

Converting Linear-Time Temporal Logic to Generalized Büchi Automata

Alexander Schimpf and Peter Lammich

May 4, 2022

Abstract

We formalize linear-time temporal logic (LTL) and the algorithm by Gerth et al. to convert LTL formulas to generalized Büchi automata. We also formalize some syntactic rewrite rules that can be applied to optimize the LTL formula before conversion. Moreover, we integrate the Stuttering Equivalence AFP-Entry by Stefan Merz, adapting the lemma that next-free LTL formula cannot distinguish between stuttering equivalent runs to our setting.

We use the Isabelle Refinement and Collection framework, as well as the Autoref tool, to obtain a refined version of our algorithm, from which efficiently executable code can be extracted.

Contents

1	Introduction	3
2	LTL to GBA translation	3
2.1	Statistics	3
2.2	Preliminaries	4
2.3	Creation of States	5
2.4	Creation of GBA	36
3	Refinement to Efficient Code	70
3.1	Parametricity Setup Boilerplate	70
3.1.1	LTL Formulas	70
3.1.2	Nodes	75
3.2	Massaging the Abstract Algorithm	77
3.2.1	Creation of the Nodes	77
3.2.2	Creation of GBA from Nodes	80
3.3	Refinement to Efficient Data Structures	85
3.3.1	Creation of GBA from Nodes	85
3.3.2	Creation of Graph	87

1 Introduction

In LTL model checking obtaining an equivalent automaton from a linear temporal logic (LTL) formula makes up an important nontrivial part of the whole process. Gerth et al. [2] present a simple tableau-based construction, which takes an LTL formula and decomposes it according to its structure gaining the desired automaton step-by-step.

In this entry, we formalize Linear Temporal Logic (LTL), some optimizing syntactic rewrite rules on LTL formulas, and Gerth's algorithm. Using the Isabelle Refinement Framework, we extract efficient code from our formalization.

Moreover, we connect our LTL formalization to the one of Stefan Merz [3], adapting the lemma that next-free LTL formula cannot distinguish between stuttering equivalent runs to our setting.

This work is part of the CAVA project [1] to implement an executable fully verified LTL model checker.

2 LTL to GBA translation

```
theory LTL-to-GBA
imports
  CAVA-Base.CAVA-Base
  LTL.LTL
  CAVA-Automata.Automata
begin
```

2.1 Statistics

```
code-printing
code-module Gerth-Statistics  $\rightarrow$  (SML)  $\langle$ 
  structure Gerth-Statistics = struct
    val active = Unsynchronized.ref false
    val data = Unsynchronized.ref (0,0,0)

    fun is-active () = !active
    fun set-data num-states num-init num-acc = (
      active := true;
      data := (num-states, num-init, num-acc)
    )

    fun to-string () = let
      val (num-states, num-init, num-acc) = !data
    in
      Num states: ^ IntInf.toString (num-states) ^\n
      ^ Num initial: ^ IntInf.toString num-init ^\n
      ^ Num acc-classes: ^ IntInf.toString num-acc ^\n
    end
  end
code-end
```

```

    end

    val - = Statistics.register-stat (Gerth LTL-to-GBA,is-active,to-string)
  end
>
code-reserved SML Gerth-Statistics

consts
  stat-set-data-int :: integer ⇒ integer ⇒ integer ⇒ unit

code-printing
  constant stat-set-data-int ↪ (SML) Gerth'-Statistics.set'-data

definition stat-set-data ns ni na
  ≡ stat-set-data-int (integer-of-nat ns) (integer-of-nat ni) (integer-of-nat na)

lemma [autoref-rules]:
  (stat-set-data,stat-set-data) ∈ nat-rel → nat-rel → nat-rel → unit-rel
  by auto

abbreviation stat-set-data-nres ns ni na ≡ RETURN (stat-set-data ns ni na)

lemma discard-stat-refine[refine]:
  m1 ≤ m2 ⇒ stat-set-data-nres ns ni na ≫ m1 ≤ m2 by simp-all

```

2.2 Preliminaries

Some very special lemmas for reasoning about the nres-monad

lemma SPEC-rule-nested2:

```

[[m ≤ SPEC P; ∧r1 r2. P (r1, r2) ⇒ g (r1, r2) ≤ SPEC P]]
⇒ m ≤ SPEC (λr'. g r' ≤ SPEC P)
by (simp add: pw-le-iff) blast

```

lemma SPEC-rule-param2:

```

assumes f x ≤ SPEC (P x)
  and ∧r1 r2. (P x) (r1, r2) ⇒ (P x') (r1, r2)
  shows f x ≤ SPEC (P x')
using assms
by (simp add: pw-le-iff)

```

lemma SPEC-rule-weak:

```

assumes f x ≤ SPEC (Q x) and f x ≤ SPEC (P x)
  and ∧r1 r2. [[(Q x) (r1, r2); (P x) (r1, r2)]] ⇒ (P x') (r1, r2)
  shows f x ≤ SPEC (P x')
using assms
by (simp add: pw-le-iff)

```

lemma SPEC-rule-weak-nested2: [[f ≤ SPEC Q; f ≤ SPEC P;
 ∧r1 r2. [[Q (r1, r2); P (r1, r2)]] ⇒ g (r1, r2) ≤ SPEC P]]

$\implies f \leq \text{SPEC } (\lambda r'. g r' \leq \text{SPEC } P)$
by (*simp add: pw-le-iff*) *blast*

2.3 Creation of States

In this section, the first part of the algorithm, which creates the states of the automaton, is formalized.

type-synonym *node-name* = *nat*

type-synonym *'a frml* = *'a ltlr*

type-synonym *'a interpr* = *'a set word*

record *'a node* =
name :: *node-name*
incoming :: *node-name set*
new :: *'a frml set*
old :: *'a frml set*
next :: *'a frml set*

context

begin

fun *new1* **where**
new1 (μ *and_r* ψ) = $\{\mu, \psi\}$
| *new1* (μ *U_r* ψ) = $\{\mu\}$
| *new1* (μ *R_r* ψ) = $\{\psi\}$
| *new1* (μ *or_r* ψ) = $\{\mu\}$
| *new1* - = $\{\}$

fun *next1* **where**
next1 (*X_r* ψ) = $\{\psi\}$
| *next1* (μ *U_r* ψ) = $\{\mu U_r \psi\}$
| *next1* (μ *R_r* ψ) = $\{\mu R_r \psi\}$
| *next1* - = $\{\}$

fun *new2* **where**
new2 (μ *U_r* ψ) = $\{\psi\}$
| *new2* (μ *R_r* ψ) = $\{\mu, \psi\}$
| *new2* (μ *or_r* ψ) = $\{\psi\}$
| *new2* - = $\{\}$

definition *expand-init* $\equiv 0$

definition *expand-new-name* $\equiv \text{Suc}$

lemma *expand-new-name-expand-init*: *expand-init* < *expand-new-name nm*

by (auto simp add: expand-init-def expand-new-name-def)

lemma *expand-new-name-step*[intro]:
fixes $n :: 'a \text{ node}$
shows $\text{name } n < \text{expand-new-name } (\text{name } n)$
by (auto simp add: expand-new-name-def)

lemma *expand-new-name--less-zero*[intro]: $0 < \text{expand-new-name } nm$
proof –
from *expand-new-name-expand-init* **have** $\text{expand-init} < \text{expand-new-name } nm$
by *auto*
then show *?thesis*
by (*metis grOI less-zeroE*)
qed

abbreviation

$\text{upd-incoming-f } n \equiv (\lambda n'.$
if $(\text{old } n' = \text{old } n \wedge \text{next } n' = \text{next } n)$ then
 $n' \setminus \{ \text{incoming} := \text{incoming } n \cup \text{incoming } n' \}$
else n'
)

definition

$\text{upd-incoming } n \text{ ns} \equiv ((\text{upd-incoming-f } n) \text{ ' ns})$

lemma *upd-incoming--elem*:
assumes $nd \in \text{upd-incoming } n \text{ ns}$
shows $nd \in \text{ns}$
 $\vee (\exists nd' \in \text{ns}. nd = nd' \setminus \{ \text{incoming} := \text{incoming } n \cup \text{incoming } nd' \}) \wedge$
 $\text{old } nd' = \text{old } n \wedge$
 $\text{next } nd' = \text{next } n)$

proof –

obtain nd' **where** $nd' \in \text{ns}$ **and** *nd-eq*: $nd = \text{upd-incoming-f } n \text{ nd}'$
using *assms* **unfolding** *upd-incoming-def* **by** *blast*
then show *?thesis* **by** *auto*

qed

lemma *upd-incoming--ident-node*:

assumes $nd \in \text{upd-incoming } n \text{ ns}$ **and** $nd \in \text{ns}$
shows $\text{incoming } n \subseteq \text{incoming } nd \vee \neg (\text{old } nd = \text{old } n \wedge \text{next } nd = \text{next } n)$

proof (*rule ccontr*)

assume $\neg ?thesis$

then have *nsubset*: $\neg (\text{incoming } n \subseteq \text{incoming } nd)$ **and**

old-next-eq: $\text{old } nd = \text{old } n \wedge \text{next } nd = \text{next } n$

by *auto*

moreover

obtain nd' **where** $nd' \in \text{ns}$ **and** *nd-eq*: $nd = \text{upd-incoming-f } n \text{ nd}'$

using *assms* **unfolding** *upd-incoming-def* **by** *blast*

{ **assume** $\text{old } nd' = \text{old } n \wedge \text{next } nd' = \text{next } n$

with *nd-eq* **have** $nd = nd'(\text{incoming} := \text{incoming } n \cup \text{incoming } nd')$ **by** *auto*
with *nsubset* **have** *False* **by** *auto* }
moreover
{ **assume** $\neg (\text{old } nd' = \text{old } n \wedge \text{next } nd' = \text{next } n)$
with *nd-eq old-next-eq* **have** *False* **by** *auto* }
ultimately show *False* **by** *fast*
qed

lemma *upd-incoming--ident*:
assumes $\forall n \in ns. P \ n$
and $\bigwedge n. \llbracket n \in ns; P \ n \rrbracket \implies (\bigwedge v. P \ (n(\text{incoming} := v)))$
shows $\forall n \in \text{upd-incoming } n \ ns. P \ n$

proof
fix *nd v*
let $?f = \lambda n'.$
 if $(\text{old } n' = \text{old } n \wedge \text{next } n' = \text{next } n)$ **then**
 $n'(\text{incoming} := \text{incoming } n \cup \text{incoming } n')$
 else n'
assume $nd \in \text{upd-incoming } n \ ns$
then obtain *nd'* **where** $nd' \in ns$ **and** *nd-eq*: $nd = ?f \ nd'$
 unfolding *upd-incoming-def* **by** *blast*
{ **assume** $\text{old } nd' = \text{old } n \wedge \text{next } nd' = \text{next } n$
 then obtain *v* **where** $nd = nd'(\text{incoming} := v)$ **using** *nd-eq* **by** *auto*
 with *assms* $\langle nd' \in ns \rangle$ **have** $P \ nd$ **by** *auto* }
then show $P \ nd$ **using** *nd-eq* $\langle nd' \in ns \rangle$ *assms* **by** *auto*
qed

lemma *name-upd-incoming-f[simp]*: $\text{name } (\text{upd-incoming-f } n \ x) = \text{name } x$
by *auto*

lemma *name-upd-incoming[simp]*:
 $\text{name } '(\text{upd-incoming } n \ ns) = \text{name } ' ns$ (**is** $?lhs = ?rhs$)
unfolding *upd-incoming-def*
proof *safe*
 fix *x*
 assume $x \in ns$
 then have $\text{upd-incoming-f } n \ x \in (\lambda n'. \text{local.upd-incoming-f } n \ n')$ $' ns$ **by** *auto*
 then have $\text{name } (\text{upd-incoming-f } n \ x) \in \text{name } '(\lambda n'. \text{local.upd-incoming-f } n \ n')$
 $' ns$
 by *blast*
 then show $\text{name } x \in \text{name } '(\lambda n'. \text{local.upd-incoming-f } n \ n')$ $' ns$ **by** *simp*
qed *auto*

abbreviation *expand-body*
where
 $\text{expand-body} \equiv (\lambda \text{expand } (n, ns).$

```

if new n = {} then (
  if ( $\exists n' \in ns. old\ n' = old\ n \wedge next\ n' = next\ n$ ) then
    RETURN (name n, upd-incoming n ns)
  else
    expand (
      (
        name=expand-new-name (name n),
        incoming={name n},
        new=next n,
        old={},
        next={}
      ),
      {n}  $\cup$  ns)
    ) else do {
       $\varphi \leftarrow SPEC (\lambda x. x \in (new\ n))$ ;
      let n = n( $new := new\ n - \{\varphi\}$ );
      if ( $\exists q. \varphi = prop_r(q) \vee \varphi = nprop_r(q)$ ) then
        (if ( $not_r\ \varphi \in old\ n$ ) then RETURN (name n, ns)
          else expand (n( $old := \{\varphi\} \cup old\ n$ ), ns))
        )
      else if  $\varphi = true_r$  then expand (n( $old := \{\varphi\} \cup old\ n$ ), ns)
      else if  $\varphi = false_r$  then RETURN (name n, ns)
      else if ( $\exists \nu\ \mu. (\varphi = \nu\ and_r\ \mu) \vee (\varphi = X_r\ \nu)$ ) then
        expand (
          n(
            new := new1  $\varphi \cup new\ n$ ,
            old :=  $\{\varphi\} \cup old\ n$ ,
            next := next1  $\varphi \cup next\ n$ 
          ),
          ns)
        )
      else do {
        (nm, nds)  $\leftarrow$  expand (
          n(
            new := new1  $\varphi \cup new\ n$ ,
            old :=  $\{\varphi\} \cup old\ n$ ,
            next := next1  $\varphi \cup next\ n$ 
          ),
          ns);
        expand (n( $name := nm, new := new2\ \varphi \cup new\ n, old := \{\varphi\} \cup old\ n$ ),
          nds)
        )
      }
    }
  )
)

```

lemma *expand-body-mono*: trimono *expand-body* by *refine-mono*

definition *expand* :: ('a node \times ('a node set)) \Rightarrow (node-name \times 'a node set) nres
where *expand* $\equiv REC\ expand\ body$

lemma *REC-rule-old*:

fixes $x::'x$
assumes M : *trimono body*
assumes $I0$: Φx
assumes IS : $\bigwedge f x. [\bigwedge x. \Phi x \implies f x \leq M x; \Phi x; f \leq REC \text{ body}]$
 $\implies \text{body } f x \leq M x$
shows $REC \text{ body } x \leq M x$
by (*rule REC-rule-arb*[**where** $pre=\lambda-. \Phi$ **and** $M=\lambda-. M$, *OF assms*])

lemma *expand-rec-rule*:

assumes $I0$: Φx
assumes IS : $\bigwedge f x. [\bigwedge x. f x \leq \text{expand } x; \bigwedge x. \Phi x \implies f x \leq M x; \Phi x]$
 $\implies \text{expand-body } f x \leq M x$
shows $\text{expand } x \leq M x$
unfolding *expand-def*
apply (*rule REC-rule-old*[**where** $\Phi=\Phi$])
apply (*rule expand-body-mono*)
apply (*rule I0*)
apply (*rule IS*[*unfolded expand-def*])
apply (*blast dest: le-funD*)
apply *blast*
apply *blast*
done

abbreviation

expand-assm-incoming n-ns
 $\equiv (\forall nm \in \text{incoming } (fst \ n\text{-ns}). \text{name } (fst \ n\text{-ns}) > nm)$
 $\wedge 0 < \text{name } (fst \ n\text{-ns})$
 $\wedge (\forall q \in \text{snd } n\text{-ns}.$
 $\text{name } (fst \ n\text{-ns}) > \text{name } q$
 $\wedge (\forall nm \in \text{incoming } q. \text{name } (fst \ n\text{-ns}) > nm))$

abbreviation

expand-rslt-incoming nm-nds
 $\equiv (\forall q \in \text{snd } nm\text{-nds}. (fst \ nm\text{-nds} > \text{name } q \wedge (\forall nm' \in \text{incoming } q. fst \ nm\text{-nds} > nm')))$

abbreviation

expand-rslt-name n-ns nm-nds
 $\equiv (\text{name } (fst \ n\text{-ns}) \leq fst \ nm\text{-nds} \wedge \text{name } '(snd \ n\text{-ns}) \subseteq \text{name } '(snd \ nm\text{-nds}))$
 $\wedge \text{name } '(snd \ nm\text{-nds})$
 $= \text{name } '(snd \ n\text{-ns}) \cup \text{name } '\{nd \in \text{snd } nm\text{-nds}. \text{name } nd \geq \text{name } (fst \ n\text{-ns})\}$

abbreviation

expand-name-ident nds
 $\equiv (\forall q \in nds. \exists !q' \in nds. \text{name } q = \text{name } q')$

abbreviation

expand-assm-exist ξ n-ns
 $\equiv \{\eta. \exists \mu. \mu \ U_r \ \eta \in \text{old } (fst \ n\text{-ns}) \wedge \xi \models_r \eta\} \subseteq \text{new } (fst \ n\text{-ns}) \cup \text{old } (fst \ n\text{-ns})$

$$\begin{aligned} & \wedge (\forall \psi \in \text{new } (fst \ n\text{-}ns). \xi \models_r \psi) \\ & \wedge (\forall \psi \in \text{old } (fst \ n\text{-}ns). \xi \models_r \psi) \\ & \wedge (\forall \psi \in \text{next } (fst \ n\text{-}ns). \xi \models_r X_r \psi) \end{aligned}$$

abbreviation

$$\begin{aligned} & \text{expand-rslt-exist--node-prop } \xi \ n \ nd \\ & \equiv \text{incoming } n \subseteq \text{incoming } nd \\ & \wedge (\forall \psi \in \text{old } nd. \xi \models_r \psi) \wedge (\forall \psi \in \text{next } nd. \xi \models_r X_r \psi) \\ & \wedge \{ \eta. \exists \mu. \mu \ U_r \ \eta \in \text{old } nd \wedge \xi \models_r \eta \} \subseteq \text{old } nd \end{aligned}$$

abbreviation

$$\begin{aligned} & \text{expand-rslt-exist } \xi \ n\text{-}ns \ nm\text{-}nds \\ & \equiv (\exists nd \in \text{snd } nm\text{-}nds. \text{expand-rslt-exist--node-prop } \xi \ (fst \ n\text{-}ns) \ nd) \end{aligned}$$

abbreviation

$$\begin{aligned} & \text{expand-rslt-exist-eq--node } n \ nd \\ & \equiv \text{name } n = \text{name } nd \\ & \wedge \text{old } n = \text{old } nd \\ & \wedge \text{next } n = \text{next } nd \\ & \wedge \text{incoming } n \subseteq \text{incoming } nd \end{aligned}$$

abbreviation

$$\begin{aligned} & \text{expand-rslt-exist-eq } n\text{-}ns \ nm\text{-}nds \equiv \\ & (\forall n \in \text{snd } n\text{-}ns. \exists nd \in \text{snd } nm\text{-}nds. \text{expand-rslt-exist-eq--node } n \ nd) \end{aligned}$$

lemma *expand-name-propag*:

assumes *expand-assm-incoming* $n\text{-}ns \wedge \text{expand-name-ident } (\text{snd } n\text{-}ns)$ (**is** $?Q$ $n\text{-}ns$)

shows $\text{expand } n\text{-}ns \leq \text{SPEC } (\lambda r. \text{expand-rslt-incoming } r$
 $\wedge \text{expand-rslt-name } n\text{-}ns \ r$
 $\wedge \text{expand-name-ident } (\text{snd } r))$

(**is** $\text{expand} - \leq \text{SPEC } (?P \ n\text{-}ns)$)

using *assms*

proof (*rule-tac expand-rec-rule*[**where** $\Phi = ?Q$], *simp, intro refine-vcg, goal-cases*)

case *prems*: (1 - - $n \ ns$)

then have $Q: ?Q \ (n, \ ns)$ **by** *fast*

let $?nds = \text{upd-incoming } n \ ns$

from *prems* **have** $\forall q \in ?nds. \text{name } q < \text{name } n$

by (*rule-tac upd-incoming--ident*) *auto*

moreover

have $\forall q \in ?nds. \forall nm' \in \text{incoming } q. nm' < \text{name } n$ (**is** $\forall q \in -. ?sg \ q$)

proof

fix q

assume $q\text{-in}: q \in ?nds$

show $?sg \ q$

proof (*cases* $q \in ns$)

case *True*

with *prems* **show** $?thesis$ **by** *auto*

```

next
case False
with upd-incoming--elem[OF q-in]
obtain nd' where
  nd'-def:nd'∈ns ∧ q = nd'(incoming := incoming n ∪ incoming nd')
by blast

{ fix nm'
  assume nm'∈incoming q
  moreover
  { assume nm'∈incoming n
    with Q have nm' < name n by auto }
  moreover
  { assume nm'∈incoming nd'
    with Q nd'-def have nm' < name n by auto }
  ultimately have nm' < name n using nd'-def by auto }

then show ?thesis by fast
qed
qed
moreover
have expand-name-ident ?nds
proof (rule upd-incoming--ident, goal-cases)
case 1
show ?case
proof
fix q
assume q∈ns

with Q have ∃!q'∈ns. name q = name q' by auto
then obtain q' where q'∈ns and name q = name q'
and q'-all: ∀ q''∈ns. name q' = name q'' → q' = q''

by auto
let ?q' = upd-incoming-f n q'
have P-a: ?q'∈?nds ∧ name q = name ?q'
using ⟨q'∈ns⟩ ⟨name q = name q'⟩ q'-all
unfolding upd-incoming-def by auto

have P-all: ∀ q''∈?nds. name ?q' = name q'' → ?q' = q''
proof(clarify)
fix q''
assume q''∈?nds and q''-name-eq: name ?q' = name q''
{ assume q''∉ns
  with upd-incoming--elem[OF ⟨q''∈?nds⟩]
  obtain nd'' where
    nd''∈ns
    and q''-is: q'' = nd''(incoming := incoming n ∪ incoming nd'')
    ∧ old nd'' = old n ∧ next nd'' = next n
  by auto

```

```

then have  $\text{name } nd'' = \text{name } q'$ 
  using  $q''\text{-name-eq}$ 
  by  $(\text{cases } \text{old } q' = \text{old } n \wedge \text{next } q' = \text{next } n) \text{ auto}$ 
with  $\langle nd'' \in ns \rangle$   $q'\text{-all}$  have  $nd'' = q'$  by  $\text{auto}$ 
then have  $?q' = q''$  using  $q''\text{-is}$  by  $\text{simp}$  }
moreover
{ assume  $q'' \in ns$ 
  moreover
  have  $\text{name } q' = \text{name } q''$ 
    using  $q''\text{-name-eq}$ 
    by  $(\text{cases } \text{old } q' = \text{old } n \wedge \text{next } q' = \text{next } n) \text{ auto}$ 
  moreover
  then have  $\text{incoming } n \subseteq \text{incoming } q''$ 
     $\implies \text{incoming } q'' = \text{incoming } n \cup \text{incoming } q''$ 
    by  $\text{auto}$ 
  ultimately have  $?q' = q''$ 
    using  $\text{upd-incoming--ident-node}[OF \langle q'' \in ?nds \rangle] q'\text{-all}$ 
    by  $\text{auto}$ 
}
ultimately show  $?q' = q''$  by  $\text{fast}$ 
qed

show  $\exists !q' \in \text{upd-incoming } n \text{ ns. } \text{name } q = \text{name } q'$ 
proof  $(\text{rule } \text{ex1I}[\text{of } - ?q], \text{goal-cases})$ 
  case 1
  then show  $?case$  using  $P\text{-a}$  by  $\text{simp}$ 
next
  case 2
  then show  $?case$ 
    using  $P\text{-all}$  unfolding  $P\text{-a}[THEN \text{conjunct2}, THEN \text{sym}]$ 
    by  $\text{blast}$ 
  qed
qed
qed  $\text{simp}$ 
ultimately show  $?case$  using  $\text{prems}$  by  $\text{auto}$ 
next
case  $\text{prems}: (2 \text{ f } x \text{ n } ns)$ 
then have  $\text{step}: \bigwedge x. ?Q \ x \implies \text{f } x \leq \text{SPEC } (?P \ x)$  by  $\text{simp}$ 
from  $\text{prems}$  have  $Q: ?Q \ (n, ns)$  by  $\text{auto}$ 

show  $?case$  unfolding  $\langle x = (n, ns) \rangle$ 
proof  $(\text{rule-tac } \text{SPEC-rule-param2}[\text{where } P = ?P], \text{rule-tac } \text{step}, \text{goal-cases})$ 
  case 1
  with  $\text{expand-new-name-step}[\text{of } n]$  show  $?case$ 
    using  $Q$ 
    by  $(\text{auto } \text{elim: } \text{order-less-trans}[\text{rotated}])$ 
next
case  $\text{prems}': (2 - nds)$ 
then have  $\text{name } ' ns \subseteq \text{name } ' nds$  by  $\text{auto}$ 

```

```

    with expand-new-name-step[of n] show ?case
      using prems' by auto
qed
next
case 3
then show ?case by auto
next
case prems: (4 f)
then have step:  $\bigwedge x. ?Q x \implies f x \leq SPEC (?P x)$ 
  by simp-all
with prems show ?case
  by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case prems: (5 f)
then have step:  $\bigwedge x. ?Q x \implies f x \leq SPEC (?P x)$ 
  by simp-all
from prems show ?case
  by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case 6
then show ?case by auto
next
case prems: (7 f)
then have step:  $\bigwedge x. ?Q x \implies f x \leq SPEC (?P x)$ 
  by simp-all
from prems show ?case
  by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case prems: (8 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in new\ n \wedge (\exists \nu\ \mu. \psi = \nu\ or_r\ \mu \vee \psi = \nu\ U_r\ \mu \vee \psi = \nu\ R_r\ \mu)$ 
  by (cases  $\psi$ ) auto
from prems have Q:  $?Q (n, ns)$  and step:  $\bigwedge x. ?Q x \implies f x \leq SPEC (?P x)$ 
  by simp-all
show ?case
  using goal-assms Q
  unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
proof (rule-tac SPEC-rule-nested2, goal-cases)
  case 1
  then show ?case
    by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case prems: (2 nm nds)
then have P-x:  $(name\ n \leq nm \wedge name\ 'ns \subseteq name\ 'nds) \wedge name\ 'nds = name\ 'ns \cup name\ '\{nd \in nds. name\ nd \geq name\ n\}$ 
  (is -  $\wedge ?nodes-eq\ nds\ ns\ (name\ n)$ ) by auto
show ?case
proof (rule-tac SPEC-rule-param2[where P = ?P], goal-cases)
  case 1

```

```

    with prems show ?case by (rule-tac step) auto
  next
  case prems': (2 nm' nds')
  then have  $\forall q \in nds'. \text{name } q < nm' \wedge (\forall nm'' \in \text{incoming } q. nm'' < nm')$  by
auto
  moreover
  have ?nodes-eq nds ns (name n) and ?nodes-eq nds' nds nm and name n  $\leq$ 
nm
  using prems' P-x by auto
  then have ?nodes-eq nds' ns (name n) by auto
  then have expand-rslt-name (n, ns) (nm', nds')
  using prems' P-x subset-trans[of name ' ns name ' nds] by auto
  ultimately show ?case using prems' by auto
qed
qed
qed

```

```

lemmas expand-name-propag--incoming = SPEC-rule-conjunct1[OF expand-name-propag]
lemmas expand-name-propag--name =
  SPEC-rule-conjunct1[OF SPEC-rule-conjunct2[OF expand-name-propag]]
lemmas expand-name-propag--name-ident =
  SPEC-rule-conjunct2[OF SPEC-rule-conjunct2[OF expand-name-propag]]

```

```

lemma expand-rslt-exist-eq:
  shows expand n-ns  $\leq$  SPEC (expand-rslt-exist-eq n-ns)
  (is -  $\leq$  SPEC (?P n-ns))
proof (rule-tac expand-rec-rule[where  $\Phi = \lambda -. \text{True}$ ], simp, intro refine-vcg, goal-cases)
  case prems: (1 f x n ns)
  let ?r = (name n, upd-incoming n ns)
  have expand-rslt-exist-eq (n, ns) ?r
  unfolding snd-conv
  proof
    fix n'
    assume n'  $\in$  ns
    { assume old n' = old n  $\wedge$  next n' = next n
      with  $\langle n' \in ns \rangle$ 
      have n'  $\in$  (incoming := incoming n  $\cup$  incoming n')  $\in$  upd-incoming n ns
      unfolding upd-incoming-def by auto
    }
    moreover
    { assume  $\neg$  (old n' = old n  $\wedge$  next n' = next n)
      with  $\langle n' \in ns \rangle$  have n'  $\in$  upd-incoming n ns
      unfolding upd-incoming-def by auto
    }
  }
  ultimately show  $\exists nd \in \text{upd-incoming } n \text{ ns. expand-rslt-exist-eq--node } n' nd$ 
  by force
qed
with prems show ?case by auto

