Converting Linear Temporal Logic to Deterministic (Generalized) Rabin Automata

Salomon Sickert

March 17, 2025

Abstract

Recently a new method directly translating linear temporal logic (LTL) formulas to deterministic (generalized) Rabin automata was described in [1].

Compared to the existing approaches of constructing a non-deterministic Buechi-automaton in the first step and then applying a determinization procedure (e.g. some variant of Safra's construction) in a second step, this new approach preservers a relation between the formula and the states of the resulting automaton. While the old approach produced a monolithic structure, the new method is compositional. Furthermore it was shown in some cases the resulting automata were much smaller than the automata generated by existing approaches. In order to guarantee the correctness of the construction this entry contains a complete formalisation and verification of the translation. Furthermore from this basis executable code is generated.

Contents

1 Auxiliary Facts

theory *Preliminaries2*
imports *Main HOL−Library.Infinite-Set*
begin

1.1 Finite and Infinite Sets

lemma *finite-product*:
assumes *fst*: *finite* (*fst* ‘ *A*)
and *snd*: *finite* (*snd* ‘ *A*)
shows *finite A*
proof −

have $A \subseteq (\mathit{fst}\ `\ A) \times (\mathit{snd}\ `\ A)$
by *force*
thus *?thesis*
using *snd fst finite-subset* **by** *blast*
qed

1.2 Cofinite Filters

lemma *almost-all-commutative*:
$\mathit{finite}\ S \Longrightarrow (\forall x \in S.\ \forall_\infty i.\ P\ x\ (i{::}\mathit{nat})) = (\forall_\infty i.\ \forall x \in S.\ P\ x\ i)$
proof (*induction rule*: *finite-induct*)
case (*insert x S*)
{
assume $\forall x \in \mathit{insert}\ x\ S.\ \forall_\infty i.\ P\ x\ i$
hence $\forall_\infty i.\ \forall x \in S.\ P\ x\ i$ **and** $\forall_\infty i.\ P\ x\ i$
using *insert* **by** *simp+*
then obtain $i_1\ i_2$ **where** $\bigwedge j.\ j \geq i_1 \Longrightarrow \forall x \in S.\ P\ x\ j$
and $\bigwedge j.\ j \geq i_2 \Longrightarrow P\ x\ j$
unfolding *MOST-nat-le* **by** *auto*
hence $\bigwedge j.\ j \geq \mathit{max}\ i_1\ i_2 \Longrightarrow \forall x \in S \cup \{x\}.\ P\ x\ j$
by *simp*
hence $\forall_\infty i.\ \forall x \in \mathit{insert}\ x\ S.\ P\ x\ i$
unfolding *MOST-nat-le* **by** *blast*
}
moreover
have $\forall_\infty i.\ \forall x \in \mathit{insert}\ x\ S.\ P\ x\ i \Longrightarrow \forall x \in \mathit{insert}\ x\ S.\ \forall_\infty i.\ P\ x\ i$
unfolding *MOST-nat-le* **by** *auto*
ultimately
show *?case*
by *blast*
qed *simp*

lemma *almost-all-commutative'*:
$\mathit{finite}\ S \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow \forall_\infty i.\ P\ x\ (i{::}\mathit{nat})) \Longrightarrow (\forall_\infty i.\ \forall x \in S.\ P\ x\ i)$
using *almost-all-commutative* **by** *blast*

fun *index*
where
$\mathit{index}\ P = (\mathit{if}\ \forall_\infty i.\ P\ i\ \mathit{then}\ \mathit{Some}\ (\mathit{LEAST}\ i.\ \forall j \geq i.\ P\ j)\ \mathit{else}\ \mathit{None})$

lemma *index-properties*:
fixes $i :: \mathit{nat}$
shows $\mathit{index}\ P = \mathit{Some}\ i \Longrightarrow 0 < i \Longrightarrow \neg\ P\ (i - 1)$
and $\mathit{index}\ P = \mathit{Some}\ i \Longrightarrow j \geq i \Longrightarrow P\ j$

proof −
  assume *index P* = *Some i*
  moreover
  hence *i-def*: $i = (LEAST\ i.\ \forall j \geq i.\ P\ j)$ **and** $\forall_\infty i.\ P\ i$
    unfolding *index.simps* **using** *option.distinct(2) option.sel*
    by (*metis* (*erased*, *lifting*))+
  then obtain i' **where** $\forall j \geq i'.\ P\ j$
    unfolding *MOST-nat-le* **by** *blast*
  ultimately
  show $\bigwedge j.\ j \geq i \Longrightarrow P\ j$
    using *LeastI*[*of* $\lambda i.\ \forall j \geq i.\ P\ j$] **by** (*metis i-def*)
  {
    assume $0 < i$
    then obtain *j* **where** $i = Suc\ j$ **and** $j < i$
      using *lessE* **by** *blast*
    hence $\bigwedge j'.\ j' > j \Longrightarrow P\ j'$
      using ‹$\bigwedge j.\ j \geq i \Longrightarrow P\ j$› **by** *force*
    hence $\neg\ P\ j$
    using *not-less-Least*[*OF* ‹$j < i$›[*unfolded i-def*]] **by** (*metis leI le-antisym*)
    thus $\neg\ P\ (i - 1)$
      unfolding ‹$i = Suc\ j$› **by** *simp*
  }
qed

end

2 Auxiliary Map Facts

theory *Map2*
  imports *Main*
begin

lemma *map-of-tabulate*:
  map-of (*map* ($\lambda x.$ (*x*, *f x*)) *xs*) $x \neq None \longleftrightarrow x \in set\ xs$
  by (*induct xs*) *auto*

lemma *map-of-tabulate-simp*:
  map-of (*map* ($\lambda x.$ (*x*, *f x*)) *xs*) *x* = (*if* $x \in set\ xs$ *then Some* (*f x*) *else None*)
  by (*metis* (*mono-tags*, *lifting*) *comp-eq-dest-lhs map-of-map-restrict restrict-map-def*)

lemma *dom-map-update*:

$dom\ (m\ (k \mapsto v)) = dom\ m \cup \{k\}$
by *simp*

lemma *map-equal*:
$dom\ m = dom\ m' \Longrightarrow (\bigwedge x.\ x \in dom\ m \Longrightarrow m\ x = m'\ x) \Longrightarrow m = m'$
by *fastforce*

lemma *map-reduce*:
assumes $dom\ m = \{a\} \cup B$
shows $\exists\, m'.\ dom\ m' = B \wedge (\forall\, x \in B.\ m\ x = m'\ x)$
proof (*cases* $a \in B$)
case *True*
thus *?thesis*
using *assms* **by** (*metis insert-absorb insert-is-Un*)
next
case *False*
with *assms* **have** $dom\ (m\ (a := None)) = B \wedge (\forall\, x{\in}B.\ m\ x = (m\ (a := None))\ x)$
by *simp*
thus *?thesis*
by *blast*
qed

end

3 Auxiliary Mapping Facts

theory *Mapping2*
imports *Main Map2 HOL−Library.Mapping*
begin

lemma *lookup-delete*:
Mapping.lookup (*Mapping.delete k m*) $k = None$
by (*transfer*; *simp*)

lemma *lookup-tabulate*:
Mapping.lookup (*Mapping.tabulate xs f*) $x =$ (*if* $x \in$ *set xs then Some* (f x) *else None*)
by (*transfer*; *insert map-of-tabulate-simp*)

lemma *lookup-tabulate-Some*:
$x \in$ *set xs* $\Longrightarrow$ *the* (*Mapping.lookup* (*Mapping.tabulate xs f*) x) $= f\ x$
by (*simp add*: *lookup-tabulate*)

lemma *finite-keys-tabulate*:
finite (*Mapping.keys* (*Mapping.tabulate xs f*))
by *simp*

lemma *keys-empty-iff-map-empty*:
Mapping.keys m = {} $\longleftrightarrow$ *m* = *Mapping.empty*
by (*transfer*; *simp*)

