

# Converting Linear-Time Temporal Logic to Generalized Büchi Automata

Alexander Schimpf and Peter Lammich

April 20, 2020

## 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

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>LTL to GBA translation</b>	<b>3</b>
2.1	Statistics . . . . .	3
2.2	Preliminaries . . . . .	4
2.3	Creation of States . . . . .	5
2.4	Creation of GBA . . . . .	14
<b>3</b>	<b>Refinement to Efficient Code</b>	<b>22</b>
3.1	Parametricity Setup Boilerplate . . . . .	22
3.1.1	LTL Formulas . . . . .	22
3.1.2	Nodes . . . . .	26
3.2	Massaging the Abstract Algorithm . . . . .	28
3.2.1	Creation of the Nodes . . . . .	28
3.2.2	Creation of GBA from Nodes . . . . .	30
3.3	Refinement to Efficient Data Structures . . . . .	33
3.3.1	Creation of GBA from Nodes . . . . .	33
3.3.2	Creation of Graph . . . . .	35

# 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) (
  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

    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
  ⟨proof⟩

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 ⟨proof⟩

```

## 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)
⟨proof⟩

```

**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')
⟨proof⟩

```

**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')
⟨proof⟩

```

**lemma** SPEC-rule-weak-nested2: [[f ≤ SPEC Q; f ≤ SPEC P;

```

  ∧r1 r2. [[Q (r1, r2); P (r1, r2)]] ⇒ g (r1, r2) ≤ SPEC P]]
⇒ f ≤ SPEC (λr'. g r' ≤ SPEC P)
⟨proof⟩

```

## 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 *ltrl*

**type-synonym** 'a *interp* = '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* ( $\mu$  *and<sub>r</sub>*  $\psi$ ) =  $\{\mu, \psi\}$   
| *new1* ( $\mu$  *U<sub>r</sub>*  $\psi$ ) =  $\{\mu\}$   
| *new1* ( $\mu$  *R<sub>r</sub>*  $\psi$ ) =  $\{\psi\}$   
| *new1* ( $\mu$  *or<sub>r</sub>*  $\psi$ ) =  $\{\mu\}$   
| *new1* - =  $\{\}$

**fun** *next1* **where**  
  *next1* (*X<sub>r</sub>*  $\psi$ ) =  $\{\psi\}$   
| *next1* ( $\mu$  *U<sub>r</sub>*  $\psi$ ) =  $\{\mu$  *U<sub>r</sub>*  $\psi\}$   
| *next1* ( $\mu$  *R<sub>r</sub>*  $\psi$ ) =  $\{\mu$  *R<sub>r</sub>*  $\psi\}$   
| *next1* - =  $\{\}$

**fun** *new2* **where**  
  *new2* ( $\mu$  *U<sub>r</sub>*  $\psi$ ) =  $\{\psi\}$   
| *new2* ( $\mu$  *R<sub>r</sub>*  $\psi$ ) =  $\{\mu, \psi\}$   
| *new2* ( $\mu$  *or<sub>r</sub>*  $\psi$ ) =  $\{\psi\}$   
| *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*  
  <*proof*>

**lemma** *expand-new-name-step*[*intro*]:

**fixes**  $n :: 'a \text{ node}$   
**shows**  $\text{name } n < \text{expand-new-name } (\text{name } n)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{expand-new-name--less-zero}[\text{intro}]$ :  $0 < \text{expand-new-name } nm$   
 $\langle \text{proof} \rangle$

**abbreviation**

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

**definition**

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

**lemma**  $\text{upd-incoming--elem}$ :

**assumes**  $nd \in \text{upd-incoming } n \text{ ns}$

**shows**  $nd \in \text{ns}$

$\vee (\exists nd' \in \text{ns}. nd = nd'(\text{incoming} := \text{incoming } n \cup \text{incoming } nd') \wedge$   
 $\text{old } nd' = \text{old } n \wedge$   
 $\text{next } nd' = \text{next } n)$

$\langle \text{proof} \rangle$

**lemma**  $\text{upd-incoming--ident-node}$ :

**assumes**  $nd \in \text{upd-incoming } n \text{ ns}$  **and**  $nd \in \text{ns}$

**shows**  $\text{incoming } n \subseteq \text{incoming } nd \vee \neg (\text{old } nd = \text{old } n \wedge \text{next } nd = \text{next } n)$

$\langle \text{proof} \rangle$

**lemma**  $\text{upd-incoming--ident}$ :

**assumes**  $\forall n \in \text{ns}. P \ n$

**and**  $\bigwedge n. \llbracket n \in \text{ns}; P \ n \rrbracket \implies (\bigwedge v. P \ (n(\text{incoming} := v)))$

**shows**  $\forall n \in \text{upd-incoming } n \ \text{ns}. P \ n$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma**  $\text{name-upd-incoming}[\text{simp}]$ :

$\text{name } ' (\text{upd-incoming } n \ \text{ns}) = \text{name } ' \ \text{ns}$  (**is**  $?lhs = ?rhs$ )