```

```

next
  case prems: (2 f)
  then have step:  $\bigwedge x. f x \leq SPEC (?P x)$  by simp
  with prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case 3 then show ?case by auto
next
  case prems: (4 f)
  then have step:  $\bigwedge x. f x \leq SPEC (?P x)$  by simp
  with prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case prems: (5 f)
  then have step:  $\bigwedge x. f x \leq SPEC (?P x)$  by simp
  with prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case 6 then show ?case by auto
next
  case prems: (7 f)
  then have step:  $\bigwedge x. f x \leq SPEC (?P x)$  by simp
  with prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case prems: (8 f x n ns)
  then have step:  $\bigwedge x. f x \leq SPEC (?P x)$  by simp

show ?case
proof (rule-tac SPEC-rule-nested2, goal-cases)
  case 1
  with prems show ?case
  by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case (2 nm nds)
  with prems have P-x:  $?P (n, ns) (nm, nds)$  by fast
  show ?case
  unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
  proof (rule-tac SPEC-rule-param2[where P = ?P], goal-cases)
    case 1
    then show ?case by (rule-tac step)
  next
  case prems': (2 nm' nds')
  {
    fix n'
    assume n' ∈ ns
    with P-x obtain nd where nd ∈ nds and n'-split: expand-rslt-exist-eq--node
n' nd
    by auto
    with prems' obtain nd' where nd' ∈ nds' and expand-rslt-exist-eq--node nd
nd'
    by auto
    then have  $\exists nd' \in nds'. \text{expand-rslt-exist-eq--node } n' nd'$ 

```

```

    using  $n'$ -split subset-trans[of incoming  $n'$ ] by auto
  }
  then have expand-rslt-exist-eq ( $n$ ,  $ns$ ) ( $nm'$ ,  $nds'$ ) by auto
  with prems show ?case by auto
qed
qed
qed

lemma expand-prop-exist:
  expand  $n$ - $ns$   $\leq$  SPEC ( $\lambda r$ . expand-asm-exist  $\xi$   $n$ - $ns$   $\longrightarrow$  expand-rslt-exist  $\xi$   $n$ - $ns$ 
 $r$ )
  (is  $\leq$  SPEC (?P  $n$ - $ns$ ))
proof (rule-tac expand-rec-rule[where  $\Phi = \lambda \cdot$ . True], simp, intro refine-vcg, goal-cases)
  case prems: (1  $f$   $x$   $n$   $ns$ )
  let ?nds = upd-incoming  $n$   $ns$ 
  let ?r = (name  $n$ , ?nds)
  { assume  $Q$ : expand-asm-exist  $\xi$  ( $n$ ,  $ns$ )
    note  $\langle \exists n' \in ns. \text{old } n' = \text{old } n \wedge \text{next } n' = \text{next } n \rangle$ 
    then obtain  $n'$  where  $n' \in ns$  and assm-eq:  $\text{old } n' = \text{old } n \wedge \text{next } n' = \text{next } n$ 
      by auto
    let ?nd =  $n' \setminus \text{incoming} := \text{incoming } n \cup \text{incoming } n'$ 
    have ?nd  $\in$  ?nds using  $\langle n' \in ns \rangle$  assm-eq unfolding upd-incoming-def by auto
    moreover
    have incoming  $n \subseteq$  incoming ?nd by auto
    moreover
    have expand-rslt-exist--node-prop  $\xi$   $n$  ?nd using  $Q$  assm-eq  $\langle \text{new } n = \{\} \rangle$ 
      by simp
    ultimately have expand-rslt-exist  $\xi$  ( $n$ ,  $ns$ ) ?r
      unfolding fst-conv snd-conv by blast
  }
  with prems show ?case
  by auto
next
  case prems: (2  $f$   $x$   $n$   $ns$ )
  then have step:  $\bigwedge x. f x \leq$  SPEC (?P  $x$ )
    and f-sup:  $\bigwedge x. f x \leq$  expand  $x$  by auto
  show ?case
    unfolding  $\langle x = (n, ns) \rangle$ 
  proof (rule-tac SPEC-rule-weak[where  $Q =$  expand-rslt-exist-eq], goal-cases)
    case 1
    then show ?case
      by (rule-tac order-trans, rule-tac f-sup, rule-tac expand-rslt-exist-eq)
  next
    case 2
    then show ?case by (rule-tac step)
  next
  case prems': (3  $nm$   $nds$ )
  then have name '  $ns \subseteq$  name '  $nds$  by auto
  moreover

```



```

{ assume assm-ex: expand-assm-exist  $\xi$  (n, ns)
  with prems' obtain nd where nd ∈ nds and expand-rslt-exist-eq--node n nd
  by force+
  then have expand-rslt-exist--node-prop  $\xi$  n nd
  using assm-ex  $\langle \text{new } n = \{\} \rangle$  by auto
  then have expand-rslt-exist  $\xi$  (n, ns) (nm, nds) using  $\langle \text{nd} \in \text{nds} \rangle$  by auto }
ultimately show ?case
  using expand-new-name-step[of n] prems' by auto
qed
next
case prems: (3 f x n ns  $\psi$ )
{ assume expand-assm-exist  $\xi$  (n, ns)
  with prems have  $\xi \models_r \psi$  and  $\xi \models_r \text{not}_r \psi$ 
  by (metis (no-types, lifting) fstI node.select-convs(4) node.surjective node.update-convs(3))+
  then have False by simp }
with prems show ?case by auto
next
case prems: (4 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in \text{new } n \wedge (\exists q. \psi = \text{prop}_r(q) \vee \psi = \text{nprop}_r(q))$ 
  and step:  $\bigwedge x. f x \leq \text{SPEC } (?P x)$  by simp-all
show ?case
  using goal-assms unfolding  $\langle x = (n, ns) \rangle$ 
proof (rule-tac SPEC-rule-param2, rule-tac step, goal-cases)
  case prems': (1 nm nds)
  { assume expand-assm-exist  $\xi$  (n, ns)
    with prems' have expand-rslt-exist  $\xi$  (n, ns) (nm, nds) by auto }
  then show ?case by auto
qed
next
case prems: (5 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in \text{new } n \wedge \psi = \text{true}_r$ 
  and step:  $\bigwedge x. f x \leq \text{SPEC } (?P x)$  by simp-all
show ?case
  using goal-assms unfolding  $\langle x = (n, ns) \rangle$ 
proof (rule-tac SPEC-rule-param2, rule-tac step, goal-cases)
  case prems': (1 nm nds)
  { assume expand-assm-exist  $\xi$  (n, ns)
    with prems' have expand-rslt-exist  $\xi$  (n, ns) (nm, nds) by auto }
  then show ?case by auto
qed
next
case prems: (6 f x n ns  $\psi$ )
{ assume expand-assm-exist  $\xi$  (n, ns)
  with prems have  $\xi \models_r \text{false}_r$  by auto }
with prems show ?case by auto
next
case prems: (7 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in \text{new } n \wedge (\exists \nu \mu. \psi = \nu \text{ and}_r \mu \vee \psi = X_r \nu)$ 
  and step:  $\bigwedge x. f x \leq \text{SPEC } (?P x)$  by simp-all

```

```

show ?case
  using goal-assms unfolding ⟨x = (n, ns)⟩
proof (rule-tac SPEC-rule-param2, rule-tac step, goal-cases)
  case prems': (1 nm nds)
  { assume expand-assm-exist ξ (n, ns)
    with prems' have expand-rslt-exist ξ (n, ns) (nm, nds) by auto }
  then show ?case by auto
qed
next
  case prems: (8 f x n ns ψ)
  then have goal-assms: ψ ∈ new n ∧ (∃ν μ. ψ = ν orr μ ∨ ψ = ν Ur μ ∨ ψ =
ν Rr μ)
  by (cases ψ) auto
  from prems have step: ∧x. f x ≤ SPEC (?P x)
  and f-sup: ∧x. f x ≤ expand x by auto
  let ?x1 = (n(new := new n - {ψ}, new := new1 ψ ∪ new (n(new := new n -
{ψ}))),
    old := {ψ} ∪ old n, next := next1 ψ ∪ next n), ns)

let ?new1-assm-sel = λψ. (case ψ of μ Ur η => η | μ Rr η => μ | μ orr η => η)

{ assume new1-assm: ¬ (ξ ⊨r (?new1-assm-sel ψ))
  then have ?case
    using goal-assms unfolding ⟨x = (n, ns)⟩
  proof (rule-tac SPEC-rule-nested2, goal-cases)
  case prems': 1
  then show ?case
  proof (rule-tac SPEC-rule-param2, rule-tac step, goal-cases)
  case prems'': (1 nm nds)
  { assume expand-assm-exist ξ (n, ns)
    with prems'' have expand-assm-exist ξ ?x1
      unfolding fst-conv
    proof (cases ψ, goal-cases)
      case ψ: (8 μ η)
      then have ξ ⊨r μ Ur η by fast
      then have ξ ⊨r μ and ξ ⊨r Xr (μ Ur η)
        using ψ ltlr-expand-Until[of ξ μ η] by auto
      with ψ show ?case by auto
    next
      case ψ: (9 μ η)
      then have *: ξ ⊨r μ Rr η by fast
      with ψ have ξ ⊨r η and ξ ⊨r Xr (μ Rr η)
        using ltlr-expand-Release[of ξ μ η] by auto
      with ψ * show ?case by auto
    qed auto
    with prems'' have expand-rslt-exist ξ (n, ns) (nm, nds) by force }
  with prems'' show ?case by auto
qed
next

```

```

case prems': (2 nm nds)
then have P-x: ?P (n, ns) (nm, nds) by fast

show ?case
  unfolding case-prod-unfold
proof (rule-tac SPEC-rule-weak[where P = ?P and Q = expand-rslt-exist-eq],
goal-cases)
  case 1
  then show ?case
    by (rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-rslt-exist-eq)
  next
  case 2
  then show ?case by (rule-tac step)
next
case prems'': (3 nm' nds')
{ assume expand-assm-exist  $\xi$  (n, ns)
  with P-x have expand-rslt-exist  $\xi$  (n, ns) (nm, nds) by simp
  then obtain nd where nd: nd ∈ nds expand-rslt-exist--node-prop  $\xi$  n nd
    using goal-assms by auto
  with prems'' obtain nd' where
    nd' ∈ nds' and expand-rslt-exist-eq--node nd nd'
    by force
  with nd have expand-rslt-exist--node-prop  $\xi$  n nd'
    using subset-trans[of incoming n incoming nd] by auto
  then have expand-rslt-exist  $\xi$  (n, ns) (nm', nds')
    using  $\langle nd' \in nds' \rangle$  goal-assms by auto }
  then show ?case by fast
qed
qed
}
moreover
{ assume new1-assm:  $\xi \models_r$  (?new1-assm-sel  $\psi$ )
  let ?x2f =  $\lambda(nm::node-name, nds::'a\ node\ set).$  (
    n(new := new n - { $\psi$ },
      name := nm,
      new := new2  $\psi \cup new$  (n(new := new n - { $\psi$ })),
      old := { $\psi$ }  $\cup old$  n),
    nds)
  have P-x: f ?x1 ≤ SPEC (?P ?x1) by (rule step)
  moreover
  { fix r :: node-name × 'a node set
    let ?x2 = ?x2f r

    assume assm: (?P ?x1) r
    have f ?x2 ≤ SPEC (?P (n, ns))
      unfolding case-prod-unfold fst-conv
    proof (rule-tac SPEC-rule-param2, rule-tac step, goal-cases)
  }
}

```

```

case prems': (1 nm' nds')
{ assume expand-asm-exist  $\xi$  (n, ns)
  with new1-asm goal-assms have expand-asm-exist  $\xi$  ?x2
  proof (cases r, cases  $\psi$ , goal-cases)
    case prems'': (g - -  $\mu$   $\eta$ )
      then have *:  $\xi \models_r \mu R_r \eta$  unfolding fst-conv by fast
      with ltr-expand-Release[of  $\xi$   $\mu$   $\eta$ ] have  $\xi \models_r \eta$  by auto
      with prems'' * show ?case by auto
    qed auto
    with prems' have expand-rslt-exist  $\xi$  ?x2 (nm', nds')
      unfolding case-prod-unfold fst-conv snd-conv by fast
      then have expand-rslt-exist  $\xi$  (n, ns) (nm', nds') by (cases r, auto) }
  then show ?case by simp
qed
}
then have SPEC (?P ?x1)
   $\leq$  SPEC ( $\lambda r$ . (case r of (nm, nds) =>
    f (?x2f (nm, nds)))  $\leq$  SPEC (?P (n, ns)))
  using goal-assms by (rule-tac SPEC-rule) force
  finally have ?case unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$  by simp
}
ultimately show ?case by fast
qed

```

Termination proof

definition *expand_T* :: ('*a node* \times ('*a node set*)) \Rightarrow (*node-name* \times '*a node set*) *nres*
where *expand_T* *n-ns* \equiv *REC_T* *expand-body* *n-ns*

abbreviation *old-next-pair* *n* \equiv (*old n*, *next n*)

abbreviation *old-next-limit* φ \equiv *Pow* (*subfrmlsr* φ) \times *Pow* (*subfrmlsr* φ)

lemma *old-next-limit-finite*: *finite* (*old-next-limit* φ)
using *subfrmlsr-finite* **by** *auto*

definition

expand-ord φ \equiv
inv-image (*finite-psupset* (*old-next-limit* φ) $\langle *lex* \rangle$ *less-than*)
($\lambda(n, ns)$. (*old-next-pair* '*ns*, *size-set* (*new n*)))

lemma *expand-ord-wf*[*simp*]: *wf* (*expand-ord* φ)
using *finite-psupset-wf*[*OF old-next-limit-finite*]
unfolding *expand-ord-def* **by** *auto*

abbreviation

expand-inv-node φ *n*
 \equiv *finite* (*new n*) \wedge *finite* (*old n*) \wedge *finite* (*next n*)
 \wedge (*new n*) \cup (*old n*) \cup (*next n*) \subseteq *subfrmlsr* φ

abbreviation

$expand\text{-}inv\text{-}result\ \varphi\ ns \equiv finite\ ns \wedge (\forall n' \in ns. (new\ n') \cup (old\ n') \cup (next\ n') \subseteq subfrmlsr\ \varphi)$

definition

$expand\text{-}inv\ \varphi\ n\text{-}ns \equiv (case\ n\text{-}ns\ of\ (n, ns) \Rightarrow expand\text{-}inv\text{-}node\ \varphi\ n \wedge expand\text{-}inv\text{-}result\ \varphi\ ns)$

lemma new1-less-sum:

$size\text{-}set\ (new1\ \varphi) < size\text{-}set\ \{\varphi\}$

proof (cases φ)

case (And-ltlr $\nu\ \mu$)

thus ?thesis

by (cases $\nu = \mu$; simp)

qed (simp-all)

lemma new2-less-sum:

$size\text{-}set\ (new2\ \varphi) < size\text{-}set\ \{\varphi\}$

proof (cases φ)

case (Release-ltlr $\nu\ \mu$)

thus ?thesis

by (cases $\nu = \mu$; simp)

qed (simp-all)

lemma new1-finite[intro]: finite (new1 ψ)

by (cases ψ) auto

lemma new1-subset-frmls: $\varphi \in new1\ \psi \implies \varphi \in subfrmlsr\ \psi$

by (cases ψ) auto

lemma new2-finite[intro]: finite (new2 ψ)

by (cases ψ) auto

lemma new2-subset-frmls: $\varphi \in new2\ \psi \implies \varphi \in subfrmlsr\ \psi$

by (cases ψ) auto

lemma next1-finite[intro]: finite (next1 ψ)

by (cases ψ) auto

lemma next1-subset-frmls: $\varphi \in next1\ \psi \implies \varphi \in subfrmlsr\ \psi$

by (cases ψ) auto

lemma expand-inv-impl[intro!]:

assumes $expand\text{-}inv\ \varphi\ (n, ns)$

and $newn: \psi \in new\ n$

and $old\text{-}next\text{-}pair\ 'ns \subseteq old\text{-}next\text{-}pair\ 'ns'$

and $expand\text{-}inv\text{-}result\ \varphi\ ns'$

and $(n' = n \setminus new := new\ n - \{\psi\},$

$old := \{\psi\} \cup old\ n \setminus) \vee$

$(n' = n \setminus new := new1\ \psi \cup (new\ n - \{\psi\}),$

$old := \{\psi\} \cup old\ n,$

$next := next1\ \psi \cup next\ n \setminus) \vee$

```

      (n' = n | name := nm,
        new := new2  $\psi \cup (new\ n - \{\psi\})$ ,
        old :=  $\{\psi\} \cup old\ n$  |)
    (is ?case1  $\vee$  ?case2  $\vee$  ?case3)
  shows ((n', ns'), (n, ns))  $\in$  expand-ord  $\varphi \wedge$  expand-inv  $\varphi$  (n', ns')
    (is ?concl1  $\wedge$  ?concl2)
proof
from assms consider ?case1 | ?case2 | ?case3 by blast
then show ?concl1
proof cases
  case n'def: 1
    with assms show ?thesis
      unfolding expand-ord-def expand-inv-def finite-psupset-def
      apply (cases old-next-pair ' ns  $\subset$  old-next-pair ' ns')
      apply simp-all
      apply auto [1]
      apply (metis (no-types, lifting) add-Suc diff-Suc-less psubsetI sum.remove
        sum-diff1-nat zero-less-Suc)
      done
  next
    case n'def: 2
      have  $\psi$ innew:  $\psi \in new\ n$  and fin-new: finite (new n)
        using assms unfolding expand-inv-def by auto
      then have size-set (new n -  $\{\psi\}$ ) = size-set (new n) - size-set  $\{\psi\}$ 
        using size-set-diff by fastforce
      moreover
      from fin-new sum-Un-nat[OF new1-finite -, of new n -  $\{\psi\}$  size  $\psi$ ]
      have size-set (new n')  $\leq$  size-set (new1  $\psi$ ) + size-set (new n -  $\{\psi\}$ )
        unfolding n'def by (simp add: new1-finite sum-Un-nat)
      moreover
      have sum-leq: size-set (new n)  $\geq$  size-set  $\{\psi\}$ 
        using  $\psi$ innew sum-mono2[OF fin-new, of  $\{\psi\}$ ]
        by blast
      ultimately
      have size-set (new n')  $<$  size-set (new n)
        using new1-less-sum[of  $\psi$ ] by auto
      with assms show ?thesis
        unfolding expand-ord-def finite-psupset-def by auto
  next
    case n'def: 3
      have  $\psi$ innew:  $\psi \in new\ n$  and fin-new: finite (new n)
        using assms unfolding expand-inv-def by auto
      from  $\psi$ innew sum-diff1-nat[of size new n  $\psi$ ]
      have size-set (new n -  $\{\psi\}$ ) = size-set (new n) - size-set  $\{\psi\}$ 
        using size-set-diff[of new n  $\{\psi\}$ ] by fastforce
      moreover
      from fin-new sum-Un-nat[OF new2-finite -, of new n -  $\{\psi\}$  size  $\psi$ ]
      have size-set (new n')  $\leq$  size-set (new2  $\psi$ ) + size-set (new n -  $\{\psi\}$ )
        unfolding n'def by (simp add: new2-finite sum-Un-nat)

```

```

moreover
have sum-leq:size-set (new n) ≥ size-set {ψ}
  using ψinnew sum-mono2[OF fin-new, of {ψ}] by blast
ultimately
have size-set (new n') < size-set (new n)
  using new2-less-sum[of ψ] sum-leq by auto
with assms show ?thesis
  unfolding expand-ord-def finite-psupset-def by auto
qed
next
have new1 ψ ⊆ subfrmlsr φ
and new2 ψ ⊆ subfrmlsr φ
and next1 ψ ⊆ subfrmlsr φ
using assms subfrmlsr-subset[OF new1-subset-frmls[of - ψ]]
  subfrmlsr-subset[of ψ φ, OF rev-subsetD[of - new n]]
  subfrmlsr-subset[OF new2-subset-frmls[of - ψ]]
  subfrmlsr-subset[OF next1-subset-frmls[of - ψ]]
unfolding expand-inv-def

apply –
apply (clarsimp split: prod.splits)
apply (metis in-mono new1-subset-frmls)
apply (clarsimp split: prod.splits)
apply (metis new2-subset-frmls rev-subsetD)
apply (clarsimp split: prod.splits)
apply (metis in-mono next1-subset-frmls)
done
with assms show ?concl2
  unfolding expand-inv-def
  by auto
qed

lemma expand-term-prop-help:
assumes ((n', ns'), (n, ns)) ∈ expand-ord φ ∧ expand-inv φ (n', ns')
and assm-rule: [expand-inv φ (n', ns'); ((n', ns'), n, ns) ∈ expand-ord φ]
   $\implies f (n', ns') \leq SPEC P$ 
shows  $f (n', ns') \leq SPEC P$ 
using assms by (rule-tac assm-rule) auto

lemma expand-inv-upd-incoming:
assumes expand-inv φ (n, ns)
shows expand-inv-result φ (upd-incoming n ns)
using assms unfolding expand-inv-def upd-incoming-def by force

lemma upd-incoming-eq-old-next-pair: old-next-pair ' ns = old-next-pair ' (upd-incoming
n ns)
  (is ?A = ?B)
proof

```

```

show ?A ⊆ ?B
proof
  fix x
  let ?f = λn'. n'(incoming := incoming n ∪ incoming n')
  assume x ∈ ?A
  then obtain n' where n' ∈ ns and xeq: x = (old n', next n')
  by auto
  have x ∈ (old-next-pair ' (λn'. n'(incoming := incoming n ∪ incoming n'))
    ' (ns ∩ {n' ∈ ns. old n' = old n ∧ next n' = next n}))
    ∪ (old-next-pair ' (ns ∩ {n' ∈ ns. old n' ≠ old n ∨ next n' ≠ next n}))
  proof (cases old n' = old n ∧ next n' = next n)
  case True
  with ⟨n' ∈ ns⟩
  have ?f n' ∈ ?f ' (ns ∩ {n' ∈ ns. old n' = old n ∧ next n' = next n}) (is - ∈
?C)
    by auto
  then have old-next-pair (?f n') ∈ old-next-pair ' ?C
    by (rule-tac imageI) auto
  with xeq have x ∈ old-next-pair ' ?C by auto
  then show ?thesis by blast
next
case False
with ⟨n' ∈ ns⟩ xeq
have x ∈ old-next-pair ' (ns ∩ {n' ∈ ns. old n' ≠ old n ∨ next n' ≠ next n})
  by auto
then show ?thesis by blast
qed
then show x ∈ ?B
  using ⟨x ∈ ?A⟩ unfolding upd-incoming-def by auto
qed
show ?B ⊆ ?A
  unfolding upd-incoming-def by (force intro:imageI)
qed

```

lemma *expand-term-prop*:

```

expand-inv φ n-ns ⇒
  expandT n-ns ≤ SPEC (λ(-, nds). old-next-pair ' snd n-ns ⊆ old-next-pair ' nds
  ∧ expand-inv-result φ nds)
(is - ⇒ - ≤ SPEC (?P n-ns))
unfolding expandT-def
  apply (rule-tac RECT-rule[where pre=λ(n, ns). expand-inv φ (n, ns) and
V=expand-ord φ])
  apply (refine-mono)
  apply simp
  apply simp
proof (intro refine-vcg, goal-cases)
case prems: (1 - - n ns)
have old-next-pair ' ns ⊆ old-next-pair ' (upd-incoming n ns)
  by (rule equalityD1[OF upd-incoming-eq-old-next-pair])

```



```

with prems show ?case
  using expand-inv-upd-incoming[of  $\varphi$  n ns] by auto
next
case prems: ( $\exists$  expand x n ns)
let ?n' = ()
  name = expand-new-name (name n),
  incoming = {name n},
  new = next n,
  old = {},
  next = {}
let ?ns' = {n}  $\cup$  ns
from prems have SPEC-sub:SPEC (?P (?n', ?ns'))  $\leq$  SPEC (?P x)
  by (rule-tac SPEC-rule) auto
from prems have old-next-pair n  $\notin$  old-next-pair ' ns
  by auto
then have old-next-pair ' ns  $\subset$  old-next-pair ' (insert n ns)
  by auto
moreover from prems have expand-inv  $\varphi$  (n, ns)
  by simp
ultimately have ((?n', ?ns'), (n, ns))  $\in$  expand-ord  $\varphi$ 
  by (auto simp add: expand-ord-def finite-psupset-def expand-inv-def)
moreover from prems have expand-inv  $\varphi$  (?n', ?ns')
  unfolding expand-inv-def by auto
ultimately have expand (?n', ?ns')  $\leq$  SPEC (?P (?n', ?ns'))
  using prems by fast
with SPEC-sub show ?case
  by (rule-tac order-trans) fast+
next
case 3
  then show ?case by (auto simp add: expand-inv-def)
next
case 4
  then show ?case
    apply (rule-tac expand-term-prop-help[OF expand-inv-impl])
    apply (simp add: expand-inv-def)+
    apply force
    done
next
case 5
  then show ?case
    apply (rule-tac expand-term-prop-help[OF expand-inv-impl])
    apply (simp add: expand-inv-def)+
    apply force
    done
next
case 6
  then show ?case by (simp add: expand-inv-def)
next
case 7

```

```

then show ?case
  apply (rule-tac expand-term-prop-help[OF expand-inv-impl])
  apply (simp add: expand-inv-def)+
  apply force
  done
next
case prems: (∑ f x a b xa)
let ?n' = a⟦
  new := new1 xa ∪ (new a - {xa}),
  old := insert xa (old a),
  next := next1 xa ∪ next a⟧
let ?n'' = λnm. a⟦
  name := nm,
  new := new2 xa ∪ (new a - {xa}),
  old := insert xa (old a)⟧
have step:((?n', b), (a, b)) ∈ expand-ord φ ∧ expand-inv φ (?n', b)
  using prems by (rule-tac expand-inv-impl) (auto simp add: expand-inv-def)
with prems have assm1: f (?n', b) ≤ SPEC (?P (a, b))
  by auto
moreover
{
  fix nm::node-name and nds::'a node set
  assume assm1: old-next-pair ' b ⊆ old-next-pair ' nds
  and assm2: expand-inv-result φ nds
  with prems step have ((?n'' nm, nds), (a, b)) ∈ expand-ord φ ∧ expand-inv φ
  (?n'' nm, nds)
  by (rule-tac expand-inv-impl) auto
  with prems have f (?n'' nm, nds) ≤ SPEC (?P (?n'' nm, nds))
  by auto
  moreover
  have SPEC (?P (?n'' nm, nds)) ≤ SPEC (?P (a, b))
  using assm2 subset-trans[OF assm1] by auto
  ultimately have f (?n'' nm, nds) ≤ SPEC (?P (a, b))
  by (rule order-trans)
}
then have assm2: SPEC (?P (a, b))
  ≤ SPEC (λr. (case r of (nm, nds) ⇒ f (?n'' nm, nds)) ≤ SPEC (?P (a, b)))
  by (rule-tac SPEC-rule) auto
from prems order-trans[OF assm1 assm2] show ?case
  by auto
qed

```

```

lemma expand-eq-expandT:
  assumes inv: expand-inv φ n-ns
  shows expandT n-ns = expand n-ns
  unfolding expandT-def expand-def
  apply (rule RECT-eq-REC)
  unfolding expandT-def[symmetric]
  using expand-term-prop[OF inv] apply auto

```

done

lemma *expand-nofail*:

assumes *inv*: *expand-inv* φ *n-ns*

shows *nofail* (*expand_T* *n-ns*)

using *expand-term-prop*[*OF inv*] **by** (*simp add: pw-le-iff*)

lemma [*intro!*]: *expand-inv* φ (

(
 name = *expand-new-name expand-init*,
 incoming = {*expand-init*},
 new = { φ },
 old = {},
 next = {})

