(1) assms(2) assms(3) assms(5) to_tree_order_same_root by blast
  then have "h ⊢ get_root_node parent →r root_ptr"
    using assms(6)  get_root_node_parent_same by blast
  then have "h ⊢ get_dom_component parent →r c"
    using c get_dom_component_def by auto
  then show ?thesis
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
qed

lemma get_dom_component_subset:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "ptr' ∈ set c"
  shows "h ⊢ get_dom_component ptr' →r c"
proof(insert assms(1) assms(5), induct ptr' rule: heap_wellformed_induct_rev )
  case (step child)
  then show ?case
  proof (cases "is_node_ptr_kind child")
    case True
    obtain node_ptr where
      node_ptr: "cast_{node_ptr2object_ptr} node_ptr = child"
      using ‹is_node_ptr_kind child› node_ptr_casts_commute3 by blast
    have "child |∈| object_ptr_kinds h"
      using to_tree_order_ptrs_in_heap assms(1) assms(2) assms(3) assms(4) step(2)
      unfolding get_dom_component_def
      by (meson bind_returns_result_E2 get_root_node_pure)
    with node_ptr have "node_ptr |∈| node_ptr_kinds h"
      by auto
    then obtain parent_opt where
      parent: "h ⊢ get_parent node_ptr →r parent_opt"
      using get_parent_ok ‹type_wf h› ‹known_ptrs h›
      by fast
    then show ?thesis
    proof (induct parent_opt)
      case None
      then have "h ⊢ get_root_node child →r child"
        using assms(1) get_root_node_no_parent node_ptr by blast
      then show ?case
        using ‹type_wf h› ‹known_ptrs h› node_ptr step(2) assms(4) assms(1)
        by (metis (no_types) bind_pure_returns_result_I2 bind_returns_result_E2
            get_dom_component_def get_root_node_pure is_OK_returns_result_I returns_result_eq
            to_tree_order_same_root)
    next
      case (Some parent_ptr)
      then have "h ⊢ get_dom_component parent_ptr →r c"
        using step get_dom_component_parent_inside assms node_ptr by blast
      then show ?case
        using Some node_ptr
        apply(auto simp add: get_dom_component_def elim!: bind_returns_result_E2
            del: bind_pure_returns_result_I intro!: bind_pure_returns_result_I)[1]
```

```
      using get_root_node_parent_same by blast
    qed
  next
    case False
    then have "child |∈| object_ptr_kinds h"
      using assms(1) assms(4) step(2)
      by (metis (no_types, lifting) assms(2) assms(3) bind_returns_result_E2 get_root_node_pure
          get_dom_component_def to_tree_order_ptrs_in_heap)
    then have "h ⊢ get_root_node child →r child"
      using assms(1) False get_root_node_not_node_same by blast
    then show ?thesis
      using assms(1) assms(2) assms(3) assms(4) step.prems
      by (metis (no_types) False ‹child |∈| object_ptr_kinds h› bind_pure_returns_result_I2
          bind_returns_result_E2 get_dom_component_def get_root_node_ok get_root_node_pure returns_result_eq
          to_tree_order_node_ptrs)
  qed
qed

lemma get_dom_component_to_tree_order_subset:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ to_tree_order ptr →r nodes"
  assumes "h ⊢ get_dom_component ptr →r c"
  shows "set nodes ⊆ set c"
  using assms
  apply(auto simp add: get_dom_component_def elim!: bind_returns_result_E2)[1]
  by (meson to_tree_order_subset  assms(5) contra_subsetD get_dom_component_ptr)

lemma get_dom_component_to_tree_order:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ to_tree_order ptr' →r to"
  assumes "ptr ∈ set to"
  shows "h ⊢ get_dom_component ptr' →r c"
  by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3) assms(4) assms(5) assms(6)
      get_dom_component_ok get_dom_component_subset get_dom_component_to_tree_order_subset
      is_OK_returns_result_E local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap
      select_result_I2 subsetCE)

lemma get_dom_component_root_node_same:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ get_root_node ptr →r root_ptr"
  assumes "x ∈ set c"
  shows "h ⊢ get_root_node x →r root_ptr"
proof(insert assms(1) assms(6), induct x rule: heap_wellformed_induct_rev )
  case (step child)
  then show ?case
  proof (cases "is_node_ptr_kind child")
    case True
    obtain node_ptr where
      node_ptr: "cast_{node_ptr2object_ptr} node_ptr = child"
      using ‹is_node_ptr_kind child› node_ptr_casts_commute3 by blast
    have "child |∈| object_ptr_kinds h"
      using to_tree_order_ptrs_in_heap assms(1) assms(2) assms(3) assms(4) step(2)
      unfolding get_dom_component_def
      by (meson bind_returns_result_E2 get_root_node_pure)
    with node_ptr have "node_ptr |∈| node_ptr_kinds h"
      by auto
    then obtain parent_opt where
      parent: "h ⊢ get_parent node_ptr →r parent_opt"
      using get_parent_ok ‹type_wf h› ‹known_ptrs h›
      by fast
    then show ?thesis
```

```
    proof (induct parent_opt)
      case None
      then have "h ⊢ get_root_node child →r child"
        using assms(1) get_root_node_no_parent  node_ptr by blast
      then show ?case
        using ‹type_wf h› ‹known_ptrs h› node_ptr step(2) assms(4) assms(1) assms(5)
        by (metis (no_types) ‹child |∈| object_ptr_kinds h› bind_pure_returns_result_I
            get_dom_component_def get_dom_component_ptr get_dom_component_subset get_root_node_pure
            is_OK_returns_result_E returns_result_eq to_tree_order_ok to_tree_order_same_root)
    next
      case (Some parent_ptr)
      then have "h ⊢ get_dom_component parent_ptr →r c"
        using step get_dom_component_parent_inside assms node_ptr
        by (meson get_dom_component_subset)
      then show ?case
        using Some node_ptr
        apply(auto simp add: get_dom_component_def elim!: bind_returns_result_E2)[1]
        using get_root_node_parent_same
        using ‹h ⊢ get_dom_component parent_ptr →r c› assms(1) assms(2) assms(3)
          get_dom_component_ptr step.hyps by blast
    qed
  next
    case False
    then have "child |∈| object_ptr_kinds h"
      using assms(1) assms(4) step(2)
      by (metis (no_types, lifting) assms(2) assms(3) bind_returns_result_E2 get_root_node_pure
          get_dom_component_def to_tree_order_ptrs_in_heap)
    then have "h ⊢ get_root_node child →r child"
      using assms(1) False get_root_node_not_node_same by auto
    then show ?thesis
      using assms(1) assms(2) assms(3) assms(4) step.prems assms(5)
      by (metis (no_types, opaque_lifting) bind_returns_result_E2 get_dom_component_def
          get_root_node_pure returns_result_eq to_tree_order_same_root)
  qed
qed


lemma get_dom_component_no_overlap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ get_dom_component ptr' →r c'"
  shows "set c ∩ set c' = {} ∨ c = c'"
proof (rule ccontr, auto)
  fix x
  assume 1: "c ≠ c'" and 2: "x ∈ set c" and 3: "x ∈ set c'"
  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    using assms(4) unfolding get_dom_component_def
    by (meson bind_is_OK_E is_OK_returns_result_I)
  moreover obtain root_ptr' where root_ptr': "h ⊢ get_root_node ptr' →r root_ptr'"
    using assms(5) unfolding get_dom_component_def
    by (meson bind_is_OK_E is_OK_returns_result_I)
  ultimately have "root_ptr ≠ root_ptr'"
    using 1 assms
    unfolding get_dom_component_def
    by (meson bind_returns_result_E3 get_root_node_pure returns_result_eq)

  moreover have "h ⊢ get_root_node x →r root_ptr"
    using 2 root_ptr get_dom_component_root_node_same assms by blast
  moreover have "h ⊢ get_root_node x →r root_ptr'"
    using 3 root_ptr' get_dom_component_root_node_same assms by blast
  ultimately show False
    using select_result_I2 by force
qed
```

```
lemma get_dom_component_separates_tree_order:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ to_tree_order ptr →r to"
  assumes "h ⊢ get_dom_component ptr' →r c'"
  assumes "ptr' ∉ set c"
  shows "set to ∩ set c' = {}"
proof -
  have "c ≠ c'"
    using assms get_dom_component_ptr by blast
  then have "set c ∩ set c' = {}"
    using assms get_dom_component_no_overlap by blast
  moreover have "set to ⊆ set c"
    using assms get_dom_component_to_tree_order_subset by blast
  ultimately show ?thesis
    by blast
qed

lemma get_dom_component_separates_tree_order_general:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ to_tree_order ptr'' →r to''"
  assumes "ptr'' ∈ set c"
  assumes "h ⊢ get_dom_component ptr' →r c'"
  assumes "ptr' ∉ set c"
  shows "set to'' ∩ set c' = {}"
proof -
  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    using assms(4)
    by (metis bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  then have c: "h ⊢ to_tree_order root_ptr →r c"
    using assms(4) unfolding get_dom_component_def
    by (simp add: bind_returns_result_E3)
  with root_ptr show ?thesis
    using assms get_dom_component_separates_tree_order get_dom_component_subset
    by meson
qed
end

interpretation i_get_dom_component?: l_get_dom_component_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name
  by(auto simp add: l_get_dom_component_Core_DOM_def l_get_dom_component_Core_DOM_axioms_def
      get_dom_component_def is_strongly_dom_component_safe_def is_weakly_dom_component_safe_def instances)
declare l_get_dom_component_Core_DOM_axioms [instances]
```

get_child_nodes

```
locale l_get_dom_component_get_child_nodes_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_child_nodes_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ get_child_nodes ptr' →r children"
  assumes "child ∈ set children"
  shows "cast child ∈ set c ⟷ ptr' ∈ set c"
```

```
proof
  assume 1: "cast child ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node (cast child) →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →r root_ptr"
    using assms(1) assms(2) assms(3) assms(5) assms(6) local.child_parent_dual
      local.get_root_node_parent_same by blast
  then have "h ⊢ get_dom_component ptr' →r c"
    using "1" assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) local.child_parent_dual
      local.get_dom_component_parent_inside local.get_dom_component_subset by blast
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume 1: "ptr' ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node ptr' →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node (cast child) →r root_ptr"
    using assms(1) assms(2) assms(3) assms(5) assms(6) local.child_parent_dual
      local.get_root_node_parent_same by blast
  then have "h ⊢ get_dom_component ptr' →r c"
    using "1" assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) local.child_parent_dual
      local.get_dom_component_parent_inside local.get_dom_component_subset by blast
  then show "cast_{node_ptr2object_ptr} child ∈ set c"
    by (smt (verit) ‹h ⊢ get_root_node (cast_{node_ptr2object_ptr} child) →r root_ptr› assms(1) assms(2) assms(3)
        assms(5) assms(6) disjoint_iff_not_equal is_OK_returns_result_E is_OK_returns_result_I
        l_get_dom_component_{Core_DOM}.get_dom_component_no_overlap local.child_parent_dual local.get_dom_component_
        local.get_dom_component_parent_inside local.get_dom_component_ptr local.get_root_node_ptr_in_heap
        local.l_get_dom_component_{Core_DOM}_axioms)
qed


lemma get_child_nodes_get_dom_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ get_child_nodes ptr →r children"
  shows "cast ` set children ⊆ set c"
  using assms get_child_nodes_is_strongly_dom_component_safe
  using local.get_dom_component_ptr by auto


lemma
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_child_nodes ptr →r children"
  assumes "h ⊢ get_child_nodes ptr →h h'"
  shows "is_strongly_dom_component_safe {ptr} (cast ` set children) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_child_nodes_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def Bex_def)[1]
    by (smt (verit) IntI get_child_nodes_is_strongly_dom_component_safe
        is_OK_returns_result_I local.get_child_nodes_ptr_in_heap local.get_dom_component_impl
        local.get_dom_component_ok local.get_dom_component_ptr returns_result_select_result)
```

```
qed
end

interpretation i_get_dom_component_get_child_nodes?: l_get_dom_component_get_child_nodes_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_dom_component_get_child_nodes_Core_DOM_def instances)
declare l_get_dom_component_get_child_nodes_Core_DOM_axioms [instances]
```

get_parent

```
locale l_get_dom_component_get_parent_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_get_parent_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_parent_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →_r c"
  assumes "h ⊢ get_parent ptr' →_r Some parent"
  shows "parent ∈ set c ⟷ cast ptr' ∈ set c"
  by (meson assms(1) assms(2) assms(3) assms(4) assms(5) is_OK_returns_result_E
      l_to_tree_order_wf.to_tree_order_parent local.get_dom_component_parent_inside
      local.get_dom_component_subset local.get_dom_component_to_tree_order_subset
      local.get_parent_parent_in_heap local.l_to_tree_order_wf_axioms local.to_tree_order_ok
      local.to_tree_order_ptr_in_result subsetCE)


lemma get_parent_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_parent node_ptr →_r Some parent"
  assumes "h ⊢ get_parent node_ptr →_h h'"
  shows "is_strongly_dom_component_safe {cast node_ptr} {parent} h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_parent_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def)[1]
    by (metis IntI local.get_dom_component_impl local.get_dom_component_ok
        local.get_dom_component_parent_inside local.get_dom_component_ptr local.get_parent_parent_in_heap
        local.known_ptrs_known_ptr local.parent_child_rel_child_in_heap local.parent_child_rel_parent
        returns_result_select_result)
qed
end

interpretation i_get_dom_component_get_parent?: l_get_dom_component_get_parent_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_dom_component_get_parent_Core_DOM_def instances)
declare l_get_dom_component_get_parent_Core_DOM_axioms [instances]
```

get_root_node

```
locale l_get_dom_component_get_root_node_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_get_root_node_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_root_node_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →_r c"
  assumes "h ⊢ get_root_node ptr' →_r root"
  shows "root ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "root ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node root →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  moreover have "h ⊢ get_root_node ptr' →_r root_ptr"
    by (metis (no_types, lifting) calculation assms(1) assms(2) assms(3) assms(5)
        is_OK_returns_result_E local.get_root_node_root_in_heap local.to_tree_order_ok
        local.to_tree_order_ptr_in_result local.to_tree_order_same_root select_result_I2)
  ultimately have "h ⊢ get_dom_component ptr' →_r c"
    apply(auto simp add: get_dom_component_def)[1]
    using assms(4) bind_returns_result_E3 local.get_dom_component_def root_ptr by fastforce
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume 1: "ptr' ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node ptr' →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node root →_r root_ptr"
    by (metis assms(1) assms(2) assms(3) assms(5) is_OK_returns_result_E
        local.get_root_node_root_in_heap local.to_tree_order_ok local.to_tree_order_ptr_in_result
        local.to_tree_order_same_root returns_result_eq)
  then have "h ⊢ get_dom_component ptr' →_r c"
    using "1" assms(1) assms(2) assms(3) assms(4) local.get_dom_component_subset by blast
  then show "root ∈ set c"
    using assms(5) bind_returns_result_E3 local.get_dom_component_def local.to_tree_order_ptr_in_result
    by fastforce
qed

lemma get_root_node_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node ptr →_r root"
  assumes "h ⊢ get_root_node ptr →_h h'"
  shows "is_strongly_dom_component_safe {ptr} {root} h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_root_node_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def)[1]
    by (metis (no_types, opaque_lifting) IntI get_root_node_is_strongly_dom_component_safe_step
        is_OK_returns_result_I local.get_dom_component_impl local.get_dom_component_ok
```

```
    local.get_dom_component_ptr local.get_root_node_ptr_in_heap returns_result_select_result)
qed
end

interpretation i_get_dom_component_get_root_node?: l_get_dom_component_get_root_node_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_dom_component_get_root_node_Core_DOM_def instances)
declare l_get_dom_component_get_root_node_Core_DOM_axioms [instances]
```

get_element_by_id

```
locale l_get_dom_component_get_element_by_id_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_first_in_tree_order_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_element_by_id_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →_r c"
  assumes "h ⊢ get_element_by_id ptr' idd →_r Some result"
  shows "cast result ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast result ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  then have "h ⊢ to_tree_order root_ptr →_r c"
    using ‹h ⊢ get_dom_component ptr →_r c›
    by (simp add: get_dom_component_def bind_returns_result_E3)

  obtain to' where to': "h ⊢ to_tree_order ptr' →_r to'"
    using ‹h ⊢ get_element_by_id ptr' idd →_r Some result›
    apply(simp add: get_element_by_id_def first_in_tree_order_def)
    by (meson bind_is_OK_E is_OK_returns_result_I)
  then have "cast result ∈ set to'"
    using ‹h ⊢ get_element_by_id ptr' idd →_r Some result› get_element_by_id_result_in_tree_order
    by blast

  have "h ⊢ get_root_node (cast result) →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →_r root_ptr"
    using ‹cast result ∈ set to'› ‹h ⊢ to_tree_order ptr' →_r to'›
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_ptr get_dom_component_root_node_same
      get_dom_component_subset get_dom_component_to_tree_order
    by blast
  then have "h ⊢ get_dom_component ptr' →_r c"
    using ‹h ⊢ to_tree_order root_ptr →_r c›
    using get_dom_component_def by auto
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume "ptr' ∈ set c"
  moreover obtain to' where to': "h ⊢ to_tree_order ptr' →_r to'"
    by (meson assms(1) assms(2) assms(3) assms(4) calculation get_dom_component_ptr_in_heap
        get_dom_component_subset is_OK_returns_result_E is_OK_returns_result_I to_tree_order_ok)
  ultimately have "set to' ⊆ set c"
```

```
    using assms(1) assms(2) assms(3) assms(4) get_dom_component_subset
      get_dom_component_to_tree_order_subset
    by blast
  moreover have "cast result ∈ set to'"
    using assms(5) get_element_by_id_result_in_tree_order to' by blast
  ultimately show "cast result ∈ set c"
    by blast
qed


lemma get_element_by_id_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_element_by_id ptr idd →_r Some result"
  assumes "h ⊢ get_element_by_id ptr idd →_h h'"
  shows "is_strongly_dom_component_safe {ptr} {cast result} h h'"
proof -
  have "h = h'"
    using assms(5)
    by(auto simp add: preserved_def get_element_by_id_def first_in_tree_order_def
        elim!: bind_returns_heap_E2 intro!: map_filter_M_pure bind_pure_I
        split: option.splits list.splits)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_element_by_id_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →_r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast result ∈ set to"
    using assms(4) local.get_element_by_id_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_dom_component ptr →_r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_dom_component_impl
      local.get_dom_component_ok by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def get_element_by_id_def
        first_in_tree_order_def elim!: bind_returns_result_E2 intro!: map_filter_M_pure bind_pure_I
        split: option.splits list.splits)[1]
    by (metis (no_types, opaque_lifting) Int_iff ‹ptr |∈| object_ptr_kinds h› assms(4)
        get_element_by_id_is_strongly_dom_component_safe_step local.get_dom_component_impl
        local.get_dom_component_ptr select_result_I2)
qed
end

interpretation i_get_dom_component_get_element_by_id?: l_get_dom_component_get_element_by_id_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_dom_component_get_element_by_id_Core_DOM_def instances)
declare l_get_dom_component_get_element_by_id_Core_DOM_axioms [instances]
```

get_elements_by_class_name

```
locale l_get_dom_component_get_elements_by_class_name_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_first_in_tree_order_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_element_by_Core_DOM
begin
```

```
lemma get_elements_by_class_name_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →_r c"
  assumes "h ⊢ get_elements_by_class_name ptr' idd →_r results"
  assumes "result ∈ set results"
  shows "cast result ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast result ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  then have "h ⊢ to_tree_order root_ptr →_r c"
    using ‹h ⊢ get_dom_component ptr →_r c›
    by (simp add: get_dom_component_def bind_returns_result_E3)

  obtain to' where to': "h ⊢ to_tree_order ptr' →_r to'"
    using assms(5)
    apply(simp add: get_elements_by_class_name_def first_in_tree_order_def)
    by (meson bind_is_OK_E is_OK_returns_result_I)
  then have "cast result ∈ set to'"
    using assms get_elements_by_class_name_result_in_tree_order by blast

  have "h ⊢ get_root_node (cast result) →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →_r root_ptr"
    using ‹cast result ∈ set to'› ‹h ⊢ to_tree_order ptr' →_r to'›
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_ptr get_dom_component_root_node_same
      get_dom_component_subset get_dom_component_to_tree_order
    by blast
  then have "h ⊢ get_dom_component ptr' →_r c"
    using ‹h ⊢ to_tree_order root_ptr →_r c›
    using get_dom_component_def by auto
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume "ptr' ∈ set c"
  moreover obtain to' where to': "h ⊢ to_tree_order ptr' →_r to'"
    by (meson assms(1) assms(2) assms(3) assms(4) calculation get_dom_component_ptr_in_heap
        get_dom_component_subset is_OK_returns_result_E is_OK_returns_result_I to_tree_order_ok)
  ultimately have "set to' ⊆ set c"
    using assms(1) assms(2) assms(3) assms(4) get_dom_component_subset
      get_dom_component_to_tree_order_subset
    by blast
  moreover have "cast result ∈ set to'"
    using assms get_elements_by_class_name_result_in_tree_order to' by blast
  ultimately show "cast result ∈ set c"
    by blast
qed

lemma get_elements_by_class_name_pure [simp]:
  "pure (get_elements_by_class_name ptr name) h"
  by(auto simp add: get_elements_by_class_name_def intro!: bind_pure_I map_filter_M_pure
      split: option.splits)

lemma get_elements_by_class_name_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_elements_by_class_name ptr name →_r results"
  assumes "h ⊢ get_elements_by_class_name ptr name →_h h'"
  shows "is_strongly_dom_component_safe {ptr} (cast ` set results) h h'"
proof -
  have "h = h'"
```

```
    using assms(5)
    by (meson get_elements_by_class_name_pure pure_returns_heap_eq)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_elements_by_class_name_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast ` set results ⊆ set to"
    using assms(4) local.get_elements_by_class_name_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_dom_component ptr →r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_dom_component_impl
      local.get_dom_component_ok by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def
        get_elements_by_class_name_def first_in_tree_order_def elim!: bind_returns_result_E2
        intro!: map_filter_M_pure bind_pure_I split: option.splits list.splits)[1]
    by (metis (no_types, opaque_lifting) Int_iff ‹ptr |∈| object_ptr_kinds h› assms(4)
        get_elements_by_class_name_is_strongly_dom_component_safe_step local.get_dom_component_impl
        local.get_dom_component_ptr select_result_I2)
qed
end

interpretation i_get_dom_component_get_elements_by_class_name?:
  l_get_dom_component_get_elements_by_class_name_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_dom_component_get_elements_by_class_name_Core_DOM_def instances)
declare l_get_dom_component_get_elements_by_class_name_Core_DOM_axioms [instances]
```

get_elements_by_tag_name

```
locale l_get_dom_component_get_elements_by_tag_name_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_first_in_tree_order_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_elements_by_tag_name_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_dom_component ptr →r c"
  assumes "h ⊢ get_elements_by_tag_name ptr' idd →r results"
  assumes "result ∈ set results"
  shows "cast result ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast result ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_dom_component_def is_OK_returns_result_I)
  then have "h ⊢ to_tree_order root_ptr →r c"
    using ‹h ⊢ get_dom_component ptr →r c›
    by (simp add: get_dom_component_def bind_returns_result_E3)

  obtain to' where to': "h ⊢ to_tree_order ptr' →r to'"
    using assms(5)
```

```
    apply(simp add: get_elements_by_tag_name_def first_in_tree_order_def)
    by (meson bind_is_OK_E is_OK_returns_result_I)
  then have "cast result ∈ set to'"
    using assms get_elements_by_tag_name_result_in_tree_order by blast

  have "h ⊢ get_root_node (cast result) →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →r root_ptr"
    using ‹cast result ∈ set to'› ‹h ⊢ to_tree_order ptr' →r to'›
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_ptr
      get_dom_component_root_node_same get_dom_component_subset get_dom_component_to_tree_order
    by blast
  then have "h ⊢ get_dom_component ptr' →r c"
    using ‹h ⊢ to_tree_order root_ptr →r c›
    using get_dom_component_def by auto
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume "ptr' ∈ set c"
  moreover obtain to' where to': "h ⊢ to_tree_order ptr' →r to'"
    by (meson assms(1) assms(2) assms(3) assms(4) calculation get_dom_component_ptr_in_heap
        get_dom_component_subset is_OK_returns_result_E is_OK_returns_result_I to_tree_order_ok)
  ultimately have "set to' ⊆ set c"
    using assms(1) assms(2) assms(3) assms(4) get_dom_component_subset
      get_dom_component_to_tree_order_subset
    by blast
  moreover have "cast result ∈ set to'"
    using assms get_elements_by_tag_name_result_in_tree_order to' by blast
  ultimately show "cast result ∈ set c"
    by blast
qed

lemma get_elements_by_tag_name_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_elements_by_tag_name ptr name →r results"
  assumes "h ⊢ get_elements_by_tag_name ptr name →h h'"
  shows "is_strongly_dom_component_safe {ptr} (cast ` set results) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson get_elements_by_tag_name_pure pure_returns_heap_eq)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_elements_by_tag_name_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast ` set results ⊆ set to"
    using assms(4) local.get_elements_by_tag_name_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_dom_component ptr →r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_dom_component_impl
      local.get_dom_component_ok by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def
        get_elements_by_class_name_def first_in_tree_order_def elim!: bind_returns_result_E2
        intro!: map_filter_M_pure bind_pure_I split: option.splits list.splits)[1]
    by (metis (no_types, opaque_lifting) IntI ‹ptr |∈| object_ptr_kinds h›
        get_elements_by_tag_name_is_strongly_dom_component_safe_step local.get_dom_component_impl
```

```
        local.get_dom_component_ptr select_result_I2)
qed
end

interpretation i_get_dom_component_get_elements_by_tag_name?:
  l_get_dom_component_get_elements_by_tag_name_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_dom_component_get_elements_by_tag_name_Core_DOM_def instances)
declare l_get_dom_component_get_elements_by_tag_name_Core_DOM_axioms [instances]
```

remove_child

```
lemma remove_child_unsafe: "¬(∀ (h
    :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
  ) h' ptr child. heap_is_wellformed h ⟶ type_wf h ⟶ known_ptrs h ⟶
h ⊢ remove_child ptr child →_h h' ⟶ is_weakly_dom_component_safe {ptr, cast child} {} h h')"
proof -
  obtain h document_ptr e1 e2 where h: "Inr ((document_ptr, e1, e2), h) = (Heap (fmempty)
      :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
    ) ⊢ (do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      append_child (cast e1) (cast e2);
      return (document_ptr, e1, e2)
  })"
    by(code_simp, auto simp add: equal_eq)+
  then obtain h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    h': "h ⊢ remove_child (cast e1) (cast e2) →_h h'" and
    "¬(is_weakly_dom_component_safe {cast e1, cast e2} {} h h')"
    apply(code_simp)
    apply(clarify)
    by(code_simp, auto simp add: equal_eq)+
  then show ?thesis
    by auto
qed
```

adopt_node

```
lemma adopt_node_unsafe: "¬(∀ (h
    :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
  ) h' document_ptr child. heap_is_wellformed h ⟶ type_wf h ⟶ known_ptrs h ⟶ h ⊢ adopt_node document_ptr
child →_h h' ⟶ is_weakly_dom_component_safe {cast document_ptr, cast child} {} h h')"
proof -
  obtain h document_ptr document_ptr2 e1 where h: "Inr ((document_ptr, document_ptr2, e1), h) = (Heap (fmempty)
      :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
```

```
    ) ⊢ (do {
      document_ptr ← create_document;
      document_ptr2 ← create_document;
      e1 ← create_element document_ptr ''div'';
      adopt_node document_ptr2 (cast e1);
      return (document_ptr, document_ptr2, e1)
  })"
    by(code_simp, auto simp add: equal_eq)+
  then obtain h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    h': "h ⊢ adopt_node document_ptr (cast e1) →_h h'" and
    "¬(is_weakly_dom_component_safe {cast document_ptr, cast e1} {} h h')"
    apply(code_simp)
    apply(clarify)
    by(code_simp, auto simp add: equal_eq)+
  then show ?thesis
    by auto
qed
```

create_element

```
lemma create_element_not_strongly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and document_ptr and new_element_ptr and tag where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ create_element document_ptr tag →_r new_element_ptr →_h h'" and
    "¬ is_strongly_dom_component_safe {cast document_ptr} {cast new_element_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "create_document"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?document_ptr = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and document_ptr="?document_ptr"])
    by code_simp+
qed


locale l_get_dom_component_create_element_Core_DOM =
  l_get_dom_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name +
  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 create_element known_ptr +
  l_get_disconnected_nodes_Core_DOM type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_disconnected_nodes_Core_DOM type_wf set_disconnected_nodes set_disconnected_nodes_locs +
  l_set_tag_name_Core_DOM type_wf set_tag_name set_tag_name_locs +
  l_new_element_get_disconnected_nodes get_disconnected_nodes get_disconnected_nodes_locs +
  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_create_element_wf_Core_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
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    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_dom_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and is_strongly_dom_component_safe :: "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_)
heap ⇒ bool"
    and is_weakly_dom_component_safe :: "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap
⇒ bool"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option)
prog"
    and get_elements_by_class_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    and get_elements_by_tag_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    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 set_tag_name :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
    and set_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and create_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
begin

lemma create_element_is_weakly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_element document_ptr tag →_h h'"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast document_ptr)|_r"
  assumes "ptr ≠ cast |h ⊢ create_element document_ptr tag|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain new_element_ptr h2 h3 disc_nodes 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 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_element_ptr # disc_nodes) →_h h'"
    using assms(4)
    by(auto simp add: create_element_def
        elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated])
  have "h ⊢ create_element document_ptr tag →_r new_element_ptr"
    using new_element_ptr h2 h3 disc_nodes h'
    apply(auto simp add: create_element_def intro!: bind_returns_result_I
        bind_pure_returns_result_I[OF get_disconnected_nodes_pure])[1]
     apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  have "preserved (get_M ptr getter) h h2"
    using h2 new_element_ptr
    apply(rule new_element_get_M_Object)
    using new_element_ptr assms(6) ‹h ⊢ create_element document_ptr tag →_r new_element_ptr›
```

```
    by simp

  have "preserved (get_M ptr getter) h2 h3"
    using set_tag_name_writes h3
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_tag_name_locs_impl a_set_tag_name_locs_def all_args_def)[1]
    by (metis (no_types, lifting) ‹h ⊢ create_element document_ptr tag →r new_element_ptr› assms(6)
        get_M_Element_preserved8 select_result_I2)

  have "document_ptr |∈| document_ptr_kinds h"
    using create_element_document_in_heap
    using assms(4)
    by blast
  then
  have "ptr ≠ (cast document_ptr)"
    using assms(5) assms(1) assms(2) assms(3) local.get_dom_component_ok local.get_dom_component_ptr
    by auto
  have "preserved (get_M ptr getter) h3 h'"
    using set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_disconnected_nodes_locs_def all_args_def)[1]
    by (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)

  show ?thesis
    using ‹preserved (get_M ptr getter) h h2› ‹preserved (get_M ptr getter) h2 h3›
      ‹preserved (get_M ptr getter) h3 h'›
    by(auto simp add: preserved_def)
qed


lemma create_element_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_element document_ptr tag →r result"
  assumes "h ⊢ create_element document_ptr tag →h h'"
  shows "is_weakly_dom_component_safe {cast document_ptr} {cast result} h 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(5)
    by(auto simp add: create_element_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF 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 local.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)

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_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
have "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
    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 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 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 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 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 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 "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 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 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(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 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 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 ‹cast_element_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 "acyclic (parent_child_rel h)"
  using ‹heap_is_wellformed h›
  by (simp add: heap_is_wellformed_def acyclic_heap_def)
also have "parent_child_rel h = parent_child_rel h2"
proof(auto simp add: 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: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
also have "... = parent_child_rel h'"
  by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
finally have "a_acyclic_heap h'"
  by (simp add: acyclic_heap_def)

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)[1]
   apply (metis ‹known_ptrs h2› ‹parent_child_rel h = parent_child_rel h2› ‹type_wf h2› assms
      funion_iff local.get_child_nodes_ok local.known_ptrs_known_ptr local.parent_child_rel_child_in_heap
      local.parent_child_rel_child_nodes2 node_ptr_kinds_commutes node_ptr_kinds_eq_h
      returns_result_select_result)
  by (metis assms disconnected_nodes_eq2_h document_ptr_kinds_eq_h funion_iff
      local.get_disconnected_nodes_ok local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_eq_h
      returns_result_select_result)
then have "a_all_ptrs_in_heap h3"
```

```
  by (simp add: children_eq2_h2 disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
      local.a_all_ptrs_in_heap_def node_ptr_kinds_eq_h2 object_ptr_kinds_eq_h2)
then have "a_all_ptrs_in_heap h'"
  using assms(1) assms(2) assms(3) assms(5) local.create_element_preserves_wellformedness(1) local.heap_is_wellfo
  by blast