lemma *mapping-equal*:
Mapping.keys m = *Mapping.keys m'* $\Longrightarrow$ ($\bigwedge$*x*. *x* $\in$ *Mapping.keys m* $\Longrightarrow$ *Mapping.lookup m x* = *Mapping.lookup m' x*) $\Longrightarrow$ *m* = *m'*
by (*transfer*; *blast intro*: *map-equal*)

fun *mapping-generator* :: ('*a* $\Rightarrow$ '*b list*) $\Rightarrow$ '*a list* $\Rightarrow$ ('*a*, '*b*) *mapping set*
where
mapping-generator V [] = {*Mapping.empty*}
| *mapping-generator V* (*k*#*ks*) = {*Mapping.update k v m* | *v m*. *v* $\in$ *set* (*V k*) $\wedge$ *m* $\in$ *mapping-generator V ks*}

fun *mapping-generator-list* :: ('*a* $\Rightarrow$ '*b list*) $\Rightarrow$ '*a list* $\Rightarrow$ ('*a*, '*b*) *mapping list*
where
mapping-generator-list V [] = [*Mapping.empty*]
| *mapping-generator-list V* (*k*#*ks*) = *concat* (*map* (λ*m*. *map* (λ*v*. *Mapping.update k v m*) (*V k*)) (*mapping-generator-list V ks*))

lemma *mapping-generator-code* [*code*]:
mapping-generator V K = *set* (*mapping-generator-list V K*)
by (*induction K*) *auto*

lemma *mapping-generator-set-eq*:
mapping-generator V K = {*m*. *Mapping.keys m* = *set K* $\wedge$ ($\forall$ *k* $\in$ (*set K*). *the* (*Mapping.lookup m k*) $\in$ *set* (*V k*))}
proof (*induction K*)
case (*Cons k ks*)
let *?l* = {*m*(*k* $\mapsto$ *v*) |*v m*. *v* $\in$ *set* (*V k*) $\wedge$ *m* $\in$ {*m*. *dom m* = *set ks* $\wedge$ ($\forall$ *k*$\in$*set ks*. *the* (*m k*) $\in$ *set* (*V k*))}}
let *?r* = {*m*. *dom m* = *set* (*k* # *ks*) $\wedge$ ($\forall$ *k*$\in$*set* (*k* # *ks*). *the* (*m k*) $\in$ *set* (*V k*))}

have *?l* $\subseteq$ *?r*
by *fastforce*
moreover
{

fix *m*
assume $m \in ?r$
hence $dom\ m = set\ (k\#ks)$
and $\forall\, k \in set\ (k\#ks).\ the\ (m\ k) \in set\ (V\ k)$
and $\forall\, k' \in set\ (k\#ks).\ m\ k \neq None$
by *auto*
moreover
then obtain m' **where** $dom\ m' = set\ ks$
and $\forall\, x \in set\ ks.\ m\ x = m'\ x$
using *map-reduce*[*of m k set ks*] **by** *auto*
ultimately
have $the\ (m\ k) \in set\ (V\ k)$
and $dom\ m' = set\ ks$
and $\forall\, k \in (set\ ks).\ the\ (m'\ k) \in set\ (V\ k)$
and $m = m'(k \mapsto the\ (m\ k))$
apply (*simp*, *blast*, *auto*)
apply (*insert map-equal*[*of m* $m'(k \mapsto the\ (m\ k))$])
apply (*unfold dom-map-update* ‹$dom\ m = set\ (k\#ks)$› ‹$dom\ m' = set\ ks$›)
by *fastforce*
moreover
hence $m \in set\ (map\ (\lambda v.\ m'(k \mapsto v))\ (V\ k))$
by *simp*
ultimately
have $m \in ?l$
using ‹$dom\ m = set\ (k\#ks)$› **by** *blast*
}
ultimately
have $\{Mapping.update\ k\ v\ m\ |v\ m.\ v \in set\ (V\ k) \wedge m \in \{m.\ Mapping.keys\ m = set\ ks \wedge (\forall\, k{\in}set\ ks.\ the\ (Mapping.lookup\ m\ k) \in set\ (V\ k))\}\}$
$= \{m.\ Mapping.keys\ m = set\ (k\ \#\ ks) \wedge (\forall\, k{\in}set\ (k\ \#\ ks).\ the\ (Mapping.lookup\ m\ k) \in set\ (V\ k))\}$
by (*transfer*; *blast*)
thus *?case*
by (*simp add*: *Cons*)
qed (*force simp add*: *keys-empty-iff-map-empty*)

end

4 Deterministic Transition Systems

theory *DTS*
imports *Main HOL−Library.Omega-Words-Fun Auxiliary/Mapping2 KBPs.DFS*

begin

— DTS are realised by functions

type-synonym $('a,\ 'b)\ DTS = 'a \Rightarrow 'b \Rightarrow 'a$
type-synonym $('a,\ 'b)\ transition = ('a \times 'b \times 'a)$

4.1 Infinite Runs

fun $run :: ('q,'a)\ DTS \Rightarrow 'q \Rightarrow 'a\ word \Rightarrow 'q\ word$
where
$run\ \delta\ q_0\ w\ 0 = q_0$
$|\ run\ \delta\ q_0\ w\ (Suc\ i) = \delta\ (run\ \delta\ q_0\ w\ i)\ (w\ i)$

fun $run_t :: ('q,'a)\ DTS \Rightarrow 'q \Rightarrow 'a\ word \Rightarrow ('q * 'a * 'q)\ word$
where
$run_t\ \delta\ q_0\ w\ i = (run\ \delta\ q_0\ w\ i,\ w\ i,\ run\ \delta\ q_0\ w\ (Suc\ i))$

lemma *run-foldl*:
$run\ \Delta\ q_0\ w\ i = foldl\ \Delta\ q_0\ (map\ w\ [0..<i])$
by (*induction i*; *simp*)

lemma run_t*-foldl*:
$run_t\ \Delta\ q_0\ w\ i = (foldl\ \Delta\ q_0\ (map\ w\ [0..<i]),\ w\ i,\ foldl\ \Delta\ q_0\ (map\ w\ [0..<Suc\ i]))$
unfolding run_t*.simps run-foldl* **..**

4.2 Reachable States and Transitions

definition $reach :: 'a\ set \Rightarrow ('b,\ 'a)\ DTS \Rightarrow 'b \Rightarrow 'b\ set$
where
$reach\ \Sigma\ \delta\ q_0 = \{run\ \delta\ q_0\ w\ n\ |\ w\ n.\ range\ w \subseteq \Sigma\}$

definition $reach_t :: 'a\ set \Rightarrow ('b,\ 'a)\ DTS \Rightarrow 'b \Rightarrow ('b,\ 'a)\ transition\ set$
where
$reach_t\ \Sigma\ \delta\ q_0 = \{run_t\ \delta\ q_0\ w\ n\ |\ w\ n.\ range\ w \subseteq \Sigma\}$

lemma *reach-foldl-def*:
assumes $\Sigma \neq \{\}$
shows $reach\ \Sigma\ \delta\ q_0 = \{foldl\ \delta\ q_0\ w\ |\ w.\ set\ w \subseteq \Sigma\}$
proof −
{
fix w **assume** $set\ w \subseteq \Sigma$
moreover

obtain *a* **where** $a \in \Sigma$
using $\langle\Sigma \neq \{\}\rangle$ **by** *blast*
ultimately
have *foldl* δ q_0 *w* = *foldl* δ q_0 (*prefix* (*length w*) (*w* $\frown$ (*iter* [*a*])))
and *range* (*w* $\frown$ (*iter* [*a*])) $\subseteq \Sigma$
by (*unfold prefix-conc-length*, *auto simp add*: *iter-def conc-def*)
hence $\exists$ *w'* *n*. *foldl* δ q_0 *w* = *run* δ q_0 *w'* *n* $\wedge$ *range w'* $\subseteq \Sigma$
unfolding *run-foldl subsequence-def* **by** *blast*
}
thus *?thesis*
by (*fastforce simp add*: *reach-def run-foldl*)
qed