{})
by (*auto simp add: expand-inv-def*)

definition *create-graph* :: 'a *frml* \Rightarrow 'a *node set nres*

where

create-graph $\varphi \equiv$
 do {
 (-, *nds*) \leftarrow *expand* (
 (
 name = *expand-new-name expand-init*,
 incoming = {*expand-init*},
 new = { φ },
 old = {},
 next = {}
)::'a *node*,
 { }::'a *node set*);
 RETURN *nds*
 }

definition *create-graph_T* :: 'a *frml* \Rightarrow 'a *node set nres*

where

create-graph_T $\varphi \equiv$ do {
 (-, *nds*) \leftarrow *expand_T* (
 (
 name = *expand-new-name expand-init*,
 incoming = {*expand-init*},
 new = { φ },
 old = {},
 next = {}
)::'a *node*,
 { }::'a *node set*);
 RETURN *nds*
}

lemma *create-graph-eq-create-graph_T*: *create-graph* $\varphi = \text{create-graph}_T \varphi$
unfolding *create-graph_T-def* *create-graph-def*
unfolding *eq-iff*
proof (*standard, goal-cases*)
case 1
then show ?*case*
by *refine-mono (unfold expand-def expand_T-def, rule REC-le-RECT)*
next
case 2
then show ?*case*
by (*refine-mono, rule expand-eq-expand_T[unfolded eq-iff, THEN conjunct1]*)
auto
qed

lemma *create-graph-finite*: *create-graph* $\varphi \leq \text{SPEC finite}$
unfolding *create-graph-def* *expand-def*
apply (*intro refine-vcg*)
apply (*rule-tac order-trans*)
apply (*rule-tac REC-le-RECT*)
apply (*fold expand_T-def*)
apply (*rule-tac order-trans[OF expand-term-prop]*)
apply *auto[1]*
apply (*rule-tac SPEC-rule*)
apply *auto*
done

lemma *create-graph-nofail*: *nofail (create-graph φ)*
by (*rule-tac pwD1[OF create-graph-finite]*) *auto*

abbreviation

create-graph-rslt-exist ξ *nds*
 $\equiv \exists nd \in nds.$
 $\quad \text{expand-init} \in \text{incoming } nd$
 $\quad \wedge (\forall \psi \in \text{old } nd. \xi \models_r \psi) \wedge (\forall \psi \in \text{next } nd. \xi \models_r X_r \psi)$
 $\quad \wedge \{ \eta. \exists \mu. \mu U_r \eta \in \text{old } nd \wedge \xi \models_r \eta \} \subseteq \text{old } nd$

lemma *L₄-7*:

assumes $\xi \models_r \varphi$
shows *create-graph* $\varphi \leq \text{SPEC (create-graph-rslt-exist } \xi)$
using *assms* **unfolding** *create-graph-def*
by (*intro refine-vcg, rule-tac order-trans, rule-tac expand-prop-exist*) (*auto simp add: expand-new-name-expand-init*)

lemma *expand-incoming-name-exist*:

assumes *name (fst n-ns) > expand-init*
 $\wedge (\forall nm \in \text{incoming (fst n-ns)}. nm \neq \text{expand-init} \longrightarrow nm \in \text{name ' (snd n-ns)})$

```

  ∧ expand-assm-incoming n-ns ∧ expand-name-ident (snd n-ns) (is ?Q n-ns)
and ∀ nd ∈ snd n-ns.
  name nd > expand-init
  ∧ (∀ nm ∈ incoming nd. nm ≠ expand-init → nm ∈ name ' (snd n-ns))
  (is ?P (snd n-ns))
shows expand n-ns ≤ SPEC (λ nm-nds. ?P (snd nm-nds))
using assms
apply (rule-tac expand-rec-rule[where Φ = λ n-ns. ?Q n-ns ∧ ?P (snd n-ns)])
apply simp
apply (intro refine-vcg)
proof goal-cases
case (1 f x n ns)
then show ?case
proof (simp, clarify, goal-cases)
  case prems: (1 - - nd)
  { assume nd ∈ ns
    with prems have ?case by auto }
  moreover
  { assume nd ∉ ns
    with upd-incoming--elem[OF ‹nd ∈ upd-incoming n ns›]
    obtain nd' where nd' ∈ ns and nd = nd' (incoming :=
      incoming n ∪ incoming nd') ∧
      old nd' = old n ∧
      next nd' = next n by auto
    with prems have ?case by auto }
  ultimately show ?case by fast
qed
next
case (2 f x n ns)
then have step: ∧ x. ?Q x ∧ ?P (snd x) ⇒ f x ≤ SPEC (λ x. ?P (snd x))
  and QP: ?Q (n, ns) ∧ ?P ns
  and f-sup: ∧ x. f x ≤ expand x by auto
show ?case
  unfolding ‹x = (n, ns)›
  using QP expand-new-name-expand-init
proof (rule-tac step, goal-cases)
  case prems: 1
  then have name-less: name n < expand-new-name (name n)
    by auto
  moreover
  from prems name-less have ∀ nm ∈ incoming n. nm < expand-new-name (name
n)
    by auto
  moreover
  from prems name-less have **: ∀ q ∈ ns. name q < expand-new-name (name n)
  ∧
  (∀ nm ∈ incoming q. nm < expand-new-name (name n))
  by force
  moreover

```

```

from  $QP$  have  $\exists!q'. (q' = n \vee q' \in ns) \wedge \text{name } n = \text{name } q'$ 
  by force
moreover
have  $\forall q \in ns. \exists!q'.$ 
   $(q' = n \vee q' \in ns) \wedge$ 
   $\text{name } q = \text{name } q'$  using  $QP$  by auto
ultimately show  $?case$  using  $\text{prems}$  by simp
qed
next
  case 3
  then show  $?case$  by simp
next
  case 4
  then show  $?case$  by simp
next
  case 5
  then show  $?case$  by simp
next
  case 6
  then show  $?case$  by simp
next
  case 7
  then show  $?case$  by simp
next
  case  $\text{prems}: (8 f x n ns \psi)$ 
  then have  $\text{goal-assms}: \psi \in \text{new } n \wedge (\exists \nu \mu. \psi = \nu \text{ or}_r \mu \vee \psi = \nu U_r \mu \vee \psi =$ 
 $\nu R_r \mu)$ 
  by  $(\text{cases } \psi)$  auto
  with  $\text{prems}$  have  $QP: ?Q (n, ns) \wedge ?P ns$ 
  and  $\text{step}: \bigwedge x. ?Q x \wedge ?P (\text{snd } x) \implies f x \leq \text{SPEC } (\lambda x'. ?P (\text{snd } x'))$ 
  and  $f\text{-sup}: \bigwedge x. f x \leq \text{expand } x$  by auto
  let  $?x = (n(\text{new} := \text{new } n - \{\psi\}, \text{new} := \text{new1 } \psi \cup \text{new } (n(\text{new} := \text{new } n -$ 
 $\{\psi\}))),$ 
   $\text{old} := \{\psi\} \cup \text{old } (n(\text{new} := \text{new } n - \{\psi\})),$ 
   $\text{next} := \text{next1 } \psi \cup \text{next } (n(\text{new} := \text{new } n - \{\psi\})), ns)$ 
  let  $?props = \lambda x r. \text{expand-rslt-incoming } r$ 
   $\wedge \text{expand-rslt-name } x r$ 
   $\wedge \text{expand-name-ident } (\text{snd } r)$ 

show  $?case$ 
  using  $\text{goal-assms } QP$  unfolding  $\text{case-prod-unfold } \langle x = (n, ns) \rangle$ 
proof  $(\text{rule-tac } \text{SPEC-rule-weak-nested2}[\text{where } Q = ?props ?x], \text{goal-cases})$ 
  case 1
  then show  $?case$ 
  by  $(\text{rule-tac } \text{order-trans}, \text{rule-tac } f\text{-sup}, \text{rule-tac } \text{expand-name-propag})$  simp
next
  case 2
  then show  $?case$ 
  by  $(\text{rule-tac } \text{SPEC-rule-param2}[\text{where } P = \lambda x r. ?P (\text{snd } r)], \text{rule-tac } \text{step})$ 

```

```

      auto
next
  case ( $\exists$  nm nds)
  then show ?case
  proof (rule-tac SPEC-rule-weak[where P =  $\lambda x r. ?P$  (snd r)
    and Q =  $\lambda x r. \text{expand-rslt-exist-eq } x r \wedge ?\text{props } x r$ ], goal-cases)
  case 1
  then show ?case
  by (rule-tac SPEC-rule-conjI,
    rule-tac order-trans,
    rule-tac f-sup,
    rule-tac expand-rslt-exist-eq,
    rule-tac order-trans,
    rule-tac f-sup,
    rule-tac expand-name-propag) force
next
  case 2
  then show ?case
  by (rule-tac SPEC-rule-param2[where P =  $\lambda x r. ?P$  (snd r)],
    rule-tac step) force
next
  case ( $\exists$  nm' nds')
  then show ?case
  by simp
qed
qed
qed

```

lemma *create-graph--incoming-name-exist*:
 $\text{create-graph } \varphi \leq \text{SPEC } (\lambda \text{nds}. \forall \text{nd} \in \text{nds}. \text{expand-init} < \text{name nd} \wedge (\forall \text{nm} \in \text{incoming}$
 $\text{nd}. \text{nm} \neq \text{expand-init} \longrightarrow \text{nm} \in \text{name ' nds}))$
unfolding *create-graph-def*
by (*intro refine-vcg*,
rule-tac order-trans,
rule-tac expand-incoming-name-exist) (
auto simp add: expand-new-name-expand-init)

abbreviation

$$\begin{aligned}
&\text{expand-rslt-all--ex-equiv } \xi \text{ nd nds} \equiv \\
&(\exists \text{nd}' \in \text{nds}. \\
&\text{name nd} \in \text{incoming nd}' \\
&\wedge (\forall \psi \in \text{old nd}'. \text{suffix } 1 \xi \models_r \psi) \wedge (\forall \psi \in \text{next nd}'. \text{suffix } 1 \xi \models_r X_r \psi) \\
&\wedge \{\eta. \exists \mu. \mu U_r \eta \in \text{old nd}' \wedge \text{suffix } 1 \xi \models_r \eta\} \subseteq \text{old nd}')
\end{aligned}$$

abbreviation

$$\begin{aligned}
&\text{expand-rslt-all } \xi \text{ n-ns nm-nds} \equiv \\
&(\forall \text{nd} \in \text{snd nm-nds}. \text{name nd} \notin \text{name ' (snd n-ns)} \wedge \\
&(\forall \psi \in \text{old nd}. \xi \models_r \psi) \wedge (\forall \psi \in \text{next nd}. \xi \models_r X_r \psi))
\end{aligned}$$

→ *expand-rslt-all-ex-equiv* ξ *nd* (*snd nm-nds*)

lemma *expand-prop-all*:

assumes *expand-assm-incoming* *n-ns* \wedge *expand-name-ident* (*snd n-ns*) (**is** *?Q* *n-ns*)

shows *expand n-ns* \leq *SPEC* (*expand-rslt-all* ξ *n-ns*)
(is \leq *SPEC* (*?P n-ns*)**)**

using *assms*

apply (*rule-tac expand-rec-rule*[**where** $\Phi = ?Q$])

apply *simp*

apply (*intro refine-vcg*)

proof *goal-cases*

case 1

then show *?case* **by** (*simp*, *rule-tac upd-incoming-ident*) *simp-all*

next

case ($2 f x n ns$)

then have *step*: $\bigwedge x. ?Q x \implies f x \leq SPEC (?P x)$

and *Q*: *?Q* (*n*, *ns*)

and *f-sup*: $\bigwedge x. f x \leq \text{expand } x$ **by** *auto*

let *?x* = ($(name = \text{expand-new-name } (name\ n),$

incoming = $\{name\ n\}$, *new* = *next n*, *old* = $\{\}$, *next* = $\{\}$), $\{n\} \cup ns$)

from *Q* **have** *name-le*: *name n* < *expand-new-name* (*name n*) **by** *auto*

show *?case*

unfolding $\langle x = (n, ns) \rangle$

proof (*rule-tac SPEC-rule-weak*[**where**

Q = $\lambda p r.$

(*expand-assm-exist* (*suffix 1* ξ) *?x* → *expand-rslt-exist* (*suffix 1* ξ) *?x r*)

\wedge *expand-rslt-exist-eq* *p r* \wedge (*expand-name-ident* (*snd r*))], *goal-cases*)

case 1

then show *?case*

proof (*rule-tac SPEC-rule-conjI*,

rule-tac order-trans,

rule-tac f-sup,

rule-tac expand-prop-exist,

rule-tac SPEC-rule-conjI,

rule-tac order-trans,

rule-tac f-sup,

rule-tac expand-rslt-exist-eq,

rule-tac order-trans,

rule-tac f-sup,

rule-tac expand-name-propag-name-ident,

goal-cases)

case 1

then show *?case* **using** *Q name-le* **by** *force*

qed

next

case 2

then show *?case* **using** *Q name-le* **by** (*rule-tac step*) *force*

next


```

case prems: ( $\exists$  nm nds)
then obtain n' where  $n' \in nds$ 
  and eq-node: expand-rslt-exist-eg--node n n' by auto
with prems have ex1-name:  $\exists! q \in nds. \text{name } n = \text{name } q$  by auto
then have nds-eg:  $nds = \{n'\} \cup \{x \in nds. \text{name } n \neq \text{name } x\}$ 
  using eq-node  $\langle n' \in nds \rangle$  by blast
have name-notin:  $\text{name } n \notin \text{name } 'ns$  using Q by auto
from prems have P-x: expand-rslt-all  $\xi$   $?x$  (nm, nds) by fast
show ?case
  unfolding snd-conv
proof clarify
  fix nd
  assume  $nd \in nds$  and name-img:  $\text{name } nd \notin \text{name } 'ns$ 
    and nd-old-equiv:  $\forall \psi \in \text{old } nd. \xi \models_r \psi$ 
    and nd-next-equiv:  $\forall \psi \in \text{next } nd. \xi \models_r X_r \psi$ 
  show expand-rslt-all--ex-equiv  $\xi$  nd nds
  proof (cases  $\text{name } nd = \text{name } n$ )
    case True
      with nds-eg eq-node  $\langle nd \in nds \rangle$  have  $nd = n'$  by auto
      with prems(1)[THEN conjunct1, simplified]
        nd-old-equiv nd-next-equiv eq-node
      show ?thesis by simp
    next
      case False
      with name-img  $\langle nd \in nds \rangle$  nd-old-equiv nd-next-equiv P-x
      show ?thesis by simp
  qed
qed
qed
next
  case  $\exists$ 
  then show ?case by auto
next
  case prems: ( $\wedge$  f)
  then have step:  $\wedge x. ?Q x \implies f x \leq \text{SPEC } (?P x)$  by simp
  from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case prems: ( $\wedge$  f)
  then have step:  $\wedge x. ?Q x \implies f x \leq \text{SPEC } (?P x)$  by simp
  from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case  $\exists$ 
  then show ?case by auto
next
  case prems: ( $\wedge$  f)
  then have step:  $\wedge x. ?Q x \implies f x \leq \text{SPEC } (?P x)$  by simp
  from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case prems: ( $\exists$  f x n ns  $\psi$ )

```

```

then have goal-assms:  $\psi \in \text{new } n \wedge (\exists \nu \mu. \psi = \nu \text{ or}_r \mu \vee \psi = \nu U_r \mu \vee \psi = \nu R_r \mu)$ 
  by (cases  $\psi$ ) auto
from prems have Q:  $?Q (n, ns)$ 
  and step:  $\bigwedge x. ?Q x \implies f x \leq \text{SPEC } (?P x)$ 
  and f-sup:  $\bigwedge x. f x \leq \text{expand } x$  by auto
let ?x = (n⟦new := new n - { $\psi$ }, new := new1  $\psi \cup \text{new } (n⟦\text{new} := \text{new } n - \{\psi\})$ ⟧),
      old := { $\psi$ }  $\cup$  old (n⟦new := new n - { $\psi$ })⟧),
      next := next1  $\psi \cup \text{next } (n⟦\text{new} := \text{new } n - \{\psi\})$ ⟧),
      ns)
let ?props =  $\lambda x r. \text{expand-rslt-incoming } r$ 
   $\wedge \text{expand-rslt-name } x r$ 
   $\wedge \text{expand-name-ident } (\text{snd } r)$ 
show ?case
  using goal-assms Q
  unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
proof (rule-tac SPEC-rule-weak-nested2[where Q = ?props ?x], goal-cases)
  case 1
  then show ?case
    by (rule-tac order-trans, rule-tac f-sup, rule-tac expand-name-propag) simp
  next
  case 2
  then show ?case
    by (rule-tac SPEC-rule-param2[where P = ?P], rule-tac step) auto
  next
  case prems: ( $\exists nm \text{ nds}$ )
  then show ?case
  proof (rule-tac SPEC-rule-weak[where
    P = ?P and
    Q =  $\lambda x r. \text{expand-rslt-exist-eq } x r \wedge ?props x r$ ], goal-cases)
  case 1
  then show ?case
    by (rule-tac SPEC-rule-conjI,
      rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-rslt-exist-eq,
      rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-name-propag) auto
  next
  case 2
  then show ?case
    by (rule-tac SPEC-rule-param2[where P = ?P], rule-tac step) auto
  next
  case prems': ( $\exists nm' \text{ nds}'$ )
  then have P-x:  $?P (n, ns) (nm, nds)$ 
  and P-x':  $?P (n, nds) (nm', nds')$  by simp-all
  show ?case

```

```

unfolding snd-conv
proof clarify
  fix nd
  assume  $nd \in nds'$ 
    and name-nd-notin:  $name\ nd \notin name\ 'ns$ 
    and old-equiv:  $\forall \psi \in old\ nd. \xi \models_r \psi$ 
    and next-equiv:  $\forall \psi \in next\ nd. \xi \models_r X_r \psi$ 
  show expand-rslt-all-ex-equiv  $\xi\ nd\ nds'$ 
  proof (cases  $name\ nd \in name\ 'nds$ )
    case True
      then obtain  $n'$  where  $n' \in nds$  and  $name\ nd = name\ n'$  by auto
      with prems' obtain  $nd'$  where  $nd' \in nds'$ 
        and nd'-eq: expand-rslt-exist-eq--node  $n'\ nd'$ 
        by auto
      moreover from prems' have  $\forall q \in nds'. \exists !q' \in nds'. name\ q = name\ q'$ 
        by simp
      ultimately have  $nd' = nd$ 
        using  $\langle name\ nd = name\ n' \rangle \langle nd \in nds' \rangle$  by auto
      with nd'-eq have n'-eq: expand-rslt-exist-eq--node  $n'\ nd$ 
        by simp
      then have  $name\ n' \notin name\ 'ns$  and  $\forall \psi \in old\ n'. \xi \models_r \psi$  and  $\forall \psi \in next\ n'. \xi \models_r X_r \psi$ 
        using name-nd-notin old-equiv next-equiv  $\langle n' \in nds \rangle$ 
        by auto
      then have expand-rslt-all-ex-equiv  $\xi\ n'\ nds$  (is  $\exists nd' \in nds. ?sthm\ n'\ nd'$ )
        using P-x  $\langle n' \in nds \rangle$  unfolding snd-conv by blast
      then obtain sucnd where sucnd:  $sucnd \in nds$  and sthm:  $?sthm\ n'\ sucnd$ 
        by blast
      moreover
        from prems' sucnd sthm obtain  $sucnd'$  where  $sucnd' \in nds'$ 
          and sucnd'-eq: expand-rslt-exist-eq--node  $sucnd\ sucnd'$ 
          by auto
        ultimately have  $?sthm\ n'\ sucnd'$  by auto
        then show ?thesis
          using  $\langle sucnd' \in nds' \rangle$ 
          unfolding  $\langle name\ nd = name\ n' \rangle$  by blast
    next
      case False
        with  $\langle nd \in nds' \rangle$  P-x' old-equiv next-equiv
        show ?thesis unfolding snd-conv by blast
    qed
  qed
qed
qed

```

abbreviation

```

create-graph-rslt-all  $\xi\ nds$ 
 $\equiv \forall q \in nds. (\forall \psi \in old\ q. \xi \models_r \psi) \wedge (\forall \psi \in next\ q. \xi \models_r X_r \psi)$ 

```

$$\begin{aligned} &\longrightarrow (\exists q' \in \text{nds}. \text{name } q \in \text{incoming } q') \\ &\quad \wedge (\forall \psi \in \text{old } q'. \text{suffix } 1 \ \xi \models_r \psi) \\ &\quad \wedge (\forall \psi \in \text{next } q'. \text{suffix } 1 \ \xi \models_r X_r \psi) \\ &\quad \wedge \{\eta. \exists \mu. \mu \ U_r \ \eta \in \text{old } q' \wedge \text{suffix } 1 \ \xi \models_r \eta\} \subseteq \text{old } q' \end{aligned}$$

lemma *L4-5: create-graph $\varphi \leq \text{SPEC}(\text{create-graph-rslt-all } \xi)$*
unfolding *create-graph-def*
apply (*refine-vcg expand-prop-all*)
apply (*auto simp add: expand-new-name-expand-init*)
done

2.4 Creation of GBA

This section formalizes the second part of the algorithm, that creates the actual generalized Büchi automata from the set of nodes.

definition *create-gba-from-nodes* :: 'a frml \Rightarrow 'a node set \Rightarrow ('a node, 'a set) gba-rec
where *create-gba-from-nodes* φ *qs* \equiv (
 $g\text{-}V = \text{qs}$,
 $g\text{-}E = \{(q, q'). q \in \text{qs} \wedge q' \in \text{qs} \wedge \text{name } q \in \text{incoming } q'\}$,
 $g\text{-}V0 = \{q \in \text{qs}. \text{expand-init} \in \text{incoming } q\}$,
 $gbg\text{-}F = \{\{q \in \text{qs}. \mu \ U_r \ \eta \in \text{old } q \longrightarrow \eta \in \text{old } q\} \mid \mu \ \eta. \mu \ U_r \ \eta \in \text{subfrmlsr } \varphi\}$,
 $gba\text{-}L = \lambda q \ l. q \in \text{qs} \wedge \{p. \text{prop}_r(p) \in \text{old } q\} \subseteq l \wedge \{p. \text{nprop}_r(p) \in \text{old } q\} \cap l = \{\}$
 \rangle

end

locale *create-gba-from-nodes-precond* =
fixes φ :: 'a ltlr
fixes *qs* :: 'a node set
assumes *res*: *inres* (*create-graph* φ) *qs*
begin

lemma *finite-qs[simp, intro!]: finite qs*
using *res create-graph-finite* **by** (*auto simp add: pw-le-iff*)

lemma *create-gba-from-nodes--invar: gba* (*create-gba-from-nodes* φ *qs*)
using [[*simproc finite-Collect*]]
apply *unfold-locales*
apply (*auto*
 $\text{intro!}: \text{finite-vimageI subfrmlsr-finite injI}$
 $\text{simp}: \text{create-gba-from-nodes-def}$)
done

sublocale *gba: gba* *create-gba-from-nodes* φ *qs*
by (*rule create-gba-from-nodes--invar*)

lemma *create-gba-from-nodes--fin: finite* ($g\text{-}V$ (*create-gba-from-nodes* φ *qs*))
unfolding *create-gba-from-nodes-def* **by** *auto*

lemma *create-gba-from-nodes--ipath*:
ipath *gba.E* $r \longleftrightarrow (\forall i. r\ i \in qs \wedge name\ (r\ i) \in incoming\ (r\ (Suc\ i)))$
unfolding *create-gba-from-nodes-def ipath-def*
by *auto*

lemma *create-gba-from-nodes--is-run*:
gba.is-run $r \longleftrightarrow expand-init \in incoming\ (r\ 0)$
 $\wedge (\forall i. r\ i \in qs \wedge name\ (r\ i) \in incoming\ (r\ (Suc\ i)))$
unfolding *gba.is-run-def*
apply (*simp add: create-gba-from-nodes--ipath*)
apply (*auto simp: create-gba-from-nodes-def*)
done

context
begin

abbreviation

auto-run-j $j\ \xi\ q \equiv$
 $(\forall \psi \in old\ q. suffix\ j\ \xi \models_r \psi) \wedge (\forall \psi \in next\ q. suffix\ j\ \xi \models_r X_r\ \psi) \wedge$
 $\{\eta. \exists \mu. \mu\ U_r\ \eta \in old\ q \wedge suffix\ j\ \xi \models_r \eta\} \subseteq old\ q$

fun *auto-run* :: [*'a* *interpret*, *'a* *node set*] \Rightarrow *'a* *node word*
where

auto-run $\xi\ nds\ 0$
 $= (SOME\ q. q \in nds \wedge expand-init \in incoming\ q \wedge auto-run-j\ 0\ \xi\ q)$
| *auto-run* $\xi\ nds\ (Suc\ k)$
 $= (SOME\ q'. q' \in nds \wedge name\ (auto-run\ \xi\ nds\ k) \in incoming\ q'$
 $\wedge auto-run-j\ (Suc\ k)\ \xi\ q')$

lemma *run-propag-on-create-graph*:

assumes *ipath* *gba.E* σ
shows $\sigma\ k \in qs \wedge name\ (\sigma\ k) \in incoming\ (\sigma\ (k+1))$
using *assms*
by (*auto simp: create-gba-from-nodes--ipath*)

lemma *expand-false-propag*:

assumes $false_r \notin old\ (fst\ n-ns) \wedge (\forall nd \in snd\ n-ns. false_r \notin old\ nd)$
(is *?Q* *n-ns*)
shows $expand\ n-ns \leq SPEC\ (\lambda nm-nds. \forall nd \in snd\ nm-nds. false_r \notin old\ nd)$
using *assms*
proof (*rule-tac expand-rec-rule*[**where** $\Phi = ?Q$], *simp*, *intro refine-vcg*, *goal-cases*)
case 1
then show *?case* **by** (*simp*, *rule-tac upd-incoming--ident*) *auto*
next
case 8

then show *?case* **by** (*rule-tac SPEC-rule-nested2*) *auto*
qed *auto*

lemma *false-propag-on-create-graph*: *create-graph* $\varphi \leq \text{SPEC} (\lambda nds. \forall nd \in nds. \text{false}_r \notin \text{old } nd)$

unfolding *create-graph-def*

by (*intro refine-vcg, rule-tac order-trans, rule-tac expand-false-propag*) *auto*

lemma *expand-and-propag*:

assumes $\mu \text{ and}_r \eta \in \text{old } (fst \ n\ ns)$

$\longrightarrow \{\mu, \eta\} \subseteq \text{old } (fst \ n\ ns) \cup \text{new } (fst \ n\ ns)$ (**is** *?Q* *n-ns*)

and $\forall nd \in \text{snd } n\ ns. \mu \text{ and}_r \eta \in \text{old } nd \longrightarrow \{\mu, \eta\} \subseteq \text{old } nd$ (**is** *?P* (*snd* *n-ns*))

shows *expand* *n-ns* $\leq \text{SPEC} (\lambda nm\ nds. ?P (\text{snd } nm\ nds))$

using *assms*

proof (*rule-tac expand-rec-rule*[**where** $\Phi = \lambda x. ?Q \ x \wedge ?P (\text{snd } x)$],

simp, intro refine-vcg, goal-cases)

case *1*

then show *?case* **by** (*simp, rule-tac upd-incoming-ident*) *auto*

next

case *prems*: (*4* *f* *x* *n* *ns*)

then have *step*: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \Longrightarrow f \ x \leq \text{SPEC} (\lambda x'. ?P (\text{snd } x'))$ **by**

simp

with *prems* **show** *?case* **by** (*rule-tac step*) *auto*

next

case *prems*: (*5* *f* *x* *n* *ns*)

then have *step*: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \Longrightarrow f \ x \leq \text{SPEC} (\lambda x'. ?P (\text{snd } x'))$ **by**

simp

with *prems* **show** *?case* **by** (*rule-tac step*) *auto*

next

case (*6* *f* *x* *n* *ns*)

then show *?case* **by** *auto*

next

case *prems*: (*7* *f* *x* *n* *ns*)

then have *step*: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \Longrightarrow f \ x \leq \text{SPEC} (\lambda x'. ?P (\text{snd } x'))$ **by**

simp

with *prems* **show** *?case* **by** (*rule-tac step*) *auto*

next

case *prems*: (*8* *f* *x* *n* *ns* ψ)

then have *goal-assms*: $\psi \in \text{new } n$

$\wedge \neg (\exists q. \psi = \text{prop}_r(q) \vee \psi = \text{nprop}_r(q))$

$\wedge \psi \neq \text{true}_r \wedge \psi \neq \text{false}_r$

$\wedge \neg (\exists \nu \mu. \psi = \nu \text{ and}_r \mu \vee \psi = X_r \ \nu)$

$\wedge (\exists \nu \mu. \psi = \nu \text{ or}_r \mu \vee \psi = \nu \ U_r \ \mu \vee \psi = \nu \ R_r \ \mu)$

by (*cases* ψ) *auto*

from *prems* **have** *QP*: $?Q (n, ns) \wedge ?P \ ns$

and *step*: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \Longrightarrow f \ x \leq \text{SPEC} (\lambda x'. ?P (\text{snd } x'))$

by *simp-all*

show *?case*

```

using goal-assms QP unfolding case-prod-unfold ⟨x = (n, ns)⟩
proof (rule-tac SPEC-rule-nested2, goal-cases)
  case 1
  then show ?case by (rule-tac step) auto
next
  case 2
  then show ?case by (rule-tac step) auto
qed
qed auto