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 a_all_ptrs_in_heap_def assms fset_mp
      fset_of_list_elem 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 "a_distinct_lists h"
  using ‹heap_is_wellformed h›
  by (simp add: heap_is_wellformed_def)
then have "a_distinct_lists h2"

  using ‹h2 ⊢ get_child_nodes (cast new_element_ptr) →_r []›
  apply(auto simp add: 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 empty_iff local.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 ‹local.a_distinct_lists h› ‹type_wf h2› disconnected_nodes_eq_h document_ptr_kinds_eq_h
      local.distinct_lists_no_parent local.get_disconnected_nodes_ok returns_result_select_result)

then have "a_distinct_lists h3"
  by(auto simp add: a_distinct_lists_def disconnected_nodes_eq2_h2 document_ptr_kinds_eq_h2
      children_eq2_h2 object_ptr_kinds_eq_h2)
then have "a_distinct_lists h'"
proof(auto simp add: 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›
        ‹a_distinct_lists h3› ‹type_wf h'› disc_nodes_h3 distinct.simps(2)
        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 (metis (mono_tags, lifting) Int_Collect Int_iff ‹type_wf h'› assms(1) assms(2) assms(3)
            assms(5) bot_set_def document_ptr_kinds_eq_h3 empty_Collect_eq
            l_get_disconnected_nodes_Core_DOM.get_disconnected_nodes_ok
            l_heap_is_wellformed_Core_DOM.heap_is_wellformed_one_disc_parent
            local.create_element_preserves_wellformedness(1)
            local.l_get_disconnected_nodes_Core_DOM_axioms
            local.l_heap_is_wellformed_Core_DOM_axioms returns_result_select_result)
    by (metis (no_types, opaque_lifting) ‹type_wf h'› assms(1) assms(2) assms(3) assms(5)
        disjoint_iff document_ptr_kinds_eq_h3 local.create_element_preserves_wellformedness(1)
        local.get_disconnected_nodes_ok local.heap_is_wellformed_one_disc_parent
        returns_result_select_result)
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›
        ‹a_distinct_lists h3› children_eq2_h3 disc_nodes_h3 distinct_lists_no_parent h'
        select_result_I2 set_ConsD set_disconnected_nodes_get_disconnected_nodes)
    by (metis "3" "4" "5" "6" ‹a_distinct_lists h3› ‹type_wf h3› children_eq2_h3
        distinct_lists_no_parent get_disconnected_nodes_ok returns_result_select_result)
qed

have "a_owner_document_valid h"
  using ‹heap_is_wellformed h› by (simp add: heap_is_wellformed_def)
then have "a_owner_document_valid h'"
  using disc_nodes_h3 ‹document_ptr |∈| document_ptr_kinds h›
  apply(auto simp add: 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 "parent_child_rel h = parent_child_rel h'"
proof -
  have "parent_child_rel h = parent_child_rel h2"
  proof(auto simp add: 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_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 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 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: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
  also have "... = parent_child_rel h'"
    by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
  finally show ?thesis
    by simp
qed
have root: "h ⊢ get_root_node (cast document_ptr) →_r cast document_ptr"
  by (simp add: ‹document_ptr |∈| document_ptr_kinds h› local.get_root_node_not_node_same)
then
have root': "h' ⊢ get_root_node (cast document_ptr) →_r cast document_ptr"
  by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
      local.get_root_node_not_node_same object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3)

have "heap_is_wellformed h'" and "known_ptrs h'"
  using create_element_preserves_wellformedness assms
  by blast+

have "cast result |∉| object_ptr_kinds h"
  using ‹cast new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r›
  by (metis (full_types) ObjectMonad.ptr_kinds_ptr_kinds_M
      ‹h ⊢ create_element document_ptr tag →_r new_element_ptr› assms(4) returns_result_eq)

obtain to where to: "h ⊢ to_tree_order (cast document_ptr) →_r to"
  by (meson ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
      document_ptr_kinds_commutes is_OK_returns_result_E local.to_tree_order_ok)
then
```

```
  have "h ⊢ a_get_dom_component (cast document_ptr) →r to"
    using root
    by(auto simp add: a_get_dom_component_def)
  moreover
  obtain to' where to': "h' ⊢ to_tree_order (cast document_ptr) →r to'"
    by (meson ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹type_wf h'› is_OK_returns_result_E
        local.get_root_node_root_in_heap local.to_tree_order_ok root')
  then
  have "h' ⊢ a_get_dom_component (cast document_ptr) →r to'"
    using root'
    by(auto simp add: a_get_dom_component_def)
  moreover
  have "⋀child. child ∈ set to ⟷ child ∈ set to'"
    by (metis ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹parent_child_rel h = parent_child_rel h'›
        ‹type_wf h'› assms(1) assms(2) assms(3) local.to_tree_order_parent_child_rel to to')
  ultimately
  have "set |h ⊢ local.a_get_dom_component (cast document_ptr)|r =
set |h' ⊢ local.a_get_dom_component (cast document_ptr)|r"
    by(auto simp add: a_get_dom_component_def)

  show ?thesis
    apply(auto simp add: is_weakly_dom_component_safe_def Let_def)[1]
    using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →r []› assms(2) assms(3)
      children_eq_h local.get_child_nodes_ok local.get_child_nodes_ptr_in_heap local.known_ptrs_known_ptr
      object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 returns_result_select_result
       apply (metis is_OK_returns_result_I)
      apply (metis ‹h ⊢ create_element document_ptr tag →r new_element_ptr› assms(4)
        element_ptr_kinds_commutes h2 new_element_ptr new_element_ptr_in_heap node_ptr_kinds_eq_h2
        node_ptr_kinds_eq_h3 returns_result_eq returns_result_heap_def)
    using ‹cast_element_ptr2object_ptr result |∉| object_ptr_kinds h› element_ptr_kinds_commutes
      node_ptr_kinds_commutes apply blast
    using assms(1) assms(2) assms(3) ‹h ⊢ create_element document_ptr tag →h h'›
    apply(rule create_element_is_weakly_dom_component_safe_step)
     apply (simp add: local.get_dom_component_impl)
    using ‹cast_element_ptr2object_ptr new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|r›
      ‹h ⊢ create_element document_ptr tag →r new_element_ptr›
    by auto

qed
end

interpretation i_get_dom_component_create_element?: l_get_dom_component_create_element_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs set_tag_name
  set_tag_name_locs create_element
  by(auto simp add: l_get_dom_component_create_element_Core_DOM_def instances)
declare l_get_dom_component_create_element_Core_DOM_axioms [instances]
```

create_character_data

```
lemma create_character_data_not_strongly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder},
'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and document_ptr and create_character_datanew_character_data_ptr and tag where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ create_character_data document_ptr tag →r create_character_datanew_character_data_ptr →h h'"
```

```
and
    "¬ is_strongly_dom_component_safe {cast document_ptr} {cast create_character_datanew_character_data_ptr}
h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "create_document"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?document_ptr = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and document_ptr="?document_ptr"])
    by code_simp+
qed

locale l_get_dom_component_create_character_data_Core_DOM =
  l_get_dom_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name +
  l_create_character_data_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs set_val set_val_locs type_wf create_character_data known_ptr +
  l_get_disconnected_nodes_Core_DOM type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_disconnected_nodes_Core_DOM type_wf set_disconnected_nodes set_disconnected_nodes_locs +
  l_set_val_Core_DOM type_wf set_val set_val_locs +
  l_create_character_data_wf_Core_DOM known_ptr type_wf get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs heap_is_wellformed parent_child_rel set_val
  set_val_locs set_disconnected_nodes set_disconnected_nodes_locs
  create_character_data known_ptrs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    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_dom_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and is_strongly_dom_component_safe :: "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_)
heap ⇒ bool"
    and is_weakly_dom_component_safe :: "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap
⇒ bool"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option)
prog"
    and get_elements_by_class_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    and get_elements_by_tag_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    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 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 create_character_data :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) character_data_ptr)
prog"
begin

lemma create_character_data_is_weakly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_character_data document_ptr text →_h h'"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast document_ptr)|_r"
  assumes "ptr ≠ cast |h ⊢ create_character_data document_ptr text|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain new_character_data_ptr h2 h3 disc_nodes 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 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_character_data_ptr # disc_nodes) →_h h'"
    using assms(4)
    by(auto simp add: create_character_data_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated])

  have "h ⊢ create_character_data document_ptr text →_r new_character_data_ptr"
    using new_character_data_ptr h2 h3 disc_nodes h'
    apply(auto simp add: create_character_data_def intro!: bind_returns_result_I
        bind_pure_returns_result_I[OF get_disconnected_nodes_pure])[1]
     apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  have "preserved (get_M ptr getter) h h2"
    using h2 new_character_data_ptr
    apply(rule new_character_data_get_M_Object)
    using new_character_data_ptr assms(6)
      ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr›
    by simp

  have "preserved (get_M ptr getter) h2 h3"
    using set_val_writes h3
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_val_locs_impl a_set_val_locs_def all_args_def)[1]
    by (metis (mono_tags) CharacterData_simp11
        ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr› assms(4) assms(6)
        is_OK_returns_heap_I is_OK_returns_result_E returns_result_eq select_result_I2)

  have "document_ptr |∈| document_ptr_kinds h"
    using create_character_data_document_in_heap
    using assms(4)
    by blast
  then
  have "ptr ≠ (cast document_ptr)"
    using assms(5) assms(1) assms(2) assms(3) local.get_dom_component_ok local.get_dom_component_ptr
    by auto
  have "preserved (get_M ptr getter) h3 h'"
    using set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_disconnected_nodes_locs_def all_args_def)[1]
    by (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)

  show ?thesis
    using ‹preserved (get_M ptr getter) h h2› ‹preserved (get_M ptr getter) h2 h3›
      ‹preserved (get_M ptr getter) h3 h'›
    by(auto simp add: preserved_def)
qed
```

```
lemma create_character_data_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_character_data document_ptr text →r result"
  assumes "h ⊢ create_character_data document_ptr text →h h'"
  shows "is_weakly_dom_component_safe {cast document_ptr} {cast result} h 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(5)
    by(auto simp add: create_character_data_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated] )
  then
  have "h ⊢ create_character_data document_ptr text →r new_character_data_ptr"
    apply(auto simp add: 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.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 set_val_writes h3])
    using 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 "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      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 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 set_val_writes h3])
    using 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 "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      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 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 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 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_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 get_disconnected_nodes_reads 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 "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 set_val_writes h3]
    using  set_val_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 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(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 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 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 "parent_child_rel h = parent_child_rel h'"
  proof -
    have "parent_child_rel h = parent_child_rel h2"
    proof(auto simp add: 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: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
    also have "... = parent_child_rel h'"
      by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
    finally show ?thesis
      by simp
  qed
  have root: "h ⊢ get_root_node (cast document_ptr) →r cast document_ptr"
    by (simp add: ‹document_ptr |∈| document_ptr_kinds h› local.get_root_node_not_node_same)
  then
  have root': "h' ⊢ get_root_node (cast document_ptr) →r cast document_ptr"
    by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
        local.get_root_node_not_node_same object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3)


  have "heap_is_wellformed h'" and "known_ptrs h'"
    using create_character_data_preserves_wellformedness assms
    by blast+

  have "cast result |∉| object_ptr_kinds h"
    using ‹cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|r›
    by (metis (full_types) ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹h ⊢ create_character_data document_ptr text →r new_character_data_ptr› assms(4) returns_result_eq)

  obtain to where to: "h ⊢ to_tree_order (cast document_ptr) →r to"
    by (meson ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
        document_ptr_kinds_commutes is_OK_returns_result_E local.to_tree_order_ok)
  then
  have "h ⊢ a_get_dom_component (cast document_ptr) →r to"
    using root
    by(auto simp add: a_get_dom_component_def)
  moreover
  obtain to' where to': "h' ⊢ to_tree_order (cast document_ptr) →r to'"
    by (meson ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹type_wf h'› is_OK_returns_result_E
        local.get_root_node_root_in_heap local.to_tree_order_ok root')
  then
  have "h' ⊢ a_get_dom_component (cast document_ptr) →r to'"
    using root'
    by(auto simp add: a_get_dom_component_def)
  moreover
  have "⋀child. child ∈ set to ⟷ child ∈ set to'"
    by (metis ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹parent_child_rel h = parent_child_rel h'›
        ‹type_wf h'› assms(1) assms(2) assms(3) local.to_tree_order_parent_child_rel to to')
  ultimately
  have "set |h ⊢ local.a_get_dom_component (cast document_ptr)|r =
set |h' ⊢ local.a_get_dom_component (cast document_ptr)|r"
    by(auto simp add: a_get_dom_component_def)

  show ?thesis
    apply(auto simp add: is_weakly_dom_component_safe_def Let_def)[1]
    using  assms(2) assms(3) children_eq_h local.get_child_nodes_ok
      local.get_child_nodes_ptr_in_heap local.known_ptrs_known_ptr object_ptr_kinds_eq_h2
      object_ptr_kinds_eq_h3 returns_result_select_result
       apply (metis ‹h2 ⊢ get_child_nodes (cast_character_data_ptr2object_ptr new_character_data_ptr) →r []›
        is_OK_returns_result_I)

      apply (metis ‹h ⊢ create_character_data document_ptr text →r new_character_data_ptr› assms(4)
        character_data_ptr_kinds_commutes h2 new_character_data_ptr new_character_data_ptr_in_heap
        node_ptr_kinds_eq_h2 node_ptr_kinds_eq_h3 returns_result_eq)
    using ‹h ⊢ create_character_data document_ptr text →r new_character_data_ptr›
      ‹new_character_data_ptr ∉ set |h ⊢ character_data_ptr_kinds_M|r› assms(4) returns_result_eq
     apply fastforce
    using  assms(2) assms(3) children_eq_h local.get_child_nodes_ok local.get_child_nodes_ptr_in_heap
```

```
    local.known_ptrs_known_ptr object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 returns_result_select_result
  apply (smt (verit, best) ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹cast_character_data_ptr2object_ptr new_character_data_ptr ∉ set |h
          ⊢ object_ptr_kinds_M|_r› ‹h ⊢ create_character_data document_ptr text →_r
          new_character_data_ptr› assms(1) assms(5)
        create_character_data_is_weakly_dom_component_safe local.a_get_dom_component_def
        local.get_dom_component_def select_result_I2)
  done
qed

end

interpretation i_get_dom_component_create_character_data?: l_get_dom_component_create_character_data_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name set_val set_val_locs
  get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs
  create_character_data
  by(auto simp add: l_get_dom_component_create_character_data_Core_DOM_def instances)
declare l_get_dom_component_create_character_data_Core_DOM_axioms [instances]
```

create_document

```
lemma create_document_unsafe: "¬(∀ (h
    :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
  ) h' new_document_ptr. heap_is_wellformed h ⟶ type_wf h ⟶ known_ptrs h ⟶
h ⊢ create_document →_r new_document_ptr ⟶ h ⊢ create_document →_h h' ⟶
is_strongly_dom_component_safe {} {cast new_document_ptr} h h')"
proof -
  note Set.filter_eq [simp del] Set.remove_eq [simp del] Set.is_empty_iff [simp del]
  obtain h document_ptr where h: "Inr (document_ptr, h) = (Heap (fmempty)
      :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
    ) ⊢ (do {
      document_ptr ← create_document;
      return (document_ptr)
  })"
    by(code_simp, auto simp add: equal_eq)+
  then obtain h' new_document_ptr where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    h': "h ⊢ create_document →_r new_document_ptr" and
    h': "h ⊢ create_document →_h h'" and
    "¬(is_strongly_dom_component_safe {} {cast new_document_ptr} h h')"
    by (code_simp, auto simp add: equal_eq)+
  then show ?thesis
    by blast
qed

locale l_get_dom_component_create_document_Core_DOM =
  l_get_dom_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name +
  l_create_document_Core_DOM create_document
  for known_ptr :: "('_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "('_) heap ⇒ bool"
```

```
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    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_dom_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and is_strongly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and is_weakly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and get_elements_by_class_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and get_elements_by_tag_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and create_document :: "((_) heap, exception, (_) document_ptr) prog"
begin

lemma create_document_is_weakly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_document →_h h'"
  assumes "ptr ≠ cast |h ⊢ create_document|_r"
  shows "preserved (get_M_Object ptr getter) h h'"
  using assms
  apply(auto simp add: create_document_def)[1]
  by (metis assms(4) assms(5) is_OK_returns_heap_I local.create_document_def new_document_get_M_Object
      select_result_I)

lemma create_document_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_document →_r result"
  assumes "h ⊢ create_document →_h h'"
  shows "is_weakly_dom_component_safe {} {cast result} h h'"
proof -
  have "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast result|}"
    using assms(4) assms(5) local.create_document_def new_document_new_ptr by auto
  moreover have "result |∉| document_ptr_kinds h"
    using assms(4) assms(5) local.create_document_def new_document_ptr_not_in_heap by auto
  ultimately show ?thesis
    using assms
    apply(auto simp add: is_weakly_dom_component_safe_def Let_def local.create_document_def
        new_document_ptr_not_in_heap)[1]
    by (metis ‹result |∉| document_ptr_kinds h› document_ptr_kinds_commutes new_document_get_M_Object)
qed
end

interpretation i_get_dom_component_create_document?: l_get_dom_component_create_document_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name create_document
  by(auto simp add: l_get_dom_component_create_document_Core_DOM_def instances)
declare l_get_dom_component_create_document_Core_DOM_axioms [instances]
```

insert_before

```
lemma insert_before_unsafe: "¬(∀(h
    :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
  ) h' ptr child. heap_is_wellformed h ⟶ type_wf h ⟶ known_ptrs h ⟶
h ⊢ insert_before ptr child None →_h h' ⟶ is_weakly_dom_component_safe {ptr, cast child} {} h h')"
proof -
  note Set.filter_eq [simp del] Set.remove_eq [simp del] Set.is_empty_iff [simp del]
  obtain h document_ptr e1 e2 where h: "Inr ((document_ptr, e1, e2), h) = (Heap (fmempty)
      :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
    ) ⊢ (do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      return (document_ptr, e1, e2)
  })"
    by(code_simp, auto simp add: equal_eq )+
  then obtain h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    h': "h ⊢ insert_before (cast e1) (cast e2) None →_h h'" and
    "¬(is_weakly_dom_component_safe {cast e1, cast e2} {} h h')"
    by(code_simp, auto simp add: equal_eq)+
  then show ?thesis
    by auto
qed

lemma insert_before_unsafe2: "¬(∀(h
    :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
  ) h' ptr child ref. heap_is_wellformed h ⟶ type_wf h ⟶ known_ptrs h ⟶
h ⊢ insert_before ptr child (Some ref) →_h h' ⟶
is_weakly_dom_component_safe {ptr, cast child, cast ref} {} h h')"
proof -
  obtain h document_ptr e1 e2 e3 where h: "Inr ((document_ptr, e1, e2, e3), h) = (Heap (fmempty)
      :: ('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap
    ) ⊢ (do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      e3 ← create_element document_ptr ''div'';
      append_child (cast e1) (cast e2);
      return (document_ptr, e1, e2, e3)
  })"
    by(code_simp, auto simp add: equal_eq)+
  then obtain h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    h': "h ⊢ insert_before (cast e1) (cast e3) (Some (cast e2)) →_h h'" and
    "¬(is_weakly_dom_component_safe {cast e1, cast e3, cast e2} {} h h')"
    apply(code_simp)
    apply(clarify)
    by(code_simp, auto simp add: equal_eq)+
  then show ?thesis
```

```
    by fast
qed


lemma append_child_unsafe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and ptr and child where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ append_child ptr child →_h h'" and
    "¬ is_weakly_dom_component_safe {ptr, cast child} {} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      return (e1, e2)
  }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "fst |?h0 ⊢ ?P|_r"
  let ?e2 = "snd |?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e1" and child="cast_element_ptr2node_ptr
?e2"])
    by code_simp+
qed
```

get_owner_document

```
lemma get_owner_document_unsafe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and ptr and owner_document where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_owner_document ptr →_r owner_document →_h h'" and
    "¬is_weakly_dom_component_safe {ptr} {cast owner_document} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      return (document_ptr, e1)
  }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?document_ptr = "fst |?h0 ⊢ ?P|_r"
  let ?e1 = "snd |?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e1" and owner_document="?document_ptr"])
```

```
    by code_simp+
qed

end
```

2.2 Core SC DOM Components II (Core_DOM_SC_DOM_Components)

```
theory Core_DOM_SC_DOM_Components
  imports
    Core_DOM_DOM_Components
begin
declare [[smt_timeout=2400]]
```

2.3 Scope Components (Core_DOM_SC_DOM_Components)

2.3.1 Definition

```
locale l_get_scdom_component_Core_DOM_defs =
  l_get_disconnected_nodes_defs get_disconnected_nodes get_disconnected_nodes_locs +
  l_get_owner_document_defs get_owner_document +
  l_to_tree_order_defs to_tree_order
  for get_owner_document :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
begin
definition a_get_scdom_component :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  where
    "a_get_scdom_component ptr = do {
      document ← get_owner_document ptr;
      disc_nodes ← get_disconnected_nodes document;
      tree_order ← to_tree_order (cast document);
      disconnected_tree_orders ← map_M (to_tree_order ∘ cast) disc_nodes;
      return (tree_order @ (concat disconnected_tree_orders))
    }"

definition a_is_strongly_scdom_component_safe ::
  "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  where
    "a_is_strongly_scdom_component_safe S_arg S_result h h' = (
      let removed_pointers = fset (object_ptr_kinds h) - fset (object_ptr_kinds h') in
      let added_pointers = fset (object_ptr_kinds h') - fset (object_ptr_kinds h) in
      let arg_components =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_scdom_component ptr|_r) ∩
        fset (object_ptr_kinds h).  set |h ⊢ a_get_scdom_component ptr|_r) in
      let arg_components' =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_scdom_component ptr|_r) ∩
        fset (object_ptr_kinds h').  set |h' ⊢ a_get_scdom_component ptr|_r) in
      removed_pointers ⊆ arg_components ∧
      added_pointers ⊆ arg_components' ∧
      S_result ⊆ arg_components' ∧
      (∀ outside_ptr ∈ fset (object_ptr_kinds h) ∩ fset (object_ptr_kinds h') -
        (⋃ptr ∈ S_arg. set |h  ⊢ a_get_scdom_component ptr|_r). preserved (get_M outside_ptr id) h h'))"

definition a_is_weakly_scdom_component_safe ::
  "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  where
    "a_is_weakly_scdom_component_safe S_arg S_result h h' = (
      let removed_pointers = fset (object_ptr_kinds h) - fset (object_ptr_kinds h') in
      let added_pointers = fset (object_ptr_kinds h') - fset (object_ptr_kinds h) in
```

```
      let arg_components =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_scdom_component ptr|_r) ∩
        fset (object_ptr_kinds h).  set |h ⊢ a_get_scdom_component ptr|_r) in
      let arg_components' =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_scdom_component ptr|_r) ∩
        fset (object_ptr_kinds h').  set |h' ⊢ a_get_scdom_component ptr|_r) in
      removed_pointers ⊆ arg_components ∧
      S_result ⊆ arg_components' ∪ added_pointers  ∧
      (∀ outside_ptr ∈ fset (object_ptr_kinds h) ∩ fset (object_ptr_kinds h') -
        (⋃ptr ∈ S_arg. set |h  ⊢ a_get_scdom_component ptr|_r). preserved (get_M outside_ptr id) h h'))"
end

global_interpretation l_get_scdom_component_Core_DOM_defs get_owner_document get_disconnected_nodes
  get_disconnected_nodes_locs to_tree_order
  defines get_scdom_component = "l_get_scdom_component_Core_DOM_defs.a_get_scdom_component
get_owner_document get_disconnected_nodes to_tree_order"
    and is_strongly_scdom_component_safe = a_is_strongly_scdom_component_safe
    and is_weakly_scdom_component_safe = a_is_weakly_scdom_component_safe
  .

locale l_get_scdom_component_defs =
  fixes get_scdom_component :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  fixes is_strongly_scdom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  fixes is_weakly_scdom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"


locale l_get_scdom_component_Core_DOM =
  l_get_scdom_component_defs +
  l_get_scdom_component_Core_DOM_defs +
  assumes get_scdom_component_impl: "get_scdom_component = a_get_scdom_component"
  assumes is_strongly_scdom_component_safe_impl:
    "is_strongly_scdom_component_safe = a_is_strongly_scdom_component_safe"
  assumes is_weakly_scdom_component_safe_impl:
    "is_weakly_scdom_component_safe = a_is_weakly_scdom_component_safe"
begin
lemmas get_scdom_component_def = a_get_scdom_component_def[folded get_scdom_component_impl]
lemmas is_strongly_scdom_component_safe_def =
  a_is_strongly_scdom_component_safe_def[folded is_strongly_scdom_component_safe_impl]
lemmas is_weakly_scdom_component_safe_def =
  a_is_weakly_scdom_component_safe_def[folded is_weakly_scdom_component_safe_impl]
end

interpretation i_get_scdom_component?: l_get_scdom_component_Core_DOM
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
  get_owner_document get_disconnected_nodes get_disconnected_nodes_locs to_tree_order
  by(auto simp add: l_get_scdom_component_Core_DOM_def get_scdom_component_def
      is_strongly_scdom_component_safe_def is_weakly_scdom_component_safe_def instances)
declare l_get_scdom_component_Core_DOM_axioms [instances]


locale l_get_dom_component_get_scdom_component_Core_DOM =
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_Core_DOM +
  l_heap_is_wellformed +
  l_get_owner_document +
  l_get_owner_document_wf +
  l_get_disconnected_nodes +
  l_to_tree_order +
  l_known_ptr +
  l_known_ptrs +
  l_get_owner_document_wf_get_root_node_wf +
```

```
  assumes known_ptr_impl: "known_ptr = DocumentClass.known_ptr"
begin

lemma known_ptr_node_or_document: "known_ptr ptr ⟹ is_node_ptr_kind ptr ∨ is_document_ptr_kind 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)

lemma get_scdom_component_ptr_in_heap2:
  assumes "h ⊢ ok (get_scdom_component ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms get_root_node_ptr_in_heap
  apply(auto simp add: get_scdom_component_def elim!: bind_is_OK_E3 intro!: map_M_pure_I)[1]
  by (simp add: is_OK_returns_result_I local.get_owner_document_ptr_in_heap)

lemma get_scdom_component_subset_get_dom_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "h ⊢ get_dom_component ptr →r c"
  shows "set c ⊆ set sc"
proof -
  obtain document disc_nodes tree_order disconnected_tree_orders where
    document: "h ⊢ get_owner_document ptr →r document"
    and disc_nodes: "h ⊢ get_disconnected_nodes document →r disc_nodes"
    and tree_order: "h ⊢ to_tree_order (cast document) →r tree_order"
    and disconnected_tree_orders: "h ⊢ map_M (to_tree_order ∘ cast) disc_nodes →r disconnected_tree_orders"
    and sc: "sc = tree_order @ (concat disconnected_tree_orders)"
    using assms(4)
    by(auto simp add: get_scdom_component_def elim!: bind_returns_result_E
        elim!: bind_returns_result_E2[rotated, OF get_owner_document_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF to_tree_order_pure, rotated]
        )

  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    and c: "h ⊢ to_tree_order root_ptr →r c"
    using assms(5)
    by(auto simp add: get_dom_component_def elim!: bind_returns_result_E2[rotated, OF get_root_node_pure,
rotated])

  show ?thesis
  proof (cases "is_document_ptr_kind root_ptr")
    case True
    then have "cast document = root_ptr"
      using get_root_node_document assms(1) assms(2) assms(3) root_ptr document
      by (metis document_ptr_casts_commute3 returns_result_eq)
    then have "c = tree_order"
      using tree_order c
      by auto
    then show ?thesis
      by(simp add: sc)
  next
    case False
    moreover have "root_ptr |∈| object_ptr_kinds h"
      using assms(1) assms(2) assms(3) local.get_root_node_root_in_heap root_ptr by blast
    ultimately have "is_node_ptr_kind root_ptr"
      using assms(3) known_ptrs_known_ptr known_ptr_node_or_document
      by auto
    then obtain root_node_ptr where root_node_ptr: "root_ptr = cast_{node_ptr2object_ptr} root_node_ptr"
      by (metis node_ptr_casts_commute3)
    then have "h ⊢ get_owner_document root_ptr →r document"
      using get_root_node_same_owner_document
      using assms(1) assms(2) assms(3) document root_ptr by blast
```

```
    then have "root_node_ptr ∈ set disc_nodes"
      using assms(1) assms(2) assms(3) disc_nodes in_disconnected_nodes_no_parent root_node_ptr
      using local.get_root_node_same_no_parent root_ptr by blast
    then have "c ∈ set disconnected_tree_orders"
      using c root_node_ptr
      using map_M_pure_E[OF disconnected_tree_orders]
      by (metis (mono_tags, lifting) comp_apply local.to_tree_order_pure select_result_I2)
    then show ?thesis
      by(auto simp add: sc)
  qed
qed

lemma get_scdom_component_ptrs_same_owner_document:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  assumes "h ⊢ get_owner_document ptr →r owner_document"
  shows "h ⊢ get_owner_document ptr' →r owner_document"
proof -
  obtain document disc_nodes tree_order disconnected_tree_orders where
    document: "h ⊢ get_owner_document ptr →r document"
    and disc_nodes: "h ⊢ get_disconnected_nodes document →r disc_nodes"
    and tree_order: "h ⊢ to_tree_order (cast document) →r tree_order"
    and disconnected_tree_orders: "h ⊢ map_M (to_tree_order ∘ cast) disc_nodes →r disconnected_tree_orders"
    and sc: "sc = tree_order @ (concat disconnected_tree_orders)"
    using assms(4)
    by(auto simp add: get_scdom_component_def elim!: bind_returns_result_E
        elim!: bind_returns_result_E2[rotated, OF get_owner_document_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF to_tree_order_pure, rotated]
        )
  show ?thesis
  proof (cases "ptr' ∈ set tree_order")
    case True
    have "owner_document = document"
      using assms(6) document by fastforce
    then show ?thesis
      by (metis (no_types) True assms(1) assms(2) assms(3) cast_document_ptr2object_ptr_inject document
          document_ptr_casts_commute3 document_ptr_document_ptr_cast document_ptr_kinds_commutes
          local.get_owner_document_owner_document_in_heap local.get_root_node_document
          local.get_root_node_not_node_same local.to_tree_order_same_root node_ptr_no_document_ptr_cast
tree_order)
  next
    case False
    then obtain disconnected_tree_order where disconnected_tree_order:
      "ptr' ∈ set disconnected_tree_order" and "disconnected_tree_order ∈ set disconnected_tree_orders"
      using sc ‹ptr' ∈ set sc›
      by auto
    obtain root_ptr' where
      root_ptr': "root_ptr' ∈ set disc_nodes" and
      "h ⊢ to_tree_order (cast root_ptr') →r disconnected_tree_order"
      using map_M_pure_E2[OF disconnected_tree_orders ‹disconnected_tree_order ∈ set disconnected_tree_orders›]
      by (metis comp_apply local.to_tree_order_pure)
    have "¬(∃parent ∈ fset (object_ptr_kinds h). root_ptr' ∈ set |h ⊢ get_child_nodes parent|r)"
      using disc_nodes
      by (meson assms(1) assms(2) assms(3) disjoint_iff_not_equal local.get_child_nodes_ok
          local.heap_is_wellformed_children_disc_nodes_different local.known_ptrs_known_ptr
          returns_result_select_result root_ptr')
    then
    have "h ⊢ get_parent root_ptr' →r None"
      using disc_nodes
      by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3) local.get_parent_child_dual
          local.get_parent_ok local.get_parent_parent_in_heap local.heap_is_wellformed_disc_nodes_in_heap
```

```
        returns_result_select_result root_ptr' select_result_I2 split_option_ex)
    then have "h ⊢ get_root_node ptr' →r cast root_ptr'"
      using ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr root_ptr') →r disconnected_tree_order› assms(1)
        assms(2) assms(3) disconnected_tree_order local.get_root_node_no_parent
        local.to_tree_order_get_root_node local.to_tree_order_ptr_in_result
      by blast
    then have "h ⊢ get_owner_document (cast root_ptr') →r document"
      using assms(1) assms(2) assms(3) disc_nodes local.get_owner_document_disconnected_nodes root_ptr'
      by blast

    then have "h ⊢ get_owner_document ptr' →r document"
      using ‹h ⊢ get_root_node ptr' →r cast_node_ptr2object_ptr root_ptr'› assms(1) assms(2) assms(3)
        local.get_root_node_same_owner_document
      by blast
    then show ?thesis
      using assms(6) document returns_result_eq by force
  qed
qed

lemma get_scdom_component_ptrs_same_scope_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  shows "h ⊢ get_scdom_component ptr' →r sc"
proof -
  obtain document disc_nodes tree_order disconnected_tree_orders where
    document: "h ⊢ get_owner_document ptr →r document"
    and disc_nodes: "h ⊢ get_disconnected_nodes document →r disc_nodes"
    and tree_order: "h ⊢ to_tree_order (cast document) →r tree_order"
    and disconnected_tree_orders: "h ⊢ map_M (to_tree_order ∘ cast) disc_nodes →r disconnected_tree_orders"
    and sc: "sc = tree_order @ (concat disconnected_tree_orders)"
    using assms(4)
    by(auto simp add: get_scdom_component_def elim!: bind_returns_result_E
        elim!: bind_returns_result_E2[rotated, OF get_owner_document_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF to_tree_order_pure, rotated]
        )
  show ?thesis
  proof (cases "ptr' ∈ set tree_order")
    case True
    then have "h ⊢ get_owner_document ptr' →r document"
      by (metis assms(1) assms(2) assms(3) cast_document_ptr2object_ptr_inject document
          document_ptr_casts_commute3 document_ptr_kinds_commutes known_ptr_node_or_document
          local.get_owner_document_owner_document_in_heap local.get_root_node_document
          local.get_root_node_not_node_same local.known_ptrs_known_ptr local.to_tree_order_get_root_node
          local.to_tree_order_ptr_in_result node_ptr_no_document_ptr_cast tree_order)
    then show ?thesis
      using disc_nodes tree_order disconnected_tree_orders sc
      by(auto simp add: get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  next
    case False
    then obtain disconnected_tree_order where disconnected_tree_order:
      "ptr' ∈ set disconnected_tree_order" and "disconnected_tree_order ∈ set disconnected_tree_orders"
      using sc ‹ptr' ∈ set sc›
      by auto
    obtain root_ptr' where
      root_ptr': "root_ptr' ∈ set disc_nodes" and
      "h ⊢ to_tree_order (cast root_ptr') →r disconnected_tree_order"
      using map_M_pure_E2[OF disconnected_tree_orders ‹disconnected_tree_order ∈ set disconnected_tree_orders›]
      by (metis comp_apply local.to_tree_order_pure)
    have "¬(∃parent ∈ fset (object_ptr_kinds h). root_ptr' ∈ set |h ⊢ get_child_nodes parent|r)"
      using disc_nodes
      by (meson assms(1) assms(2) assms(3) disjoint_iff_not_equal local.get_child_nodes_ok
```