lemma *reach*$_t$*-foldl-def*:
reach$_t$ Σ δ q_0 = $\{$(*foldl* δ q_0 *w*, ν, *foldl* δ q_0 (*w*@[ν])) $|$ *w* ν. *set w* $\subseteq \Sigma \wedge$ $\nu \in \Sigma\}$ (**is** *?lhs* = *?rhs*)
proof (*cases* $\Sigma \neq \{\}$)
case *True*
show *?thesis*
proof
{
fix *w* ν **assume** *set w* $\subseteq \Sigma$ $\nu \in \Sigma$
moreover
obtain *a* **where** $a \in \Sigma$
using $\langle\Sigma \neq \{\}\rangle$ **by** *blast*
moreover
have *w* = *map* (λ*n*. *if n* < *length w then w* ! *n else if n* − *length w* = *0 then* [ν] ! (*n* − *length w*) *else a*) [*0*..<*length w*]
by (*simp add*: *nth-equalityI*)
ultimately
have *foldl* δ q_0 *w* = *foldl* δ q_0 (*prefix* (*length w*) ((*w* @ [ν]) $\frown$ (*iter* [*a*])))
and*foldl* δ q_0 (*w* @ [ν]) = *foldl* δ q_0 (*prefix* (*length* (*w* @ [ν])) ((*w* @ [ν]) $\frown$ (*iter* [*a*])))
and *range* ((*w* @ [ν]) $\frown$ (*iter* [*a*])) $\subseteq \Sigma$
by (*simp-all only*: *prefix-conc-length conc-conc*[*symmetric*] *iter-def*)
(*auto simp add*: *subsequence-def conc-def upt-Suc-append*[*OF le0*])
moreover
have ((*w* @ [ν]) $\frown$ (*iter* [*a*])) (*length w*) = ν
by (*simp add*: *conc-def*)
ultimately
have $\exists$ *w'* *n*. (*foldl* δ q_0 *w*, ν, *foldl* δ q_0 (*w* @ [ν])) = *run*$_t$ δ q_0 *w'* *n* $\wedge$ *range w'* $\subseteq \Sigma$
by (*metis run*$_t$*-foldl length-append-singleton subsequence-def*)

}
thus *?lhs* $\supseteq$ *?rhs*
unfolding *reach*$_t$*-def run*$_t$*.simps* **by** *blast*
qed (*unfold reach*$_t$*-def run*$_t$*-foldl*, *fastforce simp add*: *upt-Suc-append*)
qed (*simp add*: *reach*$_t$*-def*)

lemma *reach-card-0*:
assumes $\Sigma \neq \{\}$
shows *infinite* (*reach* Σ δ q_0) $\longleftrightarrow$ *card* (*reach* Σ δ q_0) = *0*
proof −
have $\{$*run* δ q_0 *w n* | *w n*. *range w* $\subseteq \Sigma\} \neq \{\}$
using *assms* **by** *fast*
thus *?thesis*
unfolding *reach-def card-eq-0-iff* **by** *auto*
qed

lemma *reach*$_t$*-card-0*:
assumes $\Sigma \neq \{\}$
shows *infinite* (*reach*$_t$ Σ δ q_0) $\longleftrightarrow$ *card* (*reach*$_t$ Σ δ q_0) = *0*
proof −
have $\{$*run*$_t$ δ q_0 *w n* | *w n*. *range w* $\subseteq \Sigma\} \neq \{\}$
using *assms* **by** *fast*
thus *?thesis*
unfolding *reach*$_t$*-def card-eq-0-iff* **by** *blast*
qed

4.2.1 Relation to runs

lemma *run-subseteq-reach*:
assumes *range w* $\subseteq \Sigma$
shows *range* (*run* δ q_0 *w*) $\subseteq$ *reach* Σ δ q_0
and *range* (*run*$_t$ δ q_0 *w*) $\subseteq$ *reach*$_t$ Σ δ q_0
using *assms* **unfolding** *reach-def reach*$_t$*-def* **by** *blast+*

lemma *limit-subseteq-reach*:
assumes *range w* $\subseteq \Sigma$
shows *limit* (*run* δ q_0 *w*) $\subseteq$ *reach* Σ δ q_0
and *limit* (*run*$_t$ δ q_0 *w*) $\subseteq$ *reach*$_t$ Σ δ q_0
using *run-subseteq-reach*[*OF assms*] *limit-in-range* **by** *fast+*

lemma *run*$_t$*-finite*:
assumes *finite* (*reach* Σ δ q_0)
assumes *finite* Σ
assumes *range w* $\subseteq \Sigma$

defines $r \equiv run_t\ \delta\ q_0\ w$
shows *finite* (*range r*)
proof –
let $?S = (reach\ \Sigma\ \delta\ q_0) \times \Sigma \times (reach\ \Sigma\ \delta\ q_0)$
have $\bigwedge i.\ w\ i \in \Sigma$ **and** $\bigwedge i.\ set\ (map\ w\ [0..<i]) \subseteq \Sigma$ **and** $\Sigma \neq \{\}$
using ‹$range\ w \subseteq \Sigma$› **by** *auto*
hence $\bigwedge n.\ r\ n \in\ ?S$
unfolding run_t*.simps run-foldl reach-foldl-def*[*OF* ‹$\Sigma \neq \{\}$›] *r-def* **by** *blast*
hence $range\ r \subseteq\ ?S$ **and** *finite ?S*
using *assms* **by** *blast+*
thus *finite* (*range r*)
by (*blast dest*: *finite-subset*)
qed

4.2.2 Compute reach Using DFS

definition Q_L :: $'a\ list \Rightarrow ('b,\ 'a)\ DTS \Rightarrow\ 'b \Rightarrow\ 'b\ set$
where
$Q_L\ \Sigma\ \delta\ q_0 = (if\ \Sigma \neq []\ then\ gen\text{-}dfs\ (\lambda q.\ map\ (\delta\ q)\ \Sigma)\ Set.insert\ (\in)\ \{\}\ [q_0]\ else\ \{\})$

definition *list-dfs* :: $(('a,\ 'b)\ transition \Rightarrow ('a,\ 'b)\ transition\ list) \Rightarrow ('a,\ 'b)\ transition\ list => ('a,\ 'b)\ transition\ list => ('a,\ 'b)\ transition\ list$
where
$list\text{-}dfs\ succ\ S\ start \equiv gen\text{-}dfs\ succ\ List.insert\ (\lambda x\ xs.\ x \in set\ xs)\ S\ start$

definition δ_L :: $'a\ list \Rightarrow ('b,\ 'a)\ DTS \Rightarrow\ 'b \Rightarrow ('b,\ 'a)\ transition\ set$
where
$\delta_L\ \Sigma\ \delta\ q_0 = set\ ($
let
$start = map\ (\lambda\nu.\ (q_0,\ \nu,\ \delta\ q_0\ \nu))\ \Sigma;$
$succ = \lambda(\text{-},\ \text{-},\ q).\ (map\ (\lambda\nu.\ (q,\ \nu,\ \delta\ q\ \nu))\ \Sigma)$
in
(*list-dfs succ* [] *start*))