```

lemma and-propag-on-create-graph:

create-graph $\varphi \leq \text{SPEC } (\lambda nds. \forall nd \in nds. \mu \text{ and}_r \eta \in \text{old } nd \longrightarrow \{\mu, \eta\} \subseteq \text{old } nd)$

unfolding create-graph-def

by (intro refine-vcg, rule-tac order-trans, rule-tac expand-and-propag) auto

lemma expand-or-propag:

assumes $\mu \text{ or}_r \eta \in \text{old } (fst \ n\text{-}ns)$

$\longrightarrow \{\mu, \eta\} \cap (\text{old } (fst \ n\text{-}ns) \cup \text{new } (fst \ n\text{-}ns)) \neq \{\}$ (**is** ?Q n-ns)

and $\forall nd \in \text{snd } n\text{-}ns. \mu \text{ or}_r \eta \in \text{old } nd \longrightarrow \{\mu, \eta\} \cap \text{old } nd \neq \{\}$

(**is** ?P (snd n-ns))

shows expand n-ns $\leq \text{SPEC } (\lambda nm\text{-}nds. ?P (\text{snd } nm\text{-}nds))$

using assms

proof (rule-tac expand-rec-rule[**where** $\Phi = \lambda x. ?Q \ x \wedge ?P (\text{snd } x)$],

simp, intro refine-vcg, goal-cases)

case 1

then show ?case **by** (simp, rule-tac upd-incoming--ident) auto

next

case prems: (4 f x n ns)

then have step: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \implies f \ x \leq \text{SPEC } (\lambda x'. ?P (\text{snd } x'))$ **by**

simp

with prems **show** ?case **by** (rule-tac step) auto

next

case prems: (5 f x n ns)

then have step: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \implies f \ x \leq \text{SPEC } (\lambda x'. ?P (\text{snd } x'))$ **by**

simp

with prems **show** ?case **by** (rule-tac step) auto

next

case (6 f x n ns)

then show ?case **by** auto

next

case prems: (7 f x n ns)

then have step: $\bigwedge x. ?Q \ x \wedge ?P (\text{snd } x) \implies f \ x \leq \text{SPEC } (\lambda x'. ?P (\text{snd } x'))$ **by**

simp

with prems **show** ?case **by** (rule-tac step) auto

next

case prems: (8 f x n ns ψ)

then have goal-assms: $\psi \in \text{new } n$

$\wedge \neg (\exists q. \psi = \text{prop}_r(q) \vee \psi = \text{nprop}_r(q))$

$\wedge \psi \neq \text{true}_r \wedge \psi \neq \text{false}_r$
 $\wedge \neg (\exists \nu \mu. \psi = \nu \text{ and}_r \mu \vee \psi = X_r \nu)$
 $\wedge (\exists \nu \mu. \psi = \nu \text{ or}_r \mu \vee \psi = \nu U_r \mu \vee \psi = \nu R_r \mu)$
by (*cases* ψ) *auto*
from *prems* **have** $QP: ?Q (n, ns) \wedge ?P ns$
and *step*: $\bigwedge x. ?Q x \wedge ?P (\text{snd } x) \implies f x \leq \text{SPEC } (\lambda x'. ?P (\text{snd } x'))$
by *simp-all*
show $?case$
using *goal-assms* QP
unfolding *case-prod-unfold* $\langle x = (n, ns) \rangle$
proof (*rule-tac SPEC-rule-nested2, goal-cases*)
case 1
then show $?case$ **by** (*rule-tac step*) *auto*
next
case 2
then show $?case$ **by** (*rule-tac step*) *auto*
qed
qed *auto*

lemma *or-propag-on-create-graph*:

create-graph $\varphi \leq \text{SPEC } (\lambda nds. \forall nd \in nds. \mu \text{ or}_r \eta \in \text{old } nd \longrightarrow \{\mu, \eta\} \cap \text{old } nd \neq \{\})$

unfolding *create-graph-def*

by (*intro refine-vcg, rule-tac order-trans, rule-tac expand-or-propag*) *auto*

abbreviation

next-propag--assm $\mu \ n\text{-}ns \equiv$

$(X_r \mu \in \text{old } (\text{fst } n\text{-}ns) \longrightarrow \mu \in \text{next } (\text{fst } n\text{-}ns))$

$\wedge (\forall nd \in \text{snd } n\text{-}ns. X_r \mu \in \text{old } nd \wedge \text{name } nd \in \text{incoming } (\text{fst } n\text{-}ns)$

$\longrightarrow \mu \in \text{old } (\text{fst } n\text{-}ns) \cup \text{new } (\text{fst } n\text{-}ns))$

abbreviation

next-propag--rslt $\mu \ ns \equiv$

$\forall nd \in ns. \forall nd' \in ns. X_r \mu \in \text{old } nd \wedge \text{name } nd \in \text{incoming } nd' \longrightarrow \mu \in \text{old } nd'$

lemma *expand-next-propag*:

fixes $n\text{-}ns :: - \times 'a \text{ node set}$

assumes *next-propag--assm* $\mu \ n\text{-}ns$

\wedge *next-propag--rslt* $\mu \ (\text{snd } n\text{-}ns)$

\wedge *expand-assm-incoming* $n\text{-}ns$

\wedge *expand-name-ident* $(\text{snd } n\text{-}ns)$ (**is** $?Q \ n\text{-}ns$)

shows $\text{expand } n\text{-}ns \leq \text{SPEC } (\lambda r. \text{next-propag--rslt } \mu \ (\text{snd } r))$

(**is** $- \leq \text{SPEC } ?P$)

using *assms*

proof (*rule-tac expand-rec-rule*[**where** $\Phi = ?Q$], *simp, intro refine-vcg, goal-cases*)

case ($1 \ f \ x \ n \ ns$)

then show $?case$

proof (*simp, rule-tac upd-incoming--ident, goal-cases*)


```

case prems: 1
{
  fix nd :: 'a node and nd' :: 'a node
  assume nd∈ns and nd'-elem: nd'∈upd-incoming n ns
  have μ ∈ old nd' if *: Xr μ ∈ old nd and **: name nd ∈ incoming nd'
  proof (cases nd'∈ns)
    case True
    with prems * ** show ?thesis using ⟨nd∈ns⟩ by auto
  next
  case False
  with upd-incoming--elem[of nd' n ns] nd'-elem * **
  obtain nd'' where nd''∈ns
    and nd'-eq: nd' = nd''(incoming := incoming n ∪ incoming nd'')
    and old-eq: old nd'' = old n by auto
  have μ ∈ old nd'
  proof (cases name nd ∈ incoming n)
    case True
    with prems * ⟨nd∈ns⟩ have μ ∈ old n by auto
    then show ?thesis using nd'-eq old-eq by simp
  next
  case False
  then have name nd ∈ incoming nd''
    using ⟨name nd ∈ incoming nd'⟩ nd'-eq by simp
  then show ?thesis
    unfolding nd'-eq using ⟨nd∈ns⟩ ⟨nd''∈ns⟩ * prems by auto
  qed
  then show ?thesis by auto
  qed
}
then show ?case by auto
next
case 2
then show ?case by simp
qed
next
case prems: (2 f x n ns)
then have step:  $\bigwedge x. ?Q x \implies f x \leq SPEC ?P$ 
  and f-sup:  $\bigwedge x. f x \leq expand\ x$  by auto
from prems have Q: ?Q (n, ns) by auto
from Q have name-le: name n < expand-new-name (name n) by auto
let ?x' = ((name = expand-new-name (name n),
  incoming = {name n}, new = next n,
  old = {}, next = {}), {n} ∪ ns)
have Q'1: expand-assm-incoming ?x' ∧ expand-name-ident (snd ?x')
  using ⟨new n = {}⟩ Q[THEN conjunct2, THEN conjunct2] name-le by force
have Q'2: next-propag--assm μ ?x' ∧ next-propag--rslt μ (snd ?x')
  using Q ⟨new n = {}⟩ by auto
show ?case
  using ⟨new n = {}⟩

```

```

unfolding ⟨ $x = (n, ns)$ ⟩
proof (rule-tac SPEC-rule-weak[where
   $Q = \lambda r. \text{expand-name-ident } (snd\ r)$  and  $P = \lambda. ?P$ ], goal-cases)
case 1
then show ?case
using Q'1
by (rule-tac order-trans,
  rule-tac f-sup,
  rule-tac expand-name-propag--name-ident) auto
next
case 2
then show ?case
using Q'1 Q'2 by (rule-tac step) simp
next
case (3 nm nds)
then show ?case by simp
qed
next
case 3
then show ?case by auto
next
case prems: (4 f)
then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  by simp
from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case prems: (5 f)
then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  by simp
from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case 6
then show ?case by auto
next
case prems: (7 f)
then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  by simp
from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case prems: (8 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in new\ n \wedge (\exists \nu\ \mu. \psi = \nu\ or_r\ \mu \vee \psi = \nu\ U_r\ \mu \vee \psi =$ 
 $\nu\ R_r\ \mu)$ 
by (cases  $\psi$ ) auto
from prems have Q: ?Q (n, ns)
and step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$ 
and f-sup:  $\bigwedge x. f\ x \leq expand\ x$ 
by auto
let ?x = (n⟨new := new n - { $\psi$ }, new := new1  $\psi \cup new$  (n⟨new := new n -
{ $\psi$ }⟩)),
  old := { $\psi$ }  $\cup$  old (n⟨new := new n - { $\psi$ }⟩),
  next := next1  $\psi \cup next$  (n⟨new := new n - { $\psi$ }⟩)
  ⟩, ns)

```

```

let ?props =  $\lambda x r.$  expand-rslt-exist-eq  $x r$ 
   $\wedge$  expand-rslt-incoming  $r \wedge$  expand-rslt-name  $x r \wedge$  expand-name-ident (snd  $r$ )
show ?case
  using goal-assms  $Q$  unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
proof (rule-tac SPEC-rule-weak-nested2[where  $Q = ?props ?x$ ], goal-cases)
  case 1 then
    show ?case
    by (rule-tac SPEC-rule-conjI,
      rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-rslt-exist-eq,
      rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-name-propag) simp+
  next
    case 2
    then show ?case
    by (rule-tac SPEC-rule-param2[where  $P = \lambda-. ?P$ ], rule-tac step) auto
  next
    case prems': ( $\exists nm nds$ )
    let ?x' = (n(new := new  $n - \{\psi\}$ ),
      name := fst ( $nm, nds$ ),
      new := new2  $\psi \cup$  new (n(new := new  $n - \{\psi\}$ ))),
      old :=  $\{\psi\} \cup$  old (n(new := new  $n - \{\psi\}$ ))), nds)
    from prems' show ?case
    proof (rule-tac step, goal-cases)
      case prems'': 1
      then have expand-assm-incoming ?x' by auto
      moreover
      from prems'' have nds-ident: expand-name-ident (snd ?x') by simp
      moreover
      have  $X_r \mu \in$  old (fst ?x')  $\longrightarrow$   $\mu \in$  next (fst ?x')
      using  $Q$ [THEN conjunct1] goal-assms by auto
      moreover
      from prems'' have next-propag--rslt  $\mu$  (snd ?x') by simp
      moreover
      from prems'' have name-nds-eq: name ' nds = name ' ns  $\cup$  name '  $\{nd \in nds.$ 
name  $nd \geq$  name  $n\}$ 
      by auto
      have  $\forall nd \in nds. (X_r \mu \in$  old  $nd \wedge$  name  $nd \in$  incoming (fst ?x'))
         $\longrightarrow$   $\mu \in$  old (fst ?x')  $\cup$  new (fst ?x')
      (is  $\forall nd \in nds. ?assm$  (fst ?x')  $nd \longrightarrow ?concl$  (fst ?x')  $nd$ )
    proof
      fix nd
      assume  $nd \in nds$ 
      { assume loc-assm: name  $nd \in$  name ' ns
        then obtain  $n'$  where  $n': n' \in ns$  name  $n' =$  name  $nd$  by auto
        moreover
        from prems''  $n'$  obtain  $nd'$  where  $nd' \in nds$ 

```

```

    and n'-nd'-eq: expand-rslt-exist-eq--node n' nd'
    by auto
  ultimately have nd = nd'
    using nds-ident ⟨nd∈nds⟩ loc-assm by auto
  moreover from prems'' have ?assm n n' → ?concl n n'
    using ⟨n'∈ns⟩ by auto
  ultimately have ?assm (fst ?x') nd → ?concl (fst ?x') nd
    using n'-nd'-eq by auto }
moreover
{ assume name nd≠name ' ns
  with name-nds-eq ⟨nd∈nds⟩ have name nd ≥ name n by auto
  with prems'' have ¬ (?assm (fst ?x') nd) by auto }
ultimately show ?assm (fst ?x') nd → ?concl (fst ?x') nd by auto
qed
ultimately show ?case by simp
qed
qed
qed

```

lemma *next-propag-on-create-graph*:

create-graph $\varphi \leq SPEC (\lambda nds. \forall n \in nds. \forall n' \in nds. X_r \mu \in old\ n \wedge name\ n \in incoming\ n' \longrightarrow \mu \in old\ n')$

unfolding *create-graph-def*

apply (*refine-vcg expand-next-propag*)

apply (*auto simp add:expand-new-name-expand-init*)

done

abbreviation

release-propag--assm $\mu\ \eta\ n\text{-}ns \equiv$

$(\mu\ R_r\ \eta \in old\ (fst\ n\text{-}ns))$

$\longrightarrow \{\mu, \eta\} \subseteq old\ (fst\ n\text{-}ns) \cup new\ (fst\ n\text{-}ns) \vee$

$(\eta \in old\ (fst\ n\text{-}ns) \cup new\ (fst\ n\text{-}ns)) \wedge \mu\ R_r\ \eta \in next\ (fst\ n\text{-}ns))$

$\wedge (\forall nd \in snd\ n\text{-}ns.$

$\mu\ R_r\ \eta \in old\ nd \wedge name\ nd \in incoming\ (fst\ n\text{-}ns)$

$\longrightarrow \{\mu, \eta\} \subseteq old\ nd \vee$

$(\eta \in old\ nd \wedge \mu\ R_r\ \eta \in old\ (fst\ n\text{-}ns) \cup new\ (fst\ n\text{-}ns)))$

abbreviation

release-propag--rslt $\mu\ \eta\ ns \equiv$

$\forall nd \in ns.$

$\forall nd' \in ns.$

$\mu\ R_r\ \eta \in old\ nd \wedge name\ nd \in incoming\ nd'$

$\longrightarrow \{\mu, \eta\} \subseteq old\ nd \vee$

$(\eta \in old\ nd \wedge \mu\ R_r\ \eta \in old\ nd')$

lemma *expand-release-propag*:

fixes *n-ns* :: $- \times 'a$ node set

assumes *release-propag--assm* $\mu\ \eta\ n\text{-}ns$

```

       $\wedge$  release-propag--rslt  $\mu$   $\eta$  (snd n-ns)
       $\wedge$  expand-asm-incoming n-ns
       $\wedge$  expand-name-ident (snd n-ns) (is ?Q n-ns)
shows expand n-ns  $\leq$  SPEC ( $\lambda r.$  release-propag--rslt  $\mu$   $\eta$  (snd r))
  (is  $\leq$  SPEC ?P)
using assms
proof (rule-tac expand-rec-rule[where  $\Phi = ?Q$ ], simp, intro refine-vcg, goal-cases)
case (1 f x n ns)
then show ?case
proof (simp, rule-tac upd-incoming--ident, goal-cases)
  case prems: 1
  { fix nd :: 'a node and nd' :: 'a node
    let ?V-prop =  $\mu$  Rr  $\eta \in$  old nd  $\wedge$  name nd  $\in$  incoming nd'
       $\longrightarrow$   $\{\mu, \eta\} \subseteq$  old nd  $\vee$   $\eta \in$  old nd  $\wedge$   $\mu$  Rr  $\eta \in$  old nd'
    assume nd  $\in$  ns and nd'-elem: nd'  $\in$  upd-incoming n ns
    { assume nd'  $\in$  ns
      with prems have ?V-prop using  $\langle$ nd  $\in$  ns $\rangle$  by auto }
    moreover
    { assume nd'  $\notin$  ns
      and V-in-nd:  $\mu$  Rr  $\eta \in$  old nd and name nd  $\in$  incoming nd'
      with upd-incoming--elem[of nd' n ns] nd'-elem
      obtain nd'' where nd''  $\in$  ns
        and nd'-eq: nd' = nd'' (incoming := incoming n  $\cup$  incoming nd'')
        and old-eq: old nd'' = old n
        by auto
      { assume name nd  $\in$  incoming n
        with prems V-in-nd  $\langle$ nd  $\in$  ns $\rangle$ 
        have  $\{\mu, \eta\} \subseteq$  old nd  $\vee$   $\eta \in$  old nd  $\wedge$   $\mu$  Rr  $\eta \in$  old n
          by auto
        then have  $\{\mu, \eta\} \subseteq$  old nd  $\vee$   $\eta \in$  old nd  $\wedge$   $\mu$  Rr  $\eta \in$  old nd'
          using nd'-eq old-eq by simp }
      moreover
      { assume name nd  $\notin$  incoming n
        then have name nd  $\in$  incoming nd''
          using  $\langle$ name nd  $\in$  incoming nd' $\rangle$  nd'-eq by simp
        then have  $\{\mu, \eta\} \subseteq$  old nd  $\vee$   $\eta \in$  old nd  $\wedge$   $\mu$  Rr  $\eta \in$  old nd'
          unfolding nd'-eq
          using prems  $\langle$ nd  $\in$  ns $\rangle$   $\langle$ nd''  $\in$  ns $\rangle$  V-in-nd by auto }
        ultimately have  $\{\mu, \eta\} \subseteq$  old nd  $\vee$   $\eta \in$  old nd  $\wedge$   $\mu$  Rr  $\eta \in$  old nd'
          by fast
      }
    }
  }
  ultimately have ?V-prop by auto
}
then show ?case by auto
next
  case 2
  then show ?case by simp
qed
next

```

```

case prems: (2 f x n ns)
then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$ 
  and f-sup:  $\bigwedge x. f\ x \leq \text{expand}\ x$  by auto
from prems have Q:  $?Q\ (n, ns)$  by auto
from Q have name-le:  $\text{name}\ n < \text{expand-new-name}\ (\text{name}\ n)$  by auto
let  $?x' = (\{\text{name} = \text{expand-new-name}\ (\text{name}\ n),$ 
  incoming =  $\{\text{name}\ n\}$ , new = next n,
  old =  $\{\}$ , next =  $\{\}$ ),  $\{\text{name}\ n\} \cup ns)$ 
have Q'1:  $\text{expand-asm-incoming}\ ?x' \wedge \text{expand-name-ident}\ (\text{snd}\ ?x')$ 
using  $\langle \text{new}\ n = \{\} \rangle Q[\text{THEN}\ \text{conjunct}2, \text{THEN}\ \text{conjunct}2]$  name-le by force
have Q'2:  $\text{release-propag-asm}\ \mu\ \eta\ ?x' \wedge \text{release-propag-rslt}\ \mu\ \eta\ (\text{snd}\ ?x')$ 
using  $Q\ \langle \text{new}\ n = \{\} \rangle$  by auto

show ?case using  $\langle \text{new}\ n = \{\} \rangle$  unfolding  $\langle x = (n, ns) \rangle$ 

proof (rule-tac SPEC-rule-weak[where
   $Q = \lambda r. \text{expand-name-ident}\ (\text{snd}\ r)$  and  $P = \lambda \cdot. ?P]$ , goal-cases)
case 1
then show ?case using Q'1
by (rule-tac order-trans,
  rule-tac f-sup,
  rule-tac expand-name-propag-name-ident) auto
next
case 2
then show ?case using Q'1 Q'2 by (rule-tac step) simp
next
case (3 nm nds)
then show ?case by simp
qed
next
case 3 then show ?case by auto
next
case prems: (4 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in \text{new}\ n \wedge (\exists q. \psi = \text{prop}_r(q) \vee \psi = \text{nprop}_r(q))$  by
simp
from prems have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  and Q:  $?Q\ (n, ns)$ 
by simp-all
show ?case
using Q goal-assms by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
case prems: (5 f x n ns  $\psi$ )
then have goal-assms:  $\psi \in \text{new}\ n \wedge \psi = \text{true}_r$  by simp
from prems have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  and Q:  $?Q\ (n, ns)$ 
by simp-all
show ?case using Q goal-assms by (rule-tac SPEC-rule-param2, rule-tac step)
auto
next
case 6
then show ?case by auto

```

```

next
  case prems: (7 f x n ns ψ)
  then have goal-assms:  $\psi \in \text{new } n \wedge (\exists \nu \mu. \psi = \nu \text{ and}_r \mu \vee \psi = X_r \nu)$  by simp
  from prems have step:  $\bigwedge x. ?Q x \implies f x \leq \text{SPEC } ?P$  and Q:  $?Q (n, ns)$ 
  by simp-all
  show  $?case$  using Q goal-assms by (rule-tac SPEC-rule-param2, rule-tac step)
auto
next
  case prems: (8 f x n ns ψ)
  then have goal-assms:  $\psi \in \text{new } n \wedge (\exists \nu \mu. \psi = \nu \text{ or}_r \mu \vee \psi = \nu U_r \mu \vee \psi = \nu R_r \mu)$ 
  by (cases ψ) auto
  from prems have Q:  $?Q (n, ns)$ 
  and step:  $\bigwedge x. ?Q x \implies f x \leq \text{SPEC } ?P$ 
  and f-sup:  $\bigwedge x. f x \leq \text{expand } x$  by auto
  let  $?x = (n(\text{new} := \text{new } n - \{\psi\}, \text{new} := \text{new1 } \psi \cup \text{new } (n(\text{new} := \text{new } n - \{\psi\})))$ ,
     $\text{old} := \{\psi\} \cup \text{old } (n(\text{new} := \text{new } n - \{\psi\})))$ ,
     $\text{next} := \text{next1 } \psi \cup \text{next } (n(\text{new} := \text{new } n - \{\psi\})))$ ,
     $\text{old}, ns)$ 
  let  $?props = \lambda x r. \text{expand-rslt-exist-eq } x r$ 
     $\wedge \text{expand-rslt-incoming } r \wedge \text{expand-rslt-name } x r \wedge \text{expand-name-ident } (\text{snd } r)$ 

  show  $?case$  using goal-assms Q unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 

proof (rule-tac SPEC-rule-weak-nested2[where  $Q = ?props ?x$ ], goal-cases)
  case 1
  then show  $?case$ 
  by (rule-tac SPEC-rule-conjI,
    rule-tac order-trans,
    rule-tac f-sup,
    rule-tac expand-rslt-exist-eq,
    rule-tac order-trans,
    rule-tac f-sup,
    rule-tac expand-name-propag) simp+
next
  case 2
  then show  $?case$ 
  by (rule-tac SPEC-rule-param2[where  $P = \lambda-. ?P$ ], rule-tac step) auto
next
  case prems': (3 nm nds)
  let  $?x' = (n(\text{new} := \text{new } n - \{\psi\},$ 
     $\text{name} := \text{fst } (nm, nds),$ 
     $\text{new} := \text{new2 } \psi \cup \text{new } (n(\text{new} := \text{new } n - \{\psi\})))$ ,
     $\text{old} := \{\psi\} \cup \text{old } (n(\text{new} := \text{new } n - \{\psi\})))$ , nds)
  from prems' show  $?case$ 
  proof (rule-tac step, goal-cases)
  case prems'': 1
  then have expand-assm-incoming  $?x'$  by auto

```

```

moreover
from prems'' have nds-ident: expand-name-ident (snd ?x') by simp
moreover
have  $(\mu R_r \eta \in \text{old } (fst ?x') \rightarrow (\{\mu, \eta\} \subseteq \text{old } (fst ?x') \cup \text{new } (fst ?x') \vee (\eta \in \text{old } (fst ?x') \cup \text{new } (fst ?x') \wedge \mu R_r \eta \in \text{next } (fst ?x'))))$ 
using  $Q[THEN \text{conjunct1}]$  goal-assms by auto
moreover
from prems'' have release-propag--rslt  $\mu \eta$  (snd ?x') by simp
moreover
from prems'' have name-nds-eq: name ' nds = name ' ns  $\cup$  name ' {nd $\in$ nds. name nd  $\geq$  name n}
by auto
have  $\forall nd \in \text{nds. } (\mu R_r \eta \in \text{old } nd \wedge \text{name } nd \in \text{incoming } (fst ?x') \rightarrow (\{\mu, \eta\} \subseteq \text{old } nd \vee (\eta \in \text{old } nd \wedge \mu R_r \eta \in \text{old } (fst ?x') \cup \text{new } (fst ?x'))))$ 
(is  $\forall nd \in \text{nds. } ?\text{assm } (fst ?x') \text{ nd} \rightarrow ?\text{concl } (fst ?x') \text{ nd}$ )
proof
fix nd
assume nd $\in$ nds
{ assume loc-assm: name nd $\in$ name ' ns
then obtain n' where n': n' $\in$ ns name n' = name nd by auto
with prems'' obtain nd' where nd' $\in$ nds
and n'-nd'-eq: expand-rslt-exist-eq--node n' nd'
by auto
with n' have nd = nd' using nds-ident <nd $\in$ nds> loc-assm
by auto
moreover from prems'' have ?assm n n'  $\rightarrow$  ?concl n n'
using <n' $\in$ ns> by auto
ultimately have ?assm (fst ?x') nd  $\rightarrow$  ?concl (fst ?x') nd
using n'-nd'-eq by auto }
moreover
{ assume name nd $\notin$ name ' ns
with name-nds-eq <nd $\in$ nds> have name nd  $\geq$  name n by auto
with prems'' have  $\neg (?assm (fst ?x') nd)$  by auto }
ultimately show ?assm (fst ?x') nd  $\rightarrow$  ?concl (fst ?x') nd by auto
qed
ultimately show ?case by simp
qed
qed
qed

```

lemma *release-propag-on-create-graph:*

```

create-graph  $\varphi$ 
 $\leq SPEC (\lambda \text{nds. } \forall n \in \text{nds. } \forall n' \in \text{nds. } \mu R_r \eta \in \text{old } n \wedge \text{name } n \in \text{incoming } n' \rightarrow (\{\mu, \eta\} \subseteq \text{old } n \vee \eta \in \text{old } n \wedge \mu R_r \eta \in \text{old } n'))$ 

```

unfolding *create-graph-def*

apply (*refine-vcg expand-release-propag*)

by (auto simp add: expand-new-name-expand-init)

abbreviation

$$\begin{aligned} \text{until-propag--assm } f \ g \ n\text{-ns} &\equiv \\ &(f \ U_r \ g \in \text{old } (fst \ n\text{-ns}) \\ &\longrightarrow (g \in \text{old } (fst \ n\text{-ns}) \cup \text{new } (fst \ n\text{-ns}) \\ &\quad \vee (f \in \text{old } (fst \ n\text{-ns}) \cup \text{new } (fst \ n\text{-ns}) \wedge f \ U_r \ g \in \text{next } (fst \ n\text{-ns})))) \\ &\wedge (\forall nd \in \text{snd } n\text{-ns}. f \ U_r \ g \in \text{old } nd \wedge \text{name } nd \in \text{incoming } (fst \ n\text{-ns}) \\ &\longrightarrow (g \in \text{old } nd \vee (f \in \text{old } nd \wedge f \ U_r \ g \in \text{old } (fst \ n\text{-ns}) \cup \text{new } (fst \ n\text{-ns})))) \end{aligned}$$

abbreviation

$$\begin{aligned} \text{until-propag--rslt } f \ g \ ns &\equiv \\ &\forall n \in ns. \forall nd \in ns. f \ U_r \ g \in \text{old } n \wedge \text{name } n \in \text{incoming } nd \\ &\longrightarrow (g \in \text{old } n \vee (f \in \text{old } n \wedge f \ U_r \ g \in \text{old } nd)) \end{aligned}$$

lemma *expand-until-propag*:

fixes $n\text{-ns} :: - \times 'a \text{ node set}$

assumes *until-propag--assm* $\mu \ \eta \ n\text{-ns}$

\wedge *until-propag--rslt* $\mu \ \eta \ (\text{snd } n\text{-ns})$

\wedge *expand-assm-incoming* $n\text{-ns}$

\wedge *expand-name-ident* $(\text{snd } n\text{-ns})$ (**is** $?Q \ n\text{-ns}$)

shows $\text{expand } n\text{-ns} \leq \text{SPEC } (\lambda r. \text{until-propag--rslt } \mu \ \eta \ (\text{snd } r))$

(**is** $- \leq \text{SPEC } ?P$)

using *assms*

proof (*rule-tac expand-rec-rule*[**where** $\Phi = ?Q$], *simp*, *intro refine-vcg*, *goal-cases*)

case *prems*: (1 $f \ x \ n \ ns$)

then show $?case$

proof (*simp*, *rule-tac upd-incoming--ident*, *goal-cases*)

case *prems'*: 1

{ **fix** $nd :: 'a \text{ node}$ **and** $nd' :: 'a \text{ node}$

let $?U\text{-prop} = \mu \ U_r \ \eta \in \text{old } nd \wedge \text{name } nd \in \text{incoming } nd'$

$\longrightarrow \eta \in \text{old } nd \vee \mu \in \text{old } nd \wedge \mu \ U_r \ \eta \in \text{old } nd'$

assume $nd \in ns$ **and** $nd'\text{-elem}: nd' \in \text{upd-incoming } n \ ns$