```
        local.heap_is_wellformed_children_disc_nodes_different local.known_ptrs_known_ptr
        returns_result_select_result root_ptr')
    then
    have "h ⊢ get_parent root_ptr' →r None"
      using disc_nodes
      by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3)
        local.get_parent_child_dual local.get_parent_ok local.get_parent_parent_in_heap
        local.heap_is_wellformed_disc_nodes_in_heap returns_result_select_result root_ptr'
        select_result_I2 split_option_ex)
    then have "h ⊢ get_root_node ptr' →r cast root_ptr'"
      using ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr root_ptr') →r disconnected_tree_order› assms(1)
        assms(2) assms(3) disconnected_tree_order local.get_root_node_no_parent
        local.to_tree_order_get_root_node local.to_tree_order_ptr_in_result
      by blast
    then have "h ⊢ get_owner_document (cast root_ptr') →r document"
      using assms(1) assms(2) assms(3) disc_nodes local.get_owner_document_disconnected_nodes root_ptr'
      by blast

    then have "h ⊢ get_owner_document ptr' →r document"
      using ‹h ⊢ get_root_node ptr' →r cast_node_ptr2object_ptr root_ptr'› assms(1) assms(2) assms(3)
        local.get_root_node_same_owner_document
      by blast
    then show ?thesis
      using disc_nodes tree_order disconnected_tree_orders sc
      by(auto simp add: get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  qed
qed

lemma get_scdom_component_ok:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_scdom_component ptr)"
  using assms
  apply(auto simp add: get_scdom_component_def intro!: bind_is_OK_pure_I map_M_pure_I map_M_ok_I)[1]
  using get_owner_document_ok
     apply blast
    apply (simp add: local.get_disconnected_nodes_ok local.get_owner_document_owner_document_in_heap)
   apply (simp add: local.get_owner_document_owner_document_in_heap local.to_tree_order_ok)
  using local.heap_is_wellformed_disc_nodes_in_heap local.to_tree_order_ok node_ptr_kinds_commutes
  by blast

lemma get_scdom_component_ptr_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  shows "ptr' |∈| object_ptr_kinds h"
  apply(insert assms )
  apply(auto simp add: get_scdom_component_def elim!: bind_returns_result_E2 intro!: map_M_pure_I)[1]
  using local.to_tree_order_ptrs_in_heap apply blast
  by (metis (no_types, lifting) assms(4) assms(5) bind_returns_result_E
      get_scdom_component_ptrs_same_scope_component is_OK_returns_result_I get_scdom_component_def
      local.get_owner_document_ptr_in_heap)

lemma get_scdom_component_contains_get_dom_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  obtains c where "h ⊢ get_dom_component ptr' →r c" and "set c ⊆ set sc"
proof -
  have "h ⊢ get_scdom_component ptr' →r sc"
    using assms(1) assms(2) assms(3) assms(4) assms(5) get_scdom_component_ptrs_same_scope_component
    by blast
  then show ?thesis
```

```
    by (meson assms(1) assms(2) assms(3) assms(5) get_scdom_component_ptr_in_heap
        get_scdom_component_subset_get_dom_component is_OK_returns_result_E local.get_dom_component_ok that)
qed

lemma get_scdom_component_owner_document_same:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  obtains owner_document where "h ⊢ get_owner_document ptr' →r owner_document" and "cast owner_document
∈ set sc"
  using assms
  apply(auto simp add: get_scdom_component_def elim!: bind_returns_result_E2 intro!: map_M_pure_I)[1]
   apply (metis (no_types, lifting) assms(4) assms(5) document_ptr_casts_commute3
      document_ptr_document_ptr_cast get_scdom_component_contains_get_dom_component
      local.get_dom_component_ptr local.get_dom_component_root_node_same local.get_dom_component_to_tree_order
      local.get_root_node_document local.get_root_node_not_node_same local.to_tree_order_ptr_in_result
      local.to_tree_order_ptrs_in_heap node_ptr_no_document_ptr_cast)
  apply(rule map_M_pure_E2)
     apply(simp)
    apply(simp)
   apply(simp)
  using assms(4) assms(5) get_scdom_component_ptrs_same_owner_document local.to_tree_order_ptr_in_result
  by blast

lemma get_scdom_component_different_owner_documents:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_owner_document ptr →r owner_document"
  assumes "h ⊢ get_owner_document ptr' →r owner_document'"
  assumes "owner_document ≠ owner_document'"
  shows "set |h ⊢ get_scdom_component ptr|r ∩ set |h ⊢ get_scdom_component ptr'|r = {}"
  using assms get_scdom_component_ptrs_same_owner_document
  by (smt (verit) disjoint_iff_not_equal get_scdom_component_ok is_OK_returns_result_I
      local.get_owner_document_ptr_in_heap returns_result_eq returns_result_select_result)


lemma get_scdom_component_ptr:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r c"
  shows "ptr ∈ set c"
  using assms
  by (meson get_scdom_component_ptr_in_heap2 get_scdom_component_subset_get_dom_component
      is_OK_returns_result_E is_OK_returns_result_I local.get_dom_component_ok local.get_dom_component_ptr
      subsetD)
end

locale l_get_dom_component_get_scdom_component = l_get_owner_document_defs + l_heap_is_wellformed_defs +
  l_type_wf + l_known_ptrs + l_get_scdom_component_defs + l_get_dom_component_defs +
  assumes get_scdom_component_subset_get_dom_component:
    "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_scdom_component ptr →r sc ⟹
h ⊢ get_dom_component ptr →r c ⟹ set c ⊆ set sc"
  assumes get_scdom_component_ptrs_same_scope_component:
    "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_scdom_component ptr →r sc ⟹
ptr' ∈ set sc ⟹ h ⊢ get_scdom_component ptr' →r sc"
  assumes get_scdom_component_ptrs_same_owner_document:
    "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_scdom_component ptr →r sc ⟹
ptr' ∈ set sc ⟹ h ⊢ get_owner_document ptr →r owner_document ⟹ h ⊢ get_owner_document ptr' →r owner_documen
  assumes get_scdom_component_ok:
    "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ ptr |∈| object_ptr_kinds h ⟹
h ⊢ ok (get_scdom_component ptr)"
  assumes get_scdom_component_ptr_in_heap:
    "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_scdom_component ptr →r sc ⟹
ptr' ∈ set sc ⟹ ptr' |∈| object_ptr_kinds h"
  assumes get_scdom_component_contains_get_dom_component:
```

```
    "(⋀c. h ⊢ get_dom_component ptr' →r c ⟹ set c ⊆ set sc ⟹ thesis) ⟹ heap_is_wellformed h ⟹
type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_scdom_component ptr →r sc ⟹ ptr' ∈ set sc ⟹ thesis"
  assumes get_scdom_component_owner_document_same:
    "(⋀owner_document. h ⊢ get_owner_document ptr' →r owner_document ⟹ cast owner_document ∈ set sc
⟹ thesis) ⟹
    heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_scdom_component ptr →r sc ⟹
ptr' ∈ set sc ⟹ thesis"
  assumes get_scdom_component_different_owner_documents:
    "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹ h ⊢ get_owner_document ptr →r owner_document
⟹
h ⊢ get_owner_document ptr' →r owner_document' ⟹ owner_document ≠ owner_document' ⟹
set |h ⊢ get_scdom_component ptr|r ∩ set |h ⊢ get_scdom_component ptr'|r = {}"

interpretation i_get_dom_component_get_scdom_component?: l_get_dom_component_get_scdom_component_Core_DOM
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_owner_document
  get_disconnected_nodes get_disconnected_nodes_locs to_tree_order heap_is_wellformed parent_child_rel
  type_wf known_ptr known_ptrs get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors
  get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  by(auto simp add: l_get_dom_component_get_scdom_component_Core_DOM_def l_get_dom_component_get_scdom_component_C
instances)
declare l_get_dom_component_get_scdom_component_Core_DOM_axioms [instances]

lemma get_dom_component_get_scdom_component_is_l_get_dom_component_get_scdom_component [instances]:
  "l_get_dom_component_get_scdom_component get_owner_document heap_is_wellformed type_wf known_ptr
known_ptrs get_scdom_component get_dom_component"
  apply(auto simp add: l_get_dom_component_get_scdom_component_def l_get_dom_component_get_scdom_component_axioms_
instances)[1]
  using get_scdom_component_subset_get_dom_component apply fast
  using get_scdom_component_ptrs_same_scope_component apply fast
  using get_scdom_component_ptrs_same_owner_document apply fast
  using get_scdom_component_ok apply fast
  using get_scdom_component_ptr_in_heap apply fast
  using get_scdom_component_contains_get_dom_component apply blast
  using get_scdom_component_owner_document_same apply blast
  using get_scdom_component_different_owner_documents apply fast
  done
```

get_child_nodes

```
locale l_get_scdom_component_get_child_nodes_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_get_child_nodes_Core_DOM
begin
lemma get_child_nodes_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "h ⊢ get_child_nodes ptr' →r children"
  assumes "child ∈ set children"
  shows "cast child ∈ set sc ⟷ ptr' ∈ set sc"
  apply(auto)[1]
   apply (meson assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) contra_subsetD
      get_scdom_component_ptrs_same_scope_component get_scdom_component_subset_get_dom_component
      is_OK_returns_result_E local.get_child_nodes_is_strongly_dom_component_safe local.get_dom_component_ok
      local.get_dom_component_ptr local.heap_is_wellformed_children_in_heap node_ptr_kinds_commutes)
  by (meson assms(1) assms(2) assms(3) assms(4) assms(5) assms(6)
      get_scdom_component_contains_get_dom_component is_OK_returns_result_E is_OK_returns_result_I
      get_child_nodes_is_strongly_dom_component_safe local.get_child_nodes_ptr_in_heap
      local.get_dom_component_ok local.get_dom_component_ptr set_rev_mp)

lemma get_child_nodes_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
```

```
  assumes "h ⊢ get_child_nodes ptr →r children"
  assumes "h ⊢ get_child_nodes ptr →h h'"
  shows "is_strongly_scdom_component_safe {ptr} (cast ` set children) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_child_nodes_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def)[1]
    by (smt (verit, del_insts) IntI
        get_child_nodes_is_strongly_scdom_component_safe_step is_OK_returns_result_I
        local.get_child_nodes_ptr_in_heap local.get_dom_component_ok local.get_dom_component_ptr
        local.get_scdom_component_impl local.get_scdom_component_ok
        local.get_scdom_component_subset_get_dom_component returns_result_select_result subsetD)
qed
end

interpretation i_get_scdom_component_get_child_nodes?: l_get_scdom_component_get_child_nodes_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe get_disconnected_nodes
  get_disconnected_nodes_locs to_tree_order get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_scdom_component_get_child_nodes_Core_DOM_def instances)
declare l_get_scdom_component_get_child_nodes_Core_DOM_axioms [instances]
```

get_parent

```
locale l_get_scdom_component_get_parent_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_get_parent_Core_DOM
begin

lemma get_parent_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "h ⊢ get_parent ptr' →r Some parent"
  shows "parent ∈ set sc ⟷ cast ptr' ∈ set sc"
  by (meson assms(1) assms(2) assms(3) assms(4) assms(5) contra_subsetD
      get_scdom_component_contains_get_dom_component local.get_dom_component_ptr
      local.get_parent_is_strongly_dom_component_safe_step)

lemma get_parent_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_parent node_ptr →r Some parent"
  assumes "h ⊢ get_parent node_ptr →h h'"
  shows "is_strongly_scdom_component_safe {cast node_ptr} {parent} h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_parent_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def)[1]
    by (smt (verit) Int_iff get_parent_is_strongly_scdom_component_safe_step in_mono
        get_dom_component_ptr local.get_dom_component_ok
        local.get_parent_parent_in_heap local.get_scdom_component_impl local.get_scdom_component_ok
        local.get_scdom_component_ptr_in_heap local.get_scdom_component_ptrs_same_scope_component
        local.get_scdom_component_subset_get_dom_component
        returns_result_eq returns_result_select_result)
```

```
qed
end

interpretation i_get_scdom_component_get_parent?: l_get_scdom_component_get_parent_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_disconnected_nodes
  get_disconnected_nodes_locs to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_scdom_component_get_parent_Core_DOM_def instances)
declare l_get_scdom_component_get_parent_Core_DOM_axioms [instances]
```

get_root_node

```
locale l_get_scdom_component_get_root_node_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_get_root_node_Core_DOM
begin

lemma get_root_node_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →_r sc"
  assumes "h ⊢ get_root_node ptr' →_r root"
  shows "root ∈ set sc ⟷ ptr' ∈ set sc"
  by (meson assms(1) assms(2) assms(3) assms(4) assms(5) contra_subsetD
      get_scdom_component_contains_get_dom_component local.get_dom_component_ptr
      local.get_root_node_is_strongly_dom_component_safe_step)

lemma get_root_node_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node ptr →_r root"
  assumes "h ⊢ get_root_node ptr →_h h'"
  shows "is_strongly_scdom_component_safe {ptr} {root} h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_root_node_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def)[1]
    by (smt (verit) Int_iff is_OK_returns_result_I local.get_dom_component_ok
        local.get_dom_component_ptr local.get_root_node_is_strongly_dom_component_safe_step
        local.get_root_node_ptr_in_heap local.get_scdom_component_impl local.get_scdom_component_ok
        local.get_scdom_component_subset_get_dom_component returns_result_select_result subset_eq)
qed
end

interpretation i_get_scdom_component_get_root_node?: l_get_scdom_component_get_root_node_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_disconnected_nodes
  get_disconnected_nodes_locs to_tree_order get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name first_in_tree_order
  get_attribute get_attribute_locs
  by(auto simp add: l_get_scdom_component_get_root_node_Core_DOM_def instances)
declare l_get_scdom_component_get_root_node_Core_DOM_axioms [instances]
```

get_element_by_id

```
locale l_get_scdom_component_get_element_by_id_Core_DOM =
```

```
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_get_element_by_id_Core_DOM
begin

lemma get_element_by_id_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →_r sc"
  assumes "h ⊢ get_element_by_id ptr' idd →_r Some result"
  shows "cast result ∈ set sc ⟷ ptr' ∈ set sc"
  by (meson assms(1) assms(2) assms(3) assms(4) assms(5) contra_subsetD
      get_element_by_id_is_strongly_dom_component_safe_step get_scdom_component_contains_get_dom_component
      local.get_dom_component_ptr)

lemma get_element_by_id_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_element_by_id ptr idd →_r Some result"
  assumes "h ⊢ get_element_by_id ptr idd →_h h'"
  shows "is_strongly_scdom_component_safe {ptr} {cast result} h h'"
proof -
  have "h = h'"
    using assms(5)
    by(auto simp add: preserved_def get_element_by_id_def first_in_tree_order_def
        elim!: bind_returns_heap_E2 intro!: map_filter_M_pure bind_pure_I
        split: option.splits list.splits)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_element_by_id_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →_r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast result ∈ set to"
    using assms(4) local.get_element_by_id_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_scdom_component ptr →_r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_scdom_component_impl
      local.get_scdom_component_ok
    by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def
        get_element_by_id_def first_in_tree_order_def elim!: bind_returns_result_E2
        intro!: map_filter_M_pure bind_pure_I split: option.splits list.splits)[1]
    by (smt (verit) IntI ‹ptr |∈| object_ptr_kinds h› assms(4)
        get_element_by_id_is_strongly_scdom_component_safe_step local.get_dom_component_ok
        local.get_dom_component_ptr local.get_scdom_component_impl
        local.get_scdom_component_subset_get_dom_component returns_result_select_result select_result_I2
        subsetD)
qed
end

interpretation i_get_scdom_component_get_element_by_id?: l_get_scdom_component_get_element_by_id_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_disconnected_nodes
  get_disconnected_nodes_locs to_tree_order get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name first_in_tree_order
  get_attribute get_attribute_locs
  by(auto simp add: l_get_scdom_component_get_element_by_id_Core_DOM_def instances)
declare l_get_scdom_component_get_element_by_id_Core_DOM_axioms [instances]
```

get_elements_by_class_name

```
locale l_get_scdom_component_get_elements_by_class_name_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_get_elements_by_class_name_Core_DOM
begin

lemma get_elements_by_class_name_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →_r sc"
  assumes "h ⊢ get_elements_by_class_name ptr' idd →_r results"
  assumes "result ∈ set results"
  shows "cast result ∈ set sc ⟷ ptr' ∈ set sc"
  by (meson assms local.get_dom_component_ptr
      local.get_elements_by_class_name_is_strongly_dom_component_safe_step
      local.get_scdom_component_contains_get_dom_component subsetD)


lemma get_elements_by_class_name_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_elements_by_class_name ptr idd →_r results"
  assumes "h ⊢ get_elements_by_class_name ptr idd →_h h'"
  shows "is_strongly_scdom_component_safe {ptr} (cast ` set results) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_elements_by_class_name_pure pure_returns_heap_eq)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_elements_by_class_name_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →_r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast ` set results ⊆ set to"
    using assms(4) local.get_elements_by_class_name_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_scdom_component ptr →_r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_scdom_component_impl
      local.get_scdom_component_ok by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def
        get_element_by_id_def first_in_tree_order_def elim!: bind_returns_result_E2 intro!: map_filter_M_pure
        bind_pure_I split: option.splits list.splits)[1]
    by (smt (verit) IntI ‹ptr |∈| object_ptr_kinds h›
        get_elements_by_class_name_is_strongly_scdom_component_safe_step local.get_dom_component_ok
        local.get_dom_component_ptr local.get_scdom_component_impl
        local.get_scdom_component_subset_get_dom_component returns_result_select_result select_result_I2
subsetD)
qed
end

interpretation i_get_scdom_component_get_elements_by_class_name?: l_get_scdom_component_get_elements_by_class_name_
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_disconnected_nodes
  get_disconnected_nodes_locs to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_scdom_component_get_elements_by_class_name_Core_DOM_def instances)
```

```
declare l_get_scdom_component_get_element_by_id_Core_DOM_axioms [instances]
```

get_elements_by_tag_name

```
locale l_get_scdom_component_get_elements_by_tag_name_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_get_dom_component_get_elements_by_tag_name_Core_DOM
begin

lemma get_elements_by_tag_name_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →_r sc"
  assumes "h ⊢ get_elements_by_tag_name ptr' idd →_r results"
  assumes "result ∈ set results"
  shows "cast result ∈ set sc ⟷ ptr' ∈ set sc"
  by (meson assms local.get_dom_component_ptr
      local.get_elements_by_tag_name_is_strongly_dom_component_safe_step
      local.get_scdom_component_contains_get_dom_component subsetD)


lemma get_elements_by_tag_name_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_elements_by_tag_name ptr idd →_r results"
  assumes "h ⊢ get_elements_by_tag_name ptr idd →_h h'"
  shows "is_strongly_scdom_component_safe {ptr} (cast ` set results) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_elements_by_tag_name_pure pure_returns_heap_eq)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_elements_by_tag_name_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →_r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast ` set results ⊆ set to"
    using assms(4) local.get_elements_by_tag_name_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_scdom_component ptr →_r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_scdom_component_impl
      local.get_scdom_component_ok by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def
        get_element_by_id_def first_in_tree_order_def elim!: bind_returns_result_E2 intro!:
        map_filter_M_pure bind_pure_I split: option.splits list.splits)[1]
    by (smt (verit) IntI ‹ptr |∈| object_ptr_kinds h›
        get_elements_by_tag_name_is_strongly_scdom_component_safe_step local.get_dom_component_ok
        local.get_dom_component_ptr local.get_scdom_component_impl
        local.get_scdom_component_subset_get_dom_component returns_result_select_result select_result_I2
        subsetD)
qed
end

interpretation i_get_scdom_component_get_elements_by_tag_name?:
  l_get_scdom_component_get_elements_by_tag_name_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe
  get_disconnected_nodes get_disconnected_nodes_locs to_tree_order get_parent get_parent_locs
```

```
  get_child_nodes get_child_nodes_locs get_root_node get_root_node_locs get_ancestors
  get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_scdom_component_get_elements_by_tag_name_Core_DOM_def instances)
declare l_get_scdom_component_get_element_by_id_Core_DOM_axioms [instances]
```

remove_child

```
locale l_get_scdom_component_remove_child_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_dom_component_Core_DOM +
  l_get_scdom_component_Core_DOM +
  l_remove_child_Core_DOM +
  l_set_child_nodes_Core_DOM +
  l_set_disconnected_nodes_Core_DOM +
  l_get_owner_document_wf +
  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 parent_child_rel
begin
lemma remove_child_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_child ptr' child →_h h'"
  assumes "ptr ∉ set |h ⊢ get_dom_component ptr'|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document (cast_node_ptr2object_ptr child)|_r)|_r"

  shows "preserved (get_M ptr getter) h h'"
proof -
  have "ptr ≠ ptr'"
    using assms(5)
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(4) is_OK_returns_heap_I
        is_OK_returns_result_E local.get_dom_component_ok local.get_dom_component_ptr
        local.remove_child_ptr_in_heap select_result_I2)

  obtain owner_document where owner_document: "h ⊢ get_owner_document (cast_node_ptr2object_ptr child) →_r
owner_document"
    by (meson assms(1) assms(2) assms(3) assms(4) is_OK_returns_result_E local.get_owner_document_ok
        local.remove_child_child_in_heap node_ptr_kinds_commutes)
  then
  obtain c where "h ⊢ get_dom_component (cast owner_document) →_r c"
    using get_dom_component_ok owner_document assms(1) assms(2) assms(3)
    by (meson document_ptr_kinds_commutes get_owner_document_owner_document_in_heap select_result_I)
  then
  have "ptr ≠ cast owner_document"
    using assms(6) assms(1) assms(2) assms(3) local.get_dom_component_ptr owner_document
    by auto

  show ?thesis
    using remove_child_writes assms(4)
    apply(rule reads_writes_preserved2)
    apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: option.splits)[1]
       apply (metis ‹ptr ≠ ptr'› document_ptr_casts_commute3 get_M_Mdocument_preserved3)
      apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
     apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
    apply (metis ‹ptr ≠ ptr'› element_ptr_casts_commute3 get_M_Element_preserved8)
   apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
      get_M_Mdocument_preserved3 owner_document select_result_I2)
    apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
```

```
    done
qed

lemma remove_child_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_child ptr' child →_h h'"
  assumes "ptr ∉ set |h ⊢ get_scdom_component ptr'|_r"

  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast child)|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain sc where sc: "h ⊢ get_scdom_component ptr' →_r sc"
    using get_scdom_component_ok
    by (meson assms(1) assms(2) assms(3) assms(4) is_OK_returns_heap_I local.remove_child_ptr_in_heap
        returns_result_select_result)

  have "child |∈| node_ptr_kinds h"
    using assms(4) remove_child_child_in_heap by blast
  then
  obtain child_sc where child_sc: "h ⊢ get_scdom_component (cast child) →_r child_sc"
    using get_scdom_component_ok
    by (meson assms(1) assms(2) assms(3) node_ptr_kinds_commutes select_result_I)
  then obtain owner_document where owner_document: "h ⊢ get_owner_document (cast_node_ptr2object_ptr child)
→_r owner_document"
    by (meson ‹child |∈| node_ptr_kinds h› assms(1) assms(2) assms(3) contra_subsetD
        get_scdom_component_owner_document_same is_OK_returns_result_E
        get_scdom_component_subset_get_dom_component local.get_dom_component_ok local.get_dom_component_ptr
        node_ptr_kinds_commutes)
  then have "h ⊢ get_scdom_component (cast owner_document) →_r child_sc"
    using child_sc
    by (smt (verit) ‹child |∈| node_ptr_kinds h› assms(1) assms(2) assms(3) contra_subsetD
        get_scdom_component_subset_get_dom_component get_scdom_component_owner_document_same
        get_scdom_component_ptrs_same_scope_component local.get_dom_component_ok local.get_dom_component_ptr
        node_ptr_kinds_commutes returns_result_select_result select_result_I2)

  have "ptr ∉ set |h ⊢ get_dom_component ptr'|_r"
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(4) assms(5) contra_subsetD
        get_scdom_component_subset_get_dom_component is_OK_returns_heap_I local.get_dom_component_ok
        local.remove_child_ptr_in_heap returns_result_select_result sc select_result_I2)

  moreover have "ptr ∉ set |h ⊢ get_scdom_component (cast |h ⊢ get_owner_document (cast_node_ptr2object_ptr
child)|_r)|_r"
    using get_scdom_component_owner_document_same get_scdom_component_ptrs_same_scope_component
    by (metis (no_types, lifting)
        ‹h ⊢ get_scdom_component (cast_document_ptr2object_ptr owner_document) →_r child_sc› assms(6) child_sc
        owner_document select_result_I2)
  have "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document (cast_node_ptr2object_ptr child)|_r)|_r"
    using get_scdom_component_owner_document_same
    by (metis (no_types, lifting)
        ‹h ⊢ get_scdom_component (cast_document_ptr2object_ptr owner_document) →_r child_sc›
        ‹ptr ∉ set |h ⊢ get_scdom_component (cast_document_ptr2object_ptr |h ⊢ get_owner_document (cast_node_ptr2object_
child)|_r)|_r›
        assms(1) assms(2) assms(3) contra_subsetD document_ptr_kinds_commutes get_scdom_component_subset_get_dom_co
        is_OK_returns_result_E local.get_dom_component_ok local.get_owner_document_owner_document_in_heap
owner_document
        select_result_I2)
  ultimately show ?thesis
    using assms(1) assms(2) assms(3) assms(4) remove_child_is_component_unsafe by blast
qed


lemma remove_child_is_strongly_dom_component_safe:
```

```
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ remove_child ptr child →_h h'"
  shows "is_strongly_scdom_component_safe {ptr, cast child} {} h 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 object_ptr_kinds_eq3: "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)
  then have object_ptr_kinds_eq: "⋀ptrs. h ⊢ object_ptr_kinds_M →_r ptrs = h' ⊢ object_ptr_kinds_M →_r
ptrs"
    unfolding object_ptr_kinds_M_defs by simp
  then have object_ptr_kinds_eq2: "|h ⊢ object_ptr_kinds_M|_r = |h' ⊢ object_ptr_kinds_M|_r"
    using select_result_eq by force
  then have node_ptr_kinds_eq2: "|h ⊢ node_ptr_kinds_M|_r = |h' ⊢ node_ptr_kinds_M|_r"
    using node_ptr_kinds_M_eq by auto
  then have node_ptr_kinds_eq3: "node_ptr_kinds h = node_ptr_kinds h'"
    using node_ptr_kinds_M_eq by auto
  have document_ptr_kinds_eq2: "|h ⊢ document_ptr_kinds_M|_r = |h' ⊢ document_ptr_kinds_M|_r"
    using object_ptr_kinds_eq2 document_ptr_kinds_M_eq by auto
  then have document_ptr_kinds_eq3: "document_ptr_kinds h = document_ptr_kinds h'"
    using document_ptr_kinds_M_eq by auto
  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 disconnected_nodes_eq: "⋀document_ptr disconnected_nodes. document_ptr ≠ owner_document
        ⟹ h ⊢ get_disconnected_nodes document_ptr →_r disconnected_nodes
            = h' ⊢ get_disconnected_nodes document_ptr →_r disconnected_nodes"
    apply(rule reads_writes_preserved[OF get_disconnected_nodes_reads remove_child_writes assms(4)])
    unfolding remove_child_locs_def
    using set_child_nodes_get_disconnected_nodes set_disconnected_nodes_get_disconnected_nodes_different_pointers
    by (metis (no_types, lifting) Un_iff owner_document select_result_I2)
  then have disconnected_nodes_eq2:
    "⋀document_ptr. document_ptr ≠ owner_document
    ⟹ |h ⊢ get_disconnected_nodes document_ptr|_r = |h' ⊢ get_disconnected_nodes document_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 "known_ptrs h'"
    using object_ptr_kinds_eq3 known_ptrs_preserved ‹known_ptrs h› by blast

  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 assms(2)
    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_child_nodes_writes h']
    using  set_child_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  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_h2: "h2 ⊢ get_disconnected_nodes owner_document →r child # disconnected_nodes_h"
    using owner_document assms(4) h2 disconnected_nodes_h
    apply (auto simp add: remove_child_def  split: if_splits)[1]
    apply(drule bind_returns_heap_E2)
      apply(auto split: if_splits)[1]
     apply(simp)
    by(auto simp add: local.set_disconnected_nodes_get_disconnected_nodes split: if_splits)
  then have disconnected_nodes_h': "h' ⊢ get_disconnected_nodes owner_document →r child # disconnected_nodes_h"
    apply(rule reads_writes_separate_forwards[OF get_disconnected_nodes_reads set_child_nodes_writes h'])
    by (simp add: set_child_nodes_get_disconnected_nodes)

  moreover have "a_acyclic_heap h"
    using assms(1)  by (simp add: heap_is_wellformed_def)
  have "parent_child_rel h' ⊆ parent_child_rel h"
  proof (standard, safe)
    fix parent child
    assume a1: "(parent, child) ∈ parent_child_rel h'"
    then show "(parent, child) ∈ parent_child_rel h"
    proof (cases "parent = ptr")
      case True
      then show ?thesis
        using a1 remove_child_removes_parent[OF assms(1) assms(4)] children_h children_h'
          get_child_nodes_ptr_in_heap
        apply(auto simp add: parent_child_rel_def object_ptr_kinds_eq )[1]
        by (metis imageI notin_set_remove1)
    next
      case False
      then show ?thesis
```

```
      using a1
      by(auto simp add: parent_child_rel_def object_ptr_kinds_eq3 children_eq2)
    qed
  qed
  then have "a_acyclic_heap h'"
    using ‹a_acyclic_heap h› acyclic_heap_def acyclic_subset by blast

  moreover have "a_all_ptrs_in_heap h"
    using assms(1) by (simp add: heap_is_wellformed_def)
  then have "a_all_ptrs_in_heap h'"
    apply(auto simp add: a_all_ptrs_in_heap_def node_ptr_kinds_eq3 disconnected_nodes_eq)[1]
     apply (metis (no_types, lifting) ‹type_wf h'› assms local.get_child_nodes_ok local.known_ptrs_known_ptr
        local.remove_child_children_subset object_ptr_kinds_eq3 returns_result_select_result subset_code(1))
    apply (metis (no_types, lifting) assms(4) disconnected_nodes_eq2 disconnected_nodes_h disconnected_nodes_h'
        document_ptr_kinds_eq3 local.remove_child_child_in_heap node_ptr_kinds_eq3 select_result_I2
        set_ConsD subset_code(1))
    done
  moreover have "a_owner_document_valid h"
    using assms(1) by (simp add: heap_is_wellformed_def)
  then have "a_owner_document_valid h'"
    apply(auto simp add: a_owner_document_valid_def object_ptr_kinds_eq3 document_ptr_kinds_eq3
        node_ptr_kinds_eq3)[1]
  proof -
    fix node_ptr
    assume 0: "∀node_ptr∈fset (node_ptr_kinds h'). (∃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)"
      and 1: "node_ptr |∈| node_ptr_kinds h'"
      and 2: "∀parent_ptr. parent_ptr |∈| object_ptr_kinds h' ⟶ node_ptr ∉ set |h' ⊢ get_child_nodes
parent_ptr|_r"
    then show "∃document_ptr. document_ptr |∈| document_ptr_kinds h'
                    ∧ node_ptr ∈ set |h' ⊢ get_disconnected_nodes document_ptr|_r"
    proof (cases "node_ptr = child")
      case True
      show ?thesis
        apply(rule exI[where x=owner_document])
        using children_eq2 disconnected_nodes_eq2 children_h children_h' disconnected_nodes_h' True
        by (metis (no_types, lifting) get_disconnected_nodes_ptr_in_heap is_OK_returns_result_I
            list.set_intros(1) select_result_I2)
    next
      case False
      then show ?thesis
        using 0 1 2 children_eq2 children_h children_h' disconnected_nodes_eq2 disconnected_nodes_h
          disconnected_nodes_h'
        apply(auto simp add: children_eq2 disconnected_nodes_eq2 dest!: select_result_I2)[1]
        by (metis children_eq2 disconnected_nodes_eq2 in_set_remove1 list.set_intros(2))
    qed
  qed

  moreover
  {
    have h0: "a_distinct_lists h"
      using assms(1) by (simp add: heap_is_wellformed_def)
    moreover have ha1: "(⋃x∈set |h ⊢ object_ptr_kinds_M|_r. set |h ⊢ get_child_nodes x|_r)
                ∩ (⋃x∈set |h ⊢ document_ptr_kinds_M|_r. set |h ⊢ get_disconnected_nodes x|_r) = {}"
      using ‹a_distinct_lists h›
      unfolding a_distinct_lists_def
      by(auto)
    have ha2: "ptr |∈| object_ptr_kinds h"
      using children_h get_child_nodes_ptr_in_heap by blast
    have ha3: "child ∈ set |h ⊢ get_child_nodes ptr|_r"
```

```
    using child_in_children_h children_h
    by(simp)
  have child_not_in: "⋀document_ptr. document_ptr |∈| document_ptr_kinds h
                      ⟹ child ∉ set |h ⊢ get_disconnected_nodes document_ptr|_r"
    using ha1 ha2 ha3
    apply(simp)
    using IntI by fastforce
  moreover have "distinct |h ⊢ object_ptr_kinds_M|_r"
    apply(rule select_result_I)
    by(auto simp add: object_ptr_kinds_M_defs)
  moreover have "distinct |h ⊢ document_ptr_kinds_M|_r"
    apply(rule select_result_I)
    by(auto simp add: document_ptr_kinds_M_defs)
  ultimately have "a_distinct_lists h'"
  proof(simp (no_asm) add: a_distinct_lists_def, safe)
    assume 1: "a_distinct_lists h"
      and 3: "distinct |h ⊢ object_ptr_kinds_M|_r"

    assume 1: "a_distinct_lists h"
      and 3: "distinct |h ⊢ object_ptr_kinds_M|_r"
    have 4: "distinct (concat ((map (λptr. |h ⊢ get_child_nodes ptr|_r) |h ⊢ object_ptr_kinds_M|_r)))"
      using 1 by(auto simp add: a_distinct_lists_def)
    show "distinct (concat (map (λptr. |h' ⊢ get_child_nodes ptr|_r)
                  (sorted_list_of_set (fset (object_ptr_kinds h')))))"
    proof(rule distinct_concat_map_I[OF 3[unfolded object_ptr_kinds_eq2], simplified])
      fix x
      assume 5: "x |∈| object_ptr_kinds h'"
      then have 6: "distinct |h ⊢ get_child_nodes x|_r"
        using 4 distinct_concat_map_E object_ptr_kinds_eq2 by fastforce
      obtain children where children: "h ⊢ get_child_nodes x →_r children"
        and distinct_children: "distinct children"
        by (metis "5" "6" assms get_child_nodes_ok local.known_ptrs_known_ptr
            object_ptr_kinds_eq3 select_result_I)
      obtain children' where children': "h' ⊢ get_child_nodes x →_r children'"
        using children children_eq children_h' by fastforce
      then have "distinct children'"
      proof (cases "ptr = x")
        case True
        then show ?thesis
          using children distinct_children children_h children_h'
          by (metis children' distinct_remove1 returns_result_eq)
      next
        case False
        then show ?thesis
          using children distinct_children children_eq[OF False]
          using children' distinct_lists_children h0
          using select_result_I2 by fastforce
      qed

      then show "distinct |h' ⊢ get_child_nodes x|_r"
        using children' by(auto simp add: )
    next
      fix x y
      assume 5: "x |∈| object_ptr_kinds h'" and 6: "y |∈| object_ptr_kinds h'" and 7: "x ≠ y"
      obtain children_x where children_x: "h ⊢ get_child_nodes x →_r children_x"
        by (metis "5" assms get_child_nodes_ok is_OK_returns_result_E
            local.known_ptrs_known_ptr object_ptr_kinds_eq3)
      obtain children_y where children_y: "h ⊢ get_child_nodes y →_r children_y"
        by (metis "6" assms get_child_nodes_ok is_OK_returns_result_E
            local.known_ptrs_known_ptr object_ptr_kinds_eq3)
      obtain children_x' where children_x': "h' ⊢ get_child_nodes x →_r children_x'"
        using children_eq children_h' children_x by fastforce
      obtain children_y' where children_y': "h' ⊢ get_child_nodes y →_r children_y'"
```