$\langle \text{proof} \rangle$

**abbreviation**  $\text{expand-body}$

**where**

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

```

if new n = {} then (
  if ( $\exists n' \in ns. \text{old } n' = \text{old } n \wedge \text{next } n' = \text{next } n$ ) 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 {
       $\varphi \leftarrow \text{SPEC } (\lambda x. x \in (\text{new } n))$ ;
      let n = n ( new := new n - { $\varphi$ } );
      if ( $\exists q. \varphi = \text{prop}_r(q) \vee \varphi = \text{nprop}_r(q)$ ) then
        (if ( $\text{not}_r \varphi \in \text{old } n$ ) then RETURN (name n, ns)
         else expand (n ( old := { $\varphi$ }  $\cup$  old n ), ns))
      else if  $\varphi = \text{true}_r$  then expand (n ( old := { $\varphi$ }  $\cup$  old n ), ns)
      else if  $\varphi = \text{false}_r$  then RETURN (name n, ns)
      else if ( $\exists \nu \mu. (\varphi = \nu \text{ and}_r \mu) \vee (\varphi = X_r \nu)$ ) 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*  $\langle$ proof $\rangle$

**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**  $I0$ :  $\Phi x$   
**assumes**  $IS$ :  $\bigwedge f x. \llbracket \bigwedge x. \Phi x \implies f x \leq M x; \Phi x; f \leq REC \text{ body} \rrbracket$   
 $\implies \text{body } f x \leq M x$   
**shows**  $REC \text{ body } x \leq M x$   
 $\langle \text{proof} \rangle$

**lemma** *expand-rec-rule*:  
**assumes**  $I0$ :  $\Phi x$   
**assumes**  $IS$ :  $\bigwedge f x. \llbracket \bigwedge x. f x \leq \text{expand } x; \bigwedge x. \Phi x \implies f x \leq M x; \Phi x \rrbracket$   
 $\implies \text{expand-body } f x \leq M x$   
**shows**  $\text{expand } x \leq M x$   
 $\langle \text{proof} \rangle$

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

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

**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  $\xi$  n-ns*  
 $\equiv \{\eta. \exists \mu. \mu \ U_r \ \eta \in \text{old } (fst \ n\text{-ns}) \wedge \xi \models_r \eta\} \subseteq \text{new } (fst \ n\text{-ns}) \cup \text{old } (fst \ n\text{-ns})$   
 $\wedge (\forall \psi \in \text{new } (fst \ n\text{-ns}). \xi \models_r \psi)$   
 $\wedge (\forall \psi \in \text{old } (fst \ n\text{-ns}). \xi \models_r \psi)$   
 $\wedge (\forall \psi \in \text{next } (fst \ n\text{-ns}). \xi \models_r X_r \ \psi)$

**abbreviation**  
*expand-rslt-exist--node-prop  $\xi$  n nd*  
 $\equiv \text{incoming } n \subseteq \text{incoming } nd$



$$\begin{aligned} & \wedge (\forall \psi \in \text{old } nd. \xi \models_r \psi) \wedge (\forall \psi \in \text{next } nd. \xi \models_r X_r \psi) \\ & \wedge \{\eta. \exists \mu. \mu U_r \eta \in \text{old } nd \wedge \xi \models_r \eta\} \subseteq \text{old } nd \end{aligned}$$

**abbreviation**

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

**abbreviation**

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

**abbreviation**

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

**lemma** *expand-name-propag*:

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

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

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

*<proof>*

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

**lemmas** *expand-name-propag--name* =

$$\text{SPEC-rule-conjunct1}[\text{OF SPEC-rule-conjunct2}[\text{OF } \text{expand-name-propag}]]$$

**lemmas** *expand-name-propag--name-ident* =

$$\text{SPEC-rule-conjunct2}[\text{OF SPEC-rule-conjunct2}[\text{OF } \text{expand-name-propag}]]$$

**lemma** *expand-rslt-exist-eq*:

$$\text{shows } \text{expand } n\text{-ns} \leq \text{SPEC} (\text{expand-rslt-exist-eq } n\text{-ns})$$

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

*<proof>*

**lemma** *expand-prop-exist*:

$$\text{expand } n\text{-ns} \leq \text{SPEC} (\lambda r. \text{expand-asm-exist } \xi \text{ } n\text{-ns} \longrightarrow \text{expand-rslt-exist } \xi \text{ } n\text{-ns} \text{ } r)$$

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

*<proof>*

Termination proof

**definition**  $\text{expand}_T :: ('a \text{ node} \times ('a \text{ node set})) \Rightarrow (\text{node-name} \times 'a \text{ node set}) \text{ nres}$

**where**  $\text{expand}_T \text{ } n\text{-ns} \equiv \text{REC}_T \text{ } \text{expand-body } n\text{-ns}$

**abbreviation** *old-next-pair*  $n \equiv (\text{old } n, \text{next } n)$

**abbreviation** *old-next-limit*  $\varphi \equiv \text{Pow}(\text{subfrmlsr } \varphi) \times \text{Pow}(\text{subfrmlsr } \varphi)$

**lemma** *old-next-limit-finite*:  $\text{finite}(\text{old-next-limit } \varphi)$   
*<proof>*

**definition**

*expand-ord*  $\varphi \equiv$   
 $\text{inv-image}(\text{finite-psupset}(\text{old-next-limit } \varphi) \text{ <*\text{lex*}> \text{less-than})$   
 $(\lambda(n, ns). (\text{old-next-pair } 'ns, \text{size-set } (\text{new } n)))$

**lemma** *expand-ord-wf[simp]*:  $\text{wf}(\text{expand-ord } \varphi)$   
*<proof>*

**abbreviation**

*expand-inv-node*  $\varphi n$   
 $\equiv \text{finite}(\text{new } n) \wedge \text{finite}(\text{old } n) \wedge \text{finite}(\text{next } n)$   
 $\wedge (\text{new } n) \cup (\text{old } n) \cup (\text{next } n) \subseteq \text{subfrmlsr } \varphi$

**abbreviation**

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

**definition**

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

**lemma** *new1-less-sum*:

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

**lemma** *new2-less-sum*:

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

**lemma** *new1-finite[intro]*:  $\text{finite}(\text{new1 } \psi)$   
*<proof>*

**lemma** *new1-subset-frmls*:  $\varphi \in \text{new1 } \psi \implies \varphi \in \text{subfrmlsr } \psi$   
*<proof>*

**lemma** *new2-finite[intro]*:  $\text{finite}(\text{new2 } \psi)$   
*<proof>*

**lemma** *new2-subset-frmls*:  $\varphi \in \text{new2 } \psi \implies \varphi \in \text{subfrmlsr } \psi$   
*<proof>*

**lemma** *next1-finite[intro]*:  $\text{finite}(\text{next1 } \psi)$   
*<proof>*

**lemma** *next1-subset-frmls*:  $\varphi \in \text{next1 } \psi \implies \varphi \in \text{subfrmlsr } \psi$

*<proof>*

**lemma** *expand-inv-impl[intro!]*:

**assumes** *expand-inv*  $\varphi$  ( $n$ ,  $ns$ )

**and** *newn*:  $\psi \in \text{new } n$

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

**and** *expand-inv-result*  $\varphi$   $ns'$

**and** ( $n' = n \langle \text{new} := \text{new } n - \{\psi\}$ ,

$\text{old} := \{\psi\} \cup \text{old } n \rangle \vee$

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

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

$\text{next} := \text{next1 } \psi \cup \text{next } n \rangle \vee$

( $n' = n \langle \text{name} := nm$ ,

$\text{new} := \text{new2 } \psi \cup (\text{new } n - \{\psi\})$ ,

$\text{old} := \{\psi\} \cup \text{old } n \rangle$ )

(**is**  $?case1 \vee ?case2 \vee ?case3$ )

**shows**  $((n', ns'), (n, ns)) \in \text{expand-ord } \varphi \wedge \text{expand-inv } \varphi (n', ns')$

(**is**  $?concl1 \wedge ?concl2$ )

*<proof>*

**lemma** *expand-term-prop-help*:

**assumes**  $((n', ns'), (n, ns)) \in \text{expand-ord } \varphi \wedge \text{expand-inv } \varphi (n', ns')$

**and** *assm-rule*:  $\llbracket \text{expand-inv } \varphi (n', ns'); ((n', ns'), n, ns) \in \text{expand-ord } \varphi \rrbracket$

$\implies f (n', ns') \leq \text{SPEC } P$

**shows**  $f (n', ns') \leq \text{SPEC } P$

*<proof>*

**lemma** *expand-inv-upd-incoming*:

**assumes** *expand-inv*  $\varphi$  ( $n$ ,  $ns$ )

**shows** *expand-inv-result*  $\varphi$  (*upd-incoming*  $n$   $ns$ )

*<proof>*

**lemma** *upd-incoming-eq-old-next-pair*: *old-next-pair* ' $ns = \text{old-next-pair } ( \text{upd-incoming } n \ ns )$

(**is**  $?A = ?B$ )

*<proof>*

**lemma** *expand-term-prop*:

*expand-inv*  $\varphi$   $n$ - $ns \implies$

$\text{expand}_T n$ - $ns \leq \text{SPEC } (\lambda(-, nds). \text{old-next-pair } snd \ n$ - $ns \subseteq \text{old-next-pair } nds$

$\wedge \text{expand-inv-result } \varphi \ nds)$

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

*<proof>*

**lemma** *expand-eq-expand<sub>T</sub>*:

**assumes** *inv*: *expand-inv*  $\varphi$   $n$ - $ns$

**shows**  $\text{expand}_T n$ - $ns = \text{expand } n$ - $ns$

*<proof>*

**lemma** *expand-nofail*:  
**assumes** *inv*: *expand-inv*  $\varphi$  *n-ns*  
**shows** *nofail* (*expand<sub>T</sub>* *n-ns*)  
*<proof>*

**lemma** [*intro!*]: *expand-inv*  $\varphi$  (  
 ()  
*name* = *expand-new-name* *expand-init*,  
*incoming* = {*expand-init*},  
*new* = { $\varphi$ },  
*old* = {},  
*next* = {} ),  
 {})  
*<proof>*

**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 = { $\varphi$ },
    old = {},
    next = {}
  )::'a node,
  {}::'a node set);
RETURN nds
}

```

**definition** *create-graph<sub>T</sub>* :: 'a *frml*  $\Rightarrow$  'a *node set nres*  
**where**

```

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

```

**lemma** *create-graph-eq-create-graph<sub>T</sub>*: *create-graph*  $\varphi = \text{create-graph}_T \varphi$   
 ⟨*proof*⟩

**lemma** *create-graph-finite*: *create-graph*  $\varphi \leq \text{SPEC finite}$   
 ⟨*proof*⟩

**lemma** *create-graph-nofail*: *nofail* (*create-graph*  $\varphi$ )  
 ⟨*proof*⟩

### abbreviation

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

**lemma** *L4-7*:  
**assumes**  $\xi \models_r \varphi$   
**shows** *create-graph*  $\varphi \leq \text{SPEC} (\text{create-graph-rslt-exist } \xi)$   
 ⟨*proof*⟩