{ **assume** $nd' \in ns$

with *prems'* **have** $?U\text{-prop}$ **using** $\langle nd \in ns \rangle$ **by** *auto* }

moreover

{ **assume** $nd' \notin ns$ **and**

$U\text{-in-nd}: \mu \ U_r \ \eta \in \text{old } nd$ **and** $\text{name } nd \in \text{incoming } nd'$

with *upd-incoming--elem*[*of* $nd' \ n \ ns$] $nd'\text{-elem}$

obtain nd'' **where** $nd'' \in ns$

and $nd'\text{-eq}: nd' = nd'' \ (\!| \text{incoming } := \text{incoming } n \cup \text{incoming } nd'' \!|)$

and $old\text{-eq}: old \ nd'' = old \ n$ **by** *auto*

{ **assume** $\text{name } nd \in \text{incoming } n$

with *prems'* $U\text{-in-nd}$ $\langle nd \in ns \rangle$

have $\eta \in \text{old } nd \vee \mu \in \text{old } nd \wedge \mu \ U_r \ \eta \in \text{old } n$ **by** *auto*

then have $\eta \in \text{old } nd \vee \mu \in \text{old } nd \wedge \mu \ U_r \ \eta \in \text{old } nd'$

using $nd'\text{-eq}$ $old\text{-eq}$ **by** *simp* }

moreover

```

    { assume name nd  $\notin$  incoming n
      then have name nd  $\in$  incoming nd''
        using  $\langle$ name nd  $\in$  incoming nd'' $\rangle$  nd'-eq by simp
      then have  $\eta \in$  old nd  $\vee$   $\mu \in$  old nd  $\wedge$   $\mu U_r \eta \in$  old nd'
        unfolding nd'-eq
        using prems'  $\langle$ nd $\in$ ns $\rangle$   $\langle$ nd'' $\in$ ns $\rangle$  U-in-nd by auto }
      ultimately have  $\eta \in$  old nd  $\vee$   $\mu \in$  old nd  $\wedge$   $\mu U_r \eta \in$  old nd' by fast }
      ultimately have ?U-prop by auto }
    then show ?case by auto
  next
    case 2
    then show ?case by simp
  qed
next
case prems: (2 f x n ns)
then have step:  $\bigwedge x. ?Q x \implies f x \leq$  SPEC ?P and f-sup:  $\bigwedge x. f x \leq$  expand x
  by auto
from prems have Q: ?Q (n, ns) by auto
from Q have name-le: name n < expand-new-name (name n) by auto
let ?x' = ( $\{$ name = expand-new-name (name n),
  incoming = {name n}, new = next n,
  old = {}, next = {} $\}$ ), {n}  $\cup$  ns)
have Q'1: expand-asm-incoming ?x'  $\wedge$  expand-name-ident (snd ?x')
  using  $\langle$ new n = {} $\rangle$  Q[THEN conjunct2, THEN conjunct2] name-le by force
have Q'2: until-propag--asm  $\mu \eta$  ?x'  $\wedge$  until-propag--rslt  $\mu \eta$  (snd ?x')
  using Q  $\langle$ new n = {} $\rangle$  by auto

show ?case
  using  $\langle$ new n = {} $\rangle$  unfolding  $\langle$ x = (n, ns) $\rangle$ 
proof (rule-tac SPEC-rule-weak[where
  Q =  $\lambda$ - r. expand-name-ident (snd r) and P =  $\lambda$ -. ?P], goal-cases)
  case 1
  then show ?case using Q'1
    by (rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-name-propag--name-ident) auto
  next
  case 2
  then show ?case using Q'1 Q'2 by (rule-tac step) simp
  next
  case (3 nm nds)
  then show ?case by simp
qed
next
case 3
then show ?case by auto
next
case prems: (4 f)
then have step:  $\bigwedge x. ?Q x \implies f x \leq$  SPEC ?P by simp-all

```

```

from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case prems: (5 f)
  then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq \text{SPEC } ?P$  by simp-all
  from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case 6
  then show ?case by auto
next
  case prems: (7 f)
  then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq \text{SPEC } ?P$  by simp-all
  from prems show ?case by (rule-tac SPEC-rule-param2, rule-tac step) auto
next
  case prems: (8 f x n ns  $\psi$ )
  then have goal-assms:  $\psi \in \text{new } n \wedge (\exists \nu\ \mu. \psi = \nu\ \text{or}_r\ \mu \vee \psi = \nu\ U_r\ \mu \vee \psi =$ 
 $\nu\ R_r\ \mu)$ 
  by (cases  $\psi$ ) auto
  from prems have Q: ?Q (n, ns)
  and step:  $\bigwedge x. ?Q\ x \implies f\ x \leq \text{SPEC } ?P$ 
  and f-sup:  $\bigwedge x. f\ x \leq \text{expand } x$  by auto
  let ?x = (n(|new := new n - { $\psi$ }, new := new1  $\psi \cup \text{new}$  (n(|new := new n -
 $\{\psi\}$ ))),
    old := { $\psi$ }  $\cup$  old (n(|new := new n - { $\psi$ })),
    next := next1  $\psi \cup \text{next}$  (n(|new := new n - { $\psi$ }))
  ), ns)
  let ?props =  $\lambda x\ r. \text{expand-rslt-exist-eq } x\ r$ 
 $\wedge \text{expand-rslt-incoming } r \wedge \text{expand-rslt-name } x\ r \wedge \text{expand-name-ident } (\text{snd } r)$ 

show ?case
  using goal-assms Q unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
proof (rule-tac SPEC-rule-weak-nested2[where Q = ?props ?x], goal-cases)
  case 1
  then show ?case
    by (rule-tac SPEC-rule-conjI,
      rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-rslt-exist-eq,
      rule-tac order-trans,
      rule-tac f-sup,
      rule-tac expand-name-propag) simp+
next
  case 2
  then show ?case
    by (rule-tac SPEC-rule-param2[where P =  $\lambda\cdot. ?P$ ], rule-tac step) auto
next
  case prems: (3 nm nds)
  let ?x' = (n(|new := new n - { $\psi$ },
    name := fst (nm, nds),
    new := new2  $\psi \cup \text{new}$  (n(|new := new n - { $\psi$ }))),

```

```

    old := {ψ} ∪ old (n(|new := new n - {ψ}|)), nds)
from prems show ?case
proof (rule-tac step, goal-cases)
  case prems': 1
  then have expand-asm-incoming ?x' by auto
  moreover
  from prems' have nds-ident: expand-name-ident (snd ?x')
    by simp
  moreover
  have (μ Ur η ∈ old (fst ?x')
    → (η ∈ old (fst ?x') ∪ new (fst ?x')
      ∨ (μ ∈ old (fst ?x') ∪ new (fst ?x')
        ∧ μ Ur η ∈ next (fst ?x'))))
    using Q[THEN conjunct1] goal-assms by auto
  moreover
  from prems' have until-propag--rslt μ η (snd ?x')
    by simp
  moreover
  from prems' have name-nds-eq:
    name ' nds = name ' ns ∪ name ' {nd ∈ nds. name nd ≥ name n}
    by auto
  have ∀ nd ∈ nds. (μ Ur η ∈ old nd ∧ name nd ∈ incoming (fst ?x'))
    → (η ∈ old nd ∨ (μ ∈ old nd ∧ μ Ur η ∈ old (fst ?x') ∪ new (fst ?x')))
  (is ∀ nd ∈ nds. ?assm (fst ?x') nd → ?concl (fst ?x') nd)
proof
  fix nd
  assume nd ∈ nds
  { assume loc-asm: name nd ∈ name ' ns
    then obtain n' where n': n' ∈ ns name n' = name nd by auto
    moreover
    from prems' n' obtain nd' where nd' ∈ nds
      and n'-nd'-eq: expand-rslt-exist-eq--node n' nd'
      by auto
    ultimately have nd = nd'
      using nds-ident ⟨nd ∈ nds⟩ loc-asm by auto
    moreover from prems' have ?assm n n' → ?concl n n'
      using ⟨n' ∈ ns⟩ by auto
    ultimately have ?assm (fst ?x') nd → ?concl (fst ?x') nd
      using n'-nd'-eq by auto }
  moreover
  { assume name nd ∉ name ' ns
    with name-nds-eq ⟨nd ∈ nds⟩ have name nd ≥ name n by auto
    with prems' have ¬ (?assm (fst ?x') nd) by auto }
  ultimately show ?assm (fst ?x') nd → ?concl (fst ?x') nd by auto
qed
ultimately show ?case by simp
qed
qed
qed

```

lemma *until-propag-on-create-graph*:
create-graph $\varphi \leq SPEC (\lambda nds. \forall n \in nds. \forall n' \in nds. \mu U_r \eta \in old\ n \wedge name\ n \in incoming\ n')$
 $\longrightarrow (\eta \in old\ n \vee \mu \in old\ n \wedge \mu U_r \eta \in old\ n')$
unfolding *create-graph-def*
apply (*refine-vcg expand-until-propag*)
by (*auto simp add: expand-new-name-expand-init*)

definition *all-subfrmls* :: 'a node \Rightarrow 'a frml set
where *all-subfrmls* $n \equiv \bigcup (subfrmlsr\ ' (new\ n \cup old\ n \cup next\ n))$

lemma *all-subfrmls--UnionD*:
assumes $(\bigcup x \in A. subfrmlsr\ x) \subseteq B$
and $x \in A$
and $y \in subfrmlsr\ x$
shows $y \in B$
proof –
note *subfrmlsr-id*[*of* x]
also have $subfrmlsr\ x \subseteq (\bigcup x \in A. subfrmlsr\ x)$
using *assms* **by** *auto*
finally show *?thesis* **using** *assms* **by** *auto*
qed

lemma *expand-all-subfrmls-propag*:
assumes *all-subfrmls* (*fst* $n\text{-ns}$) $\subseteq B \wedge (\forall nd \in snd\ n\text{-ns}. all\text{-subfrmls}\ nd \subseteq B)$ (**is** *?Q* $n\text{-ns}$)
shows *expand* $n\text{-ns} \leq SPEC (\lambda r. \forall nd \in snd\ r. all\text{-subfrmls}\ nd \subseteq B)$
(is $\leq SPEC\ ?P)$
using *assms*
proof (*rule-tac expand-rec-rule*[**where** $\Phi = ?Q$], *simp*, *intro refine-vcg*, *goal-cases*)
case 1
then show *?case*
proof (*simp*, *rule-tac upd-incoming--ident*, *goal-cases*)
case 1
then show *?case* **by** *auto*
next
case 2
then show *?case* **by** (*simp add: all-subfrmls-def*)
qed
next
case 2
then show *?case* **by** (*auto simp add: all-subfrmls-def*)
next
case 3
then show *?case* **by** (*auto simp add: all-subfrmls-def*)
next
case *prems*: (4 f)

```

then have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  by simp-all
from prems show ?case by (rule-tac step) (auto simp add: all-subfrmls-def)
next
  case prems: (5 f - n ns  $\psi$ )
  then have goal-assms:  $\psi \in new\ n \wedge \psi = true_r$  by simp
  from prems have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  and Q:  $?Q\ (n, ns)$ 
    by simp-all
  show ?case using Q goal-assms
    by (rule-tac step) (auto dest: all-subfrmls--UnionD simp add: all-subfrmls-def)
next
  case 6
  then show ?case by auto
next
  case prems: (7 f x n ns  $\psi$ )
  then have goal-assms:  $\psi \in new\ n \wedge (\exists \nu\ \mu. \psi = \nu\ and_r\ \mu \vee \psi = X_r\ \nu)$  by simp
  from prems have step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$  and Q:  $?Q\ (n, ns)$ 
    by simp-all
  show ?case
    using Q goal-assms
    by (rule-tac step) (auto dest: all-subfrmls--UnionD simp add: all-subfrmls-def)
next
  case prems: (8 f x n ns  $\psi$ )
  then have goal-assms:  $\psi \in new\ n$ 
     $\wedge \neg (\exists q. \psi = prop_r(q) \vee \psi = nprop_r(q))$ 
     $\wedge \psi \neq true_r \wedge \psi \neq false_r$ 
     $\wedge \neg (\exists \nu\ \mu. \psi = \nu\ and_r\ \mu \vee \psi = X_r\ \nu)$ 
     $\wedge (\exists \nu\ \mu. \psi = \nu\ or_r\ \mu \vee \psi = \nu\ U_r\ \mu \vee \psi = \nu\ R_r\ \mu)$ 
    by (cases  $\psi$ ) auto
  from prems have Q:  $?Q\ (n, ns)$  and step:  $\bigwedge x. ?Q\ x \implies f\ x \leq SPEC\ ?P$ 
    by simp-all
  show ?case
    using goal-assms Q unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
  proof (rule-tac SPEC-rule-nested2, goal-cases)
    case 1
    then show ?case
      by (rule-tac step) (auto dest: all-subfrmls--UnionD simp: all-subfrmls-def)
    next
    case 2
    then show ?case
      by (rule-tac step) (auto dest: all-subfrmls--UnionD simp: all-subfrmls-def)
  qed
qed

lemma old-propag-on-create-graph:  $create\_graph\ \varphi \leq SPEC\ (\lambda nds. \forall n \in nds. old\ n \subseteq subfrmlsr\ \varphi)$ 
  unfolding create-graph-def
  by (intro refine-vcg,
    rule-tac order-trans,
    rule-tac expand-all-subfrmls-propag [where  $B = subfrmlsr\ \varphi$ ])

```

(force simp add:all-subfrmls-def expand-new-name-expand-init)+

lemma *L4-2--aux*:

assumes *run*: *ipath gba.E* σ
and $\mu U_r \eta \in \text{old } (\sigma 0)$
and $\forall j. (\forall i < j. \{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma i)) \longrightarrow \eta \notin \text{old } (\sigma j)$
shows $\forall i. \{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma i) \wedge \eta \notin \text{old } (\sigma i)$
proof –
have $\forall i < j. \{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma i)$ (**is** *?sbthm j*) **for** *j*
proof (*induct j*)
show *?sbthm 0* **by** *auto*
next
fix *k*
assume *step*: *?sbthm k*
then have $\sigma\text{-}k\text{-prop}$: $\eta \notin \text{old } (\sigma k)$
 $\wedge \sigma k \in \text{qs} \wedge \sigma (\text{Suc } k) \in \text{qs}$
 $\wedge \text{name } (\sigma k) \in \text{incoming } (\sigma (\text{Suc } k))$
using *assms run-propag-on-create-graph[OF run]* **by** *auto*
with *inres-SPEC[OF res until-propag-on-create-graph[where $\mu = \mu$ and $\eta =$*
 $\eta]$
have $\{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma k)$ (**is** *?subsetthm*)
proof (*cases k*)
assume $k = 0$
with *assms $\sigma\text{-}k\text{-prop}$*
 $\text{inres-SPEC}[OF \text{res until-propag-on-create-graph[where } \mu = \mu \text{ and } \eta = \eta]$
show *?subsetthm* **by** *auto*
next
fix *l*
assume $k = \text{Suc } l$
then have $\{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma l) \wedge \eta \notin \text{old } (\sigma l)$
 $\wedge \sigma l \in \text{qs} \wedge \sigma k \in \text{qs}$
 $\wedge \text{name } (\sigma l) \in \text{incoming } (\sigma k)$
using *step assms run-propag-on-create-graph[OF run]* **by** *auto*
with *inres-SPEC[OF res until-propag-on-create-graph[where $\mu = \mu$ and $\eta =$*
 $\eta]$
have $\mu U_r \eta \in \text{old } (\sigma k)$ **by** *auto*
with *$\sigma\text{-}k\text{-prop}$*
 $\text{inres-SPEC}[OF \text{res until-propag-on-create-graph[where } \mu = \mu \text{ and } \eta = \eta]$
show *?subsetthm* **by** *auto*
qed
with *step* **show** *?sbthm (Suc k)* **by** (*metis less-SucE*)
qed
with *assms* **show** *?thesis* **by** *auto*
qed

lemma *L4-2a*:

assumes *ipath gba.E* σ
and $\mu U_r \eta \in \text{old } (\sigma 0)$
shows $(\forall i. \{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma i) \wedge \eta \notin \text{old } (\sigma i))$

$\vee (\exists j. (\forall i < j. \{\mu, \mu U_r \eta\} \subseteq \text{old}(\sigma i)) \wedge \eta \in \text{old}(\sigma j))$
(is ?A \vee ?B)
proof (rule *disjCI*)
assume $\neg ?B$
then show ?A
using *assms* by (rule-tac *L4-2--aux*) *blast+*
qed

lemma *L4-2b*:
assumes *run*: *ipath gba.E* σ
and $\mu U_r \eta \in \text{old}(\sigma 0)$
and *ACC*: *gba.is-acc* σ
shows $\exists j. (\forall i < j. \{\mu, \mu U_r \eta\} \subseteq \text{old}(\sigma i)) \wedge \eta \in \text{old}(\sigma j)$
proof (rule *ccontr*)
assume $\neg ?thesis$
then have *contr*: $\forall i. \{\mu, \mu U_r \eta\} \subseteq \text{old}(\sigma i) \wedge \eta \notin \text{old}(\sigma i)$
using *assms* *L4-2a*[of $\sigma \mu \eta$] by *blast*

define *S* where $S = \{q \in \text{qs}. \mu U_r \eta \in \text{old } q \longrightarrow \eta \in \text{old } q\}$

from *assms* *inres-SPEC*[*OF res old-propag-on-create-graph*] *create-gba-from-nodes--ipath*
have $\mu U_r \eta \in \text{subfrmlsr } \varphi$
by (*metis* *assms*(2) *subsetD*)
then have $S \in \text{gbg-F}(\text{create-gba-from-nodes } \varphi \text{ qs})$
unfolding *S-def* *create-gba-from-nodes-def* by *auto*
with *ACC* **have** 1: $\exists_{\infty} i. \sigma i \in S$
unfolding *gba.is-acc-def* by *blast*

from *INFM-EX*[*OF 1*] **obtain** *k* where $\sigma k \in \text{qs}$ and $\mu U_r \eta \in \text{old}(\sigma k) \longrightarrow \eta \in \text{old}(\sigma k)$
unfolding *S-def* by *auto*
moreover have $\{\mu, \mu U_r \eta\} \subseteq \text{old}(\sigma k) \wedge \eta \notin \text{old}(\sigma k)$
using *contr* by *auto*
ultimately show *False* by *auto*
qed

lemma *L4-2c*:
assumes *run*: *ipath gba.E* σ
and $\mu R_r \eta \in \text{old}(\sigma 0)$
shows $\forall i. \eta \in \text{old}(\sigma i) \vee (\exists j < i. \mu \in \text{old}(\sigma j))$
proof –
have $\{\eta, \mu R_r \eta\} \subseteq \text{old}(\sigma i) \vee (\exists j < i. \mu \in \text{old}(\sigma j))$ (**is** ?*thm i*) **for** *i*
proof (*induct i*)
case 0
have $\sigma 0 \in \text{qs} \wedge \sigma 1 \in \text{qs} \wedge \text{name}(\sigma 0) \in \text{incoming}(\sigma 1)$
using *create-gba-from-nodes--ipath* *assms* by *auto*
then show ?*case*
using *assms* *inres-SPEC*[*OF res release-propag-on-create-graph*, of $\mu \eta$]
by *auto*


```

next
  case (Suc k)
  note <?thm k>
  moreover
  { assume { $\eta, \mu R_r \eta$ }  $\subseteq$  old ( $\sigma k$ )
  moreover
  have  $\sigma k \in qs \wedge \sigma (Suc k) \in qs \wedge name (\sigma k) \in incoming (\sigma (Suc k))$ 
  using create-gba-from-nodes--ipath assms by auto
  ultimately have  $\mu \in old (\sigma k) \vee \mu R_r \eta \in old (\sigma (Suc k))$ 
  using assms inres-SPEC[OF res release-propag-on-create-graph, of  $\mu \eta$ ]
  by auto
  moreover
  { assume  $\mu \in old (\sigma k)$ 
  then have ?case by blast }
  moreover
  { assume  $\mu R_r \eta \in old (\sigma (Suc k))$ 
  moreover
  have  $\sigma (Suc k) \in qs \wedge \sigma (Suc (Suc k)) \in qs$ 
   $\wedge name (\sigma (Suc k)) \in incoming (\sigma (Suc (Suc k)))$ 
  using create-gba-from-nodes--ipath assms by auto
  ultimately have ?case
  using assms
  inres-SPEC[OF res release-propag-on-create-graph, of  $\mu \eta$ ]
  by auto }
  ultimately have ?case by fast }
  moreover
  { assume  $\exists j < k. \mu \in old (\sigma j)$ 
  then have ?case by auto }
  ultimately show ?case by auto
qed
then show ?thesis by auto
qed

```

lemma L_4-8' :

```

assumes ipath gba.E  $\sigma$  (is ?inf-run  $\sigma$ )
  and gba.is-acc  $\sigma$  (is ?gbarel-accept  $\sigma$ )
  and  $\forall i. gba.L (\sigma i) (\xi i)$  (is ?lgbarel-accept  $\xi \sigma$ )
  and  $\psi \in old (\sigma 0)$ 
shows  $\xi \models_r \psi$ 
using assms
proof (induct  $\psi$  arbitrary:  $\sigma \xi$ )
  case True-ltlr
  show ?case by auto
next
  case False-ltlr
  then show ?case
  using inres-SPEC[OF res false-propag-on-create-graph]
  create-gba-from-nodes--ipath
  by (metis)

```

```

next
  case (Prop-ltlr  $p$ )
  then show ?case
    unfolding create-gba-from-nodes-def by auto
next
  case (Nprop-ltlr  $p$ )
  then show ?case
    unfolding create-gba-from-nodes-def by auto
next
  case (And-ltlr  $\mu \eta$ )
  then show ?case
    using inres-SPEC[OF res and-propag-on-create-graph, of  $\mu \eta$ ]
    create-gba-from-nodes--ipath
    by (metis insert-subset semantics-ltlr.simps(5))
next
  case (Or-ltlr  $\mu \eta$ )
  then have  $\mu \in \text{old } (\sigma \ 0) \vee \eta \in \text{old } (\sigma \ 0)$ 
    using inres-SPEC[OF res or-propag-on-create-graph, of  $\mu \eta$ ]
    create-gba-from-nodes--ipath
    by (metis (full-types) Int-empty-left Int-insert-left-if0)
  moreover have  $\xi \models_r \mu$  if  $\mu \in \text{old } (\sigma \ 0)$ 
    using Or-ltlr that by auto
  moreover have  $\xi \models_r \eta$  if  $\eta \in \text{old } (\sigma \ 0)$ 
    using Or-ltlr that by auto
  ultimately show ?case by auto
next
  case (Next-ltlr  $\mu$ )
  with create-gba-from-nodes--ipath[of  $\sigma$ ]
  have  $\sigma \ 0 \in \text{qs} \wedge \sigma \ 1 \in \text{qs} \wedge \text{name } (\sigma \ 0) \in \text{incoming } (\sigma \ 1)$ 
    by auto
  with inres-SPEC[OF res next-propag-on-create-graph, of  $\mu$ ] have  $\mu \in \text{old } (\text{suffix}$ 
  1  $\sigma \ 0)$ 
    using Next-ltlr by auto
  moreover
  have ?inf-run (suffix 1  $\sigma$ )
    and ?gbarel-accept (suffix 1  $\sigma$ )
    and ?lgbarel-accept (suffix 1  $\xi$ ) (suffix 1  $\sigma$ )
    using Next-ltlr create-gba-from-nodes--ipath
    apply –
    apply (metis ipath-suffix)
    apply (auto simp del: suffix-nth) []
    apply auto
  done
  ultimately show ?case using Next-ltlr by simp
next
  case (Until-ltlr  $\mu \eta$ )
  then have  $\exists j. (\forall i < j. \{\mu, \mu \ U_r \ \eta\} \subseteq \text{old } (\sigma \ i)) \wedge \eta \in \text{old } (\sigma \ j)$ 
    using L4-2b by auto
  then obtain  $j$  where  $\sigma\text{-pre}: \forall i < j. \{\mu, \mu \ U_r \ \eta\} \subseteq \text{old } (\sigma \ i)$  and  $\eta \in \text{old } (\text{suffix}$ 

```

```

j σ 0)
  by auto
  moreover
  have ?inf-run (suffix j σ)
    and ?gbarel-accept (suffix j σ)
    and ?lgbarel-accept (suffix j ξ) (suffix j σ)
  unfolding limit-suffix
  using Until-ltlr create-gba-from-nodes--ipath
  apply -
  apply (metis ipath-suffix)
  apply (auto simp del: suffix-nth) []
  apply auto
  done
  ultimately have suffix j ξ  $\models_r$  η
    using Until-ltlr by simp
  moreover {
    fix i
    assume i < j
    have ?inf-run (suffix i σ)
      and ?gbarel-accept (suffix i σ)
      and ?lgbarel-accept (suffix i ξ) (suffix i σ)
    unfolding limit-suffix
    using Until-ltlr create-gba-from-nodes--ipath
    apply -
    apply (metis ipath-suffix)
    apply (auto simp del: suffix-nth) []
    apply auto
    done
    moreover have μ ∈ old (suffix i σ 0)
      using σ-pre ⟨i < j⟩ by auto
    ultimately have suffix i ξ  $\models_r$  μ using Until-ltlr by simp
  }
  ultimately show ?case by auto
next
case (Release-ltlr μ η)
{ fix i
  assume η ∈ old (σ i) ∨ (∃ j < i. μ ∈ old (σ j))
  moreover
  {
    assume *: η ∈ old (σ i)
    have ?inf-run (suffix i σ)
      and ?gbarel-accept (suffix i σ)
      and ?lgbarel-accept (suffix i ξ) (suffix i σ)
    unfolding limit-suffix
    using Release-ltlr create-gba-from-nodes--ipath
    apply -
    apply (metis ipath-suffix)
    apply (auto simp del: suffix-nth) []
    apply auto
  }
}

```

```

    done
  with * have suffix i  $\xi \models_r \eta$ 
    using Release-ltlr by auto
}
moreover
{
  assume  $\exists j < i. \mu \in \text{old } (\sigma j)$ 
  then obtain j where  $j < i$  and  $\mu \in \text{old } (\sigma j)$  by auto
  moreover
  have ?inf-run (suffix j  $\sigma$ )
    and ?gbarel-accept (suffix j  $\sigma$ )
    and ?lgbarel-accept (suffix j  $\xi$ ) (suffix j  $\sigma$ ) unfolding limit-suffix
    using Release-ltlr create-gba-from-nodes--ipath
    apply -
    apply (metis ipath-suffix)
    apply (auto simp del: suffix-nth) []
    apply auto
  done
  ultimately have suffix j  $\xi \models_r \mu$ 
    using Release-ltlr by auto
  then have  $\exists j < i. \text{suffix } j \xi \models_r \mu$ 
    using <j < i> by auto
}
ultimately have suffix i  $\xi \models_r \eta \vee (\exists j < i. \text{suffix } j \xi \models_r \mu)$  by auto
}
then show ?case using Release-ltlr L4-2c by auto
qed

```

lemma *L4-8*:

```

  assumes gba.is-acc-run  $\sigma$ 
    and  $\forall i. \text{gba.L } (\sigma i) (\xi i)$ 
    and  $\psi \in \text{old } (\sigma 0)$ 
  shows  $\xi \models_r \psi$ 
  using assms
  unfolding gba.is-acc-run-def gba.is-run-def
  using L4-8' by blast

```

lemma *expand-expand-init-propag*:

```

  assumes  $(\forall nm \in \text{incoming } n'. nm < \text{name } n')$ 
     $\wedge \text{name } n' \leq \text{name } (\text{fst } n\text{-ns})$ 
     $\wedge (\text{incoming } n' \cap \text{incoming } (\text{fst } n\text{-ns}) \neq \{\})$ 
     $\longrightarrow \text{new } n' \subseteq \text{old } (\text{fst } n\text{-ns}) \cup \text{new } (\text{fst } n\text{-ns})$ 
    (is ?Q n-ns)
  and  $\forall nd \in \text{snd } n\text{-ns}. \forall nm \in \text{incoming } n'. nm \in \text{incoming } nd \longrightarrow \text{new } n' \subseteq \text{old } nd$ 
    (is ?P (snd n-ns))
  shows  $\text{expand } n\text{-ns} \leq \text{SPEC } (\lambda r. \text{name } n' \leq \text{fst } r \wedge ?P (\text{snd } r))$ 
  using assms
  proof (rule-tac expand-rec-rule[where  $\Phi = \lambda x. ?Q x \wedge ?P (\text{snd } x)$ ],