lemma Q_L*-reach*:
assumes *finite* $(reach\ (set\ \Sigma)\ \delta\ q_0)$
shows $Q_L\ \Sigma\ \delta\ q_0 = reach\ (set\ \Sigma)\ \delta\ q_0$
proof $(cases\ \Sigma \neq [])$
case *True*
hence *reach-redef*: $reach\ (set\ \Sigma)\ \delta\ q_0 = \{foldl\ \delta\ q_0\ w \mid w.\ set\ w \subseteq set\ \Sigma\}$
using *reach-foldl-def*[*of set* Σ] **unfolding** *set-empty*[*of* Σ, *symmetric*]

```
by force

  {
   fix x w y
   assume set w ⊆ set Σ x = foldl δ q0 w y ∈ set (map (δ x) Σ)
   moreover
   then obtain ν where y = δ x ν and ν ∈ set Σ
     by auto
   ultimately
   have y = foldl δ q0 (w@[ν]) and set (w@[ν]) ⊆ set Σ
     by simp+
   hence ∃ w'. set w' ⊆ set Σ ∧ y = foldl δ q0 w'
     by blast
  }
  note extend-run = this

  interpret DFS λq. map (δ q) Σ λq. q ∈ reach (set Σ) δ q0 λS. S ⊆ reach (set Σ) δ q0 Set.insert (∈) {} id
    apply (unfold-locales; auto simp add: member-rec reach-redef list-all-iff elim: extend-run)
    apply (metis extend-run image-eqI set-map)
    apply (metis assms[unfolded reach-redef])
    done

  have Nil1: set [] ⊆ set Σ and Nil2: q0 = foldl δ q0 []
    by simp+
  have list-all-init: list-all (λq. q ∈ reach (set Σ) δ q0) [q0]
    unfolding list-all-iff list.set reach-redef using Nil1 Nil2 by blast

  have reach (set Σ) δ q0 ⊆ reachable {q0}
  proof rule
    fix x
    assume x ∈ reach (set Σ) δ q0
    then obtain w where x-def: x = foldl δ q0 w and set w ⊆ set Σ
      unfolding reach-redef by blast
    hence foldl δ q0 w ∈ reachable {q0}
    proof (induction w arbitrary: x rule: rev-induct)
      case (snoc ν w)
        hence foldl δ q0 w ∈ reachable {q0} and δ (foldl δ q0 w) ν ∈ set (map (δ (foldl δ q0 w)) Σ)
          by simp+
        thus ?case
          by (simp add: rtrancl.rtrancl-into-rtrancl reachable-def)
    qed (simp add: reachable-def)
```

thus $x \in$ *reachable* $\{q_0\}$
by (*simp add*: *x-def*)
qed
moreover
have *reachable* $\{q_0\} \subseteq$ *reach* (*set* Σ) δ q_0
proof *rule*
fix x
assume $x \in$ *reachable* $\{q_0\}$
hence $(q_0, x) \in \{(x, y).\ y \in$ *set* (*map* $(\delta\ x)$ Σ)$\}^*$
unfolding *reachable-def* **by** *blast*
thus $x \in$ *reach* (*set* Σ) δ q_0
apply (*induction*)
apply (*insert reach-redef Nil1 Nil2*; *blast*)
apply (*metis r-into-rtrancl succsr-def succsr-isNode*)
done
qed
ultimately
have *reachable-redef*: *reachable* $\{q_0\}$ = *reach* (*set* Σ) δ q_0
by *blast*

moreover

have *reachable* $\{q_0\} \subseteq Q_L\ \Sigma\ \delta\ q_0$
using *reachable-imp-dfs*[*OF - list-all-init*] **unfolding** *list.set reachable-redef*
unfolding *reach-redef* Q_L*-def* **using** ‹$\Sigma \neq []$› **by** *auto*

moreover

have $Q_L\ \Sigma\ \delta\ q_0 \subseteq$ *reach* (*set* Σ) δ q_0
using *dfs-invariant*[*of* {}, *OF - list-all-init*]
by (*auto simp add*: *reach-redef* Q_L*-def*)

ultimately
show *?thesis*
using ‹$\Sigma \neq []$› *dfs-invariant*[*of* {}, *OF - list-all-init*] **by** *simp+*
qed (*simp add*: *reach-def* Q_L*-def*)

lemma δ_L*-reach*:
assumes *finite* (*reach*$_t$ (*set* Σ) δ q_0)
shows $\delta_L\ \Sigma\ \delta\ q_0$ = *reach*$_t$ (*set* Σ) δ q_0
proof −
{
fix $x\ w\ y$

assume *set* $w \subseteq$ *set* Σ $x =$ *foldl* δ q_0 w $y \in$ *set* (*map* (δ x) Σ)
moreover
then obtain ν **where** $y = \delta$ x ν **and** $\nu \in$ *set* Σ
by *auto*
ultimately
have $y =$ *foldl* δ q_0 (w@[ν]) **and** *set* (w@[ν]) $\subseteq$ *set* Σ
by *simp+*
hence $\exists w'$. *set* $w' \subseteq$ *set* $\Sigma \wedge y =$ *foldl* δ q_0 w'
by *blast*
}
note *extend-run* = *this*

let *?succs* = λ(-, -, q). (*map* ($\lambda\nu$. (q, ν, δ q ν)) Σ)

interpret *DFS* λ(-, -, q). (*map* ($\lambda\nu$. (q, ν, δ q ν)) Σ) λt. $t \in$ $reach_t$ (*set* Σ) δ q_0 λS. *set* $S \subseteq$ $reach_t$ (*set* Σ) δ q_0 *List.insert* λx *xs*. $x \in$ *set xs* [] *id*
apply (*unfold-locales*; *auto simp add*: *member-rec* $reach_t$*-foldl-def list-all-iff elim*: *extend-run*)
apply (*metis extend-run image-eqI set-map*)
using *assms* **unfolding** $reach_t$*-foldl-def* **by** *simp*

have *Nil1*: *set* [] $\subseteq$ *set* Σ **and** *Nil2*: $q_0 =$ *foldl* δ q_0 []
by *simp+*
have *list-all-init*: *list-all* (λq. $q \in$ $reach_t$ (*set* Σ) δ q_0) (*map* ($\lambda\nu$. (q_0, ν, δ q_0 ν)) Σ)
unfolding *list-all-iff* $reach_t$*-foldl-def set-map image-def* **using** *Nil2* **by** *fastforce*

let $?q_0$ = *set* (*map* ($\lambda\nu$. (q_0, ν, δ q_0 ν)) Σ)

{
fix q ν q'
assume (q, ν, q') $\in$ $reach_t$ (*set* Σ) δ q_0
then obtain w **where** *q-def*: $q =$ *foldl* δ q_0 w **and** *q'-def*: $q' =$ *foldl* δ q_0 (w@[ν])
and *set* $w \subseteq$ *set* Σ **and** $\nu \in$ *set* Σ
unfolding $reach_t$*-foldl-def* **by** *blast*
hence (*foldl* δ q_0 w, ν, *foldl* δ q_0 (w@[ν])) $\in$ *reachable* $?q_0$
proof (*induction* w *arbitrary*: q q' ν *rule*: *rev-induct*)
case (*snoc* ν' w)
hence (*foldl* δ q_0 w, ν', *foldl* δ q_0 (w@[ν'])) $\in$ *reachable* $?q_0$ (**is** ($?q$, ν', $?q'$) $\in$ -)
and $\bigwedge q$. δ q $\nu \in$ *set* (*map* (δ q) Σ)
and $\nu \in$ *set* Σ