**lemma** *expand-incoming-name-exist*:  
**assumes** *name* (*fst n-ns*)  $> \text{expand-init}$   
 $\wedge (\forall \text{ nm} \in \text{incoming} (\text{fst } n\text{-ns}). \text{nm} \neq \text{expand-init} \longrightarrow \text{nm} \in \text{name ' (snd } n\text{-ns)})$   
 $\wedge \text{expand-asm-incoming } n\text{-ns} \wedge \text{expand-name-ident} (\text{snd } n\text{-ns})$  (**is**  $?Q \text{ n-ns}$ )  
**and**  $\forall \text{ nd} \in \text{snd } n\text{-ns}.$   
 $\text{name nd} > \text{expand-init}$   
 $\wedge (\forall \text{ nm} \in \text{incoming nd}. \text{nm} \neq \text{expand-init} \longrightarrow \text{nm} \in \text{name ' (snd } n\text{-ns)})$   
 (**is**  $?P (\text{snd } n\text{-ns})$ )  
**shows** *expand n-ns*  $\leq \text{SPEC} (\lambda \text{ nm-nds}. ?P (\text{snd } \text{nm-nds}))$   
 ⟨*proof*⟩

**lemma** *create-graph--incoming-name-exist*:  
*create-graph*  $\varphi \leq \text{SPEC} (\lambda \text{ nds}. \forall \text{ nd} \in \text{nds}. \text{expand-init} < \text{name nd} \wedge (\forall \text{ nm} \in \text{incoming}$   
 $\text{nd}. \text{nm} \neq \text{expand-init} \longrightarrow \text{nm} \in \text{name ' nds}))$   
 ⟨*proof*⟩

### abbreviation

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

### abbreviation

*expand-rslt-all*  $\xi \text{ n-ns nm-nds} \equiv$

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

**lemma** *expand-prop-all*:

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

**shows**  $\text{expand } n\text{-ns} \leq \text{SPEC } (\text{expand-rslt-all } \xi \text{ } n\text{-ns})$

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

*<proof>*

**abbreviation**

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

**lemma** *L4-5*:  $\text{create-graph } \varphi \leq \text{SPEC } (\text{create-graph-rslt-all } \xi)$

*<proof>*

## 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* ::  $'a \text{ frml} \Rightarrow 'a \text{ node set} \Rightarrow ('a \text{ node}, 'a \text{ set})$   
*gba-rec*

**where** *create-gba-from-nodes*  $\varphi \text{ } qs \equiv \langle$

$g\text{-}V = qs,$

$g\text{-}E = \{(q, q'). q \in qs \wedge q' \in qs \wedge \text{name } q \in \text{incoming } q'\},$

$g\text{-}V0 = \{q \in qs. \text{expand-init} \in \text{incoming } q\},$

$gbg\text{-}F = \{\{q \in qs. \mu \text{ } U_r \eta \in \text{old } q \longrightarrow \eta \in \text{old } q\} \mid \mu \eta. \mu \text{ } U_r \eta \in \text{subfrmlsr } \varphi\},$

$gba\text{-}L = \lambda q \text{ } l. q \in qs \wedge \{p. \text{prop}_r(p) \in \text{old } q\} \subseteq l \wedge \{p. \text{nprop}_r(p) \in \text{old } q\} \cap l = \{\}$

$\rangle$

**end**

**locale** *create-gba-from-nodes-precond* =

**fixes**  $\varphi :: 'a \text{ ltlr}$

**fixes**  $qs :: 'a \text{ node set}$

**assumes**  $\text{res}: \text{inres } (\text{create-graph } \varphi) \text{ } qs$

**begin**

**lemma** *finite-qs[simp, intro!]*: *finite*  $qs$

*<proof>*

**lemma** *create-gba-from-nodes--invar*: *gba*  $(\text{create-gba-from-nodes } \varphi \text{ } qs)$

*<proof>*

**sublocale** *gba*: *gba create-gba-from-nodes*  $\varphi$  *qs*  
 ⟨*proof*⟩

**lemma** *create-gba-from-nodes--fin*: *finite* ( $g$ - $V$  (*create-gba-from-nodes*  $\varphi$  *qs*))  
 ⟨*proof*⟩

**lemma** *create-gba-from-nodes--ipath*:  
*ipath* *gba*. $E$   $r \longleftrightarrow (\forall i. r\ i \in qs \wedge name\ (r\ i) \in incoming\ (r\ (Suc\ i)))$   
 ⟨*proof*⟩

**lemma** *create-gba-from-nodes--is-run*:  
*gba.is-run*  $r \longleftrightarrow expand-init \in incoming\ (r\ 0)$   
 $\wedge (\forall i. r\ i \in qs \wedge name\ (r\ i) \in incoming\ (r\ (Suc\ i)))$   
 ⟨*proof*⟩

**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*. $E$   $\sigma$   
**shows**  $\sigma\ k \in qs \wedge name\ (\sigma\ k) \in incoming\ (\sigma\ (k+1))$   
 ⟨*proof*⟩

**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**  $expand\ n\ ns \leq SPEC\ (\lambda nm\ nds. \forall nd \in snd\ nm\ nds. false_r \notin old\ nd)$   
 ⟨*proof*⟩

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

$\langle \text{proof} \rangle$

**lemma** *expand-and-propag*:

**assumes**  $\mu$  and<sub>r</sub>  $\eta \in \text{old } (fst \ n\text{-}ns)$

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

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

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

**lemma** *expand-or-propag*:

**assumes**  $\mu$  or<sub>r</sub>  $\eta \in \text{old } (fst \ n\text{-}ns)$

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

**and**  $\forall nd \in \text{snd } n\text{-}ns. \mu$  or<sub>r</sub>  $\eta \in \text{old } nd \longrightarrow \{\mu, \eta\} \cap \text{old } nd \neq \{\}$

(**is**  $?P \ (\text{snd } n\text{-}ns)$ )

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

**abbreviation**

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

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

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

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

**abbreviation**

$\text{next-propag--rslt } \mu \ ns \equiv$

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

**lemma** *expand-next-propag*:

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

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

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

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

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

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

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

$\langle \text{proof} \rangle$



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

*create-graph*  $\varphi \leq \text{SPEC } (\lambda n ds. \forall n \in nds. \forall n' \in nds. X_r \mu \in \text{old } n \wedge \text{name } n \in \text{incoming } n' \longrightarrow \mu \in \text{old } n')$   
 ⟨proof⟩

**abbreviation**

*release-propag--assm*  $\mu \eta n\text{-ns} \equiv$   
 $(\mu R_r \eta \in \text{old } (fst \ n\text{-ns})$   
 $\longrightarrow \{\mu, \eta\} \subseteq \text{old } (fst \ n\text{-ns}) \cup \text{new } (fst \ n\text{-ns}) \vee$   
 $(\eta \in \text{old } (fst \ n\text{-ns}) \cup \text{new } (fst \ n\text{-ns})) \wedge \mu R_r \eta \in \text{next } (fst \ n\text{-ns}))$   
 $\wedge (\forall nd \in \text{snd } n\text{-ns}.$   
 $\mu R_r \eta \in \text{old } nd \wedge \text{name } nd \in \text{incoming } (fst \ n\text{-ns})$   
 $\longrightarrow \{\mu, \eta\} \subseteq \text{old } nd \vee$   
 $(\eta \in \text{old } nd \wedge \mu R_r \eta \in \text{old } (fst \ n\text{-ns}) \cup \text{new } (fst \ n\text{-ns})))$

**abbreviation**

*release-propag--rslt*  $\mu \eta ns \equiv$   
 $\forall nd \in ns.$   
 $\forall nd' \in ns.$   
 $\mu R_r \eta \in \text{old } nd \wedge \text{name } nd \in \text{incoming } nd'$   
 $\longrightarrow \{\mu, \eta\} \subseteq \text{old } nd \vee$   
 $(\eta \in \text{old } nd \wedge \mu R_r \eta \in \text{old } nd')$

**lemma** *expand-release-propag*:

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

**assumes** *release-propag--assm*  $\mu \eta n\text{-ns}$   
 $\wedge$  *release-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**  $\text{expand } n\text{-ns} \leq \text{SPEC } (\lambda r. \text{release-propag--rslt } \mu \eta (\text{snd } r))$   
 (**is**  $- \leq \text{SPEC } ?P$ )  
 ⟨proof⟩

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

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

**abbreviation**

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

**abbreviation**

*until-propag--rslt*  $f g ns \equiv$   
 $\forall n \in ns. \forall nd \in ns. f U_r g \in old n \wedge name n \in incoming nd$   
 $\longrightarrow (g \in old n \vee (f \in old n \wedge f U_r g \in old nd))$