```

```

    simp,
    intro refine-vcg, goal-cases)
case prems: (1 f x n ns)
then have goal-assms: new n = {}  $\wedge$  ?Q (n, ns)  $\wedge$  ?P ns by simp
{ fix nd nm
  assume nd $\in$ upd-incoming n ns and nm $\in$ incoming n' and nm $\in$ incoming nd
  { assume nd $\in$ ns
    with goal-assms  $\langle$ nm $\in$ incoming n'  $\langle$ nm $\in$ incoming nd  $\rangle$  have new n'  $\subseteq$  old nd
    by auto }
  moreover
  { assume nd $\notin$ ns
    with upd-incoming--elem[OF  $\langle$ nd $\in$ upd-incoming n ns  $\rangle$ ]
    obtain nd' where nd' $\in$ ns
      and nd-eq: nd = nd'( $\setminus$ incoming := incoming n  $\cup$  incoming nd')
      and old-next-eq: old nd' = old n  $\wedge$  next nd' = next n by auto
    { assume nm $\in$ incoming nd'
      with goal-assms  $\langle$ nm $\in$ incoming n'  $\langle$ nd' $\in$ ns  $\rangle$  have new n'  $\subseteq$  old nd
      unfolding nd-eq by auto }
    moreover
    { assume nm $\in$ incoming n
      with nd-eq old-next-eq goal-assms  $\langle$ nm $\in$ incoming n'  $\rangle$ 
      have new n'  $\subseteq$  old nd
      by auto }
    ultimately have new n'  $\subseteq$  old nd using  $\langle$ nm $\in$ incoming nd  $\rangle$  nd-eq by auto }
    ultimately have new n'  $\subseteq$  old nd by fast }
  with prems show ?case by auto
}
next
case prems: (2 f x n ns)
then have goal-assms: new n = {}  $\wedge$  ?Q (n, ns)  $\wedge$  ?P ns and step:  $\bigwedge$ x. ?Q x
 $\wedge$  ?P (snd x)
 $\implies$  f x  $\leq$  SPEC ( $\lambda$ r. name n'  $\leq$  fst r  $\wedge$  ?P (snd r))
  by simp-all
then show ?case
proof (rule-tac step, goal-cases)
  case prems': 1
  have expand-new-name-less: name n < expand-new-name (name n)
    by auto
  moreover have name n  $\notin$  incoming n'
    using goal-assms by auto
  ultimately show ?case using prems' by auto
qed
next
case 3 then show ?case by auto
next
case prems: (4 f x n ns)
then have step:  $\bigwedge$ x. ?Q x  $\wedge$  ?P (snd x)  $\implies$  f x  $\leq$  SPEC ( $\lambda$ r. name n'  $\leq$  fst r
 $\wedge$  ?P (snd r))
  by simp
from prems show ?case by (rule-tac step) auto

```

```

next
  case prems: (5 f x n ns)
  then have step:  $\bigwedge x. ?Q\ x \wedge ?P\ (\text{snd } x) \implies f\ x \leq \text{SPEC } (\lambda r. \text{name } n' \leq \text{fst } r \wedge ?P\ (\text{snd } r))$ 
  by simp
  from prems show ?case by (rule-tac step) auto
next
  case 6 then show ?case by auto
next
  case prems: (7 f x n ns)
  then have step:  $\bigwedge x. ?Q\ x \wedge ?P\ (\text{snd } x) \implies f\ x \leq \text{SPEC } (\lambda r. \text{name } n' \leq \text{fst } r \wedge ?P\ (\text{snd } r))$ 
  by simp
  from prems show ?case by (rule-tac step) auto
next
  case prems: (8 f x n ns  $\psi$ )
  then have goal-assms:  $\psi \in \text{new } n$ 
     $\wedge \neg (\exists q. \psi = \text{prop}_r(q) \vee \psi = \text{nprop}_r(q))$ 
     $\wedge \psi \neq \text{true}_r \wedge \psi \neq \text{false}_r$ 
     $\wedge \neg (\exists \nu \mu. \psi = \nu \text{ and}_r \mu \vee \psi = X_r \nu)$ 
     $\wedge (\exists \nu \mu. \psi = \nu \text{ or}_r \mu \vee \psi = \nu U_r \mu \vee \psi = \nu R_r \mu)$ 
  by (cases  $\psi$ ) auto
  from prems have QP:  $?Q\ (n, ns) \wedge ?P\ ns$  and
    step:  $\bigwedge x. ?Q\ x \wedge ?P\ (\text{snd } x) \implies f\ x \leq \text{SPEC } (\lambda r. \text{name } n' \leq \text{fst } r \wedge ?P\ (\text{snd } r))$ 
  by simp-all
  show ?case
    using goal-assms QP unfolding case-prod-unfold  $\langle x = (n, ns) \rangle$ 
  proof (rule-tac SPEC-rule-nested2, goal-cases)
    case 1
    then show ?case by (rule-tac step) auto
  next
    case 2
    then show ?case by (rule-tac step) auto
  qed
qed

lemma expand-init-propag-on-create-graph:
  create-graph  $\varphi \leq \text{SPEC}(\lambda nds. \forall nd \in nds. \text{expand-init} \in \text{incoming } nd \longrightarrow \varphi \in \text{old } nd)$ 
  unfolding create-graph-def
  by (intro refine-vcg, rule-tac order-trans,
    rule-tac expand-expand-init-propag where
       $n' = \{\}$   $\text{name} = \text{expand-new-name } \text{expand-init}$ ,
       $\text{incoming} = \{\text{expand-init}\}$ ,
       $\text{new} = \{\varphi\}$ ,
       $\text{old} = \{\}$ ,
       $\text{next} = \{\} \ \|\})$  (auto simp add:expand-new-name-expand-init)

```

lemma *L4-6*:
assumes $q \in \text{gba}.V0$
shows $\varphi \in \text{old } q$
using *assms inres-SPEC[OF res expand-init-propag-on-create-graph]*
unfolding *create-gba-from-nodes-def* **by** *auto*

lemma *L4-9*:
assumes $\text{acc}: \text{gba}.\text{accept } \xi$
shows $\xi \models_r \varphi$
proof –
from *acc* **obtain** σ **where** $\text{accept}: \text{gba}.\text{is-acc-run } \sigma \wedge (\forall i. \text{gba}.L (\sigma i) (\xi i))$
unfolding *gba.accept-def* **by** *auto*
then **have** $\sigma \in \text{gba}.V0$
unfolding *gba.is-acc-run-def gba.is-run-def* **by** *simp*
with *L4-6* **have** $\varphi \in \text{old } (\sigma 0)$ **by** *auto*
with *L4-8 accept* **show** *?thesis* **by** *auto*
qed

lemma *L4-10*:
assumes $\xi \models_r \varphi$
shows $\text{gba}.\text{accept } \xi$
proof –
define σ **where** $\sigma = \text{auto-run } \xi \text{ qs}$
let $?G = \text{create-graph } \varphi$

have $\sigma\text{-prop-0}: (\sigma 0) \in \text{qs} \wedge \text{expand-init} \in \text{incoming}(\sigma 0) \wedge \text{auto-run-j } 0 \ \xi (\sigma 0)$
(is ?sbthm)
using *inres-SPEC[OF res L4-7[OF $\xi \models_r \varphi$]]*
unfolding $\sigma\text{-def auto-run.simps}$ **by** *(rule-tac someI-ex, simp) blast*

have $\sigma\text{-valid}: \forall j. \sigma j \in \text{qs} \wedge \text{auto-run-j } j \ \xi (\sigma j)$ **(is** $\forall j. \text{?}\sigma\text{-valid } j$ **)**
proof
fix j
show $\text{?}\sigma\text{-valid } j$
proof(*induct j*)
from $\sigma\text{-prop-0}$ **show** $\text{?}\sigma\text{-valid } 0$ **by** *fast*
next
fix k
assume $\text{goal-assms}: \sigma k \in \text{qs} \wedge \text{auto-run-j } k \ \xi (\sigma k)$
with *inres-SPEC[OF res L4-5, of suffix k ξ]*
have $\text{sbthm}: \exists q'. q' \in \text{qs} \wedge \text{name } (\sigma k) \in \text{incoming } q' \wedge \text{auto-run-j } (\text{Suc } k) \ \xi q'$
(is $\exists q'. \text{?sbthm } q'$)
by *auto*
have $\text{?sbthm } (\sigma (\text{Suc } k))$ **using** *someI-ex[OF sbthm]*
unfolding $\sigma\text{-def auto-run.simps}$ **by** *blast*
then **show** $\text{?}\sigma\text{-valid } (\text{Suc } k)$ **by** *fast*
qed
qed

have σ -prop-Suc: $\bigwedge k. \sigma k \in qs \wedge \sigma (Suc\ k) \in qs$
 $\wedge name\ (\sigma\ k) \in incoming\ (\sigma\ (Suc\ k))$
 $\wedge auto-run-j\ (Suc\ k)\ \xi\ (\sigma\ (Suc\ k))$
proof –
fix k
from σ -valid **have** $\sigma\ k \in qs$ **and** $auto-run-j\ k\ \xi\ (\sigma\ k)$ **by** *blast+*
with *inres-SPEC[OF res L4-5, of suffix k ξ]*
have $sbthm: \exists q'. q' \in qs \wedge name\ (\sigma\ k) \in incoming\ q'$
 $\wedge auto-run-j\ (Suc\ k)\ \xi\ q'$ (**is** $\exists q'. ?sbthm\ q'$)
by *auto*
show $\sigma\ k \in qs \wedge ?sbthm\ (\sigma\ (Suc\ k))$ **using** $\langle \sigma\ k \in qs \rangle\ someI-ex[OF\ sbthm]$
unfolding σ -def $auto-run.simps$ **by** *blast*
qed

have σ -vnaccept:
 $\forall k\ \mu\ \eta. \mu\ U_r\ \eta \in old\ (\sigma\ k) \longrightarrow \neg (\forall i. \{\mu, \mu\ U_r\ \eta\} \subseteq old\ (\sigma\ (k+i)) \wedge \eta \notin old\ (\sigma\ (k+i)))$
proof *clarify*
fix $k\ \mu\ \eta$
assume $U-in: \mu\ U_r\ \eta \in old\ (\sigma\ k)$
and $cntr-prm: \forall i. \{\mu, \mu\ U_r\ \eta\} \subseteq old\ (\sigma\ (k+i)) \wedge \eta \notin old\ (\sigma\ (k+i))$
have $suffix\ k\ \xi \models_r \mu\ U_r\ \eta$
using $U-in$ σ -valid **by** *force*
then obtain i **where** $suffix\ (k+i)\ \xi \models_r \eta$ **and** $\forall j < i. suffix\ (k+j)\ \xi \models_r \mu$
by *auto*
moreover have $\mu\ U_r\ \eta \in old\ (\sigma\ (k+i)) \wedge \eta \notin old\ (\sigma\ (k+i))$
using $cntr-prm$ **by** *auto*
ultimately show *False*
using σ -valid **by** *force*
qed

have σ -exec: $gba.is-run\ \sigma$
using σ -prop-0 σ -prop-Suc σ -valid
unfolding $gba.is-run-def\ ipath-def$
unfolding $create-gba-from-nodes-def$
by *auto*
moreover
have σ -vaccept:
 $\forall k\ \mu\ \eta. \mu\ U_r\ \eta \in old\ (\sigma\ k) \longrightarrow$
 $(\exists j. (\forall i < j. \{\mu, \mu\ U_r\ \eta\} \subseteq old\ (\sigma\ (k+i))) \wedge \eta \in old\ (\sigma\ (k+j)))$
proof (*clarify*)
fix $k\ \mu\ \eta$
assume $U-in: \mu\ U_r\ \eta \in old\ (\sigma\ k)$
then have $\neg (\forall i. \{\mu, \mu\ U_r\ \eta\} \subseteq old\ (suffix\ k\ \sigma\ i) \wedge \eta \notin old\ (suffix\ k\ \sigma\ i))$
using σ -vnaccept [*THEN alle, of k*] **by** *auto*
moreover have $suffix\ k\ \sigma\ 0 \in qs$ **using** σ -valid **by** *auto*
ultimately show $\exists j. (\forall i < j. \{\mu, \mu\ U_r\ \eta\} \subseteq old\ (\sigma\ (k+i))) \wedge \eta \in old\ (\sigma\ (k+j))$
apply –
apply (*rule make-pos-rule'[OF L4-2a]*)


```

apply (fold suffix-def)
apply (rule ipath-suffix)
using  $\sigma$ -exec[unfolded gba.is-run-def]
apply simp
using U-in apply simp
apply simp
done
qed

have gba.is-acc  $\sigma$ 
  unfolding gba.is-acc-def
proof
  fix  $S$ 
  assume  $S \in \text{gba}.F$ 
  then obtain  $\mu \eta$  where  $S$ -eq:  $S = \{q \in qs. \mu U_r \eta \in \text{old } q \longrightarrow \eta \in \text{old } q\}$ 
    and  $\mu U_r \eta \in \text{subfrmlsr } \varphi$ 
    by (auto simp add: create-gba-from-nodes-def)
  have range-subset: range  $\sigma \subseteq qs$ 
  proof
    fix  $q$ 
    assume  $q \in \text{range } \sigma$ 
    with full-SetCompr-eq[of  $\sigma$ ] obtain  $k$  where  $q = \sigma k$  by auto
    then show  $q \in qs$  using  $\sigma$ -valid by auto
  qed
  with limit-nonempty[of  $\sigma$ ]
    limit-in-range[of  $\sigma$ ]
    finite-subset[OF range-subset]
    inres-SPEC[OF res create-graph-finite]
  obtain  $q$  where  $q$ -in-limit:  $q \in \text{limit } \sigma$  and  $q$ -is-node:  $q \in qs$ 
    by auto
  show  $\exists_{\infty} i. \sigma i \in S$ 
  proof (cases  $\mu U_r \eta \in \text{old } q$ )
    case False
      with  $S$ -eq  $q$ -in-limit  $q$ -is-node
      show ?thesis
      by (auto simp: limit-iff-frequent intro: INFM-mono)
    next
      case True
      obtain  $k$  where  $q$ -eq:  $q = \sigma k$  using  $q$ -in-limit
      unfolding limit-iff-frequent by (metis (lifting, full-types) INFM-nat-le)
      have  $\exists_{\infty} k. \eta \in \text{old } (\sigma k)$ 
      unfolding INFM-nat
      proof (rule ccontr)
        assume  $\neg (\forall m. \exists n > m. \eta \in \text{old } (\sigma n))$ 
        then obtain  $m$  where  $\forall n > m. \eta \notin \text{old } (\sigma n)$  by auto
        moreover
        from  $q$ -eq  $q$ -in-limit limit-iff-frequent[of  $q \sigma$ ] INFM-nat[of  $\lambda n. \sigma n = q$ ]
        obtain  $n$  where  $m < n$  and  $\sigma n$ -eq:  $\sigma n = \sigma k$  by auto
        moreover

```

```

    obtain  $j$  where  $\eta \in \text{old } (\sigma (n+j))$ 
      using  $\sigma\text{-vaccpt } \langle \mu U_r \eta \in \text{old } q \rangle$  unfolding  $q\text{-eq}$  by  $(\text{fold } \sigma n\text{-eq})$  force
      ultimately show  $\text{False}$  by  $\text{auto}$ 
  qed
  then have  $\exists_{\infty} k. \sigma k \in qs \wedge \eta \in \text{old } (\sigma k)$ 
    using  $\sigma\text{-valid}$  by  $(\text{auto intro: INF-mono})$ 
  then show  $\exists_{\infty} k. \sigma k \in S$ 
    unfolding  $S\text{-eq}$  by  $(\text{rule INFM-mono})$   $\text{simp}$ 
  qed
  qed
  moreover have  $\text{gba.L } (\sigma i) (\xi i)$  for  $i$ 
  proof –
    from  $\sigma\text{-valid}$  have  $[\text{simp}]: \sigma i \in qs$  by  $\text{auto}$ 
    have  $\forall \psi \in \text{old } (\sigma i). \text{suffix } i \xi \models_r \psi$ 
      using  $\sigma\text{-valid}$  by  $\text{auto}$ 
    then show  $?thesis$ 
      unfolding  $\text{create-gba-from-nodes-def}$  by  $\text{auto}$ 
  qed
  ultimately show  $?thesis$ 
  unfolding  $\text{gba.accept-def gba.is-acc-run-def}$  by  $\text{blast}$ 
  qed

end
end

lemma  $\text{create-graph--name-ident: create-graph } \varphi \leq \text{SPEC } (\lambda nds. \forall q \in nds. \exists ! q' \in nds. \text{name } q = \text{name } q')$ 
  unfolding  $\text{create-graph-def}$ 
  apply  $(\text{refine-vcg expand-name-propag--name-ident})$ 
  by  $(\text{auto simp add: expand-new-name-expand-init})$ 

corollary  $\text{create-graph--name-inj: create-graph } \varphi \leq \text{SPEC } (\lambda nds. \text{inj-on name } nds)$ 
  by  $(\text{rule order-trans[OF create-graph--name-ident]})$   $(\text{auto intro: inj-onI})$ 

definition
   $\text{create-gba } \varphi$ 
   $\equiv \text{do } \{ \text{nds} \leftarrow \text{create-graph}_T \varphi;$ 
     $\text{RETURN } (\text{create-gba-from-nodes } \varphi \text{ nds}) \}$ 

lemma  $\text{create-graph-precond: create-graph } \varphi$ 
   $\leq \text{SPEC } (\text{create-gba-from-nodes-precond } \varphi)$ 
  apply  $(\text{clarsimp simp: pw-le-iff create-graph-nofail})$ 
  apply  $\text{unfold-locales}$ 
  apply  $\text{simp}$ 
  done

lemma  $\text{create-gba--invar: create-gba } \varphi \leq \text{SPEC } \text{gba}$ 

```

unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph-precond]*)
by (*rule create-gba-from-nodes-precond.create-gba-from-nodes--invar*)

lemma *create-gba-acc:*

shows *create-gba* $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. \forall \xi. \text{gba.accept } \mathcal{A} \ \xi \longleftrightarrow \xi \models_r \varphi)$
unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph-precond]*)
using *create-gba-from-nodes-precond.L4-9*
using *create-gba-from-nodes-precond.L4-10*
by *blast*

lemma *create-gba--name-inj:*

shows *create-gba* $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. (\text{inj-on name } (g-V \ \mathcal{A})))$
unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph--name-inj]*)
apply (*auto simp: create-gba-from-nodes-def*)
done

lemma *create-gba--fin:*

shows *create-gba* $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. (\text{finite } (g-V \ \mathcal{A})))$
unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph-finite]*)
apply (*auto simp: create-gba-from-nodes-def*)
done

lemma *create-graph-old-finite:*

create-graph $\varphi \leq \text{SPEC}(\lambda \text{nds}. \forall \text{nd} \in \text{nds}. \text{finite } (\text{old nd}))$

proof –

show *?thesis*
unfolding *create-graph-def create-graph-eq-create-graph_T[symmetric]*
unfolding *expand-def*
apply (*intro refine-vcg*)
apply (*rule-tac order-trans*)
apply (*rule-tac REC-le-RECT*)
apply (*fold expand_T-def*)
apply (*rule-tac order-trans[OF expand-term-prop]*)
apply *auto[1]*
apply (*rule-tac SPEC-rule*)
apply *auto*
by (*metis infinite-super subfrmlsr-finite*)

qed

lemma *create-gba--old-fin:*

shows *create-gba* $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. \forall \text{nd} \in g-V \ \mathcal{A}. \text{finite } (\text{old nd}))$
unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph-old-finite]*)
apply (*simp add: create-gba-from-nodes-def*)
done

lemma *create-gba--incoming-exists*:
shows *create-gba* φ
 \leq *SPEC*($\lambda\mathcal{A}. \forall nd \in g-V \mathcal{A}. \text{incoming } nd \subseteq \text{insert } \text{expand-init } (\text{name } ' (g-V \mathcal{A}))$)
unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph--incoming-name-exist]*)
apply (*auto simp add: create-gba-from-nodes-def*)
done

lemma *create-gba--no-init*:
shows *create-gba* $\varphi \leq$ *SPEC*($\lambda\mathcal{A}. \text{expand-init} \notin \text{name } ' (g-V \mathcal{A})$)
unfolding *create-gba-def create-graph-eq-create-graph_T[symmetric]*
apply (*refine-rcg refine-vcg order-trans[OF create-graph--incoming-name-exist]*)
apply (*auto simp add: create-gba-from-nodes-def*)
done

definition *nds-invars* *nds* \equiv
inj-on name *nds*
 \wedge *finite* *nds*
 \wedge *expand-init* \notin *name* ' *nds*
 \wedge ($\forall nd \in$ *nds*.
finite (*old* *nd*)
 \wedge *incoming* *nd* \subseteq *insert* *expand-init* (*name* ' *nds*))

lemma *create-gba-nds-invars*: *create-gba* $\varphi \leq$ *SPEC* ($\lambda\mathcal{A}. \text{nds-invars } (g-V \mathcal{A})$)
using *create-gba--name-inj[of* φ *create-gba--fin[of* φ
create-gba--old-fin[of φ *create-gba--incoming-exists[of* φ
create-gba--no-init[of φ
unfolding *nds-invars-def*
by (*simp add: pw-le-iff*)

theorem *T4-1*:
create-gba $\varphi \leq$ *SPEC*(
 $\lambda\mathcal{A}. \text{gba } \mathcal{A}$
 \wedge *finite* (*g-V* \mathcal{A})
 \wedge ($\forall \xi. \text{gba.accept } \mathcal{A} \xi \longleftrightarrow \xi \models_r \varphi$)
 \wedge (*nds-invars* (*g-V* \mathcal{A})))
using *create-gba--invar create-gba--fin create-gba-acc create-gba-nds-invars*
apply (*simp add: pw-le-iff*)
apply *blast*
done

definition *create-name-gba* $\varphi \equiv$ *do* {
 $G \leftarrow$ *create-gba* φ ;
ASSERT (*nds-invars* (*g-V* G));
RETURN (*gba-rename* *name* G)
}

theorem *create-name-gba-correct*:

```
create-name-gba  $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. \text{gba } \mathcal{A} \wedge \text{finite } (g\text{-}V \ \mathcal{A}) \wedge (\forall \xi. \text{gba.accept } \mathcal{A} \ \xi \longleftrightarrow \xi \models_r \varphi))$ 
unfolding create-name-gba-def
apply (refine-rcg refine-vcg order-trans[OF T4-1])
apply (simp-all add: nds-invars-def gba-rename-correct)
done
```

definition *create-name-igba* :: *'a::linorder ltr* \Rightarrow - **where**

```
create-name-igba  $\varphi \equiv \text{do } \{$ 
   $A \leftarrow \text{create-name-gba } \varphi;$ 
   $A' \leftarrow \text{gba-to-idx } A;$ 
  stat-set-data-nres (card ( $g\text{-}V \ A$ )) (card ( $g\text{-}V0 \ A'$ )) (igbg-num-acc  $A'$ );
  RETURN  $A'$ 
}
```

lemma *create-name-igba-correct*: *create-name-igba* $\varphi \leq \text{SPEC} (\lambda G.$

```
igba  $G \wedge \text{finite } (g\text{-}V \ G) \wedge (\forall \xi. \text{igba.accept } G \ \xi \longleftrightarrow \xi \models_r \varphi))$ 
unfolding create-name-igba-def
apply (refine-rcg
  order-trans[OF create-name-gba-correct]
  order-trans[OF gba.gba-to-idx-ext-correct]
  refine-vcg)
apply clarsimp-all
```

proof –

```
fix  $G :: (\text{nat}, 'a \text{ set}) \text{ gba-rec}$ 
fix  $A :: \text{nat set}$ 
assume  $1: \text{gba } G$ 
assume  $2: \text{finite } (g\text{-}V \ G) \ A \in \text{gbg-F } G$ 
interpret gba  $G$  using  $1$  .
show finite  $A$  using finite-V-Fe  $2$  .
qed
```

context

```
notes [refine-vcg] = order-trans[OF create-name-gba-correct]
begin
```

lemma *create-name-igba* $\varphi \leq \text{SPEC} (\lambda G. \text{igba } G \wedge (\forall \xi. \text{igba.accept } G \ \xi \longleftrightarrow \xi \models_r \varphi))$

```
unfolding create-name-igba-def
proof (refine-rcg refine-vcg, clarsimp-all)
fix  $G :: (\text{nat}, 'a \text{ set}) \text{ gba-rec}$ 
assume gba  $G$ 
then interpret gba  $G$  .
note [refine-vcg] = order-trans[OF gba-to-idx-ext-correct]
```

```
assume  $\forall \xi. \text{gba.accept } G \ \xi = \xi \models_r \varphi \ \text{finite } (g\text{-}V \ G)$ 
then show gba-to-idx  $G \leq \text{SPEC} (\lambda G'. \text{igba } G' \wedge (\forall \xi. \text{igba.accept } G' \ \xi = \xi \models_r$ 
```

```

 $\varphi$ )
  by (refine-rcg refine-vcg) (auto intro: finite-V-Fe)
qed

end

end

```

3 Refinement to Efficient Code

```

theory LTL-to-GBA-impl
imports
  LTL-to-GBA
  Deriving.Compare-Instances
  CAVA-Automata.Automata-Impl
  CAVA-Base.CAVA-Code-Target
begin

```

3.1 Parametricity Setup Boilerplate

3.1.1 LTL Formulas

```

derive linorder ltlr

```

inductive-set *ltlr-rel* for *R* where

```

  (True-ltlr, True-ltlr)  $\in$  ltlr-rel R
| (False-ltlr, False-ltlr)  $\in$  ltlr-rel R
| (a,a') $\in$ R  $\implies$  (Prop-ltlr a,Prop-ltlr a')  $\in$  ltlr-rel R
| (a,a') $\in$ R  $\implies$  (Nprop-ltlr a,Nprop-ltlr a')  $\in$  ltlr-rel R
| [(a,a') $\in$ ltlr-rel R; (b,b') $\in$ ltlr-rel R]
   $\implies$  (And-ltlr a b,And-ltlr a' b')  $\in$  ltlr-rel R
| [(a,a') $\in$ ltlr-rel R; (b,b') $\in$ ltlr-rel R]
   $\implies$  (Or-ltlr a b,Or-ltlr a' b')  $\in$  ltlr-rel R
| [(a,a') $\in$ ltlr-rel R]  $\implies$  (Next-ltlr a,Next-ltlr a')  $\in$  ltlr-rel R
| [(a,a') $\in$ ltlr-rel R; (b,b') $\in$ ltlr-rel R]
   $\implies$  (Until-ltlr a b,Until-ltlr a' b')  $\in$  ltlr-rel R
| [(a,a') $\in$ ltlr-rel R; (b,b') $\in$ ltlr-rel R]
   $\implies$  (Release-ltlr a b,Release-ltlr a' b')  $\in$  ltlr-rel R

```

lemmas *ltlr-rel-induct*[*induct set*]
 = *ltlr-rel.induct*[*simplified relAPP-def*[*of ltlr-rel, symmetric*]]

lemmas *ltlr-rel-cases*[*cases set*]
 = *ltlr-rel.cases*[*simplified relAPP-def*[*of ltlr-rel, symmetric*]]

lemmas *ltlr-rel-intros*
 = *ltlr-rel.intros*[*simplified relAPP-def*[*of ltlr-rel, symmetric*]]

inductive-simps *ltlr-rel-left-simps*[*simplified relAPP-def*[*of ltlr-rel, symmetric*]]:

```

  (True-ltlr,z)  $\in$  ltlr-rel R
  (False-ltlr,z)  $\in$  ltlr-rel R

```

$(\text{Prop-ltlr } p, z) \in \text{ltlr-rel } R$
 $(\text{Nprop-ltlr } p, z) \in \text{ltlr-rel } R$
 $(\text{And-ltlr } a \ b, z) \in \text{ltlr-rel } R$
 $(\text{Or-ltlr } a \ b, z) \in \text{ltlr-rel } R$
 $(\text{Next-ltlr } a, z) \in \text{ltlr-rel } R$
 $(\text{Until-ltlr } a \ b, z) \in \text{ltlr-rel } R$
 $(\text{Release-ltlr } a \ b, z) \in \text{ltlr-rel } R$

lemma *ltlr-rel-sv*[*relator-props*]:
assumes *SV*: *single-valued* R
shows *single-valued* $\langle R \rangle \text{ltlr-rel}$
proof (*intro single-valuedI allI impI*)
fix $x \ y \ z$
assume $(x, y) \in \langle R \rangle \text{ltlr-rel} \ (x, z) \in \langle R \rangle \text{ltlr-rel}$
then show $y=z$
apply (*induction arbitrary: z*)
apply (*simp (no-asm-use) only: ltlr-rel-left-simps*
| *blast intro: single-valuedD[OF SV]*)
done
qed

lemma *ltlr-rel-id*[*relator-props*]: $\llbracket R = \text{Id} \rrbracket \implies \langle R \rangle \text{ltlr-rel} = \text{Id}$
proof (*intro equalityI subsetI, clarsimp-all*)
fix $a \ b$
assume $(a, b) \in \langle \text{Id} \rangle \text{ltlr-rel}$
then show $a=b$
by *induction auto*
next
fix a
show $(a, a) \in \langle \text{Id} \rangle \text{ltlr-rel}$
by (*induction a*) (*auto intro: ltlr-rel-intros*)
qed

lemma *ltlr-rel-id-simp*[*simp*]: $\langle \text{Id} \rangle \text{ltlr-rel} = \text{Id}$ **by** (*rule ltlr-rel-id*) *simp*

consts *i-ltlr* :: *interface* \Rightarrow *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of ltlr-rel i-ltlr*]

thm *ltlr-rel-intros*[*no-vars*]

lemma *ltlr-con-param*[*param, autoref-rules*]:
 $(\text{True-ltlr}, \text{True-ltlr}) \in \langle R \rangle \text{ltlr-rel}$
 $(\text{False-ltlr}, \text{False-ltlr}) \in \langle R \rangle \text{ltlr-rel}$
 $(\text{Prop-ltlr}, \text{Prop-ltlr}) \in R \rightarrow \langle R \rangle \text{ltlr-rel}$
 $(\text{Nprop-ltlr}, \text{Nprop-ltlr}) \in R \rightarrow \langle R \rangle \text{ltlr-rel}$
 $(\text{And-ltlr}, \text{And-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$
 $(\text{Or-ltlr}, \text{Or-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$
 $(\text{Next-ltlr}, \text{Next-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$
 $(\text{Until-ltlr}, \text{Until-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$