```
      using children_eq children_h' children_y by fastforce
    have "distinct (concat (map (λptr. |h ⊢ get_child_nodes ptr|_r) |h ⊢ object_ptr_kinds_M|_r))"
      using h0 by(auto simp add: a_distinct_lists_def)
    then have 8: "set children_x ∩ set children_y = {}"
      using "7" assms(1) children_x children_y local.heap_is_wellformed_one_parent by blast
    have "set children_x' ∩ set children_y' = {}"
    proof (cases "ptr = x")
      case True
      then have "ptr ≠ y"
        by(simp add: 7)
      have "children_x' = remove1 child children_x"
        using children_h children_h' children_x children_x' True returns_result_eq by fastforce
      moreover have "children_y' = children_y"
        using children_y children_y' children_eq[OF ‹ptr ≠ y›] by auto
      ultimately show ?thesis
        using 8 set_remove1_subset by fastforce
    next
      case False
      then show ?thesis
      proof (cases "ptr = y")
        case True
        have "children_y' = remove1 child children_y"
          using children_h children_h' children_y children_y' True returns_result_eq by fastforce
        moreover have "children_x' = children_x"
          using children_x children_x' children_eq[OF ‹ptr ≠ x›] by auto
        ultimately show ?thesis
          using 8 set_remove1_subset by fastforce
      next
        case False
        have "children_x' = children_x"
          using children_x children_x' children_eq[OF ‹ptr ≠ x›] by auto
        moreover have "children_y' = children_y"
          using children_y children_y' children_eq[OF ‹ptr ≠ y›] by auto
        ultimately show ?thesis
          using 8 by simp
      qed
    qed
    then show "set |h' ⊢ get_child_nodes x|_r ∩ set |h' ⊢ get_child_nodes y|_r = {}"
      using children_x' children_y'
      by (metis (no_types, lifting) select_result_I2)
  qed
next
  assume 2: "distinct |h ⊢ document_ptr_kinds_M|_r"
  then have 4: "distinct  (sorted_list_of_set (fset (document_ptr_kinds h')))"
    by simp
  have 3: "distinct (concat (map (λdocument_ptr. |h ⊢ get_disconnected_nodes document_ptr|_r)
                  (sorted_list_of_set (fset (document_ptr_kinds h')))))"
    using h0
    by(simp add: a_distinct_lists_def document_ptr_kinds_eq3)

  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[OF 4[unfolded document_ptr_kinds_eq3]])
    fix x
    assume 4: "x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h')))"
    have 5: "distinct |h ⊢ get_disconnected_nodes x|_r"
      using distinct_lists_disconnected_nodes[OF h0] 4 get_disconnected_nodes_ok
      by (simp add: assms document_ptr_kinds_eq3 select_result_I)
    show "distinct |h' ⊢ get_disconnected_nodes x|_r"
    proof (cases "x = owner_document")
      case True
      have "child ∉ set |h ⊢ get_disconnected_nodes x|_r"
        using child_not_in  document_ptr_kinds_eq2 "4" by fastforce
```

```
      moreover have "|h' ⊢ get_disconnected_nodes x|_r = child # |h ⊢ get_disconnected_nodes x|_r"
        using disconnected_nodes_h' disconnected_nodes_h unfolding True
        by(simp)
      ultimately show ?thesis
        using 5 unfolding True
        by simp
    next
      case False
      show ?thesis
        using "5" False disconnected_nodes_eq2 by auto
    qed
  next
    fix x y
    assume 4: "x ∈ set (sorted_list_of_set (fset (document_ptr_kinds h')))"
      and 5: "y ∈ set (sorted_list_of_set (fset (document_ptr_kinds h')))" and "x ≠ y"
    obtain disc_nodes_x where disc_nodes_x: "h ⊢ get_disconnected_nodes x →_r disc_nodes_x"
      using 4 get_disconnected_nodes_ok[OF ‹type_wf h›, of x] document_ptr_kinds_eq2
      by auto
    obtain disc_nodes_y where disc_nodes_y: "h ⊢ get_disconnected_nodes y →_r disc_nodes_y"
      using 5 get_disconnected_nodes_ok[OF ‹type_wf h›, of y] document_ptr_kinds_eq2
      by auto
    obtain disc_nodes_x' where disc_nodes_x': "h' ⊢ get_disconnected_nodes x →_r disc_nodes_x'"
      using 4 get_disconnected_nodes_ok[OF ‹type_wf h'›, of x] document_ptr_kinds_eq2
      by auto
    obtain disc_nodes_y' where disc_nodes_y': "h' ⊢ get_disconnected_nodes y →_r disc_nodes_y'"
      using 5 get_disconnected_nodes_ok[OF ‹type_wf h'›, of y] document_ptr_kinds_eq2
      by auto
    have "distinct
          (concat (map (λdocument_ptr. |h ⊢ get_disconnected_nodes document_ptr|_r) |h ⊢ document_ptr_kinds_M|
      using h0 by (simp add: a_distinct_lists_def)
    then have 6: "set disc_nodes_x ∩ set disc_nodes_y = {}"
      using ‹x ≠ y› assms(1) disc_nodes_x disc_nodes_y local.heap_is_wellformed_one_disc_parent
      by blast

    have "set disc_nodes_x' ∩ set disc_nodes_y' = {}"
    proof (cases "x = owner_document")
      case True
      then have "y ≠ owner_document"
        using ‹x ≠ y› by simp
      then have "disc_nodes_y' = disc_nodes_y"
        using disconnected_nodes_eq[OF ‹y ≠ owner_document›] disc_nodes_y disc_nodes_y'
        by auto
      have "disc_nodes_x' = child # disc_nodes_x"
        using disconnected_nodes_h' disc_nodes_x disc_nodes_x' True disconnected_nodes_h returns_result_eq
        by fastforce
      have "child ∉ set disc_nodes_y"
        using child_not_in  disc_nodes_y 5
        using document_ptr_kinds_eq2  by fastforce
      then show ?thesis
        apply(unfold ‹disc_nodes_x' = child # disc_nodes_x› ‹disc_nodes_y' = disc_nodes_y›)
        using 6 by auto
    next
      case False
      then show ?thesis
      proof (cases "y = owner_document")
        case True
        then have "disc_nodes_x' = disc_nodes_x"
          using disconnected_nodes_eq[OF ‹x ≠ owner_document›] disc_nodes_x disc_nodes_x' by auto
        have "disc_nodes_y' = child # disc_nodes_y"
          using disconnected_nodes_h' disc_nodes_y disc_nodes_y' True disconnected_nodes_h returns_result_eq
          by fastforce
        have "child ∉ set disc_nodes_x"
          using child_not_in  disc_nodes_x 4
```

```
        using document_ptr_kinds_eq2 by fastforce
      then show ?thesis
        apply(unfold ‹disc_nodes_y' = child # disc_nodes_y› ‹disc_nodes_x' = disc_nodes_x›)
        using 6 by auto
    next
      case False
      have "disc_nodes_x' = disc_nodes_x"
        using disconnected_nodes_eq[OF ‹x ≠ owner_document›] disc_nodes_x disc_nodes_x' by auto
      have "disc_nodes_y' = disc_nodes_y"
        using disconnected_nodes_eq[OF ‹y ≠ owner_document›] disc_nodes_y disc_nodes_y' by auto
      then show ?thesis
        apply(unfold ‹disc_nodes_y' = disc_nodes_y› ‹disc_nodes_x' = disc_nodes_x›)
        using 6 by auto
    qed
  qed
  then show "set |h' ⊢ get_disconnected_nodes x|_r ∩ set |h' ⊢ get_disconnected_nodes y|_r = {}"
    using disc_nodes_x' disc_nodes_y' by auto
qed
next
  fix x xa xb
  assume 1: "xa ∈ fset (object_ptr_kinds h')"
    and 2: "x ∈ set |h' ⊢ get_child_nodes xa|_r"
    and 3: "xb ∈ fset (document_ptr_kinds h')"
    and 4: "x ∈ set |h' ⊢ get_disconnected_nodes xb|_r"
  obtain disc_nodes where disc_nodes: "h ⊢ get_disconnected_nodes xb →_r disc_nodes"
    using 3 get_disconnected_nodes_ok[OF ‹type_wf h›, of xb] document_ptr_kinds_eq2 by auto
  obtain disc_nodes' where disc_nodes': "h' ⊢ get_disconnected_nodes xb →_r disc_nodes'"
    using 3 get_disconnected_nodes_ok[OF ‹type_wf h'›, of xb]  document_ptr_kinds_eq2 by auto

  obtain children where children: "h ⊢ get_child_nodes xa →_r children"
    by (metis "1" assms get_child_nodes_ok is_OK_returns_result_E
        local.known_ptrs_known_ptr object_ptr_kinds_eq3)
  obtain children' where children': "h' ⊢ get_child_nodes xa →_r children'"
    using children children_eq children_h' by fastforce
  have "⋀x. x ∈ set |h ⊢ get_child_nodes xa|_r ⟹ x ∈ set |h ⊢ get_disconnected_nodes xb|_r ⟹ False"
    using 1 3
    apply(fold ‹ object_ptr_kinds h =  object_ptr_kinds h'›)
    apply(fold ‹ document_ptr_kinds h =  document_ptr_kinds h'›)
    using children disc_nodes h0 apply(auto simp add: a_distinct_lists_def)[1]
    by (metis (no_types, lifting) h0 local.distinct_lists_no_parent select_result_I2)
  then have 5: "⋀x. x ∈ set children ⟹ x ∈ set disc_nodes ⟹ False"
    using children disc_nodes  by fastforce
  have 6: "|h' ⊢ get_child_nodes xa|_r = children'"
    using children' by simp
  have 7: "|h' ⊢ get_disconnected_nodes xb|_r = disc_nodes'"
    using disc_nodes' by simp
  have "False"
  proof (cases "xa = ptr")
    case True
    have "distinct children_h"
      using children_h distinct_lists_children h0 ‹known_ptr ptr› by blast
    have "|h' ⊢ get_child_nodes ptr|_r = remove1 child children_h"
      using children_h'
      by simp
    have "children = children_h"
      using True children children_h by auto
    show ?thesis
      using disc_nodes' children' 5 2 4 children_h ‹distinct children_h› disconnected_nodes_h'
      apply(auto simp add: 6 7
          ‹xa = ptr› ‹|h' ⊢ get_child_nodes ptr|_r = remove1 child children_h› ‹children = children_h›)[1]
      by (metis (no_types, lifting) disc_nodes disconnected_nodes_eq2 disconnected_nodes_h
          select_result_I2 set_ConsD)
  next
```

```
      case False
      have "children' = children"
        using children' children children_eq[OF False[symmetric]]
        by auto
      then show ?thesis
      proof (cases "xb = owner_document")
        case True
        then show ?thesis
          using disc_nodes disconnected_nodes_h disconnected_nodes_h'
          using "2" "4" "5" "6" "7" False ‹children' = children› assms(1) child_in_children_h
            child_parent_dual children children_h disc_nodes' get_child_nodes_ptr_in_heap
            list.set_cases list.simps(3) option.simps(1) returns_result_eq set_ConsD
          by (metis (no_types, opaque_lifting) assms)
      next
        case False
        then show ?thesis
          using "2" "4" "5" "6" "7" ‹children' = children› disc_nodes disc_nodes'
            disconnected_nodes_eq returns_result_eq
          by metis
      qed
    qed
    then show "x ∈ {}"
      by simp
  qed
}

ultimately have "heap_is_wellformed h'"
  using heap_is_wellformed_def by blast

show ?thesis
  apply(auto simp add: is_strongly_scdom_component_safe_def Let_def object_ptr_kinds_eq3)[1]
  using assms(1) assms(2) assms(3) assms(4) local.get_scdom_component_impl
    remove_child_is_strongly_dom_component_safe_step
  by blast
qed
end

interpretation i_get_scdom_component_remove_child?: l_get_scdom_component_remove_child_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_root_node get_root_node_locs get_ancestors
  get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name set_child_nodes set_child_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs remove_child remove_child_locs remove
  by(auto simp add: l_get_scdom_component_remove_child_Core_DOM_def instances)
declare l_get_scdom_component_remove_child_Core_DOM_axioms [instances]
```

adopt_node

```
locale l_get_scdom_component_adopt_node_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_adopt_node_Core_DOM +
  l_remove_child_Core_DOM +
  l_set_child_nodes_Core_DOM +
  l_set_disconnected_nodes_Core_DOM +
  l_adopt_node_wf +
  l_get_dom_component_Core_DOM +
  l_get_owner_document_wf +
  l_get_scdom_component_Core_DOM +
  l_adopt_node_wf_Core_DOM +
  l_heap_is_wellformed_Core_DOM
begin
```

```
lemma adopt_node_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ adopt_node document_ptr node_ptr →_h h'"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast document_ptr)|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast node_ptr)|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document (cast_node_ptr2object_ptr node_ptr)|_r)|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain owner_document where owner_document: "h ⊢ get_owner_document (cast_node_ptr2object_ptr node_ptr)
→_r owner_document"
    using assms(4) local.adopt_node_def by auto
  then
  obtain c where "h ⊢ get_dom_component (cast owner_document) →_r c"
    using get_dom_component_ok assms(1) assms(2) assms(3) get_owner_document_owner_document_in_heap
    by (meson document_ptr_kinds_commutes select_result_I)
  then
  have "ptr ≠ cast owner_document"
    using assms(6) assms(1) assms(2) assms(3) local.get_dom_component_ptr owner_document
    by (metis (no_types, lifting) assms(7) select_result_I2)

  have "document_ptr |∈| document_ptr_kinds h"
    using adopt_node_document_in_heap assms(1) assms(2) assms(3) assms(4) by auto
  then
  have "ptr ≠ cast document_ptr"
    using assms(5)
    using assms(1) assms(2) assms(3) local.get_dom_component_ptr get_dom_component_ok
    by (meson document_ptr_kinds_commutes returns_result_select_result)

  have "⋀parent. |h ⊢ get_parent node_ptr|_r = Some parent ⟹ parent ≠ ptr"
    by (metis assms(1) assms(2) assms(3) assms(6) is_OK_returns_result_I local.get_dom_component_ok
        local.get_dom_component_parent_inside local.get_dom_component_ptr local.get_owner_document_ptr_in_heap
        local.get_parent_ok node_ptr_kinds_commutes owner_document returns_result_select_result)

  show ?thesis
    using adopt_node_writes assms(4)
    apply(rule reads_writes_preserved2)
    apply(auto simp add: adopt_node_locs_def remove_child_locs_def set_child_nodes_locs_def set_disconnected_nodes
all_args_def)[1]
                  apply (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)
                 apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocume
owner_document select_result_I2)
                apply(drule ‹⋀parent. |h ⊢ get_parent node_ptr|_r = Some parent ⟹ parent ≠ ptr›)[1]
apply (metis element_ptr_casts_commute3 get_M_Element_preserved8 is_node_ptr_kind_none node_ptr_casts_commute3
option.case_eq_if)
               apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument
owner_document select_result_I2)
              apply (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)
             apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_p
owner_document select_result_I2)
            apply (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)
           apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_pre
owner_document select_result_I2)
          apply(drule ‹⋀parent. |h ⊢ get_parent node_ptr|_r = Some parent ⟹ parent ≠ ptr›)[1] ap-
ply (metis document_ptr_casts_commute3 get_M_Mdocument_preserved3)
         apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_prese
owner_document select_result_I2)
        apply (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)
       apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_preserv
owner_document select_result_I2)
      apply (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)
     apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_preserved
owner_document select_result_I2)
    apply(drule ‹⋀parent. |h ⊢ get_parent node_ptr|_r = Some parent ⟹ parent ≠ ptr›)[1]
```

```
      apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_preserved3
owner_document select_result_I2)
     apply (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)
    apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document› get_M_Mdocument_preserved3
owner_document select_result_I2)
    done
qed

lemma adopt_node_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ adopt_node document_ptr node_ptr →_h h'"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast document_ptr)|_r"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast node_ptr)|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  have "document_ptr |∈| document_ptr_kinds h"
    by (meson assms(1) assms(2) assms(3) assms(4) is_OK_returns_heap_I local.adopt_node_document_in_heap)
  then
  obtain sc where sc: "h ⊢ get_scdom_component (cast document_ptr) →_r sc"
    using get_scdom_component_ok
    by (meson assms(1) assms(2) assms(3) document_ptr_kinds_commutes returns_result_select_result)

  have "node_ptr |∈| node_ptr_kinds h"
    using assms(4)
    by (meson is_OK_returns_heap_I local.adopt_node_child_in_heap)
  then
  obtain child_sc where child_sc: "h ⊢ get_scdom_component (cast node_ptr) →_r child_sc"
    using get_scdom_component_ok
    by (meson assms(1) assms(2) assms(3) is_OK_returns_result_E node_ptr_kinds_commutes)

  then obtain owner_document where owner_document: "h ⊢ get_owner_document (cast node_ptr) →_r owner_document"
    by (meson ‹node_ptr |∈| node_ptr_kinds h› assms(1) assms(2) assms(3) contra_subsetD
        get_scdom_component_owner_document_same is_OK_returns_result_E
        get_scdom_component_subset_get_dom_component local.get_dom_component_ok local.get_dom_component_ptr
        node_ptr_kinds_commutes)
  then have "h ⊢ get_scdom_component (cast owner_document) →_r child_sc"
    using child_sc
    by (metis (no_types, lifting) ‹node_ptr |∈| node_ptr_kinds h› assms(1) assms(2) assms(3)
        get_scdom_component_owner_document_same get_scdom_component_ptrs_same_scope_component
        get_scdom_component_subset_get_dom_component is_OK_returns_result_E local.get_dom_component_ok
        local.get_dom_component_ptr node_ptr_kinds_commutes select_result_I2 subset_code(1))

  have "ptr ∉ set |h ⊢ get_dom_component (cast document_ptr)|_r"
    by (metis (no_types, lifting) ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
        assms(5) contra_subsetD document_ptr_kinds_commutes get_scdom_component_subset_get_dom_component
        local.get_dom_component_ok returns_result_select_result sc select_result_I2)

  moreover have "ptr ∉ set |h ⊢ get_dom_component (cast node_ptr)|_r"
    by (metis (no_types, lifting) ‹node_ptr |∈| node_ptr_kinds h› assms(1) assms(2) assms(3) assms(6)
        child_sc contra_subsetD get_scdom_component_subset_get_dom_component local.get_dom_component_ok
        node_ptr_kinds_commutes returns_result_select_result select_result_I2)

  moreover have "ptr ∉ set |h ⊢ get_scdom_component (cast |h ⊢ get_owner_document (cast_node_ptr2object_ptr
node_ptr)|_r)|_r"
    using get_scdom_component_owner_document_same get_scdom_component_ptrs_same_scope_component
    by (metis (no_types, lifting)
        ‹h ⊢ get_scdom_component (cast_document_ptr2object_ptr owner_document) →_r child_sc› assms(6) child_sc
        owner_document select_result_I2)
  have "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document (cast_node_ptr2object_ptr node_ptr)|_r)|_r"
    using get_scdom_component_owner_document_same
    by (metis (no_types, opaque_lifting)
        ‹⋀thesis. (⋀owner_document. h ⊢ get_owner_document (cast_node_ptr2object_ptr node_ptr) →_r owner_document
⟹ thesis) ⟹ thesis›
```

```
        ‹h ⊢ get_scdom_component (cast_document_ptr2object_ptr owner_document) →_r child_sc›
        ‹ptr ∉ set |h ⊢ get_scdom_component (cast_document_ptr2object_ptr |h ⊢ get_owner_document (cast_node_ptr2object_
node_ptr)|_r)|_r›
        assms(1) assms(2) assms(3) contra_subsetD document_ptr_kinds_commutes get_scdom_component_subset_get_dom_co
        is_OK_returns_result_E local.get_dom_component_ok local.get_owner_document_owner_document_in_heap
owner_document
        returns_result_eq select_result_I2)
  ultimately show ?thesis
    using assms(1) assms(2) assms(3) assms(4) adopt_node_is_component_unsafe
    by blast
qed

lemma adopt_node_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and type_wf: "type_wf h" and known_ptrs: "known_ptrs h"
  assumes "h ⊢ adopt_node document_ptr child →_h h'"
  shows "is_strongly_scdom_component_safe {cast document_ptr, cast child} {} h h'"
proof -

  obtain old_document parent_opt h2 where
    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(4)
    by(auto simp add: adopt_node_def 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)
  have "type_wf h2"
    using h2 remove_child_preserves_type_wf known_ptrs type_wf
    by (metis (no_types, lifting) assms(1) option.case_eq_if pure_returns_heap_eq return_pure)
  have "known_ptrs h2"
    using h2 remove_child_preserves_known_ptrs known_ptrs type_wf
    by (metis (no_types, lifting) assms(1) option.case_eq_if pure_returns_heap_eq return_pure)
  have "heap_is_wellformed h' ∧ known_ptrs h' ∧ type_wf h'"
  proof(cases "document_ptr = old_document")
```

```
    case True
    then show ?thesis
      using h' wellformed_h2 ‹type_wf h2› ‹known_ptrs 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_nodes_writes 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_nodes_writes 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 "known_ptrs h3"
    using known_ptrs local.known_ptrs_preserved object_ptr_kinds_h2_eq3 object_ptr_kinds_h_eq3 by blast
  then have "known_ptrs h'"
    using local.known_ptrs_preserved object_ptr_kinds_h3_eq3 by blast

  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 "a_owner_document_valid h"
      using assms(1) heap_is_wellformed_def by(simp add: 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 "a_distinct_lists h2"
      using 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: 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 "a_acyclic_heap h2"
    using wellformed_h2 by (simp add: 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 parent_child_rel_def
    by(simp)
qed
then have "a_acyclic_heap h'"
  using ‹a_acyclic_heap h2› acyclic_heap_def acyclic_subset by blast

moreover have "a_all_ptrs_in_heap h2"
  using wellformed_h2 by (simp add: heap_is_wellformed_def)
then have "a_all_ptrs_in_heap h3"
  apply(auto simp add: a_all_ptrs_in_heap_def node_ptr_kinds_eq3_h2 children_eq_h2)[1]
   apply (simp add: children_eq2_h2 object_ptr_kinds_h2_eq3 subset_code(1))
  by (metis (no_types, lifting) ‹child ∈ set disc_nodes_old_document_h2› ‹type_wf h2›
      disc_nodes_old_document_h2 disc_nodes_old_document_h3 disconnected_nodes_eq2_h2 document_ptr_kinds_eq3_h2
      in_set_remove1 local.get_disconnected_nodes_ok local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds
      returns_result_select_result select_result_I2 wellformed_h2)
then have "a_all_ptrs_in_heap h'"
  apply(auto simp add: a_all_ptrs_in_heap_def node_ptr_kinds_eq3_h3 children_eq_h3)[1]
   apply (simp add: children_eq2_h3 object_ptr_kinds_h3_eq3 subset_code(1))
  by (metis (no_types, lifting) ‹child ∈ set disc_nodes_old_document_h2› 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.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_eq3_h2 node_ptr_kinds_eq3_h3
      select_result_I2 set_ConsD subset_code(1) wellformed_h2)

moreover have "a_owner_document_valid h2"
  using wellformed_h2 by (simp add: heap_is_wellformed_def)
then have "a_owner_document_valid h'"
  apply(simp add: 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 (smt (verit) 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) node_ptr_kinds_eq3_h2 node_ptr_kinds_eq3_h3
      object_ptr_kinds_h2_eq3 object_ptr_kinds_h3_eq3 select_result_I2
      set_subset_Cons subset_code(1))

have a_distinct_lists_h2: "a_distinct_lists h2"
  using wellformed_h2 by (simp add: heap_is_wellformed_def)
then have "a_distinct_lists h'"
  apply(auto simp add: 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
  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 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 show ?thesis
  using ‹type_wf h'› ‹known_ptrs h'› ‹a_owner_document_valid h'› heap_is_wellformed_def by blast
qed
then have "heap_is_wellformed h'" and "known_ptrs h'" and "type_wf h'"
 by auto

have object_ptr_kinds_eq3: "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 adopt_node_writes assms(4)])
 unfolding adopt_node_locs_def
 using set_disconnected_nodes_pointers_preserved set_child_nodes_pointers_preserved
   remove_child_pointers_preserved
 by (auto simp add: reflp_def transp_def split: if_splits)
show ?thesis
 apply(auto simp add: is_strongly_scdom_component_safe_def Let_def object_ptr_kinds_eq3 )[1]
 using adopt_node_is_strongly_dom_component_safe_step get_scdom_component_impl assms by blast
```

```
qed
end

interpretation i_get_scdom_component_adopt_node?: l_get_scdom_component_adopt_node_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe 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 adopt_node_locs get_child_nodes get_child_nodes_locs
  set_child_nodes set_child_nodes_locs remove to_tree_order get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  by(auto simp add: l_get_scdom_component_adopt_node_Core_DOM_def instances)
declare l_get_scdom_component_adopt_node_Core_DOM_axioms [instances]
```

create_element

```
locale l_get_scdom_component_create_element_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_dom_component_create_element_Core_DOM +
  l_create_element_wf_Core_DOM +
  l_get_scdom_component_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_owner_document_Core_DOM
begin

lemma create_element_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_element document_ptr tag →_h h'"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast document_ptr)|_r"
  assumes "ptr ≠ cast |h ⊢ create_element document_ptr tag|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain new_element_ptr h2 h3 disc_nodes 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 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_element_ptr # disc_nodes) →_h h'"
    using assms(4)
    by(auto simp add: create_element_def elim!: bind_returns_heap_E bind_returns_heap_E2[rotated,
          OF get_disconnected_nodes_pure, rotated])

  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)

  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 "heap_is_wellformed h'"
    using assms(4)
    using assms(1) assms(2) assms(3) local.create_element_preserves_wellformedness(1) by blast
  have "type_wf h'"
    using assms(1) assms(2) assms(3) assms(4) local.create_element_preserves_wellformedness(2) by blast
  have "known_ptrs h'"
    using assms(1) assms(2) assms(3) assms(4) local.create_element_preserves_wellformedness(3) by blast

  have "document_ptr |∈| document_ptr_kinds h"
    by (meson assms(4) is_OK_returns_heap_I local.create_element_document_in_heap)
  then
  obtain sc where sc: "h ⊢ get_scdom_component (cast document_ptr) →r sc"
    using get_scdom_component_ok
    by (meson assms(1) assms(2) assms(3) document_ptr_kinds_commutes returns_result_select_result)

  have "document_ptr |∈| document_ptr_kinds h'"
    using ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
    using document_ptr_kinds_eq_h2 document_ptr_kinds_eq_h3 by blast
  then
  obtain sc' where sc': "h' ⊢ get_scdom_component (cast document_ptr) →r sc'"
    using get_scdom_component_ok
    by (meson ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹type_wf h'› document_ptr_kinds_commutes
        returns_result_select_result)

  obtain c where c: "h ⊢ get_dom_component (cast document_ptr) →r c"
    by (meson ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
        document_ptr_kinds_commutes is_OK_returns_result_E local.get_dom_component_ok)

  have "set c ⊆ set sc"
    using assms(1) assms(2) assms(3) c get_scdom_component_subset_get_dom_component sc by blast

  have "ptr ∉ set c"
    using ‹set c ⊆ set sc› assms(5) sc
    by auto
  then
  show ?thesis
    using create_element_is_weakly_dom_component_safe
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(4) assms(6) c
        local.create_element_is_weakly_dom_component_safe_step select_result_I2)
qed

lemma create_element_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_element document_ptr tag →r result"
  assumes "h ⊢ create_element document_ptr tag →h h'"
  shows "is_strongly_scdom_component_safe {cast document_ptr} {cast result} h 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(5)
    by(auto simp add: create_element_def returns_result_heap_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF 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 local.get_disconnected_nodes_pure pure_returns_heap_
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  then have "result = new_element_ptr"
    using assms(4) by auto


  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)

  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_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
have "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
    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' children. ptr' ≠ cast new_element_ptr
          ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
  using 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 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 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 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 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 "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 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 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(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 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 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 <cast_element_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 "parent_child_rel h = parent_child_rel h'"
  proof -
    have "parent_child_rel h = parent_child_rel h2"
    proof(auto simp add: 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: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
    also have "... = parent_child_rel h'"
      by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
    finally show ?thesis
      by simp
  qed


  have "document_ptr |∈| document_ptr_kinds h'"
    by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
        document_ptr_kinds_eq_h2 document_ptr_kinds_eq_h3)

  have "known_ptr (cast document_ptr)"
    using ‹document_ptr |∈| document_ptr_kinds h› assms(3) document_ptr_kinds_commutes
      local.known_ptrs_known_ptr by blast
  have "h ⊢ get_owner_document (cast document_ptr) →_r document_ptr"
    using ‹known_ptr (cast document_ptr)› ‹document_ptr |∈| document_ptr_kinds h›
    apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def)[1]
    apply(split invoke_splits, rule conjI)+
    by(auto simp add: known_ptr_impl known_ptr_defs CharacterDataClass.known_ptr_defs
        ElementClass.known_ptr_defs NodeClass.known_ptr_defs a_get_owner_document_document_ptr_def intro!:
bind_pure_returns_result_I split: option.splits)
  have "h' ⊢ get_owner_document (cast document_ptr) →_r document_ptr"
    using ‹known_ptr (cast document_ptr)› ‹document_ptr |∈| document_ptr_kinds h'›
    apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def)[1]
    apply(split invoke_splits, rule conjI)+
    by(auto simp add: known_ptr_impl known_ptr_defs CharacterDataClass.known_ptr_defs
        ElementClass.known_ptr_defs NodeClass.known_ptr_defs a_get_owner_document_document_ptr_def intro!:
bind_pure_returns_result_I split: option.splits)

  obtain to where to: "h ⊢ to_tree_order (cast document_ptr) →_r to"
    by (meson ‹h ⊢ get_owner_document (cast_document_ptr2object_ptr document_ptr) →_r document_ptr› assms(1)
        assms(2) assms(3) is_OK_returns_result_E is_OK_returns_result_I local.get_owner_document_ptr_in_heap
        local.to_tree_order_ok)
  obtain to' where to': "h' ⊢ to_tree_order (cast document_ptr) →_r to'"
    by (metis ‹document_ptr |∈| document_ptr_kinds h› ‹known_ptrs h'› ‹type_wf h'› assms(1) assms(2)
        assms(3) assms(5) document_ptr_kinds_commutes document_ptr_kinds_eq_h document_ptr_kinds_eq_h2
        document_ptr_kinds_eq_h3 is_OK_returns_result_E local.create_element_preserves_wellformedness(1)
        local.to_tree_order_ok)
  have "set to = set to'"
  proof safe
    fix x
    assume "x ∈ set to"
    show "x ∈ set to'"
      using to to'
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹known_ptrs h'› ‹type_wf h'› ‹x ∈ set to› assms(1) assms(2) assms(3) assms(5)
          local.create_element_preserves_wellformedness(1))
  next
    fix x
    assume "x ∈ set to'"
```

```
    show "x ∈ set to"
      using to to'
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹known_ptrs h'› ‹type_wf h'› ‹x ∈ set to'› assms(1) assms(2) assms(3) assms(5)
          local.create_element_preserves_wellformedness(1))
  qed

  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
  obtain disc_nodes_h' where disc_nodes_h': "h' ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h'"
    and "cast new_element_ptr ∈ set disc_nodes_h'"
    and "disc_nodes_h' =  cast new_element_ptr # disc_nodes_h3"
    by (simp add: ‹h' ⊢ get_disconnected_nodes document_ptr →r cast_element_ptr2node_ptr new_element_ptr #
disc_nodes_h3›)

  have "⋀disc_ptr to to'. disc_ptr ∈ set disc_nodes_h3 ⟹ h ⊢ to_tree_order (cast disc_ptr) →r to ⟹
h' ⊢ to_tree_order (cast disc_ptr) →r to' ⟹ set to = set to'"
  proof safe
    fix disc_ptr to to' x
    assume "disc_ptr ∈ set disc_nodes_h3"
    assume "h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to"
    assume "h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to'"
    assume "x ∈ set to"
    show "x ∈ set to'"
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to›
          ‹h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to'› ‹known_ptrs h'› ‹type_wf h'› ‹x ∈
set to›
          assms(1) assms(2) assms(3) assms(5) local.create_element_preserves_wellformedness(1))
  next
    fix disc_ptr to to' x
    assume "disc_ptr ∈ set disc_nodes_h3"
    assume "h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to"
    assume "h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to'"
    assume "x ∈ set to'"
    show "x ∈ set to"
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to›
          ‹h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →r to'› ‹known_ptrs h'› ‹type_wf h'› ‹x ∈
set to'›
          assms(1) assms(2) assms(3) assms(5) local.create_element_preserves_wellformedness(1))
  qed

  have "heap_is_wellformed h'"
    using assms(1) assms(2) assms(3) assms(5) local.create_element_preserves_wellformedness(1)
    by blast

  have "cast new_element_ptr |∈| object_ptr_kinds h'"
    using ‹cast_element_ptr2node_ptr new_element_ptr ∈ set disc_nodes_h'› ‹heap_is_wellformed h'› disc_nodes_h'
      local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_commutes by blast
  then
  have "new_element_ptr |∈| element_ptr_kinds h'"
    by simp

  have "⋀node_ptr. node_ptr ∈ set disc_nodes_h3 ⟹ node_ptr |∈| node_ptr_kinds h'"
    by (meson ‹heap_is_wellformed h'› h' local.heap_is_wellformed_disc_nodes_in_heap
        local.set_disconnected_nodes_get_disconnected_nodes set_subset_Cons subset_code(1))

  have "h ⊢ ok (map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h3)"
    using assms(1) assms(2) assms(3) to_tree_order_ok
    apply(auto intro!: map_M_ok_I)[1]
    using disc_nodes_document_ptr_h local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_commutes
    by blast
```