**lemma** *expand-until-propag*:

**fixes**  $n-ns :: - \times 'a \text{ node set}$   
**assumes** *until-propag--assm*  $\mu \eta n-ns$   
 $\wedge$  *until-propag--rslt*  $\mu \eta (snd n-ns)$   
 $\wedge$  *expand-assm-incoming*  $n-ns$   
 $\wedge$  *expand-name-ident*  $(snd n-ns)$  (**is**  $?Q n-ns$ )  
**shows**  $expand n-ns \leq SPEC (\lambda r. \text{until-propag--rslt } \mu \eta (snd r))$   
(**is**  $- \leq SPEC ?P$ )  
 $\langle proof \rangle$

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

*create-graph*  $\varphi \leq SPEC (\lambda nds. \forall n \in nds. \forall n' \in nds. \mu U_r \eta \in old n \wedge name$   
 $n \in incoming n'$   
 $\longrightarrow (\eta \in old n \vee \mu \in old n \wedge \mu U_r \eta \in old n'))$   
 $\langle proof \rangle$

**definition** *all-subfrmls*  $:: 'a \text{ node} \Rightarrow 'a \text{ frml set}$ 

**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**  $y \in B$   
 $\langle proof \rangle$

**lemma** *expand-all-subfrmls-propag*:

**assumes**  $all-subfrmls (fst n-ns) \subseteq B \wedge (\forall nd \in snd n-ns. all-subfrmls nd \subseteq B)$  (**is**  
 $?Q n-ns$ )  
**shows**  $expand n-ns \leq SPEC (\lambda r. \forall nd \in snd r. all-subfrmls nd \subseteq B)$   
(**is**  $- \leq SPEC ?P$ )  
 $\langle proof \rangle$

**lemma** *old-propag-on-create-graph*: *create-graph*  $\varphi \leq SPEC (\lambda nds. \forall n \in nds. old$   
 $n \subseteq subfrmlsr \varphi)$ 

$\langle proof \rangle$

**lemma** *L4-2--aux*:

**assumes** *run*: *ipath*  $gba.E \sigma$   
**and**  $\mu U_r \eta \in old (\sigma 0)$   
**and**  $\forall j. (\forall i < j. \{\mu, \mu U_r \eta\} \subseteq old (\sigma i)) \longrightarrow \eta \notin old (\sigma j)$   
**shows**  $\forall i. \{\mu, \mu U_r \eta\} \subseteq old (\sigma i) \wedge \eta \notin old (\sigma i)$

*<proof>*

**lemma** *L4-2a*:

**assumes** *ipath gba.E*  $\sigma$   
**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>*

**lemma** *L4-2b*:

**assumes** *run: ipath gba.E*  $\sigma$   
**and**  $\mu U_r \eta \in \text{old } (\sigma 0)$   
**and** *ACC: gba.is-acc*  $\sigma$   
**shows**  $\exists j. (\forall i < j. \{\mu, \mu U_r \eta\} \subseteq \text{old } (\sigma i)) \wedge \eta \in \text{old } (\sigma j)$   
*<proof>*

**lemma** *L4-2c*:

**assumes** *run: ipath gba.E*  $\sigma$   
**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>*

**lemma** *L4-8'*:

**assumes** *ipath gba.E*  $\sigma$  **(is ?inf-run  $\sigma$ )**  
**and** *gba.is-acc*  $\sigma$  **(is ?gbarel-accept  $\sigma$ )**  
**and**  $\forall i. \text{gba.L } (\sigma i) (\xi i)$  **(is ?lgbarel-accept  $\xi \sigma$ )**  
**and**  $\psi \in \text{old } (\sigma 0)$   
**shows**  $\xi \models_r \psi$   
*<proof>*

**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**  $\xi \models_r \psi$   
*<proof>*

**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  $n\text{-ns}$ )**  
**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  $n\text{-ns}$ ))**  
**shows** *expand  $n\text{-ns}$   $\leq$  SPEC*  $(\lambda r. \text{name } n' \leq \text{fst } r \wedge ?P (\text{snd } r))$   
*<proof>*

**lemma** *expand-init-propag-on-create-graph*:  
*create-graph*  $\varphi \leq \text{SPEC}(\lambda nds. \forall nd \in nds. \text{expand-init} \in \text{incoming } nd \longrightarrow \varphi \in \text{old } nd)$   
 ⟨*proof*⟩

**lemma** *L4-6*:  
**assumes**  $q \in \text{gba}.V0$   
**shows**  $\varphi \in \text{old } q$   
 ⟨*proof*⟩

**lemma** *L4-9*:  
**assumes**  $\text{acc}: \text{gba}.accept \ \xi$   
**shows**  $\xi \models_r \varphi$   
 ⟨*proof*⟩

**lemma** *L4-10*:  
**assumes**  $\xi \models_r \varphi$   
**shows**  $\text{gba}.accept \ \xi$   
 ⟨*proof*⟩

**end**  
**end**

**lemma** *create-graph--name-ident*: *create-graph*  $\varphi \leq \text{SPEC} (\lambda nds. \forall q \in nds. \exists ! q' \in nds. \text{name } q = \text{name } q')$   
 ⟨*proof*⟩

**corollary** *create-graph--name-inj*: *create-graph*  $\varphi \leq \text{SPEC} (\lambda nds. \text{inj-on name } nds)$   
 ⟨*proof*⟩

**definition**  
*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** *create-graph-precond*: *create-graph*  $\varphi$   
 $\leq \text{SPEC} (\text{create-gba-from-nodes-precond } \varphi)$   
 ⟨*proof*⟩

**lemma** *create-gba--invar*: *create-gba*  $\varphi \leq \text{SPEC } \text{gba}$   
 ⟨*proof*⟩

**lemma** *create-gba-acc*:  
**shows** *create-gba*  $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. \forall \xi. \text{gba}.accept \ \mathcal{A} \ \xi \longleftrightarrow \xi \models_r \varphi)$   
 ⟨*proof*⟩

**lemma** *create-gba--name-inj*:  
**shows**  $\text{create-gba } \varphi \leq \text{SPEC}(\lambda \mathcal{A}. (\text{inj-on name } (g-V \ \mathcal{A})))$   
 ⟨proof⟩

**lemma** *create-gba--fin*:  
**shows**  $\text{create-gba } \varphi \leq \text{SPEC}(\lambda \mathcal{A}. (\text{finite } (g-V \ \mathcal{A})))$   
 ⟨proof⟩

**lemma** *create-graph-old-finite*:  
 $\text{create-graph } \varphi \leq \text{SPEC} (\lambda \text{nds}. \forall \text{nd} \in \text{nds}. \text{finite } (\text{old nd}))$   
 ⟨proof⟩

**lemma** *create-gba--old-fin*:  
**shows**  $\text{create-gba } \varphi \leq \text{SPEC}(\lambda \mathcal{A}. \forall \text{nd} \in g-V \ \mathcal{A}. \text{finite } (\text{old nd}))$   
 ⟨proof⟩

**lemma** *create-gba--incoming-exists*:  
**shows**  $\text{create-gba } \varphi$   
 $\leq \text{SPEC}(\lambda \mathcal{A}. \forall \text{nd} \in g-V \ \mathcal{A}. \text{incoming nd} \subseteq \text{insert expand-init } (\text{name } ' (g-V \ \mathcal{A})))$   
 ⟨proof⟩

**lemma** *create-gba--no-init*:  
**shows**  $\text{create-gba } \varphi \leq \text{SPEC}(\lambda \mathcal{A}. \text{expand-init} \notin \text{name } ' (g-V \ \mathcal{A}))$   
 ⟨proof⟩

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

**lemma** *create-gba-nds-invars*:  $\text{create-gba } \varphi \leq \text{SPEC} (\lambda \mathcal{A}. \text{nds-invars } (g-V \ \mathcal{A}))$   
 ⟨proof⟩

**theorem** *T4-1*:  
 $\text{create-gba } \varphi \leq \text{SPEC}(\$   
 $\quad \lambda \mathcal{A}. \text{gba } \mathcal{A}$   
 $\quad \wedge \text{finite } (g-V \ \mathcal{A})$   
 $\quad \wedge (\forall \xi. \text{gba.accept } \mathcal{A} \ \xi \longleftrightarrow \xi \models_r \varphi)$   
 $\quad \wedge (\text{nds-invars } (g-V \ \mathcal{A})))$   
 ⟨proof⟩