$(\text{Release-ltlr}, \text{Release-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$
by (*auto intro: ltlr-rel-intros*)

lemma *case-ltlr-param*[*param, autoref-rules*]:
 $(\text{case-ltlr}, \text{case-ltlr}) \in Rv \rightarrow Rv \rightarrow (R \rightarrow Rv)$
 $\rightarrow (R \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv) \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv$
apply (*clarsimp*)
apply (*case-tac ai, simp-all add: ltlr-rel-left-simps*)
apply (*clarsimp-all*)
apply *parametricity+*
done

lemma *rec-ltlr-param*[*param, autoref-rules*]:
 $(\text{rec-ltlr}, \text{rec-ltlr}) \in Rv \rightarrow Rv \rightarrow (R \rightarrow Rv)$
 $\rightarrow (R \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$
 $\rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv$

proof (*clarsimp, goal-cases*)

case *prems: 1*
from *prems(10)*
show *?case*
apply (*induction*)
using *prems(1-9)*
apply *simp-all*
apply *parametricity+*
done

qed

lemma *case-ltlr-mono*[*refine-mono*]:
assumes $\varphi = \text{True-ltlr} \implies a \leq a'$
assumes $\varphi = \text{False-ltlr} \implies b \leq b'$
assumes $\bigwedge p. \varphi = \text{Prop-ltlr } p \implies c \leq c' \text{ } p$
assumes $\bigwedge p. \varphi = \text{Nprop-ltlr } p \implies d \leq d' \text{ } p$
assumes $\bigwedge \mu \nu. \varphi = \text{And-ltlr } \mu \nu \implies e \leq e' \text{ } \mu \nu$
assumes $\bigwedge \mu \nu. \varphi = \text{Or-ltlr } \mu \nu \implies f \leq f' \text{ } \mu \nu$
assumes $\bigwedge \mu. \varphi = \text{Next-ltlr } \mu \implies g \leq g' \text{ } \mu$
assumes $\bigwedge \mu \nu. \varphi = \text{Until-ltlr } \mu \nu \implies h \leq h' \text{ } \mu \nu$
assumes $\bigwedge \mu \nu. \varphi = \text{Release-ltlr } \mu \nu \implies i \leq i' \text{ } \mu \nu$
shows $\text{case-ltlr } a \ b \ c \ d \ e \ f \ g \ h \ i \ \varphi \leq \text{case-ltlr } a' \ b' \ c' \ d' \ e' \ f' \ g' \ h' \ i' \ \varphi$
using *assms*

apply (*cases* φ)
apply *simp-all*
done

primrec *ltlr-eq* **where**

ltlr-eq *eq* *True-ltlr* $f \longleftrightarrow (\text{case } f \text{ of } \text{True-ltlr} \Rightarrow \text{True} \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* *False-ltlr* $f \longleftrightarrow (\text{case } f \text{ of } \text{False-ltlr} \Rightarrow \text{True} \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*Prop-ltlr* p) $f \longleftrightarrow (\text{case } f \text{ of } \text{Prop-ltlr } p' \Rightarrow \text{eq } p \ p' \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*Nprop-ltlr* p) $f \longleftrightarrow (\text{case } f \text{ of } \text{Nprop-ltlr } p' \Rightarrow \text{eq } p \ p' \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*And-ltlr* $\mu \ \nu$) f
 $\longleftrightarrow (\text{case } f \text{ of } \text{And-ltlr } \mu' \ \nu' \Rightarrow \text{ltlr-eq } \text{eq } \mu \ \mu' \wedge \text{ltlr-eq } \text{eq } \nu \ \nu' \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*Or-ltlr* $\mu \ \nu$) f
 $\longleftrightarrow (\text{case } f \text{ of } \text{Or-ltlr } \mu' \ \nu' \Rightarrow \text{ltlr-eq } \text{eq } \mu \ \mu' \wedge \text{ltlr-eq } \text{eq } \nu \ \nu' \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*Next-ltlr* φ) f
 $\longleftrightarrow (\text{case } f \text{ of } \text{Next-ltlr } \varphi' \Rightarrow \text{ltlr-eq } \text{eq } \varphi \ \varphi' \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*Until-ltlr* $\mu \ \nu$) f
 $\longleftrightarrow (\text{case } f \text{ of } \text{Until-ltlr } \mu' \ \nu' \Rightarrow \text{ltlr-eq } \text{eq } \mu \ \mu' \wedge \text{ltlr-eq } \text{eq } \nu \ \nu' \mid - \Rightarrow \text{False})$
 \mid *ltlr-eq* *eq* (*Release-ltlr* $\mu \ \nu$) f
 $\longleftrightarrow (\text{case } f \text{ of}$
 $\quad \text{Release-ltlr } \mu' \ \nu' \Rightarrow \text{ltlr-eq } \text{eq } \mu \ \mu' \wedge \text{ltlr-eq } \text{eq } \nu \ \nu'$
 $\mid - \Rightarrow \text{False})$

lemma *ltlr-eq-autoref*[*autoref-rules*]:

assumes *EQP*: $(\text{eq}, (=)) \in R \rightarrow R \rightarrow \text{bool-rel}$
shows $(\text{ltlr-eq } \text{eq}, (=)) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \text{bool-rel}$

proof (*intro fun-relI*)

fix $\mu' \ \mu \ \nu' \ \nu$
assume $(\mu', \mu) \in \langle R \rangle \text{ltlr-rel}$ **and** $(\nu', \nu) \in \langle R \rangle \text{ltlr-rel}$
then show $(\text{ltlr-eq } \text{eq } \mu' \ \nu', \mu = \nu) \in \text{bool-rel}$
apply (*induction arbitrary*: $\nu' \ \nu$)
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
apply (*erule* *ltlr-rel-cases*,
 \quad *simp-all add*: *EQP*[*THEN fun-relD*, *THEN fun-relD*, *THEN IdD*]) \square
apply (*erule* *ltlr-rel-cases*,
 \quad *simp-all add*: *EQP*[*THEN fun-relD*, *THEN fun-relD*, *THEN IdD*]) \square
apply (*rotate-tac* -1)
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
apply (*rotate-tac* -1)
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
apply (*rotate-tac* -1)
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
apply (*rotate-tac* -1)
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
apply (*rotate-tac* -1)
apply (*erule* *ltlr-rel-cases*, *simp-all*) \square
done

qed

lemma *ltrl-dftt-cmp*[*autoref-rules-raw*]:
assumes *PREFER-id R*
shows
 $(dftt-cmp (\leq) (<), dftt-cmp (\leq) (<))$
 $\in \langle R \rangle ltrl-rel \rightarrow \langle R \rangle ltrl-rel \rightarrow comp-res-rel$
using *assms*
by *simp*

type-synonym
 $node-name-impl = node-name$

abbreviation $(input) node-name-rel \equiv Id :: (node-name-impl \times node-name) set$

lemma *case-lttr-gtransfer*:
assumes
 $\gamma ai \leq a$
 $\gamma bi \leq b$
 $\bigwedge a. \gamma (ci a) \leq c a$
 $\bigwedge a. \gamma (di a) \leq d a$
 $\bigwedge ltlr1 ltlr2. \gamma (ei ltlr1 ltlr2) \leq e ltlr1 ltlr2$
 $\bigwedge ltlr1 ltlr2. \gamma (fi ltlr1 ltlr2) \leq f ltlr1 ltlr2$
 $\bigwedge ltlr. \gamma (gi ltlr) \leq g ltlr$
 $\bigwedge ltlr1 ltlr2. \gamma (hi ltlr1 ltlr2) \leq h ltlr1 ltlr2$
 $\bigwedge ltlr1 ltlr2. \gamma (iiv ltlr1 ltlr2) \leq i ltlr1 ltlr2$
shows $\gamma (case-lttr ai bi ci di ei fi gi hi iiv \varphi)$
 $\leq (case-lttr a b c d e f g h i \varphi)$
apply (*cases* φ)
apply (*auto intro: assms*)
done

lemmas [*refine-transfer*]
 $= case-lttr-gtransfer[\mathbf{where} \gamma=nres-of] case-lttr-gtransfer[\mathbf{where} \gamma=RETURN]$

lemma [*refine-transfer*]:
assumes
 $ai \neq dSUCCEED$
 $bi \neq dSUCCEED$
 $\bigwedge a. ci a \neq dSUCCEED$
 $\bigwedge a. di a \neq dSUCCEED$
 $\bigwedge ltlr1 ltlr2. ei ltlr1 ltlr2 \neq dSUCCEED$
 $\bigwedge ltlr1 ltlr2. fi ltlr1 ltlr2 \neq dSUCCEED$
 $\bigwedge ltlr. gi ltlr \neq dSUCCEED$
 $\bigwedge ltlr1 ltlr2. hi ltlr1 ltlr2 \neq dSUCCEED$
 $\bigwedge ltlr1 ltlr2. iiv ltlr1 ltlr2 \neq dSUCCEED$
shows $case-lttr ai bi ci di ei fi gi hi iiv \varphi \neq dSUCCEED$
apply (*cases* φ)
apply (*simp-all add: assms*)
done

3.1.2 Nodes

record 'a node-impl =
 name-impl :: node-name-impl
 incoming-impl :: (node-name-impl,unit) RBT-Impl.rbt
 new-impl :: 'a frml list
 old-impl :: 'a frml list
 next-impl :: 'a frml list

definition node-rel **where** node-rel-def-internal: node-rel Re R \equiv {(
 (| name-impl = namei,
 incoming-impl = inci,
 new-impl = newi,
 old-impl = oldi,
 next-impl = nexti,
 ... = morei
 |),
 (| name = name,
 incoming = inc,
 new=new,
 old=old,
 next = next,
 ... = more
 |) | namei name inci inc newi new oldi old nexti next morei more.
 (namei,name) \in node-name-rel
 \wedge (inci,inc) \in (node-name-rel) dflt-rs-rel
 \wedge (newi,new) \in (\langle R \rangle ltlr-rel) lss.rel
 \wedge (oldi,old) \in (\langle R \rangle ltlr-rel) lss.rel
 \wedge (nexti,next) \in (\langle R \rangle ltlr-rel) lss.rel
 \wedge (morei,more) \in Re
 }
 }

lemma node-rel-def: $\langle Re,R \rangle$ node-rel = {(
 (| name-impl = namei,
 incoming-impl = inci,
 new-impl = newi,
 old-impl = oldi,
 next-impl = nexti,
 ... = morei
 |),
 (| name = name,
 incoming = inc,
 new=new,
 old=old,
 next = next,
 ... = more
 |) | namei name inci inc newi new oldi old nexti next morei more.
 (namei,name) \in node-name-rel
 \wedge (inci,inc) \in (node-name-rel) dflt-rs-rel
 \wedge (newi,new) \in (\langle R \rangle ltlr-rel) lss.rel
 }

$\wedge (oldi, old) \in \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}$
 $\wedge (nexti, next) \in \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}$
 $\wedge (morei, more) \in Re$
} by (*simp add: node-rel-def-internal relAPP-def*)

lemma *node-rel-sv[relator-props]*:
single-valued Re \implies single-valued R \implies single-valued $\langle \langle Re, R \rangle \text{node-rel} \rangle$
apply (*rule single-valuedI*)
apply (*simp add: node-rel-def*)
apply (*auto*
dest: single-valuedD lss.rel-sv[OF ltlr-rel-sv] map2set-rel-sv[OF ahm-rel-sv]
dest: single-valuedD[
OF map2set-rel-sv[OF rbt-map-rel-sv[OF single-valued-Id single-valued-Id]]
])
done

consts *i-node :: interface \implies interface \implies interface*

lemmas [*autoref-rel-intf*] = *REL-INTFI[of node-rel i-node]*

lemma [*autoref-rules*]: (*node-impl-ext, node-ext*) \in
node-name-rel
 $\rightarrow \langle \text{node-name-rel} \rangle \text{dflt-rs-rel}$
 $\rightarrow \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}$
 $\rightarrow \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}$
 $\rightarrow \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}$
 $\rightarrow Re$
 $\rightarrow \langle Re, R \rangle \text{node-rel}$
unfolding *node-rel-def*
by *auto*

lemma [*autoref-rules*]:
(*node-impl.name-impl-update, node.name-update*)
 $\in (\text{node-name-rel} \rightarrow \text{node-name-rel}) \rightarrow \langle Re, R \rangle \text{node-rel} \rightarrow \langle Re, R \rangle \text{node-rel}$
(*node-impl.incoming-impl-update, node.incoming-update*)
 $\in (\langle \text{node-name-rel} \rangle \text{dflt-rs-rel} \rightarrow \langle \text{node-name-rel} \rangle \text{dflt-rs-rel})$
 $\rightarrow \langle Re, R \rangle \text{node-rel}$
 $\rightarrow \langle Re, R \rangle \text{node-rel}$
(*node-impl.new-impl-update, node.new-update*)
 $\in (\langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel} \rightarrow \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}) \rightarrow \langle Re, R \rangle \text{node-rel} \rightarrow \langle Re, R \rangle \text{node-rel}$
(*node-impl.old-impl-update, node.old-update*)
 $\in (\langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel} \rightarrow \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}) \rightarrow \langle Re, R \rangle \text{node-rel} \rightarrow \langle Re, R \rangle \text{node-rel}$
(*node-impl.next-impl-update, node.next-update*)
 $\in (\langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel} \rightarrow \langle \langle R \rangle \text{ltlr-rel} \rangle \text{lss.rel}) \rightarrow \langle Re, R \rangle \text{node-rel} \rightarrow \langle Re, R \rangle \text{node-rel}$
(*node-impl.more-update, node.more-update*)
 $\in (Re \rightarrow Re) \rightarrow \langle Re, R \rangle \text{node-rel} \rightarrow \langle Re, R \rangle \text{node-rel}$
unfolding *node-rel-def*
by (*auto dest: fun-relD*)

term *name*

lemma [*autoref-rules*]:

$(node\text{-}impl.name\text{-}impl, node.name) \in \langle Re, R \rangle node\text{-}rel \rightarrow node\text{-}name\text{-}rel$
 $(node\text{-}impl.incoming\text{-}impl, node.incoming)$
 $\in \langle Re, R \rangle node\text{-}rel \rightarrow \langle node\text{-}name\text{-}rel \rangle dflt\text{-}rs\text{-}rel$
 $(node\text{-}impl.new\text{-}impl, node.new) \in \langle Re, R \rangle node\text{-}rel \rightarrow \langle \langle R \rangle ltlr\text{-}rel \rangle lss.rel$
 $(node\text{-}impl.old\text{-}impl, node.old) \in \langle Re, R \rangle node\text{-}rel \rightarrow \langle \langle R \rangle ltlr\text{-}rel \rangle lss.rel$
 $(node\text{-}impl.next\text{-}impl, node.next) \in \langle Re, R \rangle node\text{-}rel \rightarrow \langle \langle R \rangle ltlr\text{-}rel \rangle lss.rel$
 $(node\text{-}impl.more, node.more) \in \langle Re, R \rangle node\text{-}rel \rightarrow Re$
unfolding *node-rel-def* **by** *auto*

3.2 Messaging the Abstract Algorithm

In a first step, we do some refinement steps on the abstract data structures, with the goal to make the algorithm more efficient.

3.2.1 Creation of the Nodes

In the expand-algorithm, we replace nested conditionals by case-distinctions, and slightly stratify the code.

abbreviation (*input*) *expand2* *exp* *n* *ns* φ *n1* *nx1* *n2* \equiv *do* {
 (*nm*, *nds*) \leftarrow *exp* (
 n |
 new := *insert* *n1* (*new* *n*),
 old := *insert* φ (*old* *n*),
 next := *nx1* \cup *next* *n* |),
 ns);
 exp (*n* | *name* := *nm*, *new* := *n2* \cup *new* *n*, *old* := $\{\varphi\}$ \cup *old* *n* |), *nds*)
}

definition *expand-aimpl* \equiv *REC_T* (λ *expand* (*n*, *ns*).

 if *new* *n* = {} then (
 if ($\exists n' \in ns. old\ n' = old\ n \wedge next\ n' = next\ n$) then
 RETURN (*name* *n*, *upd-incoming* *n* *ns*)
 else do {
 ASSERT (*n* \notin *ns*);
 ASSERT (*name* *n* \notin *name*'*ns*);
 expand ((
 name = *expand-new-name* (*name* *n*),
 incoming = {*name* *n*},
 new = *next* *n*,
 old = {},
 next = {} |),
 insert *n* *ns*)
 }
 }

```

) else do {
   $\varphi \leftarrow SPEC (\lambda x. x \in (new\ n));$ 
  let  $n = n(\ new := new\ n - \{\varphi\} );$ 
  case  $\varphi$  of
     $prop_r(q) \Rightarrow$ 
      if  $nprop_r(q) \in old\ n$  then RETURN (name  $n$ ,  $ns$ )
      else expand ( $n(\ old := \{\varphi\} \cup old\ n );$ ,  $ns$ )
    |  $nprop_r(q) \Rightarrow$ 
      if  $prop_r(q) \in old\ n$  then RETURN (name  $n$ ,  $ns$ )
      else expand ( $n(\ old := \{\varphi\} \cup old\ n );$ ,  $ns$ )
    |  $true_r \Rightarrow$  expand ( $n(\ old := \{\varphi\} \cup old\ n );$ ,  $ns$ )
    |  $false_r \Rightarrow$  RETURN (name  $n$ ,  $ns$ )
    |  $\nu$  and  $\mu \Rightarrow$  expand ( $n(\$ 
       $new := insert\ \nu\ (insert\ \mu\ (new\ n)),$ 
       $old := \{\varphi\} \cup old\ n,$ 
       $next := next\ n );$ ,  $ns$ )
    |  $X_r\ \nu \Rightarrow$  expand
      ( $n(\ new := new\ n,$   $old := \{\varphi\} \cup old\ n,$   $next := insert\ \nu\ (next\ n) );$ ,  $ns$ )
    |  $\mu$  or  $\nu \Rightarrow$  expand2 expand  $n\ ns\ \varphi\ \mu\ \{\nu\}$ 
    |  $\mu$  U  $\nu \Rightarrow$  expand2 expand  $n\ ns\ \varphi\ \mu\ \{\varphi\}\ \{\nu\}$ 
    |  $\mu$  R  $\nu \Rightarrow$  expand2 expand  $n\ ns\ \varphi\ \nu\ \{\varphi\}\ \{\mu, \nu\}$ 
  }
)

```

lemma *expand-aimpl-refine*:

```

fixes  $n\ ns :: ('a\ node \times -)$ 
defines  $R \equiv Id \cap \{(-, (n, ns)). \forall n' \in ns. n > name\ n'\}$ 
defines  $R' \equiv Id \cap \{(-, (n, ns)). \forall n' \in ns. name\ n > name\ n'\}$ 
assumes [refine]:  $(n\ ns', n\ ns) \in R'$ 
shows  $expand\ aimpl\ n\ ns' \leq \Downarrow R (expand_T\ n\ ns)$ 
using [[goals-limit = 1]]

```

proof –

```

have [relator-props]: single-valued  $R$ 
by (auto simp add: R-def intro: single-valuedI)
have [relator-props]: single-valued  $R'$ 
by (auto simp add: R'-def intro: single-valuedI)

```

```

{
  fix  $n :: 'a\ node$  and  $ns$  and  $n'\ ns'$ 
  assume  $((n', ns'), (n, ns)) \in R'$ 
  then have  $(RETURN\ (name\ n', ns') \leq \Downarrow R (RETURN\ (name\ n, ns)))$ 
    by (auto simp: R-def R'-def pw-le-iff refine-pw-simps)
} note  $aux = this$ 

```

```

show ?thesis
  unfolding expand-aimpl-def expandT-def
  apply refine-rcg
  apply (simp add: R-def R'-def)
  apply (simp add: R-def R'-def)
  apply (auto simp add: R-def R'-def upd-incoming-def) []
  apply (auto simp add: R-def R'-def upd-incoming-def) []
  apply (auto simp add: R-def R'-def upd-incoming-def) []
  apply rprems
  apply (auto simp: R'-def expand-new-name-def) []
  apply (simp add: R'-def)

  apply (auto split: ltlr.split) []
  apply (fastforce simp: R-def R'-def) []
  apply (fastforce simp: R-def R'-def) []

  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (auto simp: R-def R'-def) []
  apply (refine-rcg, rprems, (fastforce simp: R-def R'-def)+) []
  apply (fastforce simp: R'-def) []
  apply (refine-rcg, rprems, (fastforce simp: R-def R'-def)+) []
  apply (refine-rcg, rprems, (fastforce simp: R-def R'-def)+) []
done
qed

```

```

thm create-graph-def
definition create-graph-aimpl  $\varphi = \text{do}$  {
  ( $\_, \text{nds}$ )  $\leftarrow$ 
  expand-aimpl
  ( $\backslash \text{name} = \text{expand-new-name}$   $\text{expand-init}$ ,  $\text{incoming} = \{\text{expand-init}\}$ ,
    $\text{new} = \{\varphi\}$ ,  $\text{old} = \{\}$ ,  $\text{next} = \{\}$ ),
   $\{\}$ );
  RETURN  $\text{nds}$ 
}

lemma create-graph-aimpl-refine:  $\text{create-graph-aimpl } \varphi \leq \Downarrow \text{Id } (\text{create-graph}_T \varphi)$ 
unfolding create-graph-aimpl-def create-graphT-def
apply (refine-rcg expand-aimpl-refine)
apply auto
done

```

3.2.2 Creation of GBA from Nodes

We summarize creation of the GBA and renaming of the nodes into one step

lemma *create-name-gba-alt*: *create-name-gba* $\varphi = do$ {
nds \leftarrow *create-graph_T* φ ;
ASSERT (*nds-invars* *nds*);
RETURN (*gba-rename-ext* (λ -. ()) *name* (*create-gba-from-nodes* φ *nds*))
}

proof –

have [*simp*]: $\bigwedge nds. g\text{-}V$ (*create-gba-from-nodes* φ *nds*) = *nds*
by (*auto simp: create-gba-from-nodes-def*)

show *?thesis*

unfolding *create-name-gba-def create-gba-def*
by *simp*

qed

In the following, we implement the components of the renamed GBA separately.

definition *build-succ* *nds* =

FOREACH
nds ($\lambda q' s.$
FOREACH
(*incoming* q') ($\lambda qn s.$
if *qn=expand-init* *then*
RETURN *s*
else
RETURN (*s*(*qn* \mapsto *insert* (*name* q') (*the-default* $\{\}$) (*s* *qn*))))
) *s*
) *Map.empty*

lemma *build-succ-aux1*:

assumes [*simp*]: *finite* *nds*

assumes [*simp*]: $\bigwedge q. q \in nds \implies finite$ (*incoming* q)

shows *build-succ* *nds* $\leq SPEC$ ($\lambda r. r = (\lambda qn.$

dflt-None-set $\{qn'. \exists q'.$

$q' \in nds \wedge qn' = name\ q' \wedge qn \in incoming\ q' \wedge qn \neq expand\text{-}init$

$\})$)

unfolding *build-succ-def*

apply (*refine-rcg refine-vcg*

FOREACH-rule[**where**

$I = \lambda it\ s. s = (\lambda qn. dflt\text{-}None\text{-}set\ \{qn'. \exists q'. q' \in nds\text{-}it \wedge qn' = name\ q' \wedge qn \in incoming\ q' \wedge qn \neq expand\text{-}init\ \})$)]

apply (*simp-all add: dflt-None-set-def*) [2]

apply (*rename-tac* q' *it* *s*)

apply (*rule-tac* $I = \lambda it2\ s. s =$

$(\lambda qn. \text{dflt-None-set } (\{qn'. \exists q'. q' \in \text{nds-it} \wedge qn' = \text{name } q' \wedge qn \in \text{incoming } q' \wedge qn \neq \text{expand-init}\})$
 \cup
 $\{qn'. qn' = \text{name } q' \wedge qn \in \text{incoming } q' - \text{it2} \wedge qn \neq \text{expand-init}\})$

in FOREACH-rule)

apply *auto* []

apply (*simp-all add: dflt-None-set-def*)

apply (*refine-rcg refine-vcg*)

apply (*auto simp: dflt-None-set-def intro!: ext*) []

apply (*rule ext, (auto) []*)+

done

lemma *build-succ-aux2*:

assumes *NINIT*: *expand-init* \notin *name'nds*

assumes *CL*: $\bigwedge nd. nd \in \text{nds} \implies \text{incoming } nd \subseteq \text{insert } \text{expand-init } (\text{name}'\text{nds})$

shows

$(\lambda qn. \text{dflt-None-set}$

$\{qn'. \exists q'. q' \in \text{nds} \wedge qn' = \text{name } q' \wedge qn \in \text{incoming } q' \wedge qn \neq \text{expand-init}\})$

$= (\lambda qn. \text{dflt-None-set } (\text{succ-of-E}$

$(\text{rename-E name } \{(q, q'). q \in \text{nds} \wedge q' \in \text{nds} \wedge \text{name } q \in \text{incoming } q'\}) qn))$

$(\text{is } (\lambda qn. \text{dflt-None-set } (?L qn)) = (\lambda qn. \text{dflt-None-set } (?R qn)))$

apply (*intro ext*)

apply (*fo-rule arg-cong*)

proof (*intro ext equalityI subsetI*)

fix *qn x*

assume $x \in ?R qn$

then show $x \in ?L qn$ **using** *NINIT*

by (*force simp: succ-of-E-def*)

next

fix *qn x*

assume *XL*: $x \in ?L qn$

show $x \in ?R qn$

proof (*cases qn = expand-init*)

case *False*

from *XL* **obtain** *q'* **where**

A: $q' \in \text{nds} \wedge qn \in \text{incoming } q'$

and [*simp*]: $x = \text{name } q'$

by *auto*

from *False* **obtain** *q* **where** *B*: $q \in \text{nds}$ **and** [*simp*]: $qn = \text{name } q$

using *CL A* **by** *auto*

from *A B* **show** $x \in ?R qn$

by (*auto simp: succ-of-E-def image-def*)

next

case [*simp*]: *True*

from XL show $x \in ?R \text{ } qn$ by *simp*
 qed
 qed

lemma *build-succ-correct*:

assumes $NINIT$: $expand\text{-}init \notin name\text{'}nds$

assumes FIN : *finite nds*

assumes CL : $\bigwedge nd. nd \in nds \implies incoming \text{ } nd \subseteq insert \text{ } expand\text{-}init \text{ } (name\text{'}nds)$

shows $build\text{-}succ \text{ } nds \leq SPEC \text{ } (\lambda r.$

$E\text{-of}\text{-succ} \text{ } (\lambda qn. the\text{-}default \text{ } \{\} \text{ } (r \text{ } qn))$

$= rename\text{-}E \text{ } (\lambda u. name \text{ } u) \text{ } \{(q, q'). q \in nds \wedge q' \in nds \wedge name \text{ } q \in incoming \text{ } q'\}$)

proof –

from $FIN \text{ } CL$ have FIN' : $\bigwedge q. q \in nds \implies finite \text{ } (incoming \text{ } q)$

by (*metis finite-imageI finite-insert infinite-super*)

note *build-succ-aux1* [*OF FIN FIN'*]

also note *build-succ-aux2* [*OF NINIT CL*]

finally show *?thesis*

by (*rule order-trans*) *auto*

qed

primrec *until-frmlsr* :: $'a \text{ } frml \implies ('a \text{ } frml \times 'a \text{ } frml)$ set **where**