by *simp+*
then obtain x_0 **where** *1*: $(x_0, (?q, \nu', ?q')) \in \{(x, y).\ y \in set\ (?succs\ x)\}^*$ **and** *2*: $x_0 \in ?q_0$
unfolding *reachable-def* **by** *auto*
moreover
have *3*: $((?q, \nu', ?q'), (?q', \nu, \delta\ ?q'\ \nu)) \in \{(x, y).\ y \in set\ (?succs\ x)\}$
using *snoc* ‹$\bigwedge q.\ \delta\ q\ \nu \in set\ (map\ (\delta\ q)\ \Sigma)$› **by** *simp*
ultimately
show *?case*
using *rtrancl.rtrancl-into-rtrancl*[*OF 1 3*] *2* **unfolding** *reachable-def foldl-append foldl.simps* **by** *auto*
qed (*auto simp add*: *reachable-def*)
hence $(q, \nu, q') \in reachable\ ?q_0$
by (*simp add*: *q-def q'-def*)
}
hence $reach_t\ (set\ \Sigma)\ \delta\ q_0 \subseteq reachable\ ?q_0$
by *auto*
moreover
{
fix *y*
assume $y \in reachable\ ?q_0$
then obtain *x* **where** $(x, y) \in \{(x, y).\ y \in set\ (case\ x\ of\ (\text{-}, \text{-}, q) \Rightarrow map\ (\lambda\nu.\ (q, \nu, \delta\ q\ \nu))\ \Sigma)\}^*$
and $x \in ?q_0$
unfolding *reachable-def* **by** *auto*
hence $y \in reach_t\ (set\ \Sigma)\ \delta\ q_0$
proof *induction*
case *base*
have $\forall p\ ps.\ list\text{-}all\ p\ ps = (\forall pa.\ pa \in set\ ps \longrightarrow p\ pa)$
by (*meson list-all-iff*)
hence $x \in \{(foldl\ \delta\ (foldl\ \delta\ q_0\ [])\ bs, b, foldl\ \delta\ (foldl\ \delta\ q_0\ [])\ (bs\ @\ [b])) \mid bs\ b.\ set\ bs \subseteq set\ \Sigma \wedge b \in set\ \Sigma\}$
using *base* **by** (*metis* (*no-types*) *Nil2 list-all-init* $reach_t$*-foldl-def*)
thus *?case*
unfolding $reach_t$*-foldl-def* **by** *auto*
next
case (*step* x' y')
thus *?case* **using** *succsr-def succsr-isNode* **by** *blast*
qed
}
hence $reachable\ ?q_0 \subseteq reach_t\ (set\ \Sigma)\ \delta\ q_0$
by *blast*
ultimately

have *reachable-redef*: *reachable* $?q_0$ = $reach_t$ (*set* Σ) δ q_0
by *blast*

moreover

have *reachable* $?q_0 \subseteq (\delta_L\ \Sigma\ \delta\ q_0)$
using *reachable-imp-dfs*[*OF* - *list-all-init*] **unfolding** δ_L*-def reachable-redef list-dfs-def*
by (*simp*; *blast*)

moreover

have $\delta_L\ \Sigma\ \delta\ q_0 \subseteq reach_t\ (set\ \Sigma)\ \delta\ q_0$
using *dfs-invariant*[*of* [], *OF* - *list-all-init*]
by (*auto simp add*: $reach_t$*-foldl-def* δ_L*-def list-dfs-def*)

ultimately
show *?thesis*
by *simp*
qed

lemma *reach-reach*$_t$*-fst*:
reach Σ δ q_0 = *fst* ‘ $reach_t$ Σ δ q_0
unfolding $reach_t$*-def reach-def image-def* **by** *fastforce*

lemma *finite-reach*:
finite ($reach_t$ Σ δ q_0) ⟹ *finite* (*reach* Σ δ q_0)
by (*simp add*: *reach-reach*$_t$*-fst*)

lemma *finite-reach*$_t$:
assumes *finite* (*reach* Σ δ q_0)
assumes *finite* Σ
shows *finite* ($reach_t$ Σ δ q_0)
proof –
have $reach_t\ \Sigma\ \delta\ q_0 \subseteq reach\ \Sigma\ \delta\ q_0 \times \Sigma \times reach\ \Sigma\ \delta\ q_0$
unfolding $reach_t$*-def reach-def run*$_t$*.simps* **by** *blast*
thus *?thesis*
using *assms finite-subset* **by** *blast*
qed

lemma Q_L*-eq-*δ_L:
assumes *finite* ($reach_t$ (*set* Σ) δ q_0)
shows $Q_L\ \Sigma\ \delta\ q_0 = fst$ ‘ $(\delta_L\ \Sigma\ \delta\ q_0)$
unfolding *set-map* δ_L*-reach*[*OF assms*] Q_L*-reach*[*OF finite-reach*[*OF assms*]]

reach-reach$_t$-fst ..

4.3 Product of DTS

fun *simple-product* :: $('a, 'c)$ *DTS* $\Rightarrow$ $('b, 'c)$ *DTS* $\Rightarrow$ $('a \times 'b, 'c)$ *DTS* (‹- × -›)
where
$\delta_1 \times \delta_2 = (\lambda(q_1, q_2)\ \nu.\ (\delta_1\ q_1\ \nu, \delta_2\ q_2\ \nu))$

lemma *simple-product-run*:
fixes $\delta_1\ \delta_2\ w\ q_1\ q_2$
defines $\varrho \equiv run\ (\delta_1 \times \delta_2)\ (q_1, q_2)\ w$
defines $\varrho_1 \equiv run\ \delta_1\ q_1\ w$
defines $\varrho_2 \equiv run\ \delta_2\ q_2\ w$
shows $\varrho\ i = (\varrho_1\ i, \varrho_2\ i)$
by (*induction i*) (*insert assms*, *auto*)

theorem *finite-reach-simple-product*:
assumes *finite* (*reach* $\Sigma\ \delta_1\ q_1$)
assumes *finite* (*reach* $\Sigma\ \delta_2\ q_2$)
shows *finite* (*reach* $\Sigma\ (\delta_1 \times \delta_2)\ (q_1, q_2)$)
proof –
have *reach* $\Sigma\ (\delta_1 \times \delta_2)\ (q_1, q_2) \subseteq$ *reach* $\Sigma\ \delta_1\ q_1 \times$ *reach* $\Sigma\ \delta_2\ q_2$
unfolding *reach-def simple-product-run* **by** *blast*
thus *?thesis*
using *assms finite-subset* **by** *blast*
qed

4.4 (Generalised) Product of DTS

fun *product* :: $('a \Rightarrow ('b, 'c)$ *DTS*$) \Rightarrow ('a \rightharpoonup 'b, 'c)$ *DTS* (‹$\Delta_\times$›)
where
$\Delta_\times\ \delta_m = (\lambda q\ \nu.\ (\lambda x.\ case\ q\ x\ of\ None \Rightarrow None \mid Some\ y \Rightarrow Some\ (\delta_m\ x\ y\ \nu)))$

lemma *product-run-None*:
fixes $\iota_m\ \delta_m\ w$
defines $\varrho \equiv run\ (\Delta_\times\ \delta_m)\ \iota_m\ w$
assumes $\iota_m\ k = None$
shows $\varrho\ i\ k = None$
by (*induction i*) (*insert assms*, *auto*)

lemma *product-run-Some*:
fixes $\iota_m\ \delta_m\ w\ q_0\ k$

defines $\varrho \equiv$ *run* $(\Delta_\times\ \delta_m)\ \iota_m\ w$
defines $\varrho' \equiv$ *run* $(\delta_m\ k)\ q_0\ w$
assumes $\iota_m\ k =$ *Some* q_0
shows $\varrho\ i\ k =$ *Some* $(\varrho'\ i)$
by (*induction i*) (*insert assms*, *auto*)

theorem *finite-reach-product*:
assumes *finite* (*dom* ι_m)
assumes $\bigwedge x.\ x \in$ *dom* $\iota_m \Longrightarrow$ *finite* (*reach* Σ $(\delta_m\ x)$ (*the* $(\iota_m\ x)$))
shows *finite* (*reach* Σ $(\Delta_\times\ \delta_m)\ \iota_m$)
using *assms*(*1*,*2*)
proof (*induction dom* ι_m *arbitrary*: ι_m)
case *empty*
hence $\iota_m =$ *Map.empty*
by *auto*
hence $\bigwedge w\ i.$ *run* $(\Delta_\times\ \delta_m)\ \iota_m\ w\ i = (\lambda x.$ *None*)
using *product-run-None* **by** *fast*
thus *?case*
unfolding *reach-def* **by** *simp*
next
case (*insert k K*)
define *f* **where** $f = (\lambda(q :: 'b,\ m :: 'a \rightharpoonup 'b).\ m(k :=$ *Some* $q))$
define *Reach* **where** *Reach* = (*reach* Σ $(\delta_m\ k)$ (*the* $(\iota_m\ k)$)) $\times$ ((*reach* Σ $(\Delta_\times\ \delta_m)$ $(\iota_m(k :=$ *None*))))