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

**theorem** *create-name-gba-correct*:

*create-name-gba*  $\varphi \leq \text{SPEC}(\lambda \mathcal{A}. \text{gba } \mathcal{A} \wedge \text{finite } (g\text{-}V \ \mathcal{A}) \wedge (\forall \xi. \text{gba.accept } \mathcal{A} \ \xi \longleftrightarrow \xi \models_r \varphi))$   
*<proof>*

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

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

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

$\text{igba } G \wedge \text{finite } (g\text{-}V \ G) \wedge (\forall \xi. \text{igba.accept } G \ \xi \longleftrightarrow \xi \models_r \varphi))$   
*<proof>*

**context**

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

**begin**

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

*<proof>*

**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**

$(\text{True-ltlr}, \text{True-ltlr}) \in \text{ltlr-rel } R$   
 $| (\text{False-ltlr}, \text{False-ltlr}) \in \text{ltlr-rel } R$   
 $| (a, a') \in R \implies (\text{Prop-ltlr } a, \text{Prop-ltlr } a') \in \text{ltlr-rel } R$   
 $| (a, a') \in R \implies (\text{Nprop-ltlr } a, \text{Nprop-ltlr } a') \in \text{ltlr-rel } R$   
 $| \llbracket (a, a') \in \text{ltlr-rel } R; (b, b') \in \text{ltlr-rel } R \rrbracket$   
 $\implies (\text{And-ltlr } a \ b, \text{And-ltlr } a' \ b') \in \text{ltlr-rel } R$   
 $| \llbracket (a, a') \in \text{ltlr-rel } R; (b, b') \in \text{ltlr-rel } R \rrbracket$   
 $\implies (\text{Or-ltlr } a \ b, \text{Or-ltlr } a' \ b') \in \text{ltlr-rel } R$   
 $| \llbracket (a, a') \in \text{ltlr-rel } R \rrbracket \implies (\text{Next-ltlr } a, \text{Next-ltlr } a') \in \text{ltlr-rel } R$   
 $| \llbracket (a, a') \in \text{ltlr-rel } R; (b, b') \in \text{ltlr-rel } R \rrbracket$   
 $\implies (\text{Until-ltlr } a \ b, \text{Until-ltlr } a' \ b') \in \text{ltlr-rel } R$   
 $| \llbracket (a, a') \in \text{ltlr-rel } R; (b, b') \in \text{ltlr-rel } R \rrbracket$   
 $\implies (\text{Release-ltlr } a \ b, \text{Release-ltlr } a' \ b') \in \text{ltlr-rel } R$

**lemmas** *ltlr-rel-induct*[*induct set*]  
 $= \text{ltlr-rel.induct}[\text{simplified relAPP-def}[\text{of ltlr-rel, symmetric}]]$   
**lemmas** *ltlr-rel-cases*[*cases set*]  
 $= \text{ltlr-rel.cases}[\text{simplified relAPP-def}[\text{of ltlr-rel, symmetric}]]$   
**lemmas** *ltlr-rel-intros*  
 $= \text{ltlr-rel.intros}[\text{simplified relAPP-def}[\text{of ltlr-rel, symmetric}]]$

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

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

**lemma** *ltlr-rel-sv*[*relator-props*]:  
**assumes** *SV*: *single-valued R*  
**shows** *single-valued*  $(\langle R \rangle \text{ltlr-rel})$   
 $\langle \text{proof} \rangle$

**lemma** *ltlr-rel-id*[*relator-props*]:  $\llbracket R = \text{Id} \rrbracket \implies \langle R \rangle \text{ltlr-rel} = \text{Id}$   
 $\langle \text{proof} \rangle$

**lemma** *ltlr-rel-id-simp*[*simp*]:  $\langle \text{Id} \rangle \text{ltlr-rel} = \text{Id}$   $\langle \text{proof} \rangle$

**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*]:  
 $(\text{True-ltlr}, \text{True-ltlr}) \in \langle R \rangle \text{ltlr-rel}$

$(False\text{-ltlr}, False\text{-ltlr}) \in \langle R \rangle \text{ltlr-rel}$   
 $(Prop\text{-ltlr}, Prop\text{-ltlr}) \in R \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $(Nprop\text{-ltlr}, Nprop\text{-ltlr}) \in R \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $(And\text{-ltlr}, And\text{-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $(Or\text{-ltlr}, Or\text{-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $(Next\text{-ltlr}, Next\text{-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $(Until\text{-ltlr}, Until\text{-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $(Release\text{-ltlr}, Release\text{-ltlr}) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel}$   
 $\langle \text{proof} \rangle$

**lemma** *case-ltlr-param*[*param, autoref-rules*]:

$(case\text{-ltlr}, case\text{-ltlr}) \in Rv \rightarrow Rv \rightarrow (R \rightarrow Rv)$   
 $\rightarrow (R \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv) \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv$   
 $\langle \text{proof} \rangle$

**lemma** *rec-ltlr-param*[*param, autoref-rules*]:

$(rec\text{-ltlr}, rec\text{-ltlr}) \in Rv \rightarrow Rv \rightarrow (R \rightarrow Rv)$   
 $\rightarrow (R \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$   
 $\rightarrow (\langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv \rightarrow Rv \rightarrow Rv)$   
 $\rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow Rv$   
 $\langle \text{proof} \rangle$

**lemma** *case-ltlr-mono*[*refine-mono*]:

**assumes**  $\varphi = True\text{-ltlr} \implies a \leq a'$   
**assumes**  $\varphi = False\text{-ltlr} \implies b \leq b'$   
**assumes**  $\bigwedge p. \varphi = Prop\text{-ltlr } p \implies c p \leq c' p$   
**assumes**  $\bigwedge p. \varphi = Nprop\text{-ltlr } p \implies d p \leq d' p$   
**assumes**  $\bigwedge \mu \nu. \varphi = And\text{-ltlr } \mu \nu \implies e \mu \nu \leq e' \mu \nu$   
**assumes**  $\bigwedge \mu \nu. \varphi = Or\text{-ltlr } \mu \nu \implies f \mu \nu \leq f' \mu \nu$   
**assumes**  $\bigwedge \mu. \varphi = Next\text{-ltlr } \mu \implies g \mu \leq g' \mu$   
**assumes**  $\bigwedge \mu \nu. \varphi = Until\text{-ltlr } \mu \nu \implies h \mu \nu \leq h' \mu \nu$   
**assumes**  $\bigwedge \mu \nu. \varphi = Release\text{-ltlr } \mu \nu \implies i \mu \nu \leq i' \mu \nu$   
**shows**  $case\text{-ltlr } a b c d e f g h i \varphi \leq case\text{-ltlr } a' b' c' d' e' f' g' h' i' \varphi$   
 $\langle \text{proof} \rangle$

**primrec** *ltlr-eq* **where**

$ltlr\text{-eq } eq \ True\text{-ltlr } f \iff (case \ f \ of \ True\text{-ltlr} \Rightarrow \ True \ | \ - \Rightarrow \ False)$   
 $| \ ltlr\text{-eq } eq \ False\text{-ltlr } f \iff (case \ f \ of \ False\text{-ltlr} \Rightarrow \ True \ | \ - \Rightarrow \ False)$   
 $| \ ltlr\text{-eq } eq \ (Prop\text{-ltlr } p) \ f \iff (case \ f \ of \ Prop\text{-ltlr } p' \Rightarrow \ eq \ p \ p' \ | \ - \Rightarrow \ False)$



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

**lemma** *ltlr-eq-autoref*[*autoref-rules*]:  
**assumes** *EQP*:  $(\text{eq}, (=)) \in R \rightarrow R \rightarrow \text{bool-rel}$   
**shows**  $(\text{ltlr-eq eq}, (=)) \in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \text{bool-rel}$   
*<proof>*