$until\text{-frmlsr} \text{ } (\mu \text{ } and_r \text{ } \psi) = (until\text{-frmlsr} \text{ } \mu) \cup (until\text{-frmlsr} \text{ } \psi)$

| $until\text{-frmlsr} \text{ } (X_r \text{ } \mu) = until\text{-frmlsr} \text{ } \mu$

| $until\text{-frmlsr} \text{ } (\mu \text{ } U_r \text{ } \psi) = insert \text{ } (\mu, \psi) ((until\text{-frmlsr} \text{ } \mu) \cup (until\text{-frmlsr} \text{ } \psi))$

| $until\text{-frmlsr} \text{ } (\mu \text{ } R_r \text{ } \psi) = (until\text{-frmlsr} \text{ } \mu) \cup (until\text{-frmlsr} \text{ } \psi)$

| $until\text{-frmlsr} \text{ } (\mu \text{ } or_r \text{ } \psi) = (until\text{-frmlsr} \text{ } \mu) \cup (until\text{-frmlsr} \text{ } \psi)$

| $until\text{-frmlsr} \text{ } (true_r) = \{\}$

| $until\text{-frmlsr} \text{ } (false_r) = \{\}$

| $until\text{-frmlsr} \text{ } (prop_r \text{ } (-)) = \{\}$

| $until\text{-frmlsr} \text{ } (nprop_r \text{ } (-)) = \{\}$

lemma *until-frmlsr-correct*:

$until\text{-frmlsr} \text{ } \varphi = \{(\mu, \eta). Until\text{-ltlr} \text{ } \mu \text{ } \eta \in subfrmlsr \text{ } \varphi\}$

by (*induct* φ) *auto*

definition *build-F nds* φ

$\equiv (\lambda(\mu, \eta). name \text{ } ' \{q \in nds. (Until\text{-ltlr} \text{ } \mu \text{ } \eta \in old \text{ } q \longrightarrow \eta \in old \text{ } q)\}) \text{ } ' \text{ } until\text{-frmlsr} \text{ } \varphi$

lemma *build-F-correct*: $build\text{-F} \text{ } nds \text{ } \varphi =$

$\{name \text{ } ' \text{ } A \mid A. \exists \mu \text{ } \eta. A = \{q \in nds. Until\text{-ltlr} \text{ } \mu \text{ } \eta \in old \text{ } q \longrightarrow \eta \in old \text{ } q\} \wedge$
 $Until\text{-ltlr} \text{ } \mu \text{ } \eta \in subfrmlsr \text{ } \varphi\}$

proof –

have $\{name \text{ } ' \text{ } A \mid A. \exists \mu \text{ } \eta. A = \{q \in nds. Until\text{-ltlr} \text{ } \mu \text{ } \eta \in old \text{ } q \longrightarrow \eta \in old \text{ } q\} \wedge$

$Until\text{-}ltrl\ \mu\ \eta\ \in\ subfrmlsr\ \varphi\}$

$= (\lambda(\mu,\eta).\ name'\{q\in nds.\ Until\text{-}ltrl\ \mu\ \eta\ \in\ old\ q\ \longrightarrow\ \eta\ \in\ old\ q\})$
 $\quad '\{(\mu,\ \eta).\ Until\text{-}ltrl\ \mu\ \eta\ \in\ subfrmlsr\ \varphi\}$

by *auto*

also have $\dots = (\lambda(\mu,\eta).\ name'\{q\in nds.\ Until\text{-}ltrl\ \mu\ \eta\ \in\ old\ q\ \longrightarrow\ \eta\ \in\ old\ q\})$
 $\quad '\ until\text{-}frmlsr\ \varphi$

unfolding *until-frmlsr-correct ..*

finally show *?thesis*

unfolding *build-F-def by simp*

qed

definition *pn-props ps* $\equiv FOREACHi$

$(\lambda it\ (P,N).\ P = \{p.\ Prop\text{-}ltrl\ p\ \in\ ps - it\} \wedge N = \{p.\ Nprop\text{-}ltrl\ p\ \in\ ps - it\})$
 $ps\ (\lambda p\ (P,N).$
 $\quad case\ p\ of\ Prop\text{-}ltrl\ p\ \Rightarrow\ RETURN\ (insert\ p\ P,N)$
 $\quad | Nprop\text{-}ltrl\ p\ \Rightarrow\ RETURN\ (P,\ insert\ p\ N)$
 $\quad | -\ \Rightarrow\ RETURN\ (P,N)$
 $\quad)\ (\{\},\{\})$

lemma *pn-props-correct:*

assumes *[simp]: finite ps*

shows $pn\text{-}props\ ps \leq SPEC(\lambda r.\ r =$
 $(\{p.\ Prop\text{-}ltrl\ p\ \in\ ps\}, \{p.\ Nprop\text{-}ltrl\ p\ \in\ ps\}))$

unfolding *pn-props-def*

apply *(refine-rcg refine-vcg)*

apply *(auto split: ltlr.split)*

done

definition *pn-map nds* $\equiv FOREACH\ nds$

$(\lambda nd\ m.\ do\ \{$
 $\quad PN \leftarrow pn\text{-}props\ (old\ nd);$
 $\quad RETURN\ (m(name\ nd\ \mapsto\ PN))$
 $\quad \})\ Map.empty$

lemma *pn-map-correct:*

assumes *[simp]: finite nds*

assumes *FIN': $\bigwedge nd.\ nd \in nds \implies finite\ (old\ nd)$*

assumes *INJ: inj-on name nds*

shows $pn\text{-}map\ nds \leq SPEC\ (\lambda r.\ \forall\ qn.$
 $\quad case\ r\ qn\ of$
 $\quad None \Rightarrow qn \notin name'nds$
 $\quad | Some\ (P,N) \Rightarrow qn \in name'nds$
 $\quad \wedge\ P = \{p.\ Prop\text{-}ltrl\ p \in old\ (the\text{-}inv\text{-}into\ nds\ name\ qn)\}$
 $\quad \wedge\ N = \{p.\ Nprop\text{-}ltrl\ p \in old\ (the\text{-}inv\text{-}into\ nds\ name\ qn)\}$
 $\quad)$

unfolding *pn-map-def*

apply *(refine-rcg refine-vcg)*

```

FOREACH-rule[where  $I = \lambda it r. \forall qn.$ 
  case  $r$   $qn$  of
    None  $\Rightarrow qn \notin \text{name}'(nds - it)$ 
  | Some  $(P, N) \Rightarrow qn \in \text{name}'(nds - it)$ 
     $\wedge P = \{p. \text{Prop-ltlr } p \in \text{old}(\text{the-inv-into } nds \text{ name } qn)\}$ 
     $\wedge N = \{p. \text{Nprop-ltlr } p \in \text{old}(\text{the-inv-into } nds \text{ name } qn)\}$ 
  order-trans[OF pn-props-correct]
)
apply simp-all
apply (blast dest: subsetD[THEN FIN']) []
apply (force
  split: option.splits
  simp: the-inv-into-f-f[OF INJ] it-step-insert-iff) []
apply (fastforce split: option.splits) []
done

```

definition *cr-rename-gba* $nds \varphi \equiv \text{do } \{$
 let $V = \text{name}' nds$;
 let $V0 = \text{name}' \{q \in nds. \text{expand-init} \in \text{incoming } q\}$;
succmap $\leftarrow \text{build-succ } nds$;
 let $E = E\text{-of-succ}(\text{the-default } \{\} \text{ o } \text{i succmap})$;
 let $F = \text{build-F } nds \varphi$;
pnm $\leftarrow \text{pn-map } nds$;
 let $L = (\lambda qn l. \text{case } pnm \text{ } qn \text{ of}$
 None $\Rightarrow \text{False}$
 | Some $(P, N) \Rightarrow (\forall p \in P. p \in (l ::_r \langle Id \rangle \text{fun-set-rel})) \wedge (\forall p \in N. p \notin l)$
);
 RETURN ($(\lambda g-V = V, g-E = E, g-V0 = V0, gbg-F = F, gba-L = L)$)
 $\}$

lemma *cr-rename-gba-refine*:

assumes *INV*: *nds-invars* nds

assumes *REL[simplified]*: $(nds', nds) \in Id$ $(\varphi', \varphi) \in Id$

shows *cr-rename-gba* $nds' \varphi'$

$\leq \Downarrow Id$ (RETURN (*gba-rename-ext* $(\lambda-. ())$ name (*create-gba-from-nodes* φ nds)))

unfolding *RETURN-SPEC-conv*

proof (*rule Id-SPEC-refine*)

from *INV* **have**

NINIT: $\text{expand-init} \notin \text{name}' nds$

and *FIN*: *finite* nds

and *FIN'*: $\bigwedge nd. nd \in nds \implies \text{finite}(\text{old } nd)$

and *CL*: $\bigwedge nd. nd \in nds \implies \text{incoming } nd \subseteq \text{insert } \text{expand-init}(\text{name}' nds)$

and *INJ*: *inj-on* name nds

unfolding *nds-invars-def* **by** *auto*

show *cr-rename-gba* $nds' \varphi'$

$\leq \text{SPEC}(\lambda x. x = \text{gba-rename-ext}(\lambda-. ()) \text{ name}(\text{create-gba-from-nodes } \varphi \text{ } nds))$

unfolding *REL*

```

unfolding cr-rename-gba-def
apply (refine-rcg refine-vcg
  order-trans[OF build-succ-correct[OF NINIT FIN CL]]
  order-trans[OF pn-map-correct[OF FIN FIN' INJ]]
)
unfolding gba-rename-ecnv-def gb-rename-ecnv-def
  fr-rename-ext-def create-gba-from-nodes-def
apply simp
apply (intro conjI)
apply (simp add: comp-def)
apply (simp add: build-F-correct)
apply (intro ext)
apply (drule-tac x=qn in spec)
apply (auto simp: the-inv-into-ff[OF INJ] split: option.split prod.split)
done
qed

```

```

definition create-name-gba-aimpl  $\varphi \equiv$  do {
  nds  $\leftarrow$  create-graph-aimpl  $\varphi$ ;
  ASSERT (nds-invars nds);
  cr-rename-gba nds  $\varphi$ 
}

```

```

lemma create-name-gba-aimpl-refine:
  create-name-gba-aimpl  $\varphi \leq \Downarrow Id$  (create-name-gba  $\varphi$ )
unfolding create-name-gba-aimpl-def create-name-gba-alt
apply (refine-rcg create-graph-aimpl-refine cr-rename-gba-refine)
by auto

```

3.3 Refinement to Efficient Data Structures

3.3.1 Creation of GBA from Nodes

```

schematic-goal until-frmlsr-impl-aux:
  assumes [relator-props, simp]:  $R=Id$ 
  shows (?c,until-frmlsr)
   $\in \langle \langle R::(-\times-::linorder) \text{ set} \rangle \rangle \text{ltlr-rel} \rightarrow \langle \langle R \rangle \rangle \text{ltlr-rel} \times_r \langle \langle R \rangle \rangle \text{ltlr-rel} \text{dflt-rs-rel}$ 
unfolding until-frmlsr-def
apply (autoref (keep-goal, trace))
done
concrete-definition until-frmlsr-impl uses until-frmlsr-impl-aux
lemmas [autoref-rules] = until-frmlsr-impl.refine[OF PREFER-id-D]

```

```

schematic-goal build-succ-impl-aux:
  shows (?c,build-succ)  $\in$ 

```

```

  <<Rm,R>node-rel>list-set-rel
  → <<nat-rel,<nat-rel>list-set-rel>iam-map-rel>nres-rel
unfolding build-succ-def[abs-def] expand-init-def
using [[autoref-trace-failed-id]]
apply (autoref (keep-goal, trace))
done

```

concrete-definition *build-succ-impl* **uses** *build-succ-impl-aux*
lemmas [autoref-rules] = *build-succ-impl.refine*

schematic-goal *build-succ-code-aux*: *RETURN ?c ≤ build-succ-impl x*
unfolding *build-succ-impl-def*
apply (*refine-transfer* (*post*))
done

concrete-definition *build-succ-code* **uses** *build-succ-code-aux*
lemmas [*refine-transfer*] = *build-succ-code.refine*

schematic-goal *build-F-impl-aux*:
assumes [*relator-props*]: *R = Id*
shows (*?c,build-F*) ∈
 <<Rm,R>node-rel>list-set-rel → <R>ltr-rel → <<nat-rel>list-set-rel>list-set-rel
unfolding *build-F-def*[abs-def]
using [[autoref-trace-failed-id]]
apply (autoref (keep-goal, trace))
done

concrete-definition *build-F-impl* **uses** *build-F-impl-aux*
lemmas [autoref-rules] = *build-F-impl.refine*[OF *PREFER-id-D*]

schematic-goal *pn-map-impl-aux*:
shows (*?c,pn-map*) ∈
 <<Rm,Id>node-rel>list-set-rel
 → <<nat-rel,<Id>list-set-rel ×_r <Id>list-set-rel>iam-map-rel>nres-rel
unfolding *pn-map-def*[abs-def] *pn-props-def*
using [[autoref-trace-failed-id]]
apply (autoref (keep-goal, trace))
done

concrete-definition *pn-map-impl* **uses** *pn-map-impl-aux*
lemma *pn-map-impl-autoref*[autoref-rules]:
assumes *PREFER-id R*

shows $(pn\text{-map-impl}, pn\text{-map}) \in$
 $\langle\langle Rm, R \rangle node\text{-rel} \rangle list\text{-set-rel}$
 $\rightarrow \langle\langle nat\text{-rel}, \langle R \rangle list\text{-set-rel} \times_r \langle R \rangle list\text{-set-rel} \rangle iam\text{-map-rel} \rangle nres\text{-rel}$
using *assms pn-map-impl.refine by simp*

schematic-goal *pn-map-code-aux*: *RETURN ?c ≤ pn-map-impl x*
unfolding *pn-map-impl-def*
apply (*refine-transfer (post)*)
done
concrete-definition *pn-map-code* **uses** *pn-map-code-aux*
lemmas [*refine-transfer*] = *pn-map-code.refine*

schematic-goal *cr-rename-gba-impl-aux*:
assumes *ID[relator-props]: R=Id*
notes [*autoref-tyrel del*] = *tyrel-dflt-linorder-set*
notes [*autoref-tyrel*] = *ty-REL[of ⟨nat-rel⟩list-set-rel]*
shows $(?c, cr\text{-rename-gba}) \in$
 $\langle\langle Rm, R \rangle node\text{-rel} \rangle list\text{-set-rel} \rightarrow \langle R \rangle ltlr\text{-rel} \rightarrow (?R::(?'c \times -) set)$
unfolding *ID*
unfolding *cr-rename-gba-def[abs-def] expand-init-def comp-def*
using [*autoref-trace-failed-id*]
apply (*autoref (keep-goal, trace)*)
done
concrete-definition *cr-rename-gba-impl* **uses** *cr-rename-gba-impl-aux*

thm *cr-rename-gba-impl.refine*

lemma *cr-rename-gba-autoref[autoref-rules]*:
assumes *PREFER-id R*
shows $(cr\text{-rename-gba-impl}, cr\text{-rename-gba}) \in$
 $\langle\langle Rm, R \rangle node\text{-rel} \rangle list\text{-set-rel} \rightarrow \langle R \rangle ltlr\text{-rel} \rightarrow$
 $\langle gbav\text{-impl-rel-ext unit-rel nat-rel} (\langle R \rangle fun\text{-set-rel}) \rangle nres\text{-rel}$
using *assms cr-rename-gba-impl.refine[of R Rm] by simp*

schematic-goal *cr-rename-gba-code-aux*: *RETURN ?c ≤ cr-rename-gba-impl x y*
unfolding *cr-rename-gba-impl-def*
apply (*refine-transfer (post)*)
done
concrete-definition *cr-rename-gba-code* **uses** *cr-rename-gba-code-aux*
lemmas [*refine-transfer*] = *cr-rename-gba-code.refine*

3.3.2 Creation of Graph

The implementation of the node-set. The relation enforces that there are no different nodes with the same name. This effectively establishes an additional invariant, made explicit by an assertion in the refined program. This invariant allows for a more efficient implementation.

definition *ls-nds-rel-def-internal*:

ls-nds-rel $R \equiv \langle R \rangle \text{list-set-rel} \cap \{(-,s). \text{inj-on name } s\}$

lemma *ls-nds-rel-def*: $\langle R \rangle \text{ls-nds-rel} = \langle R \rangle \text{list-set-rel} \cap \{(-,s). \text{inj-on name } s\}$
by (*simp add: relAPP-def ls-nds-rel-def-internal*)

lemmas [*autoref-rel-intf*] = *REL-INTFI*[of *ls-nds-rel i-set*]

lemma *ls-nds-rel-sv*[*relator-props*]:

assumes *single-valued* R

shows *single-valued* $(\langle R \rangle \text{ls-nds-rel})$

using *list-set-rel-sv*[*OF assms*]

unfolding *ls-nds-rel-def*

by (*metis inf.cobounded1 single-valued-subset*)

context begin interpretation *autoref-syn* .

lemma *lsnds-empty-autoref*[*autoref-rules*]:

assumes *PREFER-id* R

shows $([],\{\}) \in \langle R \rangle \text{ls-nds-rel}$

using *assms*

apply (*simp add: ls-nds-rel-def*)

by *autoref*

lemma *lsnds-insert-autoref*[*autoref-rules*]:

assumes *SIDE-PRECOND* (*name* $n \notin \text{name}'ns$)

assumes $(n',n) \in R$

assumes $(ns',ns) \in \langle R \rangle \text{ls-nds-rel}$

shows $(n' \# ns', (OP \text{ insert} :: R \rightarrow \langle R \rangle \text{ls-nds-rel} \rightarrow \langle R \rangle \text{ls-nds-rel}) \$ n \$ ns)$
 $\in \langle R \rangle \text{ls-nds-rel}$

using *assms*

unfolding *ls-nds-rel-def*

apply *simp*

proof (*elim conjE, rule conjI*)

assume [*autoref-rules*]: $(n', n) \in R$ $(ns', ns) \in \langle R \rangle \text{list-set-rel}$

assume *name* $n \notin \text{name}' ns$

and *inj-on name* ns

then have $n \notin ns$ by (*auto*)

then show $(n' \# ns', \text{insert } n \ ns) \in \langle R \rangle \text{list-set-rel}$

by *autoref*

qed *auto*

lemma *ls-nds-image-autoref-aux*:

assumes [*autoref-rules*]: $(f_i, f) \in Ra \rightarrow Rb$

assumes $(l, s) \in \langle Ra \rangle \text{ls-nds-rel}$

assumes [*simp*]: $\forall x. \text{name } (f x) = \text{name } x$

shows $(\text{map } f_i \ l, f's) \in \langle Rb \rangle \text{ls-nds-rel}$

proof –

from *assms* have

[*autoref-rules*]: $(l, s) \in \langle Ra \rangle \text{list-set-rel}$

and *INJ*: (*inj-on name* s)


```

    by (auto simp add: ls-nds-rel-def)

have [simp]: inj-on f s
  by (rule inj-onI) (metis INJ assms(3) inj-on-eq-iff)

have (map fi l, f's) ∈ ⟨Rb⟩list-set-rel
  by (autoref (keep-goal))
moreover have inj-on name (f's)
  apply (rule inj-onI)
  apply (auto simp: image-iff dest: inj-onD[OF INJ])
  done
ultimately show ?thesis
  by (auto simp: ls-nds-rel-def)
qed

lemma ls-nds-image-autoref[autoref-rules]:
  assumes (fi,f) ∈ Ra → Rb
  assumes SIDE-PRECOND (∀ x. name (f x) = name x)
  shows (map fi, (OP (·) :: (Ra→Rb) → ⟨Ra⟩ls-nds-rel → ⟨Rb⟩ls-nds-rel)$f)
    ∈ ⟨Ra⟩ls-nds-rel → ⟨Rb⟩ls-nds-rel
  using assms
  unfolding autoref-tag-defs
  using ls-nds-image-autoref-aux
  by blast

lemma list-set-autoref-to-list[autoref-ga-rules]:
  shows is-set-to-sorted-list (λ-. True) R ls-nds-rel id
  unfolding is-set-to-list-def is-set-to-sorted-list-def ls-nds-rel-def
    it-to-sorted-list-def list-set-rel-def br-def
  by auto

end

context begin interpretation autoref-syn .
lemma [autoref-itype]:
  upd-incoming
  :: ⟨Im, I⟩ii-node →i ⟨⟨Im', I⟩ii-node⟩ii-set →i ⟨⟨Im', I⟩ii-node⟩ii-set
  by simp
end

term upd-incoming
schematic-goal upd-incoming-impl-aux:
  assumes REL-IS-ID R
  shows (?c, upd-incoming) ∈ ⟨Rm1, R⟩node-rel
  → ⟨⟨Rm2, R⟩node-rel⟩ls-nds-rel
  → ⟨⟨Rm2, R⟩node-rel⟩ls-nds-rel
  using assms apply simp
  unfolding upd-incoming-def[abs-def]

```

```

using [[autoref-trace-failed-id]]
apply (autoref (keep-goal))
done

```

concrete-definition *upd-incoming-impl* **uses** *upd-incoming-impl-aux*
lemmas [*autoref-rules*] = *upd-incoming-impl.refine*[*OF PREFER-D*[*of REL-IS-ID*]]

```

schematic-goal expand-impl-aux: (?c, expand-aimpl) ∈
  ⟨unit-rel, Id⟩node-rel ×r ⟨⟨unit-rel, Id⟩node-rel⟩ls-nds-rel
  → ⟨nat-rel ×r ⟨⟨unit-rel, Id⟩node-rel⟩ls-nds-rel⟩nres-rel
unfolding expand-aimpl-def[abs-def] expand-new-name-def
using [[autoref-trace-failed-id, autoref-trace-intf-unif]]
apply (autoref (trace, keep-goal))
done

```

concrete-definition *expand-impl* **uses** *expand-impl-aux*

context begin interpretation *autoref-syn* .

```

lemma [autoref-itype]: expandT
  ::i ⟨⟨i-unit, I⟩ii-node, ⟨⟨i-unit, I⟩ii-node⟩ii-set⟩ii-prod
  →i ⟨⟨i-nat, ⟨⟨i-unit, I⟩ii-node⟩ii-set⟩ii-prod⟩ii-nres by simp

```

lemma *expand-autoref*[*autoref-rules*]:

```

assumes ID: PREFER-id R
assumes A: (n-ns', n-ns)
  ∈ ⟨unit-rel, R⟩node-rel ×r ⟨⟨unit-rel, R⟩node-rel⟩list-set-rel
assumes B: SIDE-PRECOND (
  let (n, ns) = n-ns in inj-on name ns ∧ (∀ n' ∈ ns. name n > name n')
  )

```

```

shows (expand-impl n-ns',
  (OP expand-aimpl
  ::: ⟨unit-rel, R⟩node-rel ×r ⟨⟨unit-rel, R⟩node-rel⟩list-set-rel
  → ⟨nat-rel ×r ⟨⟨unit-rel, R⟩node-rel⟩list-set-rel⟩nres-rel)$n-ns)
  ∈ ⟨nat-rel ×r ⟨⟨unit-rel, R⟩node-rel⟩list-set-rel⟩nres-rel

```

proof *simp*

from *ID A B* **have**

1: (*n-ns*, *n-ns*) ∈ *Id* ∩ {(-, (*n, ns*))}. ∀ *n' ∈ ns*. *name n* > *name n'*}

and 2: (*n-ns'*, *n-ns*)

∈ ⟨*unit-rel*, *Id*⟩*node-rel* ×_{*r*} ⟨⟨*unit-rel*, *Id*⟩*node-rel*⟩*ls-nds-rel*

unfolding *ls-nds-rel-def*

apply –

apply *auto* []

apply (*cases* *n-ns'*)

apply *auto* []

done

have [*simp*]: *single-valued* (*nat-rel* ×_{*r*} ⟨⟨*unit-rel*, *Id*⟩*node-rel*⟩*list-set-rel*)
by *tagged-solver*

```

have [relator-props]:  $\bigwedge R. \text{single-valued } (Id \cap R)$ 
  by (metis IntE single-valuedD single-valuedI single-valued-Id)

have [simp]:  $\text{single-valued } ((\text{nat-rel} \times_r \langle \langle \text{unit-rel}, Id \rangle \text{node-rel} \rangle \text{ls-nds-rel}) \cap O$ 
   $(Id \cap \{(-, n, ns). \forall n' \in ns. \text{name } n' < n\}))$ 
  by (tagged-solver)

from expand-impl.refine[THEN fun-relD, OF 2, THEN nres-relD]
show (expand-impl n-ns', expand-aimpl n-ns)
   $\in \langle \text{nat-rel} \times_r \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel} \rangle \text{nres-rel}$ 
  apply -
  apply (rule nres-relI)
  using ID
  apply (simp add: pw-le-iff refine-pw-simps)
  apply (fastforce simp: ls-nds-rel-def)
  done
qed

end

schematic-goal expand-code-aux: RETURN ?c  $\leq$  expand-impl n-ns
  unfolding expand-impl-def
  by (refine-transfer the-resI)
concrete-definition expand-code uses expand-code-aux
prepare-code-thms expand-code-def
lemmas [refine-transfer] = expand-code.refine

schematic-goal create-graph-impl-aux:
  assumes ID:  $R=Id$ 
  shows (?c, create-graph-aimpl)
   $\in \langle R \rangle \text{ttr-rel} \rightarrow \langle \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel} \rangle \text{nres-rel}$ 
  unfolding ID
  unfolding create-graph-aimpl-def[abs-def] expand-init-def expand-new-name-def
  using [[autoref-trace-failed-id]]
  apply (autoref (keep-goal))
  done
concrete-definition create-graph-impl uses create-graph-impl-aux

lemmas [autoref-rules] = create-graph-impl.refine[OF PREFER-id-D]

schematic-goal create-graph-code-aux: RETURN ?c  $\leq$  create-graph-impl  $\varphi$ 
  unfolding create-graph-impl-def
  by refine-transfer
concrete-definition create-graph-code uses create-graph-code-aux
lemmas [refine-transfer] = create-graph-code.refine

```

schematic-goal *create-name-gba-impl-aux*:
 (?c, (create-name-gba-aimpl:: 'a::linorder ltlr \Rightarrow -))
 $\in \langle Id \rangle$ ltlr-rel \rightarrow (?R::(?'c \times -) set)
unfolding create-name-gba-aimpl-def[abs-def]
using [[autoref-trace-failed-id]]
apply (autoref (keep-goal, trace))
done
concrete-definition *create-name-gba-impl* **uses** *create-name-gba-impl-aux*

lemma *create-name-gba-autoref*[autoref-rules]:
assumes PREFER-id R
shows
 (create-name-gba-impl, create-name-gba)
 $\in \langle R \rangle$ ltlr-rel \rightarrow (gbav-impl-rel-ext unit-rel nat-rel ($\langle R \rangle$ fun-set-rel))nres-rel
 (is \cdot \rightarrow $\langle ?R \rangle$ nres-rel)
proof (intro fun-relI nres-relI)
fix $\varphi \varphi'$
assume A: $(\varphi, \varphi') \in \langle R \rangle$ ltlr-rel
from assms **have** RID[simp]: $R = Id$ **by** simp

note *create-name-gba-impl.refine*[THEN fun-relD, THEN nres-relD, OF A[unfolded RID]]
also note *create-name-gba-aimpl.refine*
finally show *create-name-gba-impl* $\varphi \leq \Downarrow ?R$ (*create-name-gba* φ') **by** simp
qed

schematic-goal *create-name-gba-code-aux*: RETURN ?c \leq *create-name-gba-impl*
 φ
unfolding create-name-gba-impl-def
by (refine-transfer (post))
concrete-definition *create-name-gba-code* **uses** *create-name-gba-code-aux*
lemmas [refine-transfer] = *create-name-gba-code.refine*

schematic-goal *create-name-igba-impl-aux*:
assumes RID: $R = Id$
shows (?c, create-name-igba) \in
 $\langle R \rangle$ ltlr-rel \rightarrow (igbav-impl-rel-ext unit-rel nat-rel ($\langle R \rangle$ fun-set-rel))nres-rel
unfolding RID
unfolding create-name-igba-def[abs-def]
using [[autoref-trace-failed-id]]
apply (autoref (trace, keep-goal))
done
concrete-definition *create-name-igba-impl* **uses** *create-name-igba-impl-aux*
lemmas [autoref-rules] = *create-name-igba-impl.refine*[OF PREFER-id-D]

schematic-goal *create-name-igba-code-aux*: RETURN ?c \leq *create-name-igba-impl*
 φ

```

unfolding create-name-igba-impl-def
by (refine-transfer (post))
concrete-definition create-name-igba-code uses create-name-igba-code-aux
lemmas [refine-transfer] = create-name-igba-code.refine

export-code create-name-igba-code checking SML

end

```

References

- [1] J. Esparza, P. Lammich, R. Neumann, T. Nipkow, A. Schimpf, and J.-G. Smaus. A fully verified executable ltl model checker. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 463–478. Springer Berlin Heidelberg, 2013.
- [2] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, UK, 1996. Chapman & Hall, Ltd.
- [3] S. Merz. Stuttering equivalence. *Archive of Formal Proofs*, May 2012. http://isa-afp.org/entries/Stuttering_Equivalence.shtml, Formal proof development.