```
  then
  obtain disc_tree_orders where disc_tree_orders:
    "h ⊢ map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h3 →_r disc_tree_orders"
    by auto

  have "h' ⊢ ok (map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h')"
    apply(auto intro!: map_M_ok_I)[1]
    apply(simp add: ‹disc_nodes_h' =  cast new_element_ptr # disc_nodes_h3›)
    using ‹⋀node_ptr. node_ptr ∈ set disc_nodes_h3 ⟹ node_ptr |∈| node_ptr_kinds h'›
      ‹cast_element_ptr2node_ptr new_element_ptr ∈ set disc_nodes_h'› ‹heap_is_wellformed h'› ‹known_ptrs h'›
      ‹type_wf h'› disc_nodes_h' local.heap_is_wellformed_disc_nodes_in_heap local.to_tree_order_ok
      node_ptr_kinds_commutes by blast
  then
  obtain disc_tree_orders' where disc_tree_orders':
    "h' ⊢ map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h' →_r disc_tree_orders'"
    by auto

  have "h' ⊢ get_child_nodes (cast new_element_ptr) →_r []"
    using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []› children_eq_h2
      children_eq_h3 by auto

  obtain new_tree_order where new_tree_order:
    "h' ⊢ to_tree_order (cast new_element_ptr) →_r new_tree_order" and
    "new_tree_order ∈ set disc_tree_orders'"
    using map_M_pure_E[OF disc_tree_orders' ‹cast new_element_ptr ∈ set disc_nodes_h'›]
    by auto
  then have "new_tree_order = [cast new_element_ptr]"
    using ‹h' ⊢ get_child_nodes (cast new_element_ptr) →_r []›
    by(auto simp add: to_tree_order_def
        dest!: bind_returns_result_E3[rotated, OF ‹h' ⊢ get_child_nodes (cast new_element_ptr) →_r []›,
rotated])

  obtain foo where foo: "h' ⊢ map_M (to_tree_order ∘ cast_node_ptr2object_ptr)
(cast_element_ptr2node_ptr new_element_ptr # disc_nodes_h3) →_r [cast_element_ptr2object_ptr new_element_ptr] #
foo"
    apply(auto intro!: bind_pure_returns_result_I map_M_pure_I)[1]
    using ‹new_tree_order = [cast_element_ptr2object_ptr new_element_ptr]› new_tree_order apply auto[1]
    by (smt (verit) ‹disc_nodes_h' = cast_element_ptr2node_ptr new_element_ptr # disc_nodes_h3›
        bind_pure_returns_result_I bind_returns_result_E2 comp_apply disc_tree_orders'
        local.to_tree_order_pure map_M.simps(2) map_M_pure_I return_returns_result returns_result_eq)
  then have "set (concat foo) = set (concat disc_tree_orders)"
    apply(auto elim!: bind_returns_result_E2 intro!: map_M_pure_I)[1]
     apply (smt (verit) ‹⋀to' toa disc_ptr. ⟦disc_ptr ∈ set disc_nodes_h3; h ⊢ to_tree_order
            (cast_node_ptr2object_ptr disc_ptr) →_r toa; h' ⊢ to_tree_order
            (cast_node_ptr2object_ptr disc_ptr) →_r to'⟧ ⟹ set toa = set to'›
            comp_eq_dest_lhs disc_tree_orders local.to_tree_order_pure map_M_pure_E map_M_pure_E2)
    by (smt (verit) ‹⋀to' toa disc_ptr. ⟦disc_ptr ∈ set disc_nodes_h3; h ⊢ to_tree_order
        (cast_node_ptr2object_ptr disc_ptr) →_r toa; h' ⊢ to_tree_order
        (cast_node_ptr2object_ptr disc_ptr) →_r to'⟧ ⟹ set toa = set to'›
        comp_eq_dest_lhs disc_tree_orders local.to_tree_order_pure map_M_pure_E map_M_pure_E2)

  have "disc_tree_orders' = [cast_element_ptr2object_ptr new_element_ptr] # foo"
    using foo disc_tree_orders'
    by (simp add: ‹disc_nodes_h' = cast_element_ptr2node_ptr new_element_ptr # disc_nodes_h3› returns_result_eq)

  have "set (concat disc_tree_orders') = {cast new_element_ptr} ∪ set (concat disc_tree_orders)"
    apply(auto simp add: ‹disc_tree_orders' = [cast_element_ptr2object_ptr new_element_ptr] # foo›)[1]
    using ‹set (concat foo) = set (concat disc_tree_orders)› by auto

  have "h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr) →_r to' @ concat disc_tree_orders
    using ‹h' ⊢ get_owner_document (cast document_ptr) →_r document_ptr› disc_nodes_h' to' disc_tree_orders'
    by(auto simp add: a_get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  then
```

```
  have "set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to' ∪ set (concat disc_tree_orders')"
    by auto
  have "h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr) →_r to @ concat disc_tree_orders"
    using ‹h ⊢ get_owner_document (cast document_ptr) →_r document_ptr› disc_nodes_document_ptr_h
      to disc_tree_orders
    by(auto simp add: a_get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  then
  have "set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r = set to ∪ set (concat
disc_tree_orders)"
    by auto

  have "{cast new_element_ptr} ∪ set |h ⊢ local.a_get_scdom_component (cast document_ptr)|_r =
set |h' ⊢ local.a_get_scdom_component (cast document_ptr)|_r"
  proof(safe)
    show "cast_element_ptr2object_ptr new_element_ptr
        ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
      using ‹h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr) →_r to' @ concat disc_tree_o
      apply(auto simp add: a_get_scdom_component_def)[1]
      by (meson ‹⋀thesis. (⋀new_tree_order. ⟦h' ⊢ to_tree_order (cast_element_ptr2object_ptr new_element_ptr)
→_r new_tree_order;
new_tree_order ∈ set disc_tree_orders'⟧ ⟹ thesis) ⟹ thesis› local.to_tree_order_ptr_in_result)
  next
    fix x
    assume " x ∈ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
    then
    show "x ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
      using ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to ∪set (concat disc_tree_orders)›
      using ‹set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to' ∪ set (concat disc_tree_orders')›
      using ‹set to = set to'›
      using ‹set (concat disc_tree_orders') = {cast new_element_ptr} ∪ set (concat disc_tree_orders)›
      by(auto)
  next
    fix x
    assume " x ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
    assume "x ∉ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
    show "x = cast_element_ptr2object_ptr new_element_ptr"
      using ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to ∪ set (concat disc_tree_orders)›
      using ‹set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to' ∪ set (concat disc_tree_orders')›
      using ‹set to = set to'›
      using ‹set (concat disc_tree_orders') = {cast new_element_ptr} ∪ set (concat disc_tree_orders)›
      using ‹x ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r›
        ‹x ∉ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r›
      by auto
  qed


  have "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_element_ptr|}"
    using object_ptr_kinds_eq_h object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 by auto
  then
  show ?thesis
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def)[1]
      apply(rule bexI[where x="cast_document_ptr2object_ptr document_ptr"])
    using ‹{cast_element_ptr2object_ptr new_element_ptr} ∪
set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r›
      apply auto[2]
    using ‹set to = set to'› ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r
=
```

```
set to ∪ set (concat disc_tree_orders)› local.to_tree_order_ptr_in_result to'
        apply auto[1]
    using ‹document_ptr |∈| document_ptr_kinds h›
       apply blast
      apply(rule bexI[where x="cast_document_ptr2object_ptr document_ptr"])
    using ‹result = new_element_ptr›
      ‹{cast_element_ptr2object_ptr new_element_ptr} ∪ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr
document_ptr)|_r =
set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r› apply auto[1]
      apply(auto)[1]
    using ‹set to = set to'› ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r
=
set to ∪ set (concat disc_tree_orders)› local.to_tree_order_ptr_in_result to' apply auto[1]
      apply (simp add: ‹document_ptr |∈| document_ptr_kinds h›)
    using ‹⋀thesis. (⋀new_element_ptr h2 h3 disc_nodes_h3. ⟦h ⊢ new_element →_r new_element_ptr;
h ⊢ new_element →_h h2; h2 ⊢ set_tag_name new_element_ptr tag →_h h3;
h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes_h3;
h3 ⊢ set_disconnected_nodes document_ptr (cast_element_ptr2node_ptr new_element_ptr # disc_nodes_h3) →_h h'⟧
⟹ thesis) ⟹ thesis›
      new_element_ptr new_element_ptr_not_in_heap
     apply auto[1]
    using create_element_is_strongly_scdom_component_safe_step
    by (smt (verit, best) ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹cast_element_ptr2object_ptr new_element_ptr ∉ set |h ⊢
        object_ptr_kinds_M|_r› ‹h ⊢ create_element document_ptr tag →_r new_element_ptr› assms(1)
        assms(2) assms(3) assms(5) local.get_scdom_component_impl select_result_I2)
qed
end

interpretation i_get_scdom_component_remove_child?: l_get_scdom_component_remove_child_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs get_scdom_component
  is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_l
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name set_child_nodes set_child_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs remove_child remove_child_locs remove
  by(auto simp add: l_get_scdom_component_remove_child_Core_DOM_def instances)
declare l_get_scdom_component_remove_child_Core_DOM_axioms [instances]
```

create_character_data

```
locale l_get_scdom_component_create_character_data_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_dom_component_create_character_data_Core_DOM +
  l_get_scdom_component_Core_DOM +
  l_create_character_data_wf_Core_DOM +
  l_get_scdom_component_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_owner_document_Core_DOM
begin

lemma create_character_data_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_character_data document_ptr text →_h h'"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast document_ptr)|_r"
  assumes "ptr ≠ cast |h ⊢ create_character_data document_ptr text|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  have "document_ptr |∈| document_ptr_kinds h"
    by (meson assms(4) is_OK_returns_heap_I local.create_character_data_document_in_heap)
  then
  obtain sc where sc: "h ⊢ get_scdom_component (cast document_ptr) →_r sc"
    using get_scdom_component_ok
```

```
    by (meson assms(1) assms(2) assms(3) document_ptr_kinds_commutes returns_result_select_result)

  obtain c where c: "h ⊢ get_dom_component (cast document_ptr) →r c"
    by (meson ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
        document_ptr_kinds_commutes is_OK_returns_result_E local.get_dom_component_ok)

  have "set c ⊆ set sc"
    using assms(1) assms(2) assms(3) c get_scdom_component_subset_get_dom_component sc by blast

  have "ptr ∉ set c"
    using ‹set c ⊆ set sc› assms(5) sc
    by auto
  then
  show ?thesis
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(4) assms(6) c
        local.create_character_data_is_weakly_dom_component_safe_step select_result_I2)
qed

lemma create_character_data_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_character_data document_ptr text →r result"
  assumes "h ⊢ create_character_data document_ptr text →h h'"
  shows "is_strongly_scdom_component_safe {cast document_ptr} {cast result} h 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(5)
    by(auto simp add: create_character_data_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated] )
  then have "h ⊢ create_character_data document_ptr text →r new_character_data_ptr"
    apply(auto simp add: 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.get_disconnected_nodes_pure
        pure_returns_heap_eq)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  then have "result = new_character_data_ptr"
    using assms(4) by auto

  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 set_val_writes h3])
    using 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 "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      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' children. ptr' ≠ cast new_character_data_ptr
                ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using 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 set_val_writes h3])
    using 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 "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      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' children. ptr' ≠ cast new_character_data_ptr
              ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using 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 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 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_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 get_disconnected_nodes_reads 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 "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 set_val_writes h3]
    using  set_val_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 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(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 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 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 "parent_child_rel h = parent_child_rel h'"
  proof -
    have "parent_child_rel h = parent_child_rel h2"
    proof(auto simp add: 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: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
    also have "... = parent_child_rel h'"
      by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
    finally show ?thesis
      by simp
  qed

  have "known_ptr (cast new_character_data_ptr)"
    using ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr›
      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
  have "known_ptrs h'"
    using known_ptrs_preserved object_ptr_kinds_eq_h3 by blast




  have "document_ptr |∈| document_ptr_kinds h'"
    by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
        document_ptr_kinds_eq_h2 document_ptr_kinds_eq_h3)

  have "known_ptr (cast document_ptr)"
    using ‹document_ptr |∈| document_ptr_kinds h› assms(3) document_ptr_kinds_commutes
      local.known_ptrs_known_ptr by blast
  have "h ⊢ get_owner_document (cast document_ptr) →_r document_ptr"
    using ‹known_ptr (cast document_ptr)› ‹document_ptr |∈| document_ptr_kinds h›
    apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def)[1]
    apply(split invoke_splits, rule conjI)+
    by(auto simp add: known_ptr_impl known_ptr_defs CharacterDataClass.known_ptr_defs
        ElementClass.known_ptr_defs NodeClass.known_ptr_defs a_get_owner_document_document_ptr_def
        intro!: bind_pure_returns_result_I split: option.splits)
  have "h' ⊢ get_owner_document (cast document_ptr) →_r document_ptr"
    using ‹known_ptr (cast document_ptr)› ‹document_ptr |∈| document_ptr_kinds h'›
```

```
    apply(auto simp add: get_owner_document_def a_get_owner_document_tups_def)[1]
    apply(split invoke_splits, rule conjI)+
    by(auto simp add: known_ptr_impl known_ptr_defs CharacterDataClass.known_ptr_defs
        ElementClass.known_ptr_defs NodeClass.known_ptr_defs a_get_owner_document_document_ptr_def
        intro!: bind_pure_returns_result_I split: option.splits)

  obtain to where to: "h ⊢ to_tree_order (cast document_ptr) →_r to"
    by (meson ‹h ⊢ get_owner_document (cast_document_ptr2object_ptr document_ptr) →_r document_ptr›
        assms(1) assms(2) assms(3) is_OK_returns_result_E is_OK_returns_result_I
        local.get_owner_document_ptr_in_heap local.to_tree_order_ok)
  obtain to' where to': "h' ⊢ to_tree_order (cast document_ptr) →_r to'"
    by (metis ‹document_ptr |∈| document_ptr_kinds h› ‹known_ptrs h'› ‹type_wf h'› assms(1) assms(2)
        assms(3) assms(5) document_ptr_kinds_commutes document_ptr_kinds_eq_h document_ptr_kinds_eq_h2
        document_ptr_kinds_eq_h3 is_OK_returns_result_E local.create_character_data_preserves_wellformedness(1)
        local.to_tree_order_ok)
  have "set to = set to'"
  proof safe
    fix x
    assume "x ∈ set to"
    show "x ∈ set to'"
      using to to'
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹known_ptrs h'› ‹type_wf h'› ‹x ∈ set to› assms(1) assms(2) assms(3) assms(5)
          local.create_character_data_preserves_wellformedness(1))
  next
    fix x
    assume "x ∈ set to'"
    show "x ∈ set to"
      using to to'
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹known_ptrs h'› ‹type_wf h'› ‹x ∈ set to'› assms(1) assms(2) assms(3) assms(5)
          local.create_character_data_preserves_wellformedness(1))
  qed

  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
  obtain disc_nodes_h' where disc_nodes_h': "h' ⊢ get_disconnected_nodes document_ptr →_r disc_nodes_h'"
    and "cast new_character_data_ptr ∈ set disc_nodes_h'"
    and "disc_nodes_h' =  cast new_character_data_ptr # disc_nodes_h3"
    by (simp add: ‹h' ⊢ get_disconnected_nodes document_ptr →_r cast new_character_data_ptr # disc_nodes_h3›)

  have "⋀disc_ptr to to'. disc_ptr ∈ set disc_nodes_h3 ⟹ h ⊢ to_tree_order (cast disc_ptr) →_r to ⟹
h' ⊢ to_tree_order (cast disc_ptr) →_r to' ⟹ set to = set to'"
  proof safe
    fix disc_ptr to to' x
    assume "disc_ptr ∈ set disc_nodes_h3"
    assume "h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to"
    assume "h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to'"
    assume "x ∈ set to"
    show "x ∈ set to'"
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
      by (metis ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to›
          ‹h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to'› ‹known_ptrs h'› ‹type_wf h'› ‹x ∈
set to›
          assms(1) assms(2) assms(3) assms(5) local.create_character_data_preserves_wellformedness(1))
  next
    fix disc_ptr to to' x
    assume "disc_ptr ∈ set disc_nodes_h3"
    assume "h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to"
    assume "h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to'"
    assume "x ∈ set to'"
    show "x ∈ set to"
      using to_tree_order_parent_child_rel ‹parent_child_rel h = parent_child_rel h'›
```

```
      by (metis ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to›
          ‹h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to'› ‹known_ptrs h'› ‹type_wf h'› ‹x ∈
set to'›
          assms(1) assms(2) assms(3) assms(5) local.create_character_data_preserves_wellformedness(1))
  qed

  have "heap_is_wellformed h'"
    using assms(1) assms(2) assms(3) assms(5) local.create_character_data_preserves_wellformedness(1)
    by blast

  have "cast new_character_data_ptr |∈| object_ptr_kinds h'"
    using ‹cast new_character_data_ptr ∈ set disc_nodes_h'› ‹heap_is_wellformed h'› disc_nodes_h'
      local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_commutes by blast
  then
  have "new_character_data_ptr |∈| character_data_ptr_kinds h'"
    by simp

  have "⋀node_ptr. node_ptr ∈ set disc_nodes_h3 ⟹ node_ptr |∈| node_ptr_kinds h'"
    by (meson ‹heap_is_wellformed h'› h' local.heap_is_wellformed_disc_nodes_in_heap
        local.set_disconnected_nodes_get_disconnected_nodes set_subset_Cons subset_code(1))

  have "h ⊢ ok (map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h3)"
    using assms(1) assms(2) assms(3) to_tree_order_ok
    apply(auto intro!: map_M_ok_I)[1]
    using disc_nodes_document_ptr_h local.heap_is_wellformed_disc_nodes_in_heap node_ptr_kinds_commutes
    by blast
  then
  obtain disc_tree_orders where disc_tree_orders:
    "h ⊢ map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h3 →_r disc_tree_orders"
    by auto

  have "h' ⊢ ok (map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h')"
    apply(auto intro!: map_M_ok_I)[1]
    apply(simp add: ‹disc_nodes_h' =  cast new_character_data_ptr # disc_nodes_h3›)
    using ‹⋀node_ptr. node_ptr ∈ set disc_nodes_h3 ⟹ node_ptr |∈| node_ptr_kinds h'›
      ‹cast new_character_data_ptr ∈ set disc_nodes_h'› ‹heap_is_wellformed h'› ‹known_ptrs h'›
      ‹type_wf h'› disc_nodes_h' local.heap_is_wellformed_disc_nodes_in_heap local.to_tree_order_ok
      node_ptr_kinds_commutes by blast
  then
  obtain disc_tree_orders' where disc_tree_orders':
    "h' ⊢ map_M (to_tree_order ∘ cast_node_ptr2object_ptr) disc_nodes_h' →_r disc_tree_orders'"
    by auto

  have "h' ⊢ get_child_nodes (cast new_character_data_ptr) →_r []"
    using ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →_r []› children_eq_h2 children_eq_h3 by
auto

  obtain new_tree_order where new_tree_order:
    "h' ⊢ to_tree_order (cast new_character_data_ptr) →_r new_tree_order" and
    "new_tree_order ∈ set disc_tree_orders'"
    using map_M_pure_E[OF disc_tree_orders' ‹cast new_character_data_ptr ∈ set disc_nodes_h'›]
    by auto
  then have "new_tree_order = [cast new_character_data_ptr]"
    using ‹h' ⊢ get_child_nodes (cast new_character_data_ptr) →_r []›
    by(auto simp add: to_tree_order_def
        dest!: bind_returns_result_E3[rotated, OF ‹h' ⊢ get_child_nodes (cast new_character_data_ptr) →_r
[]›, rotated])

  obtain foo where foo: "h' ⊢ map_M (to_tree_order ∘ cast_node_ptr2object_ptr)
(cast new_character_data_ptr # disc_nodes_h3) →_r [cast new_character_data_ptr] # foo"
    apply(auto intro!: bind_pure_returns_result_I map_M_pure_I)[1]
    using ‹new_tree_order = [cast new_character_data_ptr]› new_tree_order apply auto[1]
    using ‹disc_nodes_h' = cast new_character_data_ptr # disc_nodes_h3› bind_pure_returns_result_I
```

```
      bind_returns_result_E2 comp_apply disc_tree_orders' local.to_tree_order_pure map_M.simps(2)
      map_M_pure_I return_returns_result returns_result_eq
    apply simp
    by (smt (verit) ‹disc_nodes_h' = cast new_character_data_ptr # disc_nodes_h3› bind_pure_returns_result_I
        bind_returns_result_E2 comp_apply disc_tree_orders' local.to_tree_order_pure map_M.simps(2) map_M_pure_I
        return_returns_result returns_result_eq)
  then have "set (concat foo) = set (concat disc_tree_orders)"
    apply(auto elim!: bind_returns_result_E2 intro!: map_M_pure_I)[1]
     apply (smt (verit) ‹⋀to' toa disc_ptr. ⟦disc_ptr ∈ set disc_nodes_h3;
h ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r toa; h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr)
→_r to'⟧ ⟹
set toa = set to'› comp_apply disc_tree_orders local.to_tree_order_pure map_M_pure_E map_M_pure_E2)
    by (smt (verit) ‹⋀to' toa disc_ptr. ⟦disc_ptr ∈ set disc_nodes_h3; h ⊢ to_tree_order (cast_node_ptr2object_ptr
disc_ptr) →_r toa;
h' ⊢ to_tree_order (cast_node_ptr2object_ptr disc_ptr) →_r to'⟧ ⟹ set toa = set to'› comp_apply disc_tree_orders
      local.to_tree_order_pure map_M_pure_E map_M_pure_E2)

  have "disc_tree_orders' = [cast new_character_data_ptr] # foo"
    using foo disc_tree_orders'
    by (simp add: ‹disc_nodes_h' = cast new_character_data_ptr # disc_nodes_h3› returns_result_eq)

  have "set (concat disc_tree_orders') = {cast new_character_data_ptr} ∪ set (concat disc_tree_orders)"
    apply(auto simp add: ‹disc_tree_orders' = [cast new_character_data_ptr] # foo›)[1]
    using ‹set (concat foo) = set (concat disc_tree_orders)› by auto

  have "h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr) →_r to' @ concat disc_tree_orders
    using ‹h' ⊢ get_owner_document (cast document_ptr) →_r document_ptr› disc_nodes_h' to' disc_tree_orders'
    by(auto simp add: a_get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  then
  have "set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r = set to' ∪ set
(concat disc_tree_orders')"
    by auto
  have "h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr) →_r to @ concat disc_tree_orders"
    using ‹h ⊢ get_owner_document (cast document_ptr) →_r document_ptr› disc_nodes_document_ptr_h to disc_tree_ord
    by(auto simp add: a_get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  then
  have "set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r = set to ∪ set (concat
disc_tree_orders)"
    by auto

  have "{cast new_character_data_ptr} ∪ set |h ⊢ local.a_get_scdom_component (cast document_ptr)|_r =
set |h' ⊢ local.a_get_scdom_component (cast document_ptr)|_r"
  proof(safe)
    show "cast new_character_data_ptr
        ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
      using ‹h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr) →_r to' @ concat disc_tree_o
      apply(auto simp add: a_get_scdom_component_def)[1]
      by (meson ‹⋀thesis. (⋀new_tree_order. ⟦h' ⊢ to_tree_order (cast new_character_data_ptr) →_r new_tree_order;
new_tree_order ∈ set disc_tree_orders'⟧ ⟹ thesis) ⟹ thesis› local.to_tree_order_ptr_in_result)
  next
    fix x
    assume " x ∈ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
    then
    show "x ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
      using ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to ∪ set (concat disc_tree_orders)›
      using ‹set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to' ∪ set (concat disc_tree_orders')›
      using ‹set to = set to'›
      using ‹set (concat disc_tree_orders') = {cast new_character_data_ptr} ∪ set (concat disc_tree_orders)›
      by(auto)
  next
    fix x
```

```
    assume " x ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
    assume "x ∉ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r"
    show "x = cast new_character_data_ptr"
      using ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to ∪ set (concat disc_tree_orders)›
      using ‹set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set to' ∪ set (concat disc_tree_orders')›
      using ‹set to = set to'›
      using ‹set (concat disc_tree_orders') = {cast new_character_data_ptr} ∪ set (concat disc_tree_orders)›
      using ‹x ∈ set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r›
        ‹x ∉ set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r›
      by auto
  qed


  have "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    using object_ptr_kinds_eq_h object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 by auto
  then
  show ?thesis
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def)[1]
       apply(rule bexI[where x="cast_document_ptr2object_ptr document_ptr"])
    using ‹{cast_character_data_ptr2object_ptr new_character_data_ptr} ∪ set |h ⊢ local.a_get_scdom_component
(cast_document_ptr2object_ptr document_ptr)|_r = set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr
document_ptr)|_r›
        apply auto[2]
    using ‹set to = set to'› ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r
=
set to ∪ set (concat disc_tree_orders)› local.to_tree_order_ptr_in_result to'
        apply auto[1]
    using ‹document_ptr |∈| document_ptr_kinds h›
       apply blast
      apply(rule bexI[where x="cast_document_ptr2object_ptr document_ptr"])
    using ‹result = new_character_data_ptr› ‹{cast_character_data_ptr2object_ptr new_character_data_ptr} ∪
set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r =
set |h' ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r›
       apply auto[1]
      apply(auto)[1]
    using ‹set to = set to'› ‹set |h ⊢ local.a_get_scdom_component (cast_document_ptr2object_ptr document_ptr)|_r
=
set to ∪ set (concat disc_tree_orders)› local.to_tree_order_ptr_in_result to' apply auto[1]
      apply (simp add: ‹document_ptr |∈| document_ptr_kinds h›)
    using ‹⋀thesis. (⋀new_character_data_ptr h2 h3 disc_nodes_h3. ⟦h ⊢ new_character_data →_r new_character_data_
h ⊢ new_character_data →_h h2; h2 ⊢ set_val new_character_data_ptr text →_h h3;
h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes_h3;
h3 ⊢ set_disconnected_nodes document_ptr (cast_character_data_ptr2node_ptr new_character_data_ptr # disc_nodes_h3)
→_h h'⟧ ⟹ thesis) ⟹ thesis›
      new_character_data_ptr new_character_data_ptr_not_in_heap
     apply auto[1]
    using create_character_data_is_strongly_dom_component_safe_step
    by (smt (verit) ObjectMonad.ptr_kinds_ptr_kinds_M ‹cast_character_data_ptr2object_ptr new_character_data_ptr
∉ set |h ⊢ object_ptr_kinds_M|_r›
       ‹result = new_character_data_ptr› assms(1) assms(2) assms(3) assms(4) assms(5) local.get_scdom_component_in
select_result_I2)
qed
end

interpretation i_get_scdom_component_create_character_data?: l_get_scdom_component_create_character_data_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs get_scdom_component
  is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name set_val set_val_locs get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs create_character_data
```

```
  by(auto simp add: l_get_scdom_component_create_character_data_Core_DOM_def instances)
declare l_get_scdom_component_create_character_data_Core_DOM_axioms [instances]
```

create_document

```
lemma create_document_not_strongly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and new_document_ptr where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ create_document →_r new_document_ptr →_h h'" and
    "¬ is_strongly_scdom_component_safe {} {cast new_document_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "create_document"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?document_ptr = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1"])
    by code_simp+
qed

locale l_get_scdom_component_create_document_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_dom_component_create_document_Core_DOM +
  l_get_scdom_component_Core_DOM
begin

lemma create_document_is_weakly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_document →_r result"
  assumes "h ⊢ create_document →_h h'"
  shows "is_weakly_scdom_component_safe {} {cast result} h h'"
proof -
  have "object_ptr_kinds h' = {|cast result|} |∪| object_ptr_kinds h"
    using assms(4) assms(5) local.create_document_def new_document_new_ptr by blast
  have "result |∉| document_ptr_kinds h"
    using assms(4) assms(5) local.create_document_def new_document_ptr_not_in_heap by auto
  show ?thesis
    using assms
    apply(auto simp add: is_weakly_scdom_component_safe_def Let_def)[1]
    using ‹object_ptr_kinds h' = {|cast result|} |∪| object_ptr_kinds h› apply(auto)[1]
      apply (simp add: local.create_document_def new_document_ptr_in_heap)
    using ‹result |∉| document_ptr_kinds h› apply auto[1]

    apply (metis (no_types, lifting) ‹result |∉| document_ptr_kinds h› document_ptr_kinds_commutes
        local.create_document_is_weakly_dom_component_safe_step select_result_I2)
    done
qed
end

interpretation i_get_scdom_component_create_document?: l_get_scdom_component_create_document_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe to_tree_order get_parent get_parent_locs
  get_child_nodes get_child_nodes_locs get_root_node get_root_node_locs get_ancestors get_ancestors_locs
```

```
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name create_document
  get_disconnected_nodes get_disconnected_nodes_locs
  by(auto simp add: l_get_scdom_component_create_document_Core_DOM_def instances)
declare l_get_scdom_component_create_document_Core_DOM_axioms [instances]
```

insert_before

```
locale l_get_dom_component_insert_before_Core_DOM =
  l_get_dom_component_Core_DOM +
  l_set_child_nodes_Core_DOM +
  l_set_disconnected_nodes_Core_DOM +
  l_remove_child_Core_DOM +
  l_adopt_node_Core_DOM +
  l_insert_before_Core_DOM +
  l_append_child_Core_DOM +
  l_get_owner_document_wf +
  l_get_dom_component_get_scdom_component +
  l_get_scdom_component_Core_DOM +
  l_insert_before_wf_Core_DOM +
  l_set_child_nodes_get_disconnected_nodes +
  l_remove_child +
  l_get_root_node_wf +
  l_set_disconnected_nodes_get_disconnected_nodes_wf +
  l_set_disconnected_nodes_get_ancestors +
  l_get_ancestors_wf +
  l_get_owner_document +
  l_heap_is_wellformed_Core_DOM
begin
lemma insert_before_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ insert_before ptr' child ref →_h h'"
  assumes "ptr ∉ set |h ⊢ get_dom_component ptr'|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast child)|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document ptr'|_r)|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document (cast child)|_r)|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain owner_document where owner_document: "h ⊢ get_owner_document (cast_node_ptr2object_ptr child) →_r
owner_document"
    using assms(4)
    by(auto simp add: local.adopt_node_def insert_before_def elim!:  bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF ensure_pre_insertion_validity_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_owner_document_pure, rotated] bind_returns_heap_E2[rotated,
OF next_sibling_pure, rotated] split: if_splits)
  then
  obtain c where "h ⊢ get_dom_component (cast owner_document) →_r c"
    using get_dom_component_ok assms(1) assms(2) assms(3) get_owner_document_owner_document_in_heap
    by (meson document_ptr_kinds_commutes select_result_I)
  then
  have "ptr ≠ cast owner_document"
    using assms(6) assms(1) assms(2) assms(3) local.get_dom_component_ptr owner_document
    by (metis (no_types, lifting) assms(8) select_result_I2)

  obtain owner_document' where owner_document': "h ⊢ get_owner_document ptr' →_r owner_document'"
    using assms(4)
    by(auto simp add: local.adopt_node_def insert_before_def elim!:  bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF ensure_pre_insertion_validity_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_owner_document_pure, rotated]
        bind_returns_heap_E2[rotated, OF next_sibling_pure, rotated] split: if_splits)
  then
  obtain c where "h ⊢ get_dom_component (cast owner_document') →_r c"
    using get_dom_component_ok assms(1) assms(2) assms(3) get_owner_document_owner_document_in_heap
    by (meson document_ptr_kinds_commutes select_result_I)
```

```
  then
  have "ptr ≠ cast owner_document'"
    using assms(1) assms(2) assms(3) assms(7) local.get_dom_component_ptr owner_document' by auto
  then
  have "ptr ≠ cast |h ⊢ get_owner_document ptr'|_r"
    using owner_document' by auto

  have "ptr ≠ ptr'"
    by (metis (mono_tags, opaque_lifting) assms(1) assms(2) assms(3) assms(5) assms(7) is_OK_returns_result_I
        l_get_dom_component_Core_DOM.get_dom_component_ok l_get_dom_component_Core_DOM.get_dom_component_ptr
        l_get_owner_document.get_owner_document_ptr_in_heap local.l_get_dom_component_Core_DOM_axioms
        local.l_get_owner_document_axioms owner_document' return_returns_result returns_result_select_result)
  have "⋀parent. h ⊢ get_parent child →_r Some parent ⟹ parent ≠ ptr"
    by (meson assms(1) assms(2) assms(3) assms(6) l_get_dom_component_Core_DOM.get_dom_component_ptr
        local.get_dom_component_ok local.get_dom_component_to_tree_order local.get_parent_parent_in_heap
        local.l_get_dom_component_Core_DOM_axioms local.to_tree_order_ok local.to_tree_order_parent
        local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap returns_result_select_result)
  then
  have "⋀parent. |h ⊢ get_parent child|_r = Some parent ⟹ parent ≠ ptr"
    by (metis assms(2) assms(3) assms(4) is_OK_returns_heap_I local.get_parent_ok
        local.insert_before_child_in_heap select_result_I)

  show ?thesis
    using insert_before_writes assms(4)
    apply(rule reads_writes_preserved2)
    apply(auto simp add: insert_before_locs_def adopt_node_locs_def all_args_def)[1]
          apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
          apply (metis ‹ptr ≠ cast_document_ptr2object_ptr |h ⊢ get_owner_document ptr'|_r›
        get_M_Mdocument_preserved3)
         apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
         apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
        apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
         apply (metis ‹ptr ≠ ptr'› document_ptr_casts_commute3 get_M_Mdocument_preserved3)
        apply(auto split: option.splits)[1]
        apply (metis ‹ptr ≠ ptr'› element_ptr_casts_commute3 get_M_Element_preserved8)
       apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def set_disconnected_nodes_locs_def
        all_args_def split: if_splits)[1]
       apply (metis ‹ptr ≠ cast_document_ptr2object_ptr |h ⊢ get_owner_document ptr'|_r› get_M_Mdocument_preserved3)
        apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
          apply (metis (no_types, lifting) ‹⋀parent. |h ⊢ get_parent child|_r = Some parent ⟹ parent
≠ ptr›
        element_ptr_casts_commute3 get_M_Element_preserved8 node_ptr_casts_commute option.case_eq_if option.collaps
         apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
        apply (metis ‹⋀parent. |h ⊢ get_parent child|_r = Some parent ⟹ parent ≠ ptr›
        document_ptr_casts_commute3 get_M_Mdocument_preserved3)
       apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
        apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
      apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
     apply (metis ‹ptr ≠ cast_document_ptr2object_ptr |h ⊢ get_owner_document ptr'|_r›
        get_M_Mdocument_preserved3)
    apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
    apply (metis (no_types, lifting) ‹ptr ≠ cast_document_ptr2object_ptr owner_document›
        get_M_Mdocument_preserved3 owner_document select_result_I2)
```