**lemma** *ltlr-dflt-cmp*[*autoref-rules-raw*]:  
**assumes** *PREFER-id* *R*  
**shows**  
 $(\text{dflt-cmp } (\leq) (<), \text{dflt-cmp } (\leq) (<))$   
 $\in \langle R \rangle \text{ltlr-rel} \rightarrow \langle R \rangle \text{ltlr-rel} \rightarrow \text{comp-res-rel}$   
*<proof>*

**type-synonym**  
 $\text{node-name-impl} = \text{node-name}$

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

**lemma** *case-ltlr-gtransfer*:  
**assumes**  
 $\gamma \ ai \leq a$   
 $\gamma \ bi \leq b$   
 $\bigwedge a. \gamma \ (ci \ a) \leq c \ a$   
 $\bigwedge a. \gamma \ (di \ a) \leq d \ a$   
 $\bigwedge \text{ltlr1 } \text{ltlr2}. \gamma \ (ei \ \text{ltlr1 } \ \text{ltlr2}) \leq e \ \text{ltlr1 } \ \text{ltlr2}$   
 $\bigwedge \text{ltlr1 } \ \text{ltlr2}. \gamma \ (fi \ \text{ltlr1 } \ \text{ltlr2}) \leq f \ \text{ltlr1 } \ \text{ltlr2}$   
 $\bigwedge \text{ltlr}. \gamma \ (gi \ \text{ltlr}) \leq g \ \text{ltlr}$   
 $\bigwedge \text{ltlr1 } \ \text{ltlr2}. \gamma \ (hi \ \text{ltlr1 } \ \text{ltlr2}) \leq h \ \text{ltlr1 } \ \text{ltlr2}$   
 $\bigwedge \text{ltlr1 } \ \text{ltlr2}. \gamma \ (iiv \ \text{ltlr1 } \ \text{ltlr2}) \leq i \ \text{ltlr1 } \ \text{ltlr2}$   
**shows**  $\gamma \ (\text{case-ltlr } ai \ bi \ ci \ di \ ei \ fi \ gi \ hi \ iiv \ \varphi)$   
 $\leq (\text{case-ltlr } a \ b \ c \ d \ e \ f \ g \ h \ i \ \varphi)$   
*<proof>*

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

**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**  $case-ltlr\ ai\ bi\ ci\ di\ ei\ fi\ gi\ hi\ iiv\ \varphi \neq dSUCCEED$   
*<proof>*

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

**lemma** *node-rel-def:  $\langle Re, R \rangle$  node-rel* =  $\{$   
 $($   
 $\quad$  *name-impl* = *namei*,  
 $\quad$  *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) ∈ node-name-rel
  ∧ (inci,inc) ∈ ⟨node-name-rel⟩ dflt-rs-rel
  ∧ (newi,new) ∈ ⟨⟨R⟩ ltlr-rel⟩ lss.rel
  ∧ (oldi,old) ∈ ⟨⟨R⟩ ltlr-rel⟩ lss.rel
  ∧ (nexti,next) ∈ ⟨⟨R⟩ ltlr-rel⟩ lss.rel
  ∧ (morei,more) ∈ Re
} ⟨proof⟩

```

**lemma** *node-rel-sv*[relator-props]:  
*single-valued Re*  $\implies$  *single-valued R*  $\implies$  *single-valued* ( $\langle Re, R \rangle$  node-rel)  
 ⟨proof⟩

**consts** *i-node* :: *interface*  $\Rightarrow$  *interface*  $\Rightarrow$  *interface*

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

**lemma** [*autoref-rules*]: (*node-impl-ext*, *node-ext*)  $\in$   
*node-name-rel*  
 $\rightarrow$   $\langle$ node-name-rel $\rangle$  dflt-rs-rel  
 $\rightarrow$   $\langle\langle R \rangle$  ltlr-rel $\rangle$  lss.rel  
 $\rightarrow$   $\langle\langle R \rangle$  ltlr-rel $\rangle$  lss.rel  
 $\rightarrow$   $\langle\langle R \rangle$  ltlr-rel $\rangle$  lss.rel  
 $\rightarrow$  *Re*  
 $\rightarrow$   $\langle Re, R \rangle$  node-rel  
 ⟨proof⟩

**lemma** [*autoref-rules*]:  
 (*node-impl.name-impl-update*, *node.name-update*)  
 $\in$  (*node-name-rel*  $\rightarrow$  *node-name-rel*)  $\rightarrow$   $\langle Re, R \rangle$  node-rel  $\rightarrow$   $\langle Re, R \rangle$  node-rel  
 (*node-impl.incoming-impl-update*, *node.incoming-update*)  
 $\in$  ( $\langle$ node-name-rel $\rangle$  dflt-rs-rel  $\rightarrow$   $\langle$ node-name-rel $\rangle$  dflt-rs-rel)  
 $\rightarrow$   $\langle Re, R \rangle$  node-rel  
 $\rightarrow$   $\langle Re, R \rangle$  node-rel  
 (*node-impl.new-impl-update*, *node.new-update*)  
 $\in$  ( $\langle\langle R \rangle$  ltlr-rel $\rangle$  lss.rel  $\rightarrow$   $\langle\langle R \rangle$  ltlr-rel $\rangle$  lss.rel)  $\rightarrow$   $\langle Re, R \rangle$  node-rel  $\rightarrow$   $\langle Re, R \rangle$  node-rel  
 (*node-impl.old-impl-update*, *node.old-update*)

$$\begin{aligned}
&\in (\langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel} \rightarrow \langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel}) \rightarrow \langle Re, R \rangle\text{node-rel} \rightarrow \langle Re, R \rangle\text{node-rel} \\
&\quad (\text{node-impl.next-impl-update}, \text{node.next-update}) \\
&\in (\langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel} \rightarrow \langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel}) \rightarrow \langle Re, R \rangle\text{node-rel} \rightarrow \langle Re, R \rangle\text{node-rel} \\
&\quad (\text{node-impl.more-update}, \text{node.more-update}) \\
&\in (Re \rightarrow Re) \rightarrow \langle Re, R \rangle\text{node-rel} \rightarrow \langle Re, R \rangle\text{node-rel} \\
&\quad \langle \text{proof} \rangle
\end{aligned}$$

**term** *name*

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

$$\begin{aligned}
&(\text{node-impl.name-impl}, \text{node.name}) \in \langle Re, R \rangle\text{node-rel} \rightarrow \text{node-name-rel} \\
&(\text{node-impl.incoming-impl}, \text{node.incoming}) \\
&\in \langle Re, R \rangle\text{node-rel} \rightarrow \langle \text{node-name-rel} \rangle \text{dflt-rs-rel} \\
&(\text{node-impl.new-impl}, \text{node.new}) \in \langle Re, R \rangle\text{node-rel} \rightarrow \langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel} \\
&(\text{node-impl.old-impl}, \text{node.old}) \in \langle Re, R \rangle\text{node-rel} \rightarrow \langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel} \\
&(\text{node-impl.next-impl}, \text{node.next}) \in \langle Re, R \rangle\text{node-rel} \rightarrow \langle\langle R \rangle\text{ltlr-rel}\rangle\text{lss.rel} \\
&(\text{node-impl.more}, \text{node.more}) \in \langle Re, R \rangle\text{node-rel} \rightarrow Re \\
&\quad \langle \text{proof} \rangle
\end{aligned}$$

## 3.2 Massaging 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*  $\varphi$  *n1* *nx1* *n2*  $\equiv$  *do* {  
 $(nm, nds) \leftarrow \text{exp} ($   
 $n(|$   
 $\quad \text{new} := \text{insert } n1 \text{ (new } n),$   
 $\quad \text{old} := \text{insert } \varphi \text{ (old } n),$   
 $\quad \text{next} := nx1 \cup \text{next } n |),$   
 $ns);$   
 $\text{exp} (n(| \text{ name} := nm, \text{ new} := n2 \cup \text{new } n, \text{ old} := \{\varphi\} \cup \text{old } n |), nds)$   
 $)$

**definition** *expand-aimpl*  $\equiv$   $REC_T (\lambda \text{expand} (n, ns).$

$$\begin{aligned}
&\text{if } \text{new } n = \{\} \text{ then (} \\
&\quad \text{if } (\exists n' \in ns. \text{old } n' = \text{old } n \wedge \text{next } n' = \text{next } n) \text{ then} \\
&\quad\quad \text{RETURN (name } n, \text{upd-incoming } n \text{ ns)} \\
&\quad \text{else do \{ } \\
&\quad\quad \text{ASSERT } (n \notin ns); \\
&\quad\quad \text{ASSERT } (\text{name } n \notin \text{name}'ns); \\
&\quad\quad \text{expand} (( \\
&\quad\quad\quad \text{name} = \text{expand-new-name (name } n),
\end{aligned}$$