have (*reach* Σ $(\Delta_\times\ \delta_m)\ \iota_m) \subseteq f$ ` *Reach*
proof
fix x
assume $x \in$ *reach* Σ $(\Delta_\times\ \delta_m)\ \iota_m$
then obtain $w\ n$ **where** *x-def*: $x =$ *run* $(\Delta_\times\ \delta_m)\ \iota_m\ w\ n$ **and** *range* $w \subseteq \Sigma$
unfolding *reach-def* **by** *blast*
{
fix k'
have $k' \notin$ *dom* $\iota_m \Longrightarrow x\ k' =$ *run* $(\Delta_\times\ \delta_m)$ $(\iota_m(k :=$ *None*)) $w\ n\ k'$
unfolding *x-def dom-def* **using** *product-run-None*[*of - -* δ_m] **by** *simp*
moreover
have $k' \in$ *dom* $\iota_m - \{k\} \Longrightarrow x\ k' =$ *run* $(\Delta_\times\ \delta_m)$ $(\iota_m(k :=$ *None*)) $w\ n\ k'$
unfolding *x-def dom-def* **using** *product-run-Some*[*of - - -* δ_m] **by** *auto*
ultimately
have $k' \neq k \Longrightarrow x\ k' =$ *run* $(\Delta_\times\ \delta_m)$ $(\iota_m(k :=$ *None*)) $w\ n\ k'$

by *blast*
}
hence $x(k := None) = run\ (\Delta_\times\ \delta_m)\ (\iota_m(k := None))\ w\ n$
using *product-run-None*[*of* - - δ_m] **by** *auto*
moreover
have $x\ k = Some\ (run\ (\delta_m\ k)\ (the\ (\iota_m\ k))\ w\ n)$
unfolding *x-def* **using** *product-run-Some*[*of* ι_m *k* - δ_m] *insert.hyps*(*4*)
by *force*
ultimately
have $(the\ (x\ k),\ x(k := None)) \in Reach$
unfolding *Reach-def reach-def* **using** ‹*range* $w \subseteq \Sigma$› **by** *auto*
moreover
have $x = f\ (the\ (x\ k),\ x(k := None))$
unfolding *f-def* **using** ‹$x\ k = Some\ (run\ (\delta_m\ k)\ (the\ (\iota_m\ k))\ w\ n)$›
by *auto*
ultimately
show $x \in f$ ‘ *Reach*
by *simp*
qed
moreover
have *finite* $(reach\ \Sigma\ (\Delta_\times\ \delta_m)\ (\iota_m\ (k := None)))$
using *insert insert*(*3*)[*of* ι_m (*k*:=*None*)] **by** *auto*
hence *finite Reach*
using *insert Reach-def* **by** *blast*
hence *finite* (*f* ‘ *Reach*)
..
ultimately
show *?case*
by (*rule finite-subset*)
qed

4.5 Simple Product Construction Helper Functions and Lemmas

fun *embed-transition-fst* :: $('a,\ 'c)$ *transition* $\Rightarrow$ $('a \times 'b,\ 'c)$ *transition set*
where
embed-transition-fst $(q,\ \nu,\ q') = \{((q,\ x),\ \nu,\ (q',\ y)) \mid x\ y.\ True\}$

fun *embed-transition-snd* :: $('b,\ 'c)$ *transition* $\Rightarrow$ $('a \times 'b,\ 'c)$ *transition set*
where
embed-transition-snd $(q,\ \nu,\ q') = \{((x,\ q),\ \nu,\ (y,\ q')) \mid x\ y.\ True\}$

lemma *embed-transition-snd-unfold*:
embed-transition-snd $t = \{((x,\ fst\ t),\ fst\ (snd\ t),\ (y,\ snd\ (snd\ t))) \mid x\ y.$

True}
  unfolding *embed-transition-snd.simps*[*symmetric*] **by** *simp*

fun *project-transition-fst* :: ($'a \times 'b$, $'c$) *transition* ⇒ ($'a$, $'c$) *transition*
where
  project-transition-fst (x, ν, y) = (*fst* x, ν, *fst* y)

fun *project-transition-snd* :: ($'a \times 'b$, $'c$) *transition* ⇒ ($'b$, $'c$) *transition*
where
  project-transition-snd (x, ν, y) = (*snd* x, ν, *snd* y)

lemma
  fixes δ_1 δ_2 w q_1 q_2
  defines $\varrho \equiv run_t$ ($\delta_1 \times \delta_2$) (q_1, q_2) w
  defines $\varrho_1 \equiv run_t$ δ_1 q_1 w
  defines $\varrho_2 \equiv run_t$ δ_2 q_2 w
  shows *product-run-project-fst*: *project-transition-fst* (ϱ i) = ϱ_1 i
    and *product-run-project-snd*: *project-transition-snd* (ϱ i) = ϱ_2 i
    and *product-run-embed-fst*: ϱ i ∈ *embed-transition-fst* (ϱ_1 i)
    and *product-run-embed-snd*: ϱ i ∈ *embed-transition-snd* (ϱ_2 i)
  unfolding *assms* run_t*.simps* *simple-product-run* **by** *simp-all*

lemma
  fixes δ_1 δ_2 w q_1 q_2
  defines $\varrho \equiv run_t$ ($\delta_1 \times \delta_2$) (q_1, q_2) w
  defines $\varrho_1 \equiv run_t$ δ_1 q_1 w
  defines $\varrho_2 \equiv run_t$ δ_2 q_2 w
  assumes *finite* (*range* ϱ)
  shows *product-run-finite-fst*: *finite* (*range* ϱ_1)
    and *product-run-finite-snd*: *finite* (*range* ϱ_2)
proof −
  have $\bigwedge k$. *project-transition-fst* (ϱ k) = ϱ_1 k
    and $\bigwedge k$. *project-transition-snd* (ϱ k) = ϱ_2 k
   unfolding *assms* *product-run-project-fst* *product-run-project-snd* **by** *simp+*
  hence *project-transition-fst* ‘ *range* ϱ = *range* ϱ_1
    and *project-transition-snd* ‘ *range* ϱ = *range* ϱ_2
    using *range-composition*[*symmetric*, *of* *project-transition-fst* ϱ]
   using *range-composition*[*symmetric*, *of* *project-transition-snd* ϱ] **by** *presburger+*
  thus *finite* (*range* ϱ_1) **and** *finite* (*range* ϱ_2)
    using *assms* *finite-imageI* **by** *metis+*
qed

lemma

fixes δ_1 δ_2 w q_1 q_2
defines $\varrho \equiv run_t$ $(\delta_1 \times \delta_2)$ (q_1, q_2) w
defines $\varrho_1 \equiv run_t$ δ_1 q_1 w
assumes *finite* (*range* ϱ)
shows *product-run-project-limit-fst*: *project-transition-fst* ' *limit* ϱ = *limit* ϱ_1
and *product-run-embed-limit-fst*: *limit* $\varrho \subseteq \bigcup$ (*embed-transition-fst* ' (*limit* ϱ_1))
proof −
have *finite* (*range* ϱ_1)
using *assms product-run-finite-fst* **by** *metis*

then obtain *i* **where** *limit* ϱ = *range* (*suffix i* ϱ) **and** *limit* ϱ_1 = *range* (*suffix i* ϱ_1)
using *common-range-limit assms* **by** *metis*
moreover
have $\bigwedge k$. *project-transition-fst* (*suffix i* ϱ *k*) = (*suffix i* ϱ_1 *k*)
by (*simp only*: *assms* run_t*.simps*) (*metis* ϱ_1*-def product-run-project-fst suffix-nth*)
hence *project-transition-fst* ' *range* (*suffix i* ϱ) = (*range* (*suffix i* ϱ_1))
using *range-composition*[*symmetric*, *of project-transition-fst suffix i* ϱ] **by** *presburger*
moreover
have $\bigwedge k$. (*suffix i* ϱ *k*) $\in$ *embed-transition-fst* (*suffix i* ϱ_1 *k*)
using *assms product-run-embed-fst* **by** *simp*
ultimately
show *project-transition-fst* ' *limit* ϱ = *limit* ϱ_1
and *limit* $\varrho \subseteq \bigcup$ (*embed-transition-fst* ' (*limit* ϱ_1))
by *auto*
qed