```
    apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
     apply (metis ‹ptr ≠ ptr'› document_ptr_casts_commute3 get_M_Mdocument_preserved3)
    apply (metis (no_types, lifting) ‹ptr ≠ ptr'› element_ptr_casts_commute3
        get_M_Element_preserved8 node_ptr_casts_commute option.case_eq_if option.collapse)
   apply(auto simp add: remove_child_locs_def set_child_nodes_locs_def
        set_disconnected_nodes_locs_def all_args_def split: if_splits)[1]
  by (metis ‹ptr ≠ cast_document_ptr2object_ptr |h ⊢ get_owner_document ptr'|_r› get_M_Mdocument_preserved3)
qed

lemma insert_before_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ insert_before ptr' child ref →_h h'"
  assumes "ptr ∉ set |h ⊢ get_scdom_component ptr'|_r"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast child)|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  have "ptr' |∈| object_ptr_kinds h"
    by (meson assms(4) is_OK_returns_heap_I local.insert_before_ptr_in_heap)
  then
  obtain sc' where sc': "h ⊢ get_scdom_component ptr' →_r sc'"
    by (meson assms(1) assms(2) assms(3) get_scdom_component_ok is_OK_returns_result_E)
  moreover
  obtain c' where c': "h ⊢ get_dom_component ptr' →_r c'"
    by (meson ‹ptr' |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.get_dom_component_ok)
  ultimately have "set c' ⊆ set sc'"
    using assms(1) assms(2) assms(3) get_scdom_component_subset_get_dom_component by blast

  have "child |∈| node_ptr_kinds h"
    by (meson assms(4) is_OK_returns_heap_I local.insert_before_child_in_heap)
  then
  obtain child_sc where child_sc: "h ⊢ get_scdom_component (cast child) →_r child_sc"
    by (meson assms(1) assms(2) assms(3) get_scdom_component_ok is_OK_returns_result_E
        node_ptr_kinds_commutes)
  moreover
  obtain child_c where child_c: "h ⊢ get_dom_component (cast child) →_r child_c"
    by (meson ‹child |∈| node_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.get_dom_component_ok node_ptr_kinds_commutes)
  ultimately have "set child_c ⊆ set child_sc"
    using assms(1) assms(2) assms(3) get_scdom_component_subset_get_dom_component by blast

  obtain ptr'_owner_document where ptr'_owner_document: "h ⊢ get_owner_document ptr' →_r ptr'_owner_document"
    by (meson ‹set c' ⊆ set sc'› assms(1) assms(2) assms(3) c' get_scdom_component_owner_document_same
        local.get_dom_component_ptr sc' subset_code(1))
  then
  have "h ⊢ get_scdom_component (cast ptr'_owner_document) →_r sc'"
    by (metis (no_types, lifting) ‹set c' ⊆ set sc'› assms(1) assms(2) assms(3) c'
        get_scdom_component_owner_document_same get_scdom_component_ptrs_same_scope_component
        local.get_dom_component_ptr sc' select_result_I2 subset_code(1))
  moreover
  obtain ptr'_owner_document_c where ptr'_owner_document_c:
    "h ⊢ get_dom_component (cast ptr'_owner_document) →_r ptr'_owner_document_c"
    by (meson assms(1) assms(2) assms(3) document_ptr_kinds_commutes is_OK_returns_result_E
        local.get_dom_component_ok local.get_owner_document_owner_document_in_heap ptr'_owner_document)
  ultimately have "set ptr'_owner_document_c ⊆ set sc'"
    using assms(1) assms(2) assms(3) get_scdom_component_subset_get_dom_component by blast

  obtain child_owner_document where child_owner_document: "h ⊢ get_owner_document (cast child) →_r child_owner_do
    by (meson ‹set child_c ⊆ set child_sc› assms(1) assms(2) assms(3) child_c child_sc
        get_scdom_component_owner_document_same local.get_dom_component_ptr subset_code(1))

  have "child_owner_document |∈| document_ptr_kinds h"
```

```
    using assms(1) assms(2) assms(3) child_owner_document local.get_owner_document_owner_document_in_heap
    by blast
  then
  have "h ⊢ get_scdom_component (cast child_owner_document) →r child_sc"
    using get_scdom_component_ok assms(1) assms(2) assms(3) child_sc
    by (metis (no_types, lifting) ‹set child_c ⊆ set child_sc› child_c child_owner_document
        get_scdom_component_owner_document_same get_scdom_component_ptrs_same_scope_component
        local.get_dom_component_ptr returns_result_eq set_mp)
  moreover
  obtain child_owner_document_c where child_owner_document_c:
    "h ⊢ get_dom_component (cast child_owner_document) →r child_owner_document_c"
    by (meson assms(1) assms(2) assms(3) child_owner_document document_ptr_kinds_commutes
        is_OK_returns_result_E local.get_dom_component_ok local.get_owner_document_owner_document_in_heap)
  ultimately have "set child_owner_document_c ⊆ set child_sc"
    using assms(1) assms(2) assms(3) get_scdom_component_subset_get_dom_component by blast


  have "ptr ∉ set |h ⊢ get_dom_component ptr'|r"
    using ‹set c' ⊆ set sc'› assms(5) c' sc' by auto
  moreover have "ptr ∉ set |h ⊢ get_dom_component (cast child)|r"
    using ‹set child_c ⊆ set child_sc› assms(6) child_c child_sc by auto
  moreover have "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document ptr'|r)|r"
    using ‹set ptr'_owner_document_c ⊆ set sc'› assms(5) ptr'_owner_document ptr'_owner_document_c sc'
    by auto
  moreover have "ptr ∉ set |h ⊢ get_dom_component (cast |h ⊢ get_owner_document (cast child)|r)|r"
    using ‹set child_owner_document_c ⊆ set child_sc› assms(6) child_owner_document child_owner_document_c
      child_sc by auto

  ultimately show ?thesis
    using assms insert_before_is_component_unsafe
    by blast
qed

lemma insert_before_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ insert_before ptr node child →h h'"
  shows "is_strongly_scdom_component_safe ({ptr, cast node} ∪ (case child of Some ref ⇒ {cast ref} | None
⇒ {} )) {} h h'"
proof -
  obtain ancestors reference_child owner_document h2 h3 disconnected_nodes_h2 where
    ancestors: "h ⊢ get_ancestors ptr →r ancestors" and
    node_not_in_ancestors: "cast node ∉ set ancestors" and
    reference_child:
    "h ⊢ (if Some node = child then a_next_sibling node else return child) →r reference_child" 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 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)

  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 "known_ptrs h2"
    using assms(3) object_ptr_kinds_M_eq3_h known_ptrs_preserved by blast

  have wellformed_h2: "heap_is_wellformed h2"
    using adopt_node_preserves_wellformedness[OF assms(1) h2] assms(3) assms(2) .

  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 "known_ptrs h3"
    using object_ptr_kinds_M_eq3_h2 known_ptrs_preserved <known_ptrs h2> by blast

  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 "object_ptr_kinds h = object_ptr_kinds h'"
    by (simp add: object_ptr_kinds_M_eq3_h object_ptr_kinds_M_eq3_h' object_ptr_kinds_M_eq3_h2)
  then
  show ?thesis
    using assms
    apply(auto simp add: is_strongly_scdom_component_safe_def)[1]
    using insert_before_is_strongly_dom_component_safe_step local.get_scdom_component_impl by blast
qed
```

```
lemma append_child_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ append_child ptr' child →_h h'"
  assumes "ptr ∉ set |h ⊢ get_scdom_component ptr'|_r"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast child)|_r"
  shows "preserved (get_M ptr getter) h h'"
  by (metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(6)
      insert_before_is_strongly_dom_component_safe_step local.append_child_def)

lemma append_child_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ append_child ptr child →_h h'"
  shows "is_strongly_scdom_component_safe {ptr, cast child} {} h h'"
  using assms unfolding append_child_def
  using insert_before_is_strongly_dom_component_safe
  by fastforce
end

interpretation i_get_dom_component_insert_before?: l_get_dom_component_insert_before_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name set_child_nodes set_child_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs
  get_owner_document remove_child remove_child_locs remove adopt_node adopt_node_locs insert_before
  insert_before_locs append_child get_scdom_component is_strongly_scdom_component_safe
  is_weakly_scdom_component_safe
  by(auto simp add: l_get_dom_component_insert_before_Core_DOM_def instances)
declare l_get_dom_component_insert_before_Core_DOM_axioms [instances]
```

get_owner_document

```
locale l_get_owner_document_scope_component_Core_DOM =
  l_get_scdom_component_Core_DOM +
  l_get_owner_document_wf_Core_DOM +
  l_get_dom_component_Core_DOM +
  l_get_dom_component_get_scdom_component +
  l_get_owner_document_wf_get_root_node_wf
begin
lemma get_owner_document_is_strongly_scdom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →_r sc"
  assumes "h ⊢ get_owner_document ptr' →_r owner_document"
  shows "cast owner_document ∈ set sc ⟷ ptr' ∈ set sc"
proof -
  have "h ⊢ get_owner_document (cast owner_document) →_r owner_document"
    by (metis assms(1) assms(2) assms(3) assms(5) cast_document_ptr2object_ptr_inject
        document_ptr_casts_commute3 document_ptr_document_ptr_cast document_ptr_kinds_commutes
        local.get_owner_document_owner_document_in_heap local.get_root_node_document
        local.get_root_node_not_node_same node_ptr_no_document_ptr_cast)
  then show ?thesis
    using assms
    using bind_returns_result_E contra_subsetD get_scdom_component_ok
      get_scdom_component_ptrs_same_scope_component get_scdom_component_subset_get_dom_component
      is_OK_returns_result_E is_OK_returns_result_I local.get_dom_component_ok local.get_dom_component_ptr
      local.get_owner_document_ptr_in_heap local.get_owner_document_pure local.get_scdom_component_def
      pure_returns_heap_eq returns_result_eq
    by (smt (verit) local.get_scdom_component_ptrs_same_owner_document subsetD)
qed


lemma get_owner_document_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
```

```
  assumes "h ⊢ get_owner_document ptr →r owner_document"
  assumes "h ⊢ get_owner_document ptr →h h'"
  shows "is_strongly_scdom_component_safe {ptr} {cast owner_document} h h'"
proof -
  have "h = h'"
    by (meson assms(5) local.get_owner_document_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_scdom_component_safe_def Let_def preserved_def)[1]
    by (smt (verit) get_owner_document_is_strongly_scdom_component_safe_step inf.orderE is_OK_returns_result_I
        local.get_dom_component_ok local.get_dom_component_to_tree_order_subset local.get_owner_document_ptr_in_hea
        local.get_scdom_component_impl local.get_scdom_component_ok local.get_scdom_component_ptr_in_heap
        local.get_scdom_component_subset_get_dom_component local.to_tree_order_ok
        local.to_tree_order_ptr_in_result returns_result_select_result subset_eq)
qed
end

interpretation i_get_owner_document_scope_component?: l_get_owner_document_scope_component_Core_DOM
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
  get_owner_document get_disconnected_nodes get_disconnected_nodes_locs to_tree_order known_ptr
  known_ptrs type_wf heap_is_wellformed parent_child_rel get_child_nodes get_child_nodes_locs
  get_parent get_parent_locs get_ancestors get_ancestors_locs get_root_node get_root_node_locs
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name
  by(auto simp add: l_get_owner_document_scope_component_Core_DOM_def instances)
declare l_get_owner_document_scope_component_Core_DOM_axioms [instances]

end
```

2.4 Shadow SC DOM Components (Shadow_DOM_DOM_Components)

```
theory Shadow_DOM_DOM_Components
  imports
    Shadow_SC_DOM.Shadow_DOM
    Core_DOM_DOM_Components
begin

declare [[smt_timeout = 1200]]
```

2.5 Shadow root components (Shadow_DOM_DOM_Components)

2.5.1 get_component

```
global_interpretation l_get_dom_component_Core_DOM_defs get_root_node get_root_node_locs to_tree_order
  defines get_dom_component = a_get_dom_component
    and is_strongly_dom_component_safe = a_is_strongly_dom_component_safe
    and is_weakly_dom_component_safe = a_is_weakly_dom_component_safe
  .

interpretation i_get_dom_component?: l_get_dom_component_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name
  by(auto simp add: l_get_dom_component_Core_DOM_def l_get_dom_component_Core_DOM_axioms_def
      get_dom_component_def is_strongly_dom_component_safe_def is_weakly_dom_component_safe_def instances)
declare l_get_dom_component_Core_DOM_axioms [instances]
```

2.5.2 attach_shadow_root

```
locale l_get_dom_component_attach_shadow_root_Core_DOM =
  l_get_dom_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  to_tree_order get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name +
  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_set_mode_Shadow_DOM type_wf set_mode set_mode_locs +
  l_set_shadow_root_Shadow_DOM type_wf set_shadow_root set_shadow_root_locs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    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_dom_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option)
prog"
    and get_elements_by_class_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    and get_elements_by_tag_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    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 get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and is_strongly_dom_component_safe :: "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_)
heap ⇒ bool"
    and is_weakly_dom_component_safe :: "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap
⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) 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

lemma attach_shadow_root_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →_h h'"
  assumes "ptr ≠ cast |h ⊢ attach_shadow_root element_ptr shadow_root_mode|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast element_ptr)|_r"
  shows "preserved (get_M_Object ptr getter) h h'"
proof -
  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
```

```
    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 shadow_root_mode →_r new_shadow_root_ptr"
    using new_shadow_root_ptr h2 h3 h'
    using assms(4)
    by(auto simp add: attach_shadow_root_def intro!: bind_returns_result_I
        bind_pure_returns_result_I[OF get_tag_name_pure] bind_pure_returns_result_I[OF get_shadow_root_pure]
        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 "preserved (get_M_Object ptr getter) h h2"
    using h2 new_shadow_root_ptr
    by (metis (no_types, lifting) ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr›
        assms(5) new_shadow_root_get_M_Object select_result_I2)

  have "ptr ≠ cast new_shadow_root_ptr"
    using ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr› assms(5)
    by auto

  have "preserved (get_M_Object ptr getter) h2 h3"
    using set_mode_writes h3
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_mode_locs_def all_args_def)[1]
    using ‹ptr ≠ cast_shadow_root_ptr2object_ptr new_shadow_root_ptr›
    by (metis get_M_Mshadow_root_preserved3a)

  have "element_ptr |∈| element_ptr_kinds h"
    using ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr› attach_shadow_root_element
    by blast
  have "ptr ≠ cast element_ptr"
    by (metis (no_types, lifting) ‹element_ptr |∈| element_ptr_kinds h› assms(1) assms(2) assms(3)
        assms(6) element_ptr_kinds_commutes is_OK_returns_result_E l_get_dom_component_Core_DOM.get_dom_component_
        local.get_dom_component_ptr local.l_get_dom_component_Core_DOM_axioms node_ptr_kinds_commutes select_resul

  have "preserved (get_M_Object ptr getter) h3 h'"
    using set_shadow_root_writes h'
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_shadow_root_locs_def all_args_def)[1]
    by (metis ‹ptr ≠ cast_element_ptr2object_ptr element_ptr› get_M_Element_preserved8)

  show ?thesis
    using ‹preserved (get_M ptr getter) h h2› ‹preserved (get_M ptr getter) h2 h3› ‹preserved (get_M ptr
getter) h3 h'›
    by(auto simp add: preserved_def)
qed
end

interpretation i_get_dom_component_attach_shadow_root?: l_get_dom_component_attach_shadow_root_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  set_shadow_root set_shadow_root_locs set_mode set_mode_locs attach_shadow_root get_disconnected_nodes
  get_disconnected_nodes_locs is_strongly_dom_component_safe is_weakly_dom_component_safe get_tag_name
  get_tag_name_locs get_shadow_root get_shadow_root_locs
  by(auto simp add: l_get_dom_component_attach_shadow_root_Core_DOM_def instances)
declare l_get_dom_component_attach_shadow_root_Core_DOM_axioms [instances]
```

2.5.3 get_shadow_root

```
locale l_get_shadow_root_component_Shadow_DOM =
  l_get_shadow_root +
```

```
  l_heap_is_wellformed_Shadow_DOM +
  l_get_dom_component_Core_DOM +
  l_get_root_node_Core_DOM +
  l_get_root_node_wf_Core_DOM +
  l_remove_shadow_root_get_child_nodes
begin
lemma get_shadow_root_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  shows "set |h ⊢ get_dom_component (cast host)|_r ∩ set |h ⊢ get_dom_component (cast shadow_root_ptr)|_r
= {}"
proof -
  have "cast host |∈| object_ptr_kinds h"
    using assms(4) get_shadow_root_ptr_in_heap by auto
  then obtain host_c where host_c: "h ⊢ get_dom_component (cast host) →_r host_c"
    by (meson  assms(1) assms(2) assms(3) get_dom_component_ok is_OK_returns_result_E)
  obtain host_root where host_root: "h ⊢ get_root_node (cast host) →_r host_root"
    by (metis (no_types, lifting) bind_returns_heap_E get_dom_component_def host_c
        is_OK_returns_result_I pure_def pure_eq_iff)

  have "cast shadow_root_ptr |∈| object_ptr_kinds h"
    using get_shadow_root_shadow_root_ptr_in_heap assms shadow_root_ptr_kinds_commutes
    using document_ptr_kinds_commutes by blast
  then obtain shadow_root_ptr_c where shadow_root_ptr_c: "h ⊢ get_dom_component (cast shadow_root_ptr)
→_r shadow_root_ptr_c"
    by (meson  assms(1) assms(2) assms(3) get_dom_component_ok is_OK_returns_result_E)
  have "h ⊢ get_root_node (cast shadow_root_ptr) →_r cast shadow_root_ptr"
    using ‹cast shadow_root_ptr |∈| object_ptr_kinds h›
    by(auto simp add: get_root_node_def get_ancestors_def intro!: bind_pure_returns_result_I
        split: option.splits)

  have "host_root ≠ cast shadow_root_ptr"
  proof (rule ccontr, simp)
    assume "host_root = cast_shadow_root_ptr2object_ptr shadow_root_ptr"

    have "(cast shadow_root_ptr, host_root) ∈ (parent_child_rel h)^*"
      using ‹host_root = cast_shadow_root_ptr2object_ptr shadow_root_ptr› by auto
    moreover have "(host_root, cast host) ∈ (parent_child_rel h)^*"
      using get_root_node_parent_child_rel host_root assms
      by blast
    moreover have "(cast host, cast shadow_root_ptr) ∈ (a_host_shadow_root_rel h)"
      using assms(4) apply(auto simp add: a_host_shadow_root_rel_def)[1]
      by (metis (mono_tags, lifting) get_shadow_root_ptr_in_heap image_eqI is_OK_returns_result_I
          mem_Collect_eq prod.simps(2) select_result_I2)
    moreover have " acyclic (parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel
h)"
      using assms(1)[unfolded heap_is_wellformed_def]
      by auto
    ultimately show False
      using local.parent_child_rel_node_ptr
      by (metis (no_types, lifting) Un_iff ‹host_root = cast_shadow_root_ptr2object_ptr shadow_root_ptr›
          acyclic_def in_rtrancl_UnI rtrancl_into_trancl1)
  qed
  then have "host_c ≠ shadow_root_ptr_c"
    by (metis ‹h ⊢ get_root_node (cast_shadow_root_ptr2object_ptr shadow_root_ptr) →_r cast_shadow_root_ptr2object_ptr
shadow_root_ptr›
        assms(1) assms(2) assms(3) get_dom_component_ptr get_dom_component_root_node_same host_c host_root
        local.get_root_node_parent_child_rel local.get_root_node_same_no_parent_parent_child_rel rtranclE
shadow_root_ptr_c)
  then have "set host_c ∩ set shadow_root_ptr_c = {}"
    using assms get_dom_component_no_overlap Shadow_DOM.a_heap_is_wellformed_def host_c shadow_root_ptr_c
    by blast
  then show ?thesis
```

```
    using host_c shadow_root_ptr_c
    by auto
qed
end

interpretation i_get_shadow_root_component?: l_get_shadow_root_component_Shadow_DOM
  type_wf 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 known_ptr
  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 to_tree_order get_parent
  get_parent_locs get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe
  get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name remove_shadow_root remove_shadow_root_locs
  by(auto simp add: l_get_shadow_root_component_Shadow_DOM_def instances)
declare l_get_shadow_root_component_Shadow_DOM_axioms [instances]
```

2.5.4 get_host

```
locale l_get_host_component_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM +
  l_get_host +
  l_get_dom_component_Core_DOM +
  l_get_shadow_root_component_Shadow_DOM
begin
lemma get_host_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_host shadow_root_ptr →_r host"
  shows "set |h ⊢ get_dom_component (cast host)|_r ∩ set |h ⊢ get_dom_component (cast shadow_root_ptr)|_r
= {}"
proof -
  have "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
    using assms(1) assms(2) assms(4) local.shadow_root_host_dual by blast
  then show ?thesis
    using assms(1) assms(2) assms(3) local.get_shadow_root_is_component_unsafe by blast
qed
end

interpretation i_get_host_component?: l_get_host_component_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 to_tree_order get_parent get_parent_locs get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors
  get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name remove_shadow_root
  remove_shadow_root_locs
  by(auto simp add: l_get_host_component_Shadow_DOM_def instances)
declare l_get_host_component_Shadow_DOM_axioms [instances]
```

2.5.5 get_root_node_si

```
locale l_get_dom_component_get_root_node_si_Shadow_DOM =
  l_get_root_node_si_wf_Shadow_DOM +
  l_get_dom_component_Core_DOM
begin

lemma get_root_node_si_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node_si ptr' →_r root"
  shows "set |h ⊢ get_dom_component ptr'|_r = set |h ⊢ get_dom_component root|_r ∨
set |h ⊢ get_dom_component ptr'|_r ∩ set |h ⊢ get_dom_component root|_r = {}"
proof -
  have "ptr' |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr_in_heap assms(4)
```

```
    by(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)
  then
  obtain c where "h ⊢ get_dom_component ptr' →_r c"
    by (meson assms(1) assms(2) assms(3) local.get_dom_component_ok select_result_I)
  moreover
  have "root |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr assms(4)
    apply(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) empty_iff empty_set
        get_ancestors_si_ptrs_in_heap last_in_set)
  then
  obtain c' where "h ⊢ get_dom_component root →_r c'"
    by (meson assms(1) assms(2) assms(3) local.get_dom_component_ok select_result_I)
  ultimately show ?thesis
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) local.get_dom_component_no_overlap select_result_I2)
qed
end

interpretation i_get_dom_component_get_root_node_si?: l_get_dom_component_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
  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 to_tree_order get_dom_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  by(auto simp add: l_get_dom_component_get_root_node_si_Shadow_DOM_def instances)
declare l_get_dom_component_get_root_node_si_Shadow_DOM_axioms [instances]
```

2.5.6 get_assigned_nodes

```
lemma get_shadow_root_not_weakly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    element_ptr and shadow_root_ptr_opt and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_shadow_root element_ptr →_r shadow_root_ptr_opt →_h h'" and
    "¬ is_weakly_dom_component_safe {cast element_ptr} (cast ` set_option shadow_root_ptr_opt) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e1
```

```
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and element_ptr="?e1"])
    by code_simp+
qed

lemma assigned_nodes_not_weakly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder},
'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    node_ptr and nodes and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ assigned_nodes node_ptr →_r nodes →_h h'" and
    "¬ is_weakly_dom_component_safe {cast node_ptr} (cast ` set nodes) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Closed;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e3
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e3 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and node_ptr="?e3"])
    by code_simp+
qed

lemma get_composed_root_node_not_weakly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    ptr and root and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_root_node_si ptr →_r root →_h h'" and
    "¬ is_weakly_dom_component_safe {ptr} {root} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
```

```
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Closed;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e3
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e3 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e3"])
    by code_simp+
qed

lemma assigned_slot_not_weakly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder},
'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    node_ptr and slot_opt and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ assigned_slot node_ptr →_r slot_opt →_h h'" and
    "¬ is_weakly_dom_component_safe {cast node_ptr} (cast ` set_option slot_opt) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
 'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e2
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e2 = "|?h0 ⊢ ?P|_r"
```

```
  show thesis
    apply(rule that[where h="?h1" and node_ptr="cast_element_ptr2node_ptr ?e2"])
    by code_simp+
qed

locale l_assigned_nodes_component_Shadow_DOM =
  l_get_tag_name +
  l_get_child_nodes +
  l_heap_is_wellformed_Shadow_DOM +
  l_find_slot_Shadow_DOM +
  l_assigned_nodes_Shadow_DOM +
  l_assigned_nodes_wf_Shadow_DOM +
  l_get_dom_component_Core_DOM +
  l_adopt_node_Shadow_DOM +
  l_remove_child_Core_DOM +
  l_remove_child_wf2 +
  l_insert_before_wf +
  l_insert_before_wf2 +
  l_append_child_Core_DOM +
  l_append_child_wf_Shadow_DOM +
  l_set_disconnected_nodes_get_tag_name +
  l_set_shadow_root_get_child_nodes +
  l_set_child_nodes_get_tag_name +
  l_get_shadow_root_component_Shadow_DOM +
  l_remove_shadow_root_Shadow_DOM +
  l_remove_shadow_root_get_tag_name +
  l_set_disconnected_nodes_get_shadow_root +
  l_set_child_nodes_get_shadow_root +
  l_remove_shadow_root_wf_Shadow_DOM
begin
lemma find_slot_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ find_slot open_flag node_ptr →_r Some slot"
  shows "set |h ⊢ get_dom_component (cast node_ptr)|_r ∩ set |h ⊢ get_dom_component (cast slot)|_r = {}"
proof -
  obtain host shadow_root_ptr to where
    "h ⊢ get_parent node_ptr →_r Some (cast host)" and
    "h ⊢ get_shadow_root host →_r Some shadow_root_ptr" and
    "h ⊢ to_tree_order (cast shadow_root_ptr) →_r to" and
    "cast slot ∈ set to"
    using assms(4)
    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]
    by (metis element_ptr_casts_commute3)+

  have "node_ptr |∈| node_ptr_kinds h"
    using assms(4) find_slot_ptr_in_heap by blast
  then obtain node_ptr_c where node_ptr_c: "h ⊢ get_dom_component (cast node_ptr) →_r node_ptr_c"
    using assms(1) assms(2) assms(3) get_dom_component_ok is_OK_returns_result_E
      node_ptr_kinds_commutes[symmetric]
    by metis

  then have "cast host ∈ set node_ptr_c"
    using ‹h ⊢ get_parent node_ptr →_r Some (cast host)› get_dom_component_parent_inside assms(1)
      assms(2) assms(3) get_dom_component_ptr
    by blast

  then have "h ⊢ get_dom_component (cast host) →_r node_ptr_c"
    using ‹h ⊢ get_parent node_ptr →_r Some (cast host)› get_dom_component_subset a_heap_is_wellformed_def
      assms(1) assms(2) assms(3) node_ptr_c
    by blast
```

```
  moreover have "slot |∈| element_ptr_kinds h"
    using assms(4) find_slot_slot_in_heap by blast
  then obtain slot_c where slot_c: "h ⊢ get_dom_component (cast slot) →r slot_c"
    using a_heap_is_wellformed_def assms(1) assms(2) assms(3) get_dom_component_ok is_OK_returns_result_E
      node_ptr_kinds_commutes[symmetric] element_ptr_kinds_commutes[symmetric]
    by metis
  then have "cast shadow_root_ptr ∈ set slot_c"
    using ‹h ⊢ to_tree_order (cast shadow_root_ptr) →r to› ‹cast slot ∈ set to› get_dom_component_to_tree_order
      assms(1) assms(2) assms(3) get_dom_component_ptr
    by blast
  then have "h ⊢ get_dom_component (cast shadow_root_ptr) →r slot_c"
    using  ‹h ⊢ get_shadow_root host →r Some shadow_root_ptr› get_dom_component_subset assms(1) assms(2)
      assms(3) slot_c
    by blast

  ultimately show ?thesis
    using get_shadow_root_is_component_unsafe assms ‹h ⊢ get_shadow_root host →r Some shadow_root_ptr›
      node_ptr_c slot_c
    by fastforce
qed


lemma assigned_nodes_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ assigned_nodes element_ptr →r nodes"
  assumes "node_ptr ∈ set nodes"
  shows "set |h ⊢ get_dom_component (cast element_ptr)|r ∩ set |h ⊢ get_dom_component (cast node_ptr)|r
= {}"
proof -
  have "h ⊢ find_slot False node_ptr →r Some element_ptr"
    using assms(4) assms(5)
    by(auto simp add: assigned_nodes_def elim!: bind_returns_result_E2
        dest!: filter_M_holds_for_result[where x=node_ptr] intro!: bind_pure_I split: if_splits)
  then show ?thesis
    using assms find_slot_is_component_unsafe
    by blast
qed

lemma flatten_dom_assigned_nodes_become_children:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ flatten_dom →h h'"
  assumes "h ⊢ assigned_nodes slot →r nodes"
  assumes "nodes ≠ []"
  shows "h' ⊢ get_child_nodes (cast slot) →r nodes"
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 all_tups_slot: "⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set tups ⟹
h ⊢ get_tag_name slot →_r ''slot''"
    using tups
    apply(induct element_ptrs arbitrary: tups)
    by(auto elim!: bind_returns_result_E2 split: if_splits intro!: map_filter_M2_pure bind_pure_I)

  have "distinct element_ptrs"
    using ‹h ⊢ element_ptr_kinds_M →_r element_ptrs›
    by auto
  then
  have "distinct tups"
    using tups
    apply(induct element_ptrs arbitrary: tups)
    by(auto elim!: bind_returns_result_E2 intro!: map_filter_M2_pure bind_pure_I
        split: option.splits if_splits intro: map_filter_pure_foo[rotated] )


  have "slot ∈ set element_ptrs"
    using assms(5) assigned_nodes_ptr_in_heap ‹h ⊢ element_ptr_kinds_M →_r element_ptrs›
    by auto
  then
  have "(slot, nodes) ∈ set tups"
    apply(rule map_filter_M2_in_result[OF tups])
    apply(auto intro!: bind_pure_I)[1]
    apply(intro bind_pure_returns_result_I)
    using assms assigned_nodes_slot_is_slot
    by(auto intro!: bind_pure_returns_result_I)

  have "⋀slot nodes. (slot, nodes) ∈ set tups ⟹ h ⊢ assigned_nodes slot →_r nodes"
    using tups
    apply(induct element_ptrs arbitrary: tups)
    by(auto elim!: bind_returns_result_E2 intro!: map_filter_M2_pure bind_pure_I split: if_splits)
  then
  have elementwise_eq: "⋀slot slot' nodes nodes'. (slot, nodes) ∈ set tups ⟹
(slot', nodes') ∈ set tups ⟹ slot = slot' ⟹ nodes = nodes'"
    by (meson returns_result_eq)

  have "⋀slot nodes. (slot, nodes) ∈ set tups ⟹ distinct nodes"
    using ‹⋀slot nodes. (slot, nodes) ∈ set tups ⟹ h ⊢ assigned_nodes slot →_r nodes› assigned_nodes_distinct
    using assms(1) by blast

  have "⋀slot slot' nodes nodes'. (slot, nodes) ∈ set tups ⟹ (slot', nodes') ∈ set tups ⟹ slot ≠
slot' ⟹ set nodes ∩ set nodes' = {}"
    using ‹⋀slot nodes. (slot, nodes) ∈ set tups ⟹ h ⊢ assigned_nodes slot →_r nodes›
      assigned_nodes_different_ptr assms(1) assms(2) assms(3) by blast

  have "h ⊢ get_tag_name slot →_r ''slot''"
    using ‹(slot, nodes) ∈ set tups› all_tups_slot by blast
  then have "h2 ⊢ get_tag_name slot →_r ''slot''"
    using h2
  proof(induct tups arbitrary: h, simp)
    case (Cons x xs)
    obtain xc ha hb slot' nodes' where
      "x = (slot', nodes')" and
      "h ⊢ get_child_nodes (cast_element_ptr2object_ptr slot') →_r xc" and
```

```
      ha: "h ⊢ forall_M remove xc →_h ha" and
      hb: "ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →_h hb" and
      remainder: "hb ⊢ forall_M (λ(slot, assigned_nodes). Heap_Error_Monad.bind
(get_child_nodes (cast_element_ptr2object_ptr slot))
(λx. Heap_Error_Monad.bind (forall_M remove x)
(λ_. forall_M (append_child (cast_element_ptr2object_ptr slot)) assigned_nodes))) xs →_h h2"
      using Cons(3)
      by (auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated]
          bind_returns_result_E bind_returns_result_E2[rotated, OF get_child_nodes_pure, rotated] split:
prod.splits)

    have "ha ⊢ get_tag_name slot →_r ''slot''"
      using ‹h ⊢ get_tag_name slot →_r ''slot''› ha
    proof(induct xc arbitrary: h, simp)
      case (Cons a yc)
      obtain hb1 where
        hb1: "h ⊢ remove a →_h hb1" and
        hba: "hb1 ⊢ forall_M remove yc →_h ha"
        using Cons
        by (auto elim!: bind_returns_heap_E)
      have "hb1 ⊢ get_tag_name slot →_r ''slot''"
        using ‹h ⊢ get_tag_name slot →_r ''slot''› hb1
        by(auto simp add: CD.remove_child_locs_def set_child_nodes_get_tag_name
            set_disconnected_nodes_get_tag_name dest!: reads_writes_separate_forwards[OF get_tag_name_reads
              CD.remove_writes])
      then show ?case
        using hba Cons(1) by simp
    qed
    then
    have "hb ⊢ get_tag_name slot →_r ''slot''"
      using hb
    proof (induct nodes' arbitrary: ha, simp)
      case (Cons a nodes')
      obtain ha1 where
        ha1: "ha ⊢ append_child (cast slot') a →_h ha1" and
        hb: "ha1 ⊢ forall_M (append_child (cast slot')) nodes' →_h hb"
        using Cons
        by (auto elim!: bind_returns_heap_E)
      have "ha1 ⊢ get_tag_name slot →_r ''slot''"
        using ‹ha ⊢ get_tag_name slot →_r ''slot''› ha1
        by(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def
            CD.adopt_node_locs_def CD.remove_child_locs_def set_child_nodes_get_tag_name
            set_disconnected_nodes_get_tag_name dest!: reads_writes_separate_forwards[OF get_tag_name_reads
              insert_before_writes] split: if_splits)
      then show ?case
        using ha1 hb Cons(1)
        by simp
    qed
    then
    show ?case
      using Cons(1) remainder by simp
  qed

  have "h2 ⊢ get_child_nodes (cast slot) →_r nodes ∧ heap_is_wellformed h2 ∧ type_wf h2 ∧ known_ptrs h2"
    using ‹(slot, nodes) ∈ set tups›
    using h2 assms(1) assms(2) assms(3) ‹distinct tups› all_tups_slot elementwise_eq
    using ‹⋀slot slot' assigned_nodes nodes'. (slot, assigned_nodes) ∈ set tups ⟹
(slot', nodes') ∈ set tups ⟹ slot ≠ slot' ⟹ set assigned_nodes ∩ set nodes' = {}›
    using ‹⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set tups ⟹ distinct assigned_nodes›
  proof(induct tups arbitrary: h, simp)
    case (Cons x xs)
    obtain xc ha hb slot' nodes' where
      "x = (slot', nodes')" and
```