```

    new = {φ}, old = {}, next = {}),
  {});
  RETURN nds
}

```

**lemma** *create-graph-aimpl-refine*:  $create-graph-aimpl\ \varphi \leq \Downarrow Id\ (create-graph_T\ \varphi)$   
*<proof>*

### 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:  $create-name-gba\ \varphi = do\ \{$ 
   $nds \leftarrow create-graph_T\ \varphi;$ 
   $ASSERT\ (nds-invars\ nds);$ 
   $RETURN\ (gba-rewrite-ext\ (\lambda-. ())\ name\ (create-gba-from-nodes\ \varphi\ nds))$ 
 $\}$ 
<proof>

```

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  $build-succ\ nds \leq SPEC\ (\lambda r. r = (\lambda qn.$ 
     $dflt-None-set\ \{qn'. \exists q'.$ 
       $q' \in nds \wedge qn' = name\ q' \wedge qn \in incoming\ q' \wedge qn \neq expand-init$ 
     $\}))$ 
  <proof>

```

```

lemma build-succ-aux2:
  assumes NINIT:  $expand-init \notin name'nds$ 
  assumes CL:  $\bigwedge nd. nd \in nds \implies incoming\ nd \subseteq insert\ expand-init\ (name'nds)$ 
  shows
     $(\lambda qn. dflt-None-set$ 
       $\{qn'. \exists q'. q' \in nds \wedge qn' = name\ q' \wedge qn \in incoming\ q' \wedge qn \neq expand-init\ \})$ 

```

$= (\lambda qn. \text{dflt-None-set } (\text{succ-of-E } (\text{rename-E name } \{(q, q') . q \in \text{nds} \wedge q' \in \text{nds} \wedge \text{name } q \in \text{incoming } q'\})$   
 $qn))$   
 $(\text{is } (\lambda qn. \text{dflt-None-set } (?L \text{qn})) = (\lambda qn. \text{dflt-None-set } (?R \text{qn})))$   
 $\langle \text{proof} \rangle$

**lemma** *build-succ-correct*:

**assumes** *NINIT*:  $\text{expand-init} \notin \text{name'nds}$   
**assumes** *FIN*: *finite nds*  
**assumes** *CL*:  $\bigwedge nd. nd \in \text{nds} \implies \text{incoming } nd \subseteq \text{insert } \text{expand-init } (\text{name'nds})$   
**shows**  $\text{build-succ } \text{nds} \leq \text{SPEC } (\lambda r. \text{E-of-succ } (\lambda qn. \text{the-default } \{\} (r \text{qn}))$   
 $= \text{rename-E } (\lambda u. \text{name } u) \{(q, q') . q \in \text{nds} \wedge q' \in \text{nds} \wedge \text{name } q \in \text{incoming } q'\})$   
 $\langle \text{proof} \rangle$

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

$\text{until-frmlsr } (\mu \text{ and}_r \psi) = (\text{until-frmlsr } \mu) \cup (\text{until-frmlsr } \psi)$   
 $|\ \text{until-frmlsr } (X_r \mu) = \text{until-frmlsr } \mu$   
 $|\ \text{until-frmlsr } (\mu \text{ U}_r \psi) = \text{insert } (\mu, \psi) ((\text{until-frmlsr } \mu) \cup (\text{until-frmlsr } \psi))$   
 $|\ \text{until-frmlsr } (\mu \text{ R}_r \psi) = (\text{until-frmlsr } \mu) \cup (\text{until-frmlsr } \psi)$   
 $|\ \text{until-frmlsr } (\mu \text{ or}_r \psi) = (\text{until-frmlsr } \mu) \cup (\text{until-frmlsr } \psi)$   
 $|\ \text{until-frmlsr } (\text{true}_r) = \{\}$   
 $|\ \text{until-frmlsr } (\text{false}_r) = \{\}$   
 $|\ \text{until-frmlsr } (\text{prop}_r(-)) = \{\}$   
 $|\ \text{until-frmlsr } (\text{nprop}_r(-)) = \{\}$

**lemma** *until-frmlsr-correct*:

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

**definition** *build-F nds*  $\varphi$

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

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

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

$\langle \text{proof} \rangle$

**definition** *pn-props*  $ps \equiv \text{FOREACHi}$

$(\lambda \text{it } (P, N). P = \{p. \text{Prop-ltlr } p \in ps - \text{it}\} \wedge N = \{p. \text{Nprop-ltlr } p \in ps - \text{it}\})$   
 $ps (\lambda p (P, N).$

```

    case p of Prop-ltlr p ⇒ RETURN (insert p P,N)
    | Nprop-ltlr p ⇒ RETURN (P, insert p N)
    | - ⇒ RETURN (P,N)
  ) ({} , {})

```

**lemma** *pn-props-correct*:

```

assumes [simp]: finite ps
shows pn-props ps ≤ SPEC(λr. r =
  ({p. Prop-ltlr p ∈ ps}, {p. Nprop-ltlr p ∈ ps}))
⟨proof⟩

```

**definition** *pn-map nds* ≡ FOREACH nds

```

(λnd m. do {
  PN ← pn-props (old nd);
  RETURN (m(name nd ↦ PN))
}) Map.empty

```

**lemma** *pn-map-correct*:

```

assumes [simp]: finite nds
assumes FIN': ∧nd. nd ∈ nds ⇒ finite (old nd)
assumes INJ: inj-on name nds
shows pn-map nds ≤ SPEC (λr. ∀ qn.
  case r qn of
    None ⇒ qn ∉ name'nds
  | Some (P,N) ⇒ qn ∈ name'nds
    ∧ P = {p. Prop-ltlr p ∈ old (the-inv-into nds name qn)}
    ∧ N = {p. Nprop-ltlr p ∈ old (the-inv-into nds name qn)}
  )
⟨proof⟩

```

**definition** *cr-rename-gba nds φ* ≡ do {