lemma
fixes δ_1 δ_2 w q_1 q_2
defines $\varrho \equiv run_t$ $(\delta_1 \times \delta_2)$ (q_1, q_2) w
defines $\varrho_2 \equiv run_t$ δ_2 q_2 w
assumes *finite* (*range* ϱ)
shows *product-run-project-limit-snd*: *project-transition-snd* ' *limit* ϱ = *limit* ϱ_2
and *product-run-embed-limit-snd*: *limit* $\varrho \subseteq \bigcup$ (*embed-transition-snd* ' (*limit* ϱ_2))
proof −
have *finite* (*range* ϱ_2)
using *assms product-run-finite-snd* **by** *metis*

then obtain i **where** *limit* ϱ = *range* (*suffix* i ϱ) **and** *limit* ϱ_2 = *range* (*suffix* i ϱ_2)
using *common-range-limit assms* **by** *metis*
moreover
have $\bigwedge k$. *project-transition-snd* (*suffix* i ϱ k) = (*suffix* i ϱ_2 k)
by (*simp only*: *assms* run_t*.simps*) (*metis* ϱ_2*-def product-run-project-snd suffix-nth*)
hence *project-transition-snd* ‘ *range* ((*suffix* i ϱ)) = (*range* (*suffix* i ϱ_2))
using *range-composition*[*symmetric*, *of project-transition-snd* (*suffix* i ϱ)] **by** *presburger*
moreover
have $\bigwedge k$. (*suffix* i ϱ k) ∈ *embed-transition-snd* (*suffix* i ϱ_2 k)
using *assms product-run-embed-snd* **by** *simp*
ultimately
show *project-transition-snd* ‘ *limit* ϱ = *limit* ϱ_2
and *limit* $\varrho \subseteq \bigcup$ (*embed-transition-snd* ‘ (*limit* ϱ_2))
by *auto*
qed

lemma
fixes δ_1 δ_2 w q_1 q_2
defines $\varrho \equiv run_t$ $(\delta_1 \times \delta_2)$ (q_1, q_2) w
defines $\varrho_1 \equiv run_t$ δ_1 q_1 w
defines $\varrho_2 \equiv run_t$ δ_2 q_2 w
assumes *finite* (*range* ϱ)
shows *product-run-embed-limit-finiteness-fst*: *limit* $\varrho \cap (\bigcup$ (*embed-transition-fst* ‘ S)) = {} ⟷ *limit* $\varrho_1 \cap S$ = {} (**is** *?thesis1*)
and *product-run-embed-limit-finiteness-snd*: *limit* $\varrho \cap (\bigcup$ (*embed-transition-snd* ‘ S')) = {} ⟷ *limit* $\varrho_2 \cap S'$ = {} (**is** *?thesis2*)
proof −
show *?thesis1*
using *assms product-run-project-limit-fst* **by** *fastforce*
show *?thesis2*
using *assms product-run-project-limit-snd* **by** *fastforce*
qed

4.6 Product Construction Helper Functions and Lemmas

fun *embed-transition* :: $'a \Rightarrow ('b, 'c)$ *transition* $\Rightarrow ('a \rightharpoonup 'b, 'c)$ *transition set* (‹↿-›)
where
$\upharpoonleft_x$ $(q, \nu, q') = \{(m, \nu, m') \mid m\ m'.\ m\ x = Some\ q \wedge m'\ x = Some\ q'\}$

fun *project-transition* :: $'a \Rightarrow ('a \rightharpoonup 'b, 'c)$ *transition* $\Rightarrow ('b, 'c)$ *transition*

$(\langle\downharpoonright_{-}\rangle)$
where
  $\downharpoonright_x$ $(m,\ \nu,\ m') = (\mathit{the}\ (m\ x),\ \nu,\ \mathit{the}\ (m'\ x))$

fun *embed-pair* :: $'a \Rightarrow (('b,\ 'c)\ \mathit{transition\ set} \times ('b,\ 'c)\ \mathit{transition\ set}) \Rightarrow (('a \rightharpoonup 'b,\ 'c)\ \mathit{transition\ set} \times ('a \rightharpoonup 'b,\ 'c)\ \mathit{transition\ set})$ $(\langle\upharpoonleft_{-}\rangle)$
where
  $\upharpoonleft_x (S,\ S') = (\bigcup(\upharpoonleft_x\ `\ S),\ \bigcup(\upharpoonleft_x\ `\ S'))$

fun *project-pair* :: $'a \Rightarrow (('a \rightharpoonup 'b,\ 'c)\ \mathit{transition\ set} \times ('a \rightharpoonup 'b,\ 'c)\ \mathit{transition\ set}) \Rightarrow (('b,\ 'c)\ \mathit{transition\ set} \times ('b,\ 'c)\ \mathit{transition\ set})$ $(\langle\downharpoonright_{-}\rangle)$
where
  $\downharpoonright_x (S,\ S') = (\downharpoonright_x\ `\ S,\ \downharpoonright_x\ `\ S')$

lemma *embed-transition-unfold*:
  $\mathit{embed\text{-}transition}\ x\ t = \{(m,\ \mathit{fst}\ (\mathit{snd}\ t),\ m') \mid m\ m'.\ m\ x = \mathit{Some}\ (\mathit{fst}\ t) \wedge m'\ x = \mathit{Some}\ (\mathit{snd}\ (\mathit{snd}\ t))\}$
  unfolding *embed-transition.simps*[*symmetric*] **by** *simp*

lemma
  fixes $\iota_m\ \delta_m\ w\ q_0$
  fixes $x :: 'a$
  defines $\varrho \equiv \mathit{run}_t\ (\Delta_\times\ \delta_m)\ \iota_m\ w$
  defines $\varrho' \equiv \mathit{run}_t\ (\delta_m\ x)\ q_0\ w$
  assumes $\iota_m\ x = \mathit{Some}\ q_0$
  shows *product-run-project*: $\downharpoonright_x (\varrho\ i) = \varrho'\ i$
    and *product-run-embed*: $\varrho\ i \in \upharpoonleft_x (\varrho'\ i)$
  using *assms product-run-Some*[*of* - - - δ_m] **by** *simp+*

lemma
  fixes $\iota_m\ \delta_m\ w\ q_0\ x$
  defines $\varrho \equiv \mathit{run}_t\ (\Delta_\times\ \delta_m)\ \iota_m\ w$
  defines $\varrho' \equiv \mathit{run}_t\ (\delta_m\ x)\ q_0\ w$
  assumes $\iota_m\ x = \mathit{Some}\ q_0$
  assumes *finite* (*range* ϱ)
  shows *product-run-project-limit*: $\downharpoonright_x\ `\ \mathit{limit}\ \varrho = \mathit{limit}\ \varrho'$
    and *product-run-embed-limit*: $\mathit{limit}\ \varrho \subseteq \bigcup\ (\upharpoonleft_x\ `\ (\mathit{limit}\ \varrho'))$
proof –
  have $\bigwedge k.\ \downharpoonright_x (\varrho\ k) = \varrho'\ k$
    using *assms product-run-embed*[*of* - - - δ_m] **by** *simp*
  hence $\downharpoonright_x\ `\ \mathit{range}\ \varrho = \mathit{range}\ \varrho'$
    using *range-composition*[*symmetric*, *of* $\downharpoonright_x\ \varrho$] **by** *presburger*
  hence *finite* (*range* ϱ')
    using *assms finite-imageI* **by** *metis*