```
      "h ⊢ get_child_nodes (cast_element_ptr2object_ptr slot') →_r xc" and
      ha: "h ⊢ forall_M remove xc →_h ha" and
      hb: "ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →_h hb" and
      remainder: "hb ⊢ forall_M (λ(slot, assigned_nodes). Heap_Error_Monad.bind
(get_child_nodes (cast_element_ptr2object_ptr slot)) (λx. Heap_Error_Monad.bind (forall_M remove x)
(λ_. forall_M (append_child (cast_element_ptr2object_ptr slot)) assigned_nodes))) xs →_h h2"
      using Cons(3)
      by (auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated]
          bind_returns_result_E bind_returns_result_E2[rotated, OF get_child_nodes_pure, rotated]
          split: prod.splits)

    have "⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set xs ⟹ h ⊢ get_tag_name slot →_r ''slot''"
      using Cons by auto

    have "heap_is_wellformed ha" and "type_wf ha" and "known_ptrs ha"
      using Cons(4) Cons(5) Cons(6) ‹h ⊢ forall_M remove xc →_h ha›
      apply(induct xc arbitrary: h)
      apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
          simp add: CD.remove_def split: option.splits)[1]
      apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
          simp add: CD.remove_def split: option.splits)[1]
      apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
          simp add: CD.remove_def split: option.splits)[1]
      apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
          simp add: CD.remove_def split: option.splits)[1]
      using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf remove_child_preserves_known_p
      apply metis
      apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
          simp add: CD.remove_def split: option.splits)[1]
      using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf remove_child_preserves_known_p
      apply metis
      apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
          simp add: CD.remove_def split: option.splits)[1]
      using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf remove_child_preserves_known_p
      apply metis
      done
    then
    have "heap_is_wellformed hb" and "type_wf hb" and "known_ptrs hb"
      using ‹ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →_h hb›
      apply(induct nodes' arbitrary: ha)
      apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
      apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
      apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
      apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
      using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf insert_before_preserves_know
      apply metis
      apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
      using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf insert_before_preserves_know
      apply metis
      apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
      using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf insert_before_preserves_know
      apply metis
      done

    {
      fix slot assigned_nodes
      assume "(slot, assigned_nodes) ∈ set xs"
      then have "h ⊢ get_tag_name slot →_r ''slot''"
        using ‹⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set xs ⟹ h ⊢ get_tag_name slot →_r ''slot''›
        by auto
      then have "ha ⊢ get_tag_name slot →_r ''slot''"
        using ‹h ⊢ forall_M remove xc →_h ha›
        apply(induct xc arbitrary: h)
```

```
      by(auto simp add: CD.remove_child_locs_def set_child_nodes_get_tag_name set_disconnected_nodes_get_tag_nam
          dest!: reads_writes_separate_forwards[OF get_tag_name_reads CD.remove_writes]
          elim!: bind_returns_heap_E)
    then have "hb ⊢ get_tag_name slot →r ''slot''"
      using ‹ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →h hb›
      apply(induct nodes' arbitrary: ha)
      by(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def CD.adopt_node_locs_def
          CD.remove_child_locs_def set_child_nodes_get_tag_name set_disconnected_nodes_get_tag_name
          dest!: reads_writes_separate_forwards[OF get_tag_name_reads insert_before_writes]
          elim!: bind_returns_heap_E
          split: if_splits)
  } note tag_names_same = this

  show ?case
  proof(cases "slot' = slot")
    case True
    then
    have "nodes' = nodes"
      using Cons.prems(1) Cons.prems(8) ‹x = (slot', nodes')›
      by (metis list.set_intros(1))
    then
    have "(slot, nodes) ∉ set xs"
      using Cons.prems(6) True ‹x = (slot', nodes')› by auto

    have "ha ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r []"
      using remove_for_all_empty_children Cons.prems(3) Cons.prems(4) Cons.prems(5)
        True ‹h ⊢ forall_M remove xc →h ha›
      using ‹h ⊢ get_child_nodes (cast_element_ptr2object_ptr slot') →r xc›
      by blast
    then
    have "hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes"
      using  append_child_for_all_on_no_children[OF ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs
hb›]
        True ‹nodes' = nodes›
      using ‹ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →h hb›
      using ‹(slot, nodes) ∈ set tups› ‹⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set tups ⟹
distinct assigned_nodes› ‹heap_is_wellformed ha› ‹known_ptrs ha› ‹type_wf ha› local.append_child_for_all_on_no_chi
      by blast
    with ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
    show ?thesis
      using ‹(slot, nodes) ∉ set xs› remainder
      using ‹⋀slot slot' assigned_nodes nodes'. (slot, assigned_nodes) ∈ set (x#xs) ⟹
(slot', nodes') ∈ set (x#xs) ⟹ slot = slot' ⟹ assigned_nodes = nodes'›
      using ‹(slot, nodes) ∈ set (x # xs)›
      using ‹⋀slot slot' assigned_nodes nodes'. (slot, assigned_nodes) ∈ set (x#xs) ⟹
(slot', nodes') ∈ set (x#xs) ⟹ slot ≠ slot' ⟹ set assigned_nodes ∩ set nodes' = {}›
    proof(induct xs arbitrary: hb, simp)
      case (Cons y ys)
      obtain yc hba hbb slot'' nodes'' where
        "y = (slot'', nodes'')" and
        "hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →r yc" and
        "hb ⊢ forall_M remove yc →h hba" and
        "hba ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot'')) nodes'' →h hbb" and
        remainder: "hbb ⊢ forall_M (λ(slot, assigned_nodes).
Heap_Error_Monad.bind (get_child_nodes (cast_element_ptr2object_ptr slot))
(λx. Heap_Error_Monad.bind (forall_M remove x)
(λ_. forall_M (append_child (cast_element_ptr2object_ptr slot)) assigned_nodes))) ys →h h2"
        using Cons(7)
        by (auto elim!: bind_returns_heap_E
            bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated] split: prod.splits)

      have "slot ≠ slot''"
        by (metis Cons.prems(5) Cons.prems(7) Cons.prems(8) ‹y = (slot'', nodes'')›
```

```
            list.set_intros(1) list.set_intros(2))
        then have "set nodes ∩ set nodes'' = {}"
          by (metis Cons.prems(8) Cons.prems(9) ‹y = (slot'', nodes'')› list.set_intros(1)
              list.set_intros(2))

        have "hba ⊢ get_child_nodes (cast slot) →r nodes ∧ heap_is_wellformed hba ∧ type_wf hba ∧ known_ptrs
hba"
          using ‹hb ⊢ get_child_nodes (cast slot) →r nodes›
          using ‹hb ⊢ forall_M remove yc →h hba›
          using ‹hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →r yc›
          using ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs hb›
        proof(induct yc arbitrary: hb, simp)
          case (Cons a yc)
          obtain hb1 where
            hb1: "hb ⊢ remove a →h hb1" and
            hba: "hb1 ⊢ forall_M remove yc →h hba"
            using Cons
            by (auto elim!: bind_returns_heap_E)
          have "hb ⊢ get_parent a →r Some (cast slot'')"
            using Cons.prems(3) Cons.prems(4) Cons.prems(5) Cons.prems(6) local.child_parent_dual
            by auto

          moreover
          have "heap_is_wellformed hb1" and "type_wf hb1" and "known_ptrs hb1"
            using  ‹hb ⊢ remove a →h hb1› Cons.prems(4) Cons.prems(5) Cons.prems(6)
              local.remove_child_heap_is_wellformed_preserved
            apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
                bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
            using Cons.prems(4) Cons.prems(5) Cons.prems(6) hb1 local.remove_preserves_type_wf
            apply blast
            using Cons.prems(4) Cons.prems(5) Cons.prems(6) hb1 local.remove_preserves_known_ptrs
            by blast
          moreover have "hb1 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →r yc"
            using ‹hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →r a # yc› hb1
            using remove_removes_child ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs hb›
            by simp
          moreover have "hb1 ⊢ get_child_nodes (cast slot) →r nodes"
            using Cons(2) hb1 CD.set_child_nodes_get_child_nodes_different_pointers
              ‹hb ⊢ get_parent a →r Some (cast slot'')› ‹slot ≠ slot''›
            apply(auto simp add: CD.remove_child_locs_def elim!: bind_returns_heap_E
                dest!: reads_writes_separate_forwards[OF get_child_nodes_reads CD.remove_writes])[1]
            by (metis cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject)
          ultimately show ?thesis
            using ‹hb1 ⊢ forall_M remove (yc) →h hba› Cons
            by auto
        qed


        then have "hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb"
          using ‹hba ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot'')) nodes'' →h hbb›
          using ‹set nodes ∩ set nodes'' = {}›
        proof(induct nodes'' arbitrary: hba, simp)
          case (Cons a nodes'')
          then have "hba ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes" and
            "heap_is_wellformed hba" and
            "type_wf hba" and
            "known_ptrs hba"
            by auto
          obtain hba1 where
            hba1: "hba ⊢ append_child (cast slot'') a →h hba1" and
            "hba1 ⊢ forall_M (append_child (cast slot'')) nodes'' →h hbb"
            using Cons(3)
```

```
      by (auto elim!: bind_returns_heap_E)

    have "heap_is_wellformed hba1" and "type_wf hba1" and "known_ptrs hba1"
      using Cons.prems(1) hba1 local.append_child_heap_is_wellformed_preserved(1)
      apply blast
      using ‹heap_is_wellformed hba› ‹known_ptrs hba› ‹type_wf hba› hba1 local.append_child_preserves_type_w
      apply blast
      using Cons.prems(1) hba1 local.append_child_preserves_known_ptrs
      by blast

    moreover
    have "a ∉ set nodes"
      using ‹set nodes ∩ set (a # nodes'') = {}›
      by auto


    moreover
    obtain parent_opt where "hba ⊢ get_parent a →r parent_opt"
      using insert_before_child_in_heap hba1 get_parent_ok unfolding append_child_def
      by (meson Cons.prems(1) is_OK_returns_heap_I is_OK_returns_result_E)
    then
    have "hba1 ⊢ get_child_nodes (cast slot) →r nodes"
    proof (induct parent_opt)
      case None
      then show ?case
        using ‹hba ⊢ append_child (cast slot'') a →h hba1›
        using ‹hba ⊢ get_child_nodes (cast slot) →r nodes›
        using ‹slot ≠ slot''›
        apply(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def
            CD.adopt_node_locs_def remove_child_locs_def elim!: reads_writes_separate_forwards[OF
get_child_nodes_reads
              insert_before_writes])[1]
        by (metis cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject
            CD.set_child_nodes_get_child_nodes_different_pointers)
    next
      case (Some parent)
      have "parent ≠ cast slot"
        apply(rule ccontr, simp)
        using Cons(2)
        apply -
        apply(rule get_parent_child_dual[OF ‹hba ⊢ get_parent a →r Some parent›])
        apply(auto)[1]
        using ‹a ∉ set nodes› returns_result_eq
        by fastforce
      show ?case
        apply(rule reads_writes_separate_forwards)
        apply(fact get_child_nodes_reads)
        apply(fact insert_before_writes)
        apply(fact ‹hba ⊢ append_child (cast slot'') a →h hba1›[unfolded append_child_def])
        apply(fact ‹hba ⊢ get_child_nodes (cast slot) →r nodes›)
        using ‹hba ⊢ get_parent a →r Some parent› ‹parent ≠ cast slot› ‹slot ≠ slot''›
        apply(auto simp add: insert_before_locs_def adopt_node_locs_def CD.adopt_node_locs_def
            CD.remove_child_locs_def)[1]
        apply (simp_all add: CD.set_child_nodes_get_child_nodes_different_pointers
            CD.set_disconnected_nodes_get_child_nodes)
        by (metis cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject
            CD.set_child_nodes_get_child_nodes_different_pointers)
    qed
    moreover
    have "set nodes ∩ set nodes'' = {}"
      using Cons.prems(3) by auto
    ultimately show ?case
      using Cons.hyps  ‹hba1 ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot'')) nodes'' →h
```

```
hbb›
          by blast
      qed
      show ?case
        apply(rule Cons(1))
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using Cons.prems(5) apply auto[1]
        apply (simp add: remainder)
        using Cons.prems(7) apply auto[1]
        apply (simp add: True ‹nodes' = nodes› ‹x = (slot', nodes')›)
        by (metis Cons.prems(9) insert_iff list.simps(15))
    qed
  next
    case False
    then have "nodes' ≠ nodes"
      using Cons.prems(1) Cons.prems(9) ‹x = (slot', nodes')›
      by (metis assms(6) inf.idem list.set_intros(1) set_empty2)
    then
    have "(slot, nodes) ∈ set xs"
      using Cons.prems(1) ‹x = (slot', nodes')›
      by auto
    then show ?thesis
      using Cons(1)[simplified, OF ‹(slot, nodes) ∈ set xs› remainder
          ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs hb›]
      using Cons.prems(6) tag_names_same  Cons.prems(8) Cons.prems(9)
      by (smt (verit) Cons.prems(10) distinct.simps(2) list.set_intros(2))
  qed
qed
then
show ?thesis
  using h' ‹h2 ⊢ get_tag_name slot →_r ''slot''›
proof(induct shadow_root_ptrs arbitrary: h2, simp)
  case (Cons shadow_root_ptr shadow_root_ptrs)
  then have "h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes" and
    "heap_is_wellformed h2" and
    "type_wf h2" and
    "known_ptrs h2"
    by auto
  obtain host h2a h2b h2c host_children shadow_root_children where
    "h2 ⊢ get_host shadow_root_ptr →_r host" and
    "h2 ⊢ get_child_nodes (cast host) →_r host_children" and
    h2a: "h2 ⊢ forall_M remove host_children →_h h2a" and
    "h2a ⊢ get_child_nodes (cast shadow_root_ptr) →_r shadow_root_children" and
    h2b: "h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →_h h2b" and
    "h2b ⊢ remove_shadow_root host →_h h2c" and
    remainder: "h2c ⊢ forall_M(λshadow_root_ptr. Heap_Error_Monad.bind (get_host shadow_root_ptr)
      (λhost. Heap_Error_Monad.bind (get_child_nodes (cast_element_ptr2object_ptr host))
                  (λx. Heap_Error_Monad.bind (forall_M remove x)
                         (λ_. Heap_Error_Monad.bind (get_child_nodes (cast_shadow_root_ptr2object_ptr shadow_root_pt
                                (λx. Heap_Error_Monad.bind (forall_M (append_child (cast_element_ptr2object_ptr
host)) x)
                                       (λ_. remove_shadow_root host))))))
```

```
        shadow_root_ptrs
     →h h'"
    using Cons(3)
    by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_host_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated])


  have "h2 ⊢ get_shadow_root host →r Some shadow_root_ptr"
    using ‹h2 ⊢ get_host shadow_root_ptr →r host› shadow_root_host_dual
    using ‹heap_is_wellformed h2› ‹type_wf h2› by blast
  then have "h2a ⊢ get_shadow_root host →r Some shadow_root_ptr"
    using ‹h2 ⊢ forall_M remove host_children →h h2a›
    apply(induct host_children arbitrary: h2)
    by(auto simp add: set_disconnected_nodes_get_shadow_root set_child_nodes_get_shadow_root
        CD.remove_child_locs_def elim!: bind_returns_heap_E
        dest!: reads_writes_separate_forwards[OF get_shadow_root_reads CD.remove_writes])
  then have "h2b ⊢ get_shadow_root host →r Some shadow_root_ptr"
    using ‹h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →h h2b›
    apply(induct shadow_root_children arbitrary: h2a)
    by(auto simp add: set_disconnected_nodes_get_shadow_root set_child_nodes_get_shadow_root
        append_child_def insert_before_locs_def adopt_node_locs_def CD.adopt_node_locs_def
        CD.remove_child_locs_def elim!: bind_returns_heap_E
        dest!: reads_writes_separate_forwards[OF get_shadow_root_reads insert_before_writes]
        split: if_splits)

  have "host ≠ slot"
  proof (rule ccontr, simp)
    assume "host = slot"
    show False
      using get_host_valid_tag_name[OF ‹heap_is_wellformed h2› ‹type_wf h2›
          ‹h2 ⊢ get_host shadow_root_ptr →r host›[unfolded ‹host = slot›] ‹h2 ⊢ get_tag_name slot →r
''slot''›]
      by(simp)
  qed

  have "heap_is_wellformed h2a" and "type_wf h2a" and "known_ptrs h2a"
    using ‹heap_is_wellformed h2› and ‹type_wf h2› and ‹known_ptrs h2› ‹h2 ⊢ forall_M remove host_children
→h h2a›
    apply(induct host_children arbitrary: h2)
    apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
      remove_child_preserves_known_ptrs apply metis
    apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
      remove_child_preserves_known_ptrs apply metis
    apply(auto simp add: CD.remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
      remove_child_preserves_known_ptrs apply metis
    done
  then
  have "heap_is_wellformed h2b" and "type_wf h2b" and "known_ptrs h2b"
    using ‹h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →h h2b›
    apply(induct shadow_root_children arbitrary: h2a)
    apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
```

```
    apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
    apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
    apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
    using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf insert_before_preserves_know
    apply(metis)
    apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
    using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf insert_before_preserves_know
    apply(metis)
    apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
    using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf insert_before_preserves_know
    apply(metis)
    done
  then
  have "heap_is_wellformed h2c" and "type_wf h2c" and "known_ptrs h2c"
    using remove_shadow_root_preserves ‹h2b ⊢ remove_shadow_root host →_h h2c›
    by blast+

  moreover
  have "h2a ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes"
    using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes›
    using ‹h2 ⊢ forall_M remove host_children →_h h2a›
    using ‹h2 ⊢ get_child_nodes (cast host) →_r host_children›
    using ‹heap_is_wellformed h2› ‹type_wf h2› ‹known_ptrs h2›
  proof (induct host_children arbitrary: h2, simp)
    case (Cons a host_children)
    obtain h21 where "h2 ⊢ remove a →_h h21" and
      "h21 ⊢ forall_M remove host_children →_h h2a"
      using Cons(3)
      by(auto elim!: bind_returns_heap_E)

    have "heap_is_wellformed h21" and "type_wf h21" and "known_ptrs h21"
      using Cons.prems(4) Cons.prems(5) Cons.prems(6) ‹h2 ⊢ remove a →_h h21› local.remove_heap_is_wellformed_pr
      apply blast
      using Cons.prems(4) Cons.prems(5) Cons.prems(6) ‹h2 ⊢ remove a →_h h21› local.remove_preserves_type_wf
      apply blast
      using Cons.prems(4) Cons.prems(5) Cons.prems(6) ‹h2 ⊢ remove a →_h h21› local.remove_preserves_known_ptrs
      by blast
    have "h2 ⊢ get_parent a →_r Some (cast host)"
      using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr host) →_r a # host_children›
      using ‹heap_is_wellformed h2› ‹type_wf h2› ‹known_ptrs h2› child_parent_dual
      using heap_is_wellformed_def by auto
    then have "h21 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes"
      using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes› ‹host ≠ slot›
      using ‹h2 ⊢ remove a →_h h21›
      apply(auto simp add: CD.remove_child_locs_def CD.set_disconnected_nodes_get_child_nodes
          dest!: reads_writes_preserved[OF get_child_nodes_reads CD.remove_writes])[1]
      by (meson cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject
          CD.set_child_nodes_get_child_nodes_different_pointers)
    moreover have "h21 ⊢ get_child_nodes (cast_element_ptr2object_ptr host) →_r host_children"
      using ‹h2 ⊢ remove a →_h h21›  remove_removes_child[OF ‹heap_is_wellformed h2› ‹type_wf h2›
          ‹known_ptrs h2› ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr host) →_r a # host_children›]
      by blast
    ultimately show ?case
      using ‹heap_is_wellformed h21› and ‹type_wf h21› and ‹known_ptrs h21›
        ‹h21 ⊢ forall_M remove host_children →_h h2a› Cons(1)
      using Cons.prems(3) Cons.prems(4) Cons.prems(5) Cons.prems(6) heap_is_wellformed_def
        ‹h2 ⊢ remove a →_h h21› remove_removes_child
      by blast
  qed

  then
  have "h2b ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes"
    using ‹h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →_h h2b›
```

```
      using ‹h2a ⊢ get_child_nodes (cast shadow_root_ptr) →r shadow_root_children›
      using ‹heap_is_wellformed h2a› ‹type_wf h2a› ‹known_ptrs h2a›
    proof(induct shadow_root_children arbitrary: h2a, simp)
      case (Cons a shadow_root_children)
      obtain h2a1 where "h2a ⊢ append_child (cast_element_ptr2object_ptr host) a →h h2a1" and
        "h2a1 ⊢ forall_M (append_child (cast_element_ptr2object_ptr host)) (shadow_root_children) →h h2b"
        using Cons(3)
        by(auto elim!: bind_returns_heap_E)

      have "heap_is_wellformed h2a1" and "type_wf h2a1" and "known_ptrs h2a1"
        using Cons.prems(4) Cons.prems(5) Cons.prems(6) ‹h2a ⊢ append_child (cast_element_ptr2object_ptr host)
a →h h2a1›
          local.append_child_heap_is_wellformed_preserved by blast+
      moreover have "h2a1 ⊢ get_child_nodes (cast shadow_root_ptr) →r shadow_root_children"
        using ‹h2a ⊢ get_child_nodes (cast shadow_root_ptr) →r a # shadow_root_children›
        using insert_before_removes_child ‹h2a ⊢ append_child (cast_element_ptr2object_ptr host) a →h h2a1›[unfolded
append_child_def]
        using ‹heap_is_wellformed h2a› ‹type_wf h2a› ‹known_ptrs h2a›
        using cast_document_ptr_not_node_ptr(2) by blast
      moreover have "h2a ⊢ get_parent a →r Some (cast shadow_root_ptr)"
        using ‹h2a ⊢ get_child_nodes (cast_shadow_root_ptr2object_ptr shadow_root_ptr) →r a # shadow_root_children›
        using ‹heap_is_wellformed h2a› ‹type_wf h2a› ‹known_ptrs h2a› child_parent_dual
        using heap_is_wellformed_def by auto
      then have "h2a1 ⊢ get_child_nodes (cast slot) →r nodes"
        using ‹h2a ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes›
        using ‹h2a ⊢ append_child (cast_element_ptr2object_ptr host) a →h h2a1› ‹host ≠ slot›
        apply(auto simp add:  set_disconnected_nodes_get_child_nodes append_child_def
            insert_before_locs_def adopt_node_locs_def CD.adopt_node_locs_def CD.remove_child_locs_def
            elim!: bind_returns_heap_E dest!: reads_writes_preserved[OF get_child_nodes_reads insert_before_writes]
        using CD.set_child_nodes_get_child_nodes_different_pointers cast_element_ptr2node_ptr_inject
          cast_node_ptr2object_ptr_inject cast_document_ptr_not_node_ptr(2)
        by metis+
      ultimately
      show ?case
        using Cons(1) ‹h2a1 ⊢ forall_M (append_child (cast_element_ptr2object_ptr host)) shadow_root_children
→h h2b›
        by blast
    qed
    then
    have "h2c ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes"
      using ‹h2b ⊢ remove_shadow_root host →h h2c›
      by(auto simp add: remove_shadow_root_get_child_nodes_different_pointers[OF
            cast_document_ptr_not_node_ptr(2)] dest!: reads_writes_separate_forwards[OF get_child_nodes_reads
            remove_shadow_root_writes])

    moreover
    have "h2a ⊢ get_tag_name slot →r ''slot''"
      using h2a ‹h2 ⊢ get_tag_name slot →r ''slot''›
      apply(induct host_children arbitrary: h2)
      by(auto simp add: CD.remove_child_locs_def set_disconnected_nodes_get_tag_name
          set_child_nodes_get_tag_name dest!: reads_writes_separate_forwards[OF get_tag_name_reads CD.remove_writes
          elim!: bind_returns_heap_E)
    then
    have "h2b ⊢ get_tag_name slot →r ''slot''"
      using h2b
      apply(induct shadow_root_children arbitrary: h2a)
      by(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def
          CD.adopt_node_locs_def CD.remove_child_locs_def set_disconnected_nodes_get_tag_name
          set_child_nodes_get_tag_name dest!: reads_writes_separate_forwards[OF get_tag_name_reads insert_before_wr
          elim!: bind_returns_heap_E split: if_splits)
    then
    have "h2c ⊢ get_tag_name slot →r ''slot''"
      using ‹h2b ⊢ remove_shadow_root host →h h2c›
```

```
      by(auto simp add: remove_shadow_root_get_tag_name dest!:
          reads_writes_separate_forwards[OF get_tag_name_reads remove_shadow_root_writes])

    ultimately show ?case
      using Cons(1) remainder
      by auto
  qed
qed
end

interpretation i_assigned_nodes_component?: l_assigned_nodes_component_Shadow_DOM
  type_wf get_tag_name get_tag_name_locs known_ptr get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_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 get_parent get_parent_locs get_mode get_mode_locs get_attribute
  get_attribute_locs first_in_tree_order find_slot assigned_slot known_ptrs to_tree_order assigned_nodes
  assigned_nodes_flatten flatten_dom get_root_node get_root_node_locs remove insert_before insert_before_locs
  append_child remove_shadow_root remove_shadow_root_locs set_shadow_root set_shadow_root_locs remove_child
  remove_child_locs get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe get_ancestors
  get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name get_owner_document
  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 set_child_nodes set_child_nodes_locs
  by(auto simp add: l_assigned_nodes_component_Shadow_DOM_def instances)
declare l_assigned_nodes_component_Shadow_DOM_axioms [instances]
```

get_owner_document

```
locale l_get_owner_document_component_Shadow_DOM =
  l_get_owner_document_wf_Shadow_DOM +
  l_get_dom_component_Core_DOM
begin
lemma get_owner_document_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_owner_document ptr →_r owner_document"
  assumes "¬is_document_ptr_kind |h ⊢ get_root_node ptr|_r"
  shows "set |h ⊢ get_dom_component ptr|_r ∩ set |h ⊢ get_dom_component (cast owner_document)|_r = {}"
proof -
  have "owner_document |∈| document_ptr_kinds h"
    using assms(1) assms(2) assms(3) assms(4) get_owner_document_owner_document_in_heap by blast
  have "ptr |∈| object_ptr_kinds h"
    by (meson assms(4) is_OK_returns_result_I local.get_owner_document_ptr_in_heap)
  obtain root where root: "h ⊢ get_root_node ptr →_r root"
    by (meson assms(1) assms(2) assms(3) assms(4) is_OK_returns_result_I
        local.get_owner_document_ptr_in_heap local.get_root_node_ok returns_result_select_result)
  then obtain to where to: "h ⊢ to_tree_order root →_r to"
    by (meson assms(1) assms(2) assms(3) is_OK_returns_result_E local.get_root_node_root_in_heap
        local.to_tree_order_ok)
  then have "∀p ∈ set to. ¬is_document_ptr_kind p"
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(5) document_ptr_casts_commute3
        local.to_tree_order_node_ptrs node_ptr_no_document_ptr_cast root select_result_I2)
  then have "cast owner_document ∉ set |h ⊢ get_dom_component ptr|_r"
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(5) document_ptr_document_ptr_cast
        is_OK_returns_result_I l_get_dom_component_Core_DOM.get_dom_component_ok local.get_dom_component_root_node
        local.get_root_node_not_node_same local.get_root_node_ptr_in_heap local.l_get_dom_component_Core_DOM_axiom
        node_ptr_no_document_ptr_cast returns_result_select_result root select_result_I2)
  then have "|h ⊢ get_dom_component ptr|_r ≠ |h ⊢ get_dom_component (cast owner_document)|_r"
    by (metis (no_types, lifting) ‹owner_document |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
        document_ptr_kinds_commutes l_get_dom_component_Core_DOM.get_dom_component_ok local.get_dom_component_ptr
        local.l_get_dom_component_Core_DOM_axioms returns_result_select_result)
  then show ?thesis
    by (meson ‹owner_document |∈| document_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2)
        assms(3) document_ptr_kinds_commutes l_get_dom_component_Core_DOM.get_dom_component_no_overlap
        l_get_dom_component_Core_DOM.get_dom_component_ok local.l_get_dom_component_Core_DOM_axioms returns_result
```

qed

end

interpretation *i_get_owner_document_component?: l_get_owner_document_component*$_{Shadow_DOM}$
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 to_tree_orde
get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe get_element_by_id
get_elements_by_class_name get_elements_by_tag_name
by *(auto simp add: l_get_owner_document_component*$_{Shadow_DOM}$*_def instances)*
declare *l_get_owner_document_component*$_{Shadow_DOM}$*_axioms [instances]*

definition *is_shadow_root_component :: "(_) object_ptr list ⇒ bool"*
where
"is_shadow_root_component c = is_shadow_root_ptr_kind (hd c)"

end

2.6 Shadow SC DOM Components II (Shadow_DOM_SC_DOM_Components)

theory *Shadow_DOM_SC_DOM_Components*
imports
Core_DOM_SC_DOM_Components
Shadow_DOM_DOM_Components
begin

2.7 Shadow root scope components (Shadow_DOM_SC_DOM_Components)

2.7.1 get_scope_component

global_interpretation *l_get_scdom_component*$_{Core_DOM}$*_defs get_owner_document get_disconnected_nodes*
get_disconnected_nodes_locs to_tree_order
defines *get_scdom_component = a_get_scdom_component*
and *is_strongly_scdom_component_safe = a_is_strongly_scdom_component_safe*
and *is_weakly_scdom_component_safe = a_is_weakly_scdom_component_safe*
.
interpretation *i_get_scdom_component?: l_get_scdom_component*$_{Core_DOM}$
get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
get_owner_document get_disconnected_nodes get_disconnected_nodes_locs to_tree_order
by *(auto simp add: l_get_scdom_component*$_{Core_DOM}$*_def get_scdom_component_def*
is_strongly_scdom_component_safe_def is_weakly_scdom_component_safe_def instances)
declare *l_get_scdom_component*$_{Core_DOM}$*_axioms [instances]*

get_component

locale *l_get_dom_component_get_scdom_component*$_{Shadow_DOM}$ *=*
l_get_scdom_component$_{Core_DOM}$ *+*
l_get_dom_component$_{Core_DOM}$ *+*
l_heap_is_wellformed +
l_get_owner_document +
l_get_owner_document_wf +
l_get_disconnected_nodes +
l_to_tree_order +
l_known_ptr +
l_known_ptrs +
l_get_owner_document_wf_get_root_node_wf +

```
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
begin

lemma known_ptr_node_or_document: "known_ptr ptr ⟹ is_node_ptr_kind ptr ∨ is_document_ptr_kind 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)

lemma get_scdom_component_subset_get_dom_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "h ⊢ get_dom_component ptr →r c"
  shows "set c ⊆ set sc"
proof -
  obtain document disc_nodes tree_order disconnected_tree_orders where document: "h ⊢ get_owner_document
ptr →r document"
    and disc_nodes: "h ⊢ get_disconnected_nodes document →r disc_nodes"
    and tree_order: "h ⊢ to_tree_order (cast document) →r tree_order"
    and disconnected_tree_orders: "h ⊢ map_M (to_tree_order ∘ cast) disc_nodes →r disconnected_tree_orders"
    and sc: "sc = tree_order @ (concat disconnected_tree_orders)"
    using assms(4)
    by(auto simp add: get_scdom_component_def elim!: bind_returns_result_E
        elim!: bind_returns_result_E2[rotated, OF get_owner_document_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF to_tree_order_pure, rotated]
        )

  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    and c: "h ⊢ to_tree_order root_ptr →r c"
    using assms(5)
    by(auto simp add: get_dom_component_def elim!: bind_returns_result_E2[rotated, OF get_root_node_pure,
rotated])

  show ?thesis
  proof (cases "is_document_ptr_kind root_ptr")
    case True
    then have "cast document = root_ptr"
      using get_root_node_document assms(1) assms(2) assms(3) root_ptr document
      by (metis document_ptr_casts_commute3 returns_result_eq)
    then have "c = tree_order"
      using tree_order c
      by auto
    then show ?thesis
      by(simp add: sc)
  next
    case False
    moreover have "root_ptr |∈| object_ptr_kinds h"
      using assms(1) assms(2) assms(3) local.get_root_node_root_in_heap root_ptr by blast
    ultimately have "is_node_ptr_kind root_ptr"
      using assms(3) known_ptrs_known_ptr known_ptr_node_or_document
      by auto
    then obtain root_node_ptr where root_node_ptr: "root_ptr = cast_{node_ptr2object_ptr} root_node_ptr"
      by (metis node_ptr_casts_commute3)
    then have "h ⊢ get_owner_document root_ptr →r document"
      using get_root_node_same_owner_document
      using assms(1) assms(2) assms(3) document root_ptr by blast
    then have "root_node_ptr ∈ set disc_nodes"
      using assms(1) assms(2) assms(3) disc_nodes in_disconnected_nodes_no_parent root_node_ptr
      using local.get_root_node_same_no_parent root_ptr by blast
    then have "c ∈ set disconnected_tree_orders"
      using c root_node_ptr
      using map_M_pure_E[OF disconnected_tree_orders]
      by (metis (mono_tags, lifting) comp_apply local.to_tree_order_pure select_result_I2)
    then show ?thesis
```