```

  let V = name ' nds;
  let V0 = name ' {q ∈ nds. expand-init ∈ incoming q};
  succmap ← build-succ nds;
  let E = E-of-succ (the-default {} o succmap);
  let F = build-F nds φ;
  pnm ← pn-map nds;
  let L = (λqn l. case pnm qn of
    None ⇒ False
  | Some (P,N) ⇒ (∀ p ∈ P. p ∈ (l:::r⟨Id⟩fun-set-rel)) ∧ (∀ p ∈ N. p ∉ l)
  );
  RETURN ((| 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) ∈ Id (φ',φ) ∈ Id

```



**shows** *cr-rename-gba* *nds'*  $\varphi'$   
 $\leq \Downarrow Id (RETURN (gba-rename-ext (\lambda-. ()) name (create-gba-from-nodes \varphi nds)))$   
 $\langle proof \rangle$

**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*  $\varphi \leq \Downarrow Id (create-name-gba \varphi)$   
 $\langle proof \rangle$

### 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** ( $?c, until-frmlsr$ )  
 $\in \langle (R::(-\times-::linorder) set) \rangle ltlr-rel \rightarrow \langle (R) \rangle ltlr-rel \times_r \langle (R) \rangle ltlr-rel \rangle dflt-rs-rel$   
 $\langle proof \rangle$

**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** ( $?c, build-succ$ )  $\in$   
 $\langle (Rm, R) node-rel \rangle list-set-rel$   
 $\rightarrow \langle (nat-rel, \langle nat-rel \rangle list-set-rel) iam-map-rel \rangle nres-rel$   
 $\langle proof \rangle$

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

**schematic-goal** *build-succ-code-aux*:  $RETURN ?c \leq build-succ-impl x$   
 $\langle proof \rangle$

**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**  $(?c, build-F) \in$   
 $\langle \langle Rm, R \rangle node-rel \rangle list-set-rel \rightarrow \langle R \rangle ltlr-rel \rightarrow \langle \langle nat-rel \rangle list-set-rel \rangle list-set-rel$   
 $\langle proof \rangle$

**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**  $(?c, pn-map) \in$   
 $\langle \langle Rm, Id \rangle node-rel \rangle list-set-rel$   
 $\rightarrow \langle \langle nat-rel, \langle Id \rangle list-set-rel \times_r \langle Id \rangle list-set-rel \rangle iam-map-rel \rangle nres-rel$   
 $\langle proof \rangle$

**concrete-definition** *pn-map-impl* **uses** *pn-map-impl-aux*  
**lemma** *pn-map-impl-autoref*[*autoref-rules*]:  
**assumes** *PREFER-id R*  
**shows**  $(pn-map-impl, pn-map) \in$   
 $\langle \langle Rm, R \rangle node-rel \rangle list-set-rel$   
 $\rightarrow \langle \langle nat-rel, \langle R \rangle list-set-rel \times_r \langle R \rangle list-set-rel \rangle iam-map-rel \rangle nres-rel$   
 $\langle proof \rangle$

**schematic-goal** *pn-map-code-aux*: *RETURN ?c ≤ pn-map-impl x*  
 $\langle proof \rangle$

**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*  $\langle nat-rel \rangle list-set-rel$ ]  
**shows**  $(?c, cr-rename-gba) \in$   
 $\langle \langle Rm, R \rangle node-rel \rangle list-set-rel \rightarrow \langle R \rangle ltlr-rel \rightarrow (?R::(?'c \times -) set)$   
 $\langle proof \rangle$

**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**  $(cr-rename-gba-impl, cr-rename-gba) \in$   
 $\langle \langle Rm, R \rangle node-rel \rangle list-set-rel \rightarrow \langle R \rangle ltlr-rel \rightarrow$   
 $\langle gbav-impl-rel-ext \ unit-rel \ nat-rel \ (\langle R \rangle fun-set-rel) \rangle nres-rel$   
 $\langle proof \rangle$

**schematic-goal** *cr-rename-gba-code-aux*: *RETURN ?c ≤ cr-rename-gba-impl x y*  
 ⟨*proof*⟩  
**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*:  $\langle R \rangle \text{ls-nds-rel} = \langle R \rangle \text{list-set-rel} \cap \{(-,s). \text{inj-on name } s\}$   
 ⟨*proof*⟩

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

⟨*proof*⟩

**context begin interpretation** *autoref-syn* ⟨*proof*⟩

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

**assumes** *PREFER-id*  $R$

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

⟨*proof*⟩

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

⟨*proof*⟩

**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*⟩

**lemma** *ls-nds-image-autoref*[*autoref-rules*]:

**assumes**  $(f_i, f) \in Ra \rightarrow Rb$

**assumes** *SIDE-PRECOND* ( $\forall x. \text{name } (f x) = \text{name } x$ )

**shows**  $(\text{map } f_i, (OP \text{ ' } :: (Ra \rightarrow Rb) \rightarrow \langle Ra \rangle_{ls-nds-rel} \rightarrow \langle Rb \rangle_{ls-nds-rel}) \$ f)$   
 $\in \langle Ra \rangle_{ls-nds-rel} \rightarrow \langle Rb \rangle_{ls-nds-rel}$   
 $\langle \text{proof} \rangle$

**lemma** *list-set-autoref-to-list*[*autoref-ga-rules*]:  
**shows** *is-set-to-sorted-list*  $(\lambda - . \text{True}) R \text{ } ls-nds-rel \text{ } id$   
 $\langle \text{proof} \rangle$

**end**

**context begin interpretation** *autoref-syn*  $\langle \text{proof} \rangle$

**lemma** [*autoref-itype*]:

*upd-incoming*  
 $::_i \langle Im, I \rangle_i i\text{-node} \rightarrow_i \langle \langle Im', I \rangle_i i\text{-node} \rangle_i i\text{-set} \rightarrow_i \langle \langle Im', I \rangle_i i\text{-node} \rangle_i i\text{-set}$   
 $\langle \text{proof} \rangle$

**end**

**term** *upd-incoming*

**schematic-goal** *upd-incoming-impl-aux*:

**assumes** *REL-IS-ID* *R*  
**shows**  $(?c, \text{upd-incoming}) \in \langle Rm1, R \rangle \text{node-rel}$   
 $\rightarrow \langle \langle Rm2, R \rangle \text{node-rel} \rangle_{ls-nds-rel}$   
 $\rightarrow \langle \langle Rm2, R \rangle \text{node-rel} \rangle_{ls-nds-rel}$   
 $\langle \text{proof} \rangle$

**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*:  $(?c, \text{expand-aimpl}) \in$   
 $\langle \text{unit-rel}, Id \rangle \text{node-rel} \times_r \langle \langle \text{unit-rel}, Id \rangle \text{node-rel} \rangle_{ls-nds-rel}$   
 $\rightarrow \langle \text{nat-rel} \times_r \langle \langle \text{unit-rel}, Id \rangle \text{node-rel} \rangle_{ls-nds-rel} \rangle \text{nres-rel}$   
 $\langle \text{proof} \rangle$

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

**context begin interpretation** *autoref-syn*  $\langle \text{proof} \rangle$

**lemma** [*autoref-itype*]: *expand<sub>T</sub>*

$::_i \langle \langle i\text{-unit}, I \rangle_i i\text{-node}, \langle \langle i\text{-unit}, I \rangle_i i\text{-node} \rangle_i i\text{-set} \rangle_i i\text{-prod}$   
 $\rightarrow_i \langle \langle i\text{-nat}, \langle \langle i\text{-unit}, I \rangle_i i\text{-node} \rangle_i i\text{-set} \rangle_i i\text{-prod} \rangle_i i\text{-nres} \langle \text{proof} \rangle$

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

**assumes** *ID*: *PREFER-id* *R*

**assumes** *A*:  $(n\text{-ns}', n\text{-ns})$

$\in \langle \text{unit-rel}, R \rangle \text{node-rel} \times_r \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel}$

**assumes** *B*: *SIDE-PRECOND* ( $$

$\text{let } (n, ns) = n\text{-ns} \text{ in } \text{inj-on name ns} \wedge (\forall n' \in ns. \text{name } n > \text{name } n')$

$)$

**shows** (*expand-impl* *n-ns'*,  
*(OP expand-aimpl*  
 $:: \langle \text{unit-rel}, R \rangle \text{node-rel} \times_r \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel}$   
 $\rightarrow \langle \text{nat-rel} \times_r \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel} \rangle \text{nres-rel}$ )\$*n-ns*)  
 $\in \langle \text{nat-rel} \times_r \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel} \rangle \text{nres-rel}$   
*<proof>*

**end**

**schematic-goal** *expand-code-aux*: *RETURN ?c ≤ expand-impl n-ns*  
*<proof>*

**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*)

$\in \langle R \rangle \text{ltlr-rel} \rightarrow \langle \langle \langle \text{unit-rel}, R \rangle \text{node-rel} \rangle \text{list-set-rel} \rangle \text{nres-rel}$

*<proof>*

**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 ≤ create-graph-impl φ*  
*<proof>*

**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 ⇒ -)*)

$\in \langle \text{Id} \rangle \text{ltlr-rel} \rightarrow (\text{?R}::(\text{?'c} \times -) \text{ set})$

*<proof>*

**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*)

$\in \langle R \rangle \text{ltlr-rel} \rightarrow \langle \text{gbav-impl-rel-ext unit-rel nat-rel} (\langle R \rangle \text{fun-set-rel}) \rangle \text{nres-rel}$

(**is**  $\text{-:} \rightarrow \langle \text{?R} \rangle \text{nres-rel}$ )

*<proof>*

**schematic-goal** *create-name-gba-code-aux*: *RETURN ?c ≤ create-name-gba-impl*

*φ*

*<proof>*

**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**  $(?c, \text{create-name-igba}) \in$   
 $\langle R \rangle \text{ltlr-rel} \rightarrow \langle \text{igba-impl-rel-ext unit-rel nat-rel} (\langle R \rangle \text{fun-set-rel}) \rangle \text{nres-rel}$   
 $\langle \text{proof} \rangle$

**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 ≤ create-name-igba-impl*  
 $\varphi$   
 $\langle \text{proof} \rangle$

**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](http://isa-afp.org/entries/Stuttering_Equivalence.shtml), Formal proof development.