```
    by(auto simp add: sc)
  qed
qed

lemma get_scdom_component_ptrs_same_owner_document:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →_r sc"
  assumes "ptr' ∈ set sc"
  assumes "h ⊢ get_owner_document ptr →_r owner_document"
  shows "h ⊢ get_owner_document ptr' →_r owner_document"
proof -
  obtain document disc_nodes tree_order disconnected_tree_orders where document: "h ⊢ get_owner_document
ptr →_r document"
    and disc_nodes: "h ⊢ get_disconnected_nodes document →_r disc_nodes"
    and tree_order: "h ⊢ to_tree_order (cast document) →_r tree_order"
    and disconnected_tree_orders: "h ⊢ map_M (to_tree_order ∘ cast) disc_nodes →_r disconnected_tree_orders"
    and sc: "sc = tree_order @ (concat disconnected_tree_orders)"
    using assms(4)
    by(auto simp add: get_scdom_component_def elim!: bind_returns_result_E
        elim!: bind_returns_result_E2[rotated, OF get_owner_document_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF to_tree_order_pure, rotated]
        )
  show ?thesis
  proof (cases "ptr' ∈ set tree_order")
    case True
    have "owner_document = document"
      using assms(6) document by fastforce
    then show ?thesis
      by (metis (no_types) True assms(1) assms(2) assms(3) cast_document_ptr2object_ptr_inject document
          document_ptr_casts_commute3 document_ptr_document_ptr_cast document_ptr_kinds_commutes
          local.get_owner_document_owner_document_in_heap local.get_root_node_document
          local.get_root_node_not_node_same local.to_tree_order_same_root node_ptr_no_document_ptr_cast
          tree_order)
  next
    case False
    then obtain disconnected_tree_order where disconnected_tree_order:
      "ptr' ∈ set disconnected_tree_order" and "disconnected_tree_order ∈ set disconnected_tree_orders"
      using sc ‹ptr' ∈ set sc›
      by auto
    obtain root_ptr' where
      root_ptr': "root_ptr' ∈ set disc_nodes" and
      "h ⊢ to_tree_order (cast root_ptr') →_r disconnected_tree_order"
      using map_M_pure_E2[OF disconnected_tree_orders ‹disconnected_tree_order ∈ set disconnected_tree_orders›]
      by (metis comp_apply local.to_tree_order_pure)
    have "¬(∃parent ∈ fset (object_ptr_kinds h). root_ptr' ∈ set |h ⊢ get_child_nodes parent|_r)"
      using disc_nodes
      by (meson assms(1) assms(2) assms(3) disjoint_iff_not_equal local.get_child_nodes_ok
          local.heap_is_wellformed_children_disc_nodes_different local.known_ptrs_known_ptr
          returns_result_select_result root_ptr')
    then
    have "h ⊢ get_parent root_ptr' →_r None"
      using disc_nodes
      by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3) local.get_parent_child_dual
          local.get_parent_ok local.get_parent_parent_in_heap local.heap_is_wellformed_disc_nodes_in_heap
          returns_result_select_result root_ptr' select_result_I2 split_option_ex)
    then have "h ⊢ get_root_node ptr' →_r cast root_ptr'"
      using ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr root_ptr') →_r disconnected_tree_order› assms(1)
        assms(2) assms(3) disconnected_tree_order local.get_root_node_no_parent local.to_tree_order_get_root_node
        local.to_tree_order_ptr_in_result
      by blast
    then have "h ⊢ get_owner_document (cast root_ptr') →_r document"
      using assms(1) assms(2) assms(3) disc_nodes local.get_owner_document_disconnected_nodes root_ptr'
```

```
      by blast

    then have "h ⊢ get_owner_document ptr' →r document"
      using ‹h ⊢ get_root_node ptr' →r cast_node_ptr2object_ptr root_ptr'› assms(1) assms(2) assms(3)
        local.get_root_node_same_owner_document
      by blast
    then show ?thesis
      using assms(6) document returns_result_eq by force
  qed
qed

lemma get_scdom_component_ptrs_same_scope_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  shows "h ⊢ get_scdom_component ptr' →r sc"
proof -
  obtain document disc_nodes tree_order disconnected_tree_orders where document:
    "h ⊢ get_owner_document ptr →r document"
    and disc_nodes: "h ⊢ get_disconnected_nodes document →r disc_nodes"
    and tree_order: "h ⊢ to_tree_order (cast document) →r tree_order"
    and disconnected_tree_orders: "h ⊢ map_M (to_tree_order ∘ cast) disc_nodes →r disconnected_tree_orders"
    and sc: "sc = tree_order @ (concat disconnected_tree_orders)"
    using assms(4)
    by(auto simp add: get_scdom_component_def elim!: bind_returns_result_E
        elim!: bind_returns_result_E2[rotated, OF get_owner_document_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF get_disconnected_nodes_pure, rotated]
        elim!: bind_returns_result_E2[rotated, OF to_tree_order_pure, rotated]
        )
  show ?thesis
  proof (cases "ptr' ∈ set tree_order")
    case True
    then have "h ⊢ get_owner_document ptr' →r document"
      by (metis assms(1) assms(2) assms(3) cast_document_ptr2object_ptr_inject document document_ptr_casts_commute3
          document_ptr_kinds_commutes known_ptr_node_or_document local.get_owner_document_owner_document_in_heap
          local.get_root_node_document local.get_root_node_not_node_same local.known_ptrs_known_ptr
          local.to_tree_order_get_root_node local.to_tree_order_ptr_in_result node_ptr_no_document_ptr_cast
tree_order)
    then show ?thesis
      using disc_nodes tree_order disconnected_tree_orders sc
      by(auto simp add: get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  next
    case False
    then obtain disconnected_tree_order where disconnected_tree_order:
      "ptr' ∈ set disconnected_tree_order" and "disconnected_tree_order ∈ set disconnected_tree_orders"
      using sc ‹ptr' ∈ set sc›
      by auto
    obtain root_ptr' where
      root_ptr': "root_ptr' ∈ set disc_nodes" and
      "h ⊢ to_tree_order (cast root_ptr') →r disconnected_tree_order"
      using map_M_pure_E2[OF disconnected_tree_orders ‹disconnected_tree_order ∈ set disconnected_tree_orders›]
      by (metis comp_apply local.to_tree_order_pure)
    have "¬(∃parent ∈ fset (object_ptr_kinds h). root_ptr' ∈ set |h ⊢ get_child_nodes parent|r)"
      using disc_nodes
      by (meson assms(1) assms(2) assms(3) disjoint_iff_not_equal local.get_child_nodes_ok
          local.heap_is_wellformed_children_disc_nodes_different local.known_ptrs_known_ptr
          returns_result_select_result root_ptr')
    then
    have "h ⊢ get_parent root_ptr' →r None"
      using disc_nodes
      by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3) local.get_parent_child_dual
          local.get_parent_ok local.get_parent_parent_in_heap local.heap_is_wellformed_disc_nodes_in_heap
          returns_result_select_result root_ptr' select_result_I2 split_option_ex)
```

```
    then have "h ⊢ get_root_node ptr' →r cast root_ptr'"
      using ‹h ⊢ to_tree_order (cast_node_ptr2object_ptr root_ptr') →r disconnected_tree_order› assms(1)
        assms(2) assms(3) disconnected_tree_order local.get_root_node_no_parent local.to_tree_order_get_root_node
        local.to_tree_order_ptr_in_result
      by blast
    then have "h ⊢ get_owner_document (cast root_ptr') →r document"
      using assms(1) assms(2) assms(3) disc_nodes local.get_owner_document_disconnected_nodes root_ptr'
      by blast

    then have "h ⊢ get_owner_document ptr' →r document"
      using ‹h ⊢ get_root_node ptr' →r cast_node_ptr2object_ptr root_ptr'› assms(1) assms(2) assms(3)
        local.get_root_node_same_owner_document
      by blast
    then show ?thesis
      using disc_nodes tree_order disconnected_tree_orders sc
      by(auto simp add: get_scdom_component_def intro!: bind_pure_returns_result_I map_M_pure_I)
  qed
qed

lemma get_scdom_component_ok:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_scdom_component ptr)"
  using assms
  apply(auto simp add: get_scdom_component_def intro!: bind_is_OK_pure_I map_M_pure_I map_M_ok_I)[1]
  using get_owner_document_ok
     apply blast
    apply (simp add: local.get_disconnected_nodes_ok local.get_owner_document_owner_document_in_heap)
   apply (simp add: local.get_owner_document_owner_document_in_heap local.to_tree_order_ok)
  using local.heap_is_wellformed_disc_nodes_in_heap local.to_tree_order_ok node_ptr_kinds_commutes
  by blast


lemma get_scdom_component_ptr_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  shows "ptr' |∈| object_ptr_kinds h"
  using assms
  apply(auto simp add: get_scdom_component_def elim!: bind_returns_result_E2 intro!: map_M_pure_I)[1]
  using local.to_tree_order_ptrs_in_heap apply blast
  by (metis (no_types, lifting) assms(4) assms(5) bind_returns_result_E get_scdom_component_impl
      get_scdom_component_ptrs_same_scope_component is_OK_returns_result_I
      l_get_scdom_component_Core_DOM_defs.a_get_scdom_component_def local.get_owner_document_ptr_in_heap)

lemma get_scdom_component_contains_get_dom_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
  obtains c where "h ⊢ get_dom_component ptr' →r c" and "set c ⊆ set sc"
proof -
  have "h ⊢ get_scdom_component ptr' →r sc"
    using assms(1) assms(2) assms(3) assms(4) assms(5) get_scdom_component_ptrs_same_scope_component
    by blast
  then show ?thesis
    by (meson assms(1) assms(2) assms(3) assms(5) get_scdom_component_ptr_in_heap
        get_scdom_component_subset_get_dom_component is_OK_returns_result_E local.get_dom_component_ok that)
qed

lemma get_scdom_component_owner_document_same:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "ptr' ∈ set sc"
```

```
  obtains owner_document where "h ⊢ get_owner_document ptr' →r owner_document" and "cast owner_document
∈ set sc"
  using assms
  apply(auto simp add: get_scdom_component_def elim!: bind_returns_result_E2 intro!: map_M_pure_I)[1]
   apply (metis (no_types, lifting) assms(4) assms(5) document_ptr_casts_commute3
      document_ptr_document_ptr_cast get_scdom_component_contains_get_dom_component local.get_dom_component_ptr
      local.get_dom_component_root_node_same local.get_dom_component_to_tree_order local.get_root_node_document
      local.get_root_node_not_node_same local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap
      node_ptr_no_document_ptr_cast)
  apply(rule map_M_pure_E2)
     apply(simp)
    apply(simp)
   apply(simp)
  by (smt (verit) assms(4) assms(5) comp_apply get_scdom_component_ptr_in_heap is_OK_returns_result_E
      local.get_owner_document_disconnected_nodes local.get_root_node_ok local.get_root_node_same_owner_document
      local.to_tree_order_get_root_node local.to_tree_order_ptr_in_result)

lemma get_scdom_component_different_owner_documents:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_owner_document ptr →r owner_document"
  assumes "h ⊢ get_owner_document ptr' →r owner_document'"
  assumes "owner_document ≠ owner_document'"
  shows "set |h ⊢ get_scdom_component ptr|r ∩ set |h ⊢ get_scdom_component ptr'|r = {}"
  using assms get_scdom_component_ptrs_same_owner_document
  by (smt (verit) disjoint_iff_not_equal get_scdom_component_ok is_OK_returns_result_I
      local.get_owner_document_ptr_in_heap returns_result_eq returns_result_select_result)
end

interpretation i_get_dom_component_get_scdom_component?: l_get_dom_component_get_scdom_component_Shadow_DOM
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_owner_document
  get_disconnected_nodes get_disconnected_nodes_locs to_tree_order heap_is_wellformed parent_child_rel
  type_wf known_ptr known_ptrs get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node
  get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name
  by(auto simp add: l_get_dom_component_get_scdom_component_Shadow_DOM_def l_get_dom_component_get_scdom_component
instances)
declare l_get_dom_component_get_scdom_component_Shadow_DOM_axioms [instances]

lemma get_dom_component_get_scdom_component_is_l_get_dom_component_get_scdom_component [instances]:
  "l_get_dom_component_get_scdom_component get_owner_document heap_is_wellformed type_wf known_ptr known_ptrs
get_scdom_component get_dom_component"
  apply(auto simp add: l_get_dom_component_get_scdom_component_def
      l_get_dom_component_get_scdom_component_axioms_def instances)[1]
  using get_scdom_component_subset_get_dom_component apply fast
  using get_scdom_component_ptrs_same_scope_component apply fast
  using get_scdom_component_ptrs_same_owner_document apply fast
  using get_scdom_component_ok apply fast
  using get_scdom_component_ptr_in_heap apply fast
  using get_scdom_component_contains_get_dom_component apply blast
  using get_scdom_component_owner_document_same apply blast
  using get_scdom_component_different_owner_documents apply fast
  done
```

2.7.2 attach_shadow_root

```
lemma attach_shadow_root_not_strongly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'ShadowRoot::{equal,linorder}) heap" and
    h' and host and new_shadow_root_ptr where
```

```
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ attach_shadow_root host m →r new_shadow_root_ptr →h h'" and
    "¬ is_strongly_scdom_component_safe {cast host} {cast new_shadow_root_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'ShadowRoot::{equal,linorder}) heap"
  let ?P = "do {
    doc ← create_document;
    create_element doc ''div''
}"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and host="?e1"])
    by code_simp+
qed

locale l_get_scdom_component_attach_shadow_root_Core_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_dom_component_attach_shadow_root_Core_DOM +
  l_get_dom_component_get_scdom_component_Shadow_DOM
begin
lemma attach_shadow_root_is_weakly_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →h h'"
  assumes "ptr ≠ cast |h ⊢ attach_shadow_root element_ptr shadow_root_mode|_r"
  assumes "ptr ∉ set |h ⊢ get_scdom_component (cast element_ptr)|_r"
  shows "preserved (get_M_Object ptr getter) h h'"
proof -
  have "element_ptr |∈| element_ptr_kinds h"
    by (meson assms(4) is_OK_returns_heap_I local.attach_shadow_root_element_ptr_in_heap)
  obtain sc where sc: "h ⊢ get_scdom_component (cast element_ptr) →r sc"
    using get_scdom_component_ok
    by (meson assms(1) assms(2) assms(3) assms(4) element_ptr_kinds_commutes is_OK_returns_heap_I
        local.attach_shadow_root_element_ptr_in_heap node_ptr_kinds_commutes select_result_I)
  then have "ptr ∉ set |h ⊢ get_dom_component (cast element_ptr)|_r"
    by (metis (no_types, lifting) ‹element_ptr |∈| element_ptr_kinds h› assms(1) assms(2) assms(3)
        assms(6) element_ptr_kinds_commutes local.get_dom_component_ok
        local.get_scdom_component_subset_get_dom_component node_ptr_kinds_commutes
        returns_result_select_result select_result_I2 set_rev_mp)
  then show ?thesis
    using assms(1) assms(2) assms(3) assms(4) assms(5) local.attach_shadow_root_is_weakly_dom_component_safe
    by blast
qed

lemma attach_shadow_root_is_weakly_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →r result"
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →h h'"
  shows "is_weakly_scdom_component_safe {cast element_ptr} {cast result} h h'"
proof -

  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
    h': "h3 ⊢ set_shadow_root element_ptr (Some new_shadow_root_ptr) →h h'"
    using assms(5)
    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 shadow_root_mode →r new_shadow_root_ptr"
    using new_shadow_root_ptr h2 h3 h'
    using assms(5)
    by(auto simp add: attach_shadow_root_def intro!: bind_returns_result_I
        bind_pure_returns_result_I[OF get_tag_name_pure] bind_pure_returns_result_I[OF get_shadow_root_pure]
        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 "object_ptr_kinds h2 = {|cast result|} |∪| object_ptr_kinds h"
    using h2 new_ShadowRoot_M_new_ptr
    using new_shadow_root_ptr
    using assms(4) by auto
  moreover
  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_mode_writes h3])
    using set_mode_pointers_preserved
      apply blast
    by (auto simp add: reflp_def transp_def)
  moreover
  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_shadow_root_writes h'])
    using set_shadow_root_pointers_preserved
      apply blast
    by (auto simp add: reflp_def transp_def)
  ultimately have "object_ptr_kinds h' = {|cast result|} |∪| object_ptr_kinds h"
    by simp
  moreover
  have "result |∉| shadow_root_ptr_kinds h"
    using h2 new_ShadowRoot_M_ptr_not_in_heap new_shadow_root_ptr
    using ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →r new_shadow_root_ptr› assms(4)
      returns_result_eq by metis
  ultimately
  show ?thesis
    using assms
    apply(auto simp add: is_weakly_scdom_component_safe_def Let_def)[1]
    using attach_shadow_root_is_weakly_component_safe_step
    by (smt (verit) document_ptr_kinds_commutes local.get_scdom_component_impl select_result_I2
        shadow_root_ptr_kinds_commutes)
qed
end

interpretation i_get_scdom_component_attach_shadow_root?: l_get_scdom_component_attach_shadow_root_Core_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe to_tree_order
  get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  set_shadow_root set_shadow_root_locs set_mode set_mode_locs attach_shadow_root get_disconnected_nodes
  get_disconnected_nodes_locs get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_locs
  by(auto simp add: l_get_scdom_component_attach_shadow_root_Core_DOM_def instances)
declare l_get_scdom_component_attach_shadow_root_Core_DOM_axioms [instances]
```

2.7.3 get_shadow_root

```
lemma get_shadow_root_not_weakly_scdom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder},
```

```
'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    element_ptr and shadow_root_ptr_opt and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_shadow_root_safe element_ptr →r shadow_root_ptr_opt →h h'" and
    "¬ is_weakly_scdom_component_safe {cast element_ptr} (cast ` set_option shadow_root_ptr_opt) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linor
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder},
'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e1
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and element_ptr="?e1"])
    by code_simp+
qed

locale l_get_shadow_root_scope_component_Shadow_DOM =
  l_get_dom_component_get_scdom_component +
  l_get_shadow_root_component_Shadow_DOM +
  l_create_document +
  l_get_owner_document_wf_Shadow_DOM +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_mode +
  l_get_scdom_component_Core_DOM +
  l_get_shadow_root_safe_Shadow_DOM +
  assumes known_ptrs_impl: "known_ptrs = a_known_ptrs"
begin
lemma get_shadow_root_components_disjunct:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "h ⊢ get_shadow_root host →r Some shadow_root_ptr"
  shows "set |h ⊢ get_scdom_component (cast host)|_r ∩ set |  h ⊢ get_scdom_component (cast shadow_root_ptr)|_r
= {}"
proof -
  obtain owner_document where owner_document: "h ⊢ get_owner_document (cast host) →r owner_document"
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(5) element_ptr_kinds_commutes
        is_OK_returns_result_E is_OK_returns_result_I local.get_dom_component_ok local.get_dom_component_ptr
        local.get_scdom_component_ok local.get_scdom_component_owner_document_same
        local.get_scdom_component_subset_get_dom_component local.get_shadow_root_ptr_in_heap node_ptr_kinds_commute
        subset_code(1))
  have "owner_document ≠ cast shadow_root_ptr"
  proof
    assume "owner_document = cast shadow_root_ptr"
    then have "(cast owner_document, cast host) ∈
```

```
(parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*"
      using get_owner_document_rel owner_document
      by (metis (no_types, lifting) assms(1) assms(2) assms(3) cast_document_ptr_not_node_ptr(2)
          in_rtrancl_UnI inf_sup_aci(6) inf_sup_aci(7))
    then have "(cast shadow_root_ptr, cast host) ∈
(parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)*"
      by (simp add: ‹owner_document = cast shadow_root_ptr›)
    moreover have "(cast host, cast shadow_root_ptr) ∈ a_host_shadow_root_rel h"
      by (metis (mono_tags, lifting) assms(5) is_OK_returns_result_I local.get_shadow_root_ptr_in_heap
          local.a_host_shadow_root_rel_def mem_Collect_eq pair_imageI prod.simps(2) select_result_I2)
    then have "(cast host, cast shadow_root_ptr) ∈
(parent_child_rel h ∪ local.a_host_shadow_root_rel h ∪ a_ptr_disconnected_node_rel h)"
      by (simp)
    ultimately show False
      using assms(1)
      unfolding heap_is_wellformed_def ‹owner_document = cast shadow_root_ptr› acyclic_def
      by (meson rtrancl_into_trancl1)
  qed
  moreover
  have "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
    using assms(1) assms(5) local.get_shadow_root_shadow_root_ptr_in_heap by blast
  then
  have "cast shadow_root_ptr ∈ fset (object_ptr_kinds h)"
    by auto
  have "is_shadow_root_ptr shadow_root_ptr"
    using assms(3)[unfolded known_ptrs_impl ShadowRootClass.known_ptrs_defs
        ShadowRootClass.known_ptr_defs DocumentClass.known_ptr_defs CharacterDataClass.known_ptr_defs
        ElementClass.known_ptr_defs NodeClass.known_ptr_defs, simplified, rule_format, OF
        ‹cast shadow_root_ptr ∈ fset (object_ptr_kinds h)›]
    by(auto simp add: is_document_ptr_document_ptr_def cast_shadow_root_ptr2document_ptr_def
        split: option.splits document_ptr.splits)
  then
  have "h ⊢ get_owner_document (cast shadow_root_ptr) →_r cast shadow_root_ptr"
    using ‹shadow_root_ptr |∈| shadow_root_ptr_kinds h›
    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)+)+
    by(auto simp add: is_node_ptr_kind_none a_get_owner_document_shadow_root_ptr_def
        CD.a_get_owner_document_document_ptr_def)
  ultimately show ?thesis
    using assms(1) assms(2) assms(3) get_scdom_component_different_owner_documents owner_document
    by blast
qed

lemma get_shadow_root_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_shadow_root_safe element_ptr →_r shadow_root_ptr_opt →_h h'"
  assumes "∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h). h ⊢ get_mode shadow_root_ptr →_r Closed"
  shows "is_strongly_scdom_component_safe {cast element_ptr} (cast ` set_option shadow_root_ptr_opt) h h'"
proof -
  have "h = h'"
    using assms(4)
    by(auto simp add: returns_result_heap_def pure_returns_heap_eq)
  moreover have "shadow_root_ptr_opt = None"
    using assms(4)
    apply(auto simp add: returns_result_heap_def get_shadow_root_safe_def elim!: bind_returns_result_E2
        split: option.splits if_splits)[1]
    using get_shadow_root_shadow_root_ptr_in_heap
    by (meson assms(5) is_OK_returns_result_I local.get_mode_ptr_in_heap returns_result_eq
        shadow_root_mode.distinct(1))
  ultimately show ?thesis
    by(simp add: is_strongly_scdom_component_safe_def preserved_def)
qed
```

```
end

interpretation i_get_shadow_root_scope_component?: l_get_shadow_root_scope_component_Shadow_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe get_dom_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe 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_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs to_tree_order get_parent get_parent_locs get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  remove_shadow_root remove_shadow_root_locs create_document DocumentClass.known_ptr DocumentClass.type_wf
  CD.a_get_owner_document  get_mode get_mode_locs get_shadow_root_safe get_shadow_root_safe_locs
  by(auto simp add: l_get_shadow_root_scope_component_Shadow_DOM_def l_get_shadow_root_scope_component_Shadow_DOM
instances)
declare l_get_shadow_root_scope_component_Shadow_DOM_axioms [instances]
```

2.7.4 get_host

```
lemma get_host_not_weakly_scdom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    shadow_root_ptr and element_ptr and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_host shadow_root_ptr →_r element_ptr →_h h'" and
    "¬ is_weakly_scdom_component_safe {cast shadow_root_ptr} {cast element_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return s1
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?s1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and shadow_root_ptr="?s1"])
    by code_simp+
qed

locale l_get_host_scope_component_Shadow_DOM =
  l_get_shadow_root_scope_component_Shadow_DOM +
  l_get_host_component_Shadow_DOM
begin
```

```
lemma get_host_components_disjunct:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_scdom_component ptr →r sc"
  assumes "h ⊢ get_host shadow_root_ptr →r host"
  shows "set |h ⊢ get_scdom_component (cast host)|r ∩ set |  h ⊢ get_scdom_component (cast shadow_root_ptr)|r
= {}"
  using assms get_shadow_root_components_disjunct local.shadow_root_host_dual
  by blast
end

interpretation i_get_host_scope_component?: l_get_host_scope_component_Shadow_DOM
  get_owner_document heap_is_wellformed parent_child_rel type_wf known_ptr
  known_ptrs get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe 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_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs to_tree_order get_parent get_parent_locs get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  remove_shadow_root remove_shadow_root_locs create_document DocumentClass.known_ptr DocumentClass.type_wf
  CD.a_get_owner_document get_mode get_mode_locs get_shadow_root_safe get_shadow_root_safe_locs
  by(auto simp add: l_get_host_scope_component_Shadow_DOM_def instances)
declare l_get_host_scope_component_Shadow_DOM_axioms [instances]
```

2.7.5 get_root_node_si

```
lemma get_composed_root_node_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    ptr and root and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_root_node_si ptr →r root →h h'" and
    "¬ is_weakly_scdom_component_safe {ptr} {root} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Closed;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e3
      }"
  let ?h1 = "|?h0 ⊢ ?P|h"
  let ?e3 = "|?h0 ⊢ ?P|r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e3"])
```

```
    by code_simp+
qed

locale l_get_scdom_component_get_root_node_si_Shadow_DOM =
  l_get_dom_component_get_root_node_si_Shadow_DOM +
  l_get_dom_component_get_scdom_component
begin

lemma get_root_node_si_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node_si ptr' →_r root"
  shows "set |h ⊢ get_scdom_component ptr'|_r = set |h ⊢ get_scdom_component root|_r ∨
set |h ⊢ get_scdom_component ptr'|_r ∩ set |h ⊢ get_scdom_component root|_r = {}"
proof -
  have "ptr' |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr_in_heap assms(4)
    by(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)
  then
  obtain sc where "h ⊢ get_scdom_component ptr' →_r sc"
    by (meson assms(1) assms(2) assms(3) local.get_scdom_component_ok select_result_I)
  moreover
  have "root |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr assms(4)
    apply(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) empty_iff empty_set
        get_ancestors_si_ptrs_in_heap last_in_set)
  then
  obtain sc' where "h ⊢ get_scdom_component root →_r sc'"
    by (meson assms(1) assms(2) assms(3) local.get_scdom_component_ok select_result_I)
  ultimately show ?thesis
    by (metis (no_types, opaque_lifting) IntE ‹⋀thesis. (⋀sc'. h ⊢ get_scdom_component root →_r sc' ⟹
thesis) ⟹ thesis›
        ‹⋀thesis. (⋀sc. h ⊢ get_scdom_component ptr' →_r sc ⟹ thesis) ⟹ thesis› assms(1) assms(2)
assms(3) empty_subsetI
        local.get_scdom_component_ptrs_same_scope_component returns_result_eq select_result_I2 subsetI subset_antis
qed
end

interpretation i_get_scdom_component_get_root_node_si?: l_get_scdom_component_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 get_disconnected_node
  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_disconnected_document get_disconnected_document_locs
to_tree_order
  get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  get_owner_document get_scdom_component is_strongly_scdom_component_safe
  by(auto simp add: l_get_scdom_component_get_root_node_si_Shadow_DOM_def instances)
declare l_get_scdom_component_get_root_node_si_Shadow_DOM_axioms [instances]
```

2.7.6 get_assigned_nodes

```
lemma assigned_nodes_not_weakly_scdom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    node_ptr and nodes and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ assigned_nodes node_ptr →_r nodes →_h h'" and
    "¬ is_weakly_scdom_component_safe {cast node_ptr} (cast ` set nodes) h h'"
proof -
```

```
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Closed;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e3
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e3 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and node_ptr="?e3"])
    by code_simp+
qed

lemma assigned_slot_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    node_ptr and slot_opt and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ assigned_slot node_ptr →_r slot_opt →_h h'" and
    "¬ is_weakly_scdom_component_safe {cast node_ptr} (cast ` set_option slot_opt) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e2
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
```

```
  let ?e2 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and node_ptr="cast_element_ptr2node_ptr ?e2"])
    by code_simp+
qed

locale l_assigned_nodes_scope_component_Shadow_DOM =
  l_assigned_nodes_component_Shadow_DOM +
  l_get_shadow_root_scope_component_Shadow_DOM +
  l_find_slot_Shadow_DOM
begin

lemma find_slot_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ find_slot open_flag node_ptr →_r Some slot"
  shows "set |h ⊢ get_scdom_component (cast node_ptr)|_r ∩ set |h ⊢ get_scdom_component (cast slot)|_r =
{}"
proof -
  obtain host shadow_root_ptr to where
    "h ⊢ get_parent node_ptr →_r Some (cast host)" and
    "h ⊢ get_shadow_root host →_r Some shadow_root_ptr" and
    "h ⊢ to_tree_order (cast shadow_root_ptr) →_r to" and
    "cast slot ∈ set to"
    using assms(4)
    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]
    by (metis element_ptr_casts_commute3)+

  have "node_ptr |∈| node_ptr_kinds h"
    using assms(4) find_slot_ptr_in_heap by blast
  then obtain node_ptr_c where node_ptr_c: "h ⊢ get_scdom_component (cast node_ptr) →_r node_ptr_c"
    using assms(1) assms(2) assms(3) get_scdom_component_ok is_OK_returns_result_E
      node_ptr_kinds_commutes[symmetric]
    by metis

  then have "cast host ∈ set node_ptr_c"
    using ‹h ⊢ get_parent node_ptr →_r Some (cast host)› assms(1) assms(2) assms(3)
    by (meson assms(4) is_OK_returns_result_E local.find_slot_ptr_in_heap local.get_dom_component_ok
        local.get_dom_component_parent_inside local.get_dom_component_ptr
        local.get_scdom_component_subset_get_dom_component node_ptr_kinds_commutes subsetCE)

  then have "h ⊢ get_scdom_component (cast host) →_r node_ptr_c"
    using ‹h ⊢ get_parent node_ptr →_r Some (cast host)› a_heap_is_wellformed_def assms(1) assms(2)
      assms(3) node_ptr_c
    using local.get_scdom_component_ptrs_same_scope_component by blast

  moreover have "slot |∈| element_ptr_kinds h"
    using assms(4) find_slot_slot_in_heap by blast
  then obtain slot_c where slot_c: "h ⊢ get_scdom_component (cast slot) →_r slot_c"
    using a_heap_is_wellformed_def assms(1) assms(2) assms(3) get_scdom_component_ok
      is_OK_returns_result_E node_ptr_kinds_commutes[symmetric] element_ptr_kinds_commutes[symmetric]
    by (smt (verit, best))
  then have "cast shadow_root_ptr ∈ set slot_c"
    using ‹h ⊢ to_tree_order (cast shadow_root_ptr) →_r to› ‹cast slot ∈ set to› assms(1) assms(2) assms(3)
    by (meson is_OK_returns_result_E local.get_dom_component_ok local.get_dom_component_ptr
        local.get_dom_component_to_tree_order local.get_scdom_component_subset_get_dom_component local.to_tree_orde
subsetCE)
  then have "h ⊢ get_scdom_component (cast shadow_root_ptr) →_r slot_c"
    using  ‹h ⊢ get_shadow_root host →_r Some shadow_root_ptr› assms(1) assms(2) assms(3) slot_c
    using local.get_scdom_component_ptrs_same_scope_component by blast
```

```
  ultimately show ?thesis
    using get_shadow_root_components_disjunct assms ‹h ⊢ get_shadow_root host →r Some shadow_root_ptr›
      node_ptr_c slot_c
    by (metis (no_types, lifting) select_result_I2)
qed


lemma assigned_nodes_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ assigned_nodes element_ptr →r nodes"
  assumes "node_ptr ∈ set nodes"
  shows "set |h ⊢ get_scdom_component (cast element_ptr)|r ∩ set |h ⊢ get_scdom_component (cast node_ptr)|r
= {}"
  using assms
proof -
  have "h ⊢ find_slot False node_ptr →r Some element_ptr"
    using assms(4) assms(5)
    by(auto simp add: assigned_nodes_def elim!: bind_returns_result_E2
        dest!: filter_M_holds_for_result[where x=node_ptr] intro!: bind_pure_I split: if_splits)
  then show ?thesis
    using assms find_slot_is_component_unsafe
    by blast
qed

lemma assigned_slot_is_strongly_scdom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ assigned_slot element_ptr →r slot_opt →h h'"
  assumes "∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h). h ⊢ get_mode shadow_root_ptr →r Closed"
  shows "is_strongly_scdom_component_safe {cast element_ptr} (cast ‘ set_option slot_opt) h h'"
proof -
  have "h = h'"
    using assms(4) find_slot_pure
    by(auto simp add: assigned_slot_def returns_result_heap_def pure_returns_heap_eq find_slot_impl)
  moreover have "slot_opt = None"
    using assms(4) assms(5)
    apply(auto simp add: returns_result_heap_def assigned_slot_def a_find_slot_def
        elim!: bind_returns_result_E2 split: option.splits if_splits
        dest!: get_shadow_root_shadow_root_ptr_in_heap[OF assms(1)])[1]
     apply (meson returns_result_eq shadow_root_mode.distinct(1))
    apply (meson returns_result_eq shadow_root_mode.distinct(1))
    done
  ultimately show ?thesis
    by(auto simp add: is_strongly_scdom_component_safe_def preserved_def)
qed
end

interpretation i_assigned_nodes_scope_component?: l_assigned_nodes_scope_component_Shadow_DOM
  type_wf get_tag_name get_tag_name_locs known_ptr get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_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 get_parent get_parent_locs
  get_mode get_mode_locs get_attribute get_attribute_locs first_in_tree_order find_slot
  assigned_slot known_ptrs to_tree_order assigned_nodes assigned_nodes_flatten flatten_dom
  get_root_node get_root_node_locs remove insert_before insert_before_locs append_child
  remove_shadow_root remove_shadow_root_locs set_shadow_root set_shadow_root_locs remove_child
  remove_child_locs get_dom_component is_strongly_dom_component_safe is_weakly_dom_component_safe
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  get_owner_document 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 set_child_nodes set_child_nodes_locs
  get_scdom_component is_strongly_scdom_component_safe is_weakly_scdom_component_safe create_document
  DocumentClass.known_ptr DocumentClass.type_wf CD.a_get_owner_document get_shadow_root_safe
  get_shadow_root_safe_locs
  by(auto simp add: l_assigned_nodes_scope_component_Shadow_DOM_def instances)
```

declare *l_assigned_nodes_scope_component*$_{Shadow_DOM}$*_axioms [instances]*
end

Bibliography

[1] E. Bidelman. Shadow dom v1: Self-contained web components, May 2017. URL https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom.

[2] 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.

[3] A. D. Brucker and M. Herzberg. A formal semantics of the core DOM in Isabelle/HOL. In *Proceedings of the Web Programming, Design, Analysis, And Implementation (WPDAI) track at WWW 2018*, 2018. URL https://www.brucker.ch/bibliography/abstract/brucker.ea-fdom-2018.

[4] 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.

[5] A. D. Brucker and M. Herzberg. A formalization of web components. *Archive of Formal Proofs*, Sept. 2020. ISSN 2150-914x. URL http://www.brucker.ch/bibliography/abstract/brucker.ea-afp-dom-components-2020. http://www.isa-afp.org/entries/DOM_Components.html, Formal proof development.

[6] 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.

[7] A. D. Brucker and M. Herzberg. Shadow sc dom: A formal model of the safelty composable 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-sc-dom-2020. http://www.isa-afp.org/entries/Shadow_SC_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 http://www.brucker.ch/bibliography/abstract/brucker.ea-web-components-2019.

[9] M. Herzberg. *Formal Foundations for Provably Safe Web Components*. PhD thesis, The University of Sheffield, 2020.

[10] WHATWG. DOM – living standard, Feb. 2019. URL https://dom.spec.whatwg.org/commit-snapshots/7fa83673430f767d329406d0aed901f296332216/. Last Updated 11 February 2019.