then obtain i **where** *limit* ϱ = *range* (*suffix* i ϱ) **and** *limit* ϱ' = *range* (*suffix* i ϱ')
using *common-range-limit assms* **by** *metis*
moreover
have $\bigwedge k$. $\downharpoonleft_x$ (*suffix* i ϱ k) = (*suffix* i ϱ' k)
using *assms product-run-embed*[*of* - - - δ_m] **by** *simp*
hence $\downharpoonleft_x$ ' *range* ((*suffix* i ϱ)) = (*range* (*suffix* i ϱ'))
using *range-composition*[*symmetric*, *of* $\downharpoonleft_x$ (*suffix* i ϱ)] **by** *presburger*
moreover
have $\bigwedge k$. (*suffix* i ϱ k) $\in$ $\upharpoonleft_x$ (*suffix* i ϱ' k)
using *assms product-run-embed*[*of* - - - δ_m] **by** *simp*
ultimately
show $\downharpoonleft_x$ ' *limit* ϱ = *limit* ϱ' **and** *limit* $\varrho \subseteq \bigcup$ ($\upharpoonleft_x$ ' (*limit* ϱ'))
by *auto*
qed

lemma *product-run-embed-limit-finiteness*:
fixes ι_m δ_m w q_0 k
defines $\varrho \equiv run_t$ ($\Delta_\times$ δ_m) ι_m w
defines $\varrho' \equiv run_t$ (δ_m k) q_0 w
assumes ι_m k = *Some* q_0
assumes *finite* (*range* ϱ)
shows *limit* $\varrho \cap$ ($\bigcup$ ($\upharpoonleft_k$ ' S)) = {} $\longleftrightarrow$ *limit* $\varrho' \cap S$ = {}
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
have $\downharpoonleft_k$ ' *limit* $\varrho \cap S \neq$ {} $\longrightarrow$ *limit* $\varrho \cap$ ($\bigcup$ ($\upharpoonleft_k$ ' S)) $\neq$ {}
proof
assume $\downharpoonleft_k$ ' *limit* $\varrho \cap S \neq$ {}
then obtain q ν q' **where** (q, ν, q') $\in \downharpoonleft_k$ ' *limit* ϱ **and** (q, ν, q') $\in S$
by *auto*
moreover
have $\bigwedge m$ ν m' i. (m, ν, m') = ϱ $i \Longrightarrow \exists p\ p'$. m k = *Some* $p \wedge m'$ k = *Some* p'
using *assms product-run-Some*[*of* ι_m , *OF assms*(*3*)] **by** *auto*
hence $\bigwedge m$ ν m'. (m, ν, m') $\in$ *limit* $\varrho \Longrightarrow \exists p\ p'$. m k = *Some* $p \wedge m'$ k = *Some* p'
using *limit-in-range* **by** *fast*
ultimately
obtain m m' **where** m k = *Some* q **and** m' k = *Some* q' **and** (m, ν, m') $\in$ *limit* ϱ
by *auto*
moreover
hence (m, ν, m') $\in \bigcup$ ($\upharpoonleft_k$ ' S)

using ‹$(q, \nu, q') \in S$› **by** *force*
ultimately
show *limit* $\varrho \cap (\bigcup (\upharpoonleft_k$ ‘ $S)) \neq \{\}$
by *blast*
qed
hence *?lhs* $\longleftrightarrow \downharpoonleft_k$ ‘ *limit* $\varrho \cap S = \{\}$
by *auto*
also
have ... $\longleftrightarrow$ *?rhs*
using *assms product-run-project-limit*[*of* - - - δ_m] **by** *simp*
finally
show *?thesis*
by *simp*
qed

4.7 Transfer Rules

context includes *lifting-syntax*
begin

lemma *product-parametric* [*transfer-rule*]:
((*A* ===> *B* ===> *C* ===> *B*) ===> (*A* ===> *rel-option B*) ===> *C* ===> *A* ===> *rel-option B*) *product product*
by (*auto simp add*: *rel-fun-def rel-option-iff split*: *option.split*)

lemma *run-parametric* [*transfer-rule*]:
((*A* ===> *B* ===> *A*) ===> *A* ===> ((=) ===> *B*) ===> (=) ===> *A*) *run run*
proof −
{
fix δ δ' q q' n w
fix w' :: *nat* ⇒ *'d*
assume (*A* ===> *B* ===> *A*) δ δ' *A* q q' ((=) ===> *B*) w w'
hence *A* (*run* δ q w n) (*run* δ' q' w' n)
by (*induction n*) (*simp-all add*: *rel-fun-def*)
}
thus *?thesis*
by *blast*
qed

lemma *reach-parametric* [*transfer-rule*]:
assumes *bi-total B*
assumes *bi-unique B*
shows (*rel-set B* ===> (*A* ===> *B* ===> *A*) ===> *A* ===> *rel-set*

```
A) reach reach
proof standard+
  fix Σ Σ' δ δ' q q'
  assume rel-set B Σ Σ' (A ===> B ===> A) δ δ' A q q'

  {
    fix z
    assume z ∈ reach Σ δ q
    then obtain w n where z = run δ q w n and range w ⊆ Σ
      unfolding reach-def by auto

    define w' where w' n = (SOME x. B (w n) x) for n

    have ⋀n. w n ∈ Σ
      using ‹range w ⊆ Σ› by blast
    hence ⋀n. w' n ∈ Σ'
        using assms ‹rel-set B Σ Σ'› by (simp add: w'-def bi-unique-def
rel-set-def; metis someI)
    hence run δ' q' w' n ∈ reach Σ' δ' q'
      unfolding reach-def by auto

    moreover

    have A z (run δ' q' w' n)
      apply (unfold ‹z = run δ q w n›)
      apply (insert ‹A q q'› ‹(A ===> B ===> A) δ δ'› assms(1))
      apply (induction n)
      apply (simp-all add: rel-fun-def bi-total-def w'-def)
      by (metis tfl-some)

    ultimately

    have ∃z' ∈ reach Σ' δ' q'. A z z'
      by blast
  }

  moreover

  {
    fix z
    assume z ∈ reach Σ' δ' q'
    then obtain w n where z = run δ' q' w n and range w ⊆ Σ'
      unfolding reach-def by auto
```

define w' **where** $w'\ n = (SOME\ x.\ B\ x\ (w\ n))$ **for** n

have $\bigwedge n.\ w\ n \in \Sigma'$
  using ‹*range* $w \subseteq \Sigma'$› **by** *blast*
hence $\bigwedge n.\ w'\ n \in \Sigma$
  using *assms* ‹*rel-set* $B\ \Sigma\ \Sigma'$› **by** (*simp add*: w'-*def* *bi-unique-def* *rel-set-def*; *metis someI*)
hence *run* $\delta\ q\ w'\ n \in$ *reach* $\Sigma\ \delta\ q$
  unfolding *reach-def* **by** *auto*

moreover

have A (*run* $\delta\ q\ w'\ n$) z
  apply (*unfold* ‹$z =$ *run* $\delta'\ q'\ w\ n$›)
  apply (*insert* ‹$A\ q\ q'$› ‹$(A$ ===> B ===> $A)\ \delta\ \delta'$› *assms*(*1*))
  apply (*induction* n)
  apply (*simp-all add*: *rel-fun-def* *bi-total-def* w'-*def*)
  by (*metis tfl-some*)

ultimately

have $\exists\, z' \in$ *reach* $\Sigma\ \delta\ q.\ A\ z'\ z$
  by *blast*
}
ultimately
show *rel-set* A (*reach* $\Sigma\ \delta\ q$) (*reach* $\Sigma'\ \delta'\ q'$)
  unfolding *rel-set-def* **by** *blast*
qed

end

4.8 Lift to Mapping

lift-definition *product-abs* :: $('a \Rightarrow ('b, 'c)\ DTS) \Rightarrow (('a, 'b)\ mapping, 'c)\ DTS$ (‹$\uparrow\Delta_\times$›) **is** *product*
  parametric *product-parametric* **.**

lemma *product-abs-run-None*:
  Mapping.lookup $\iota_m\ k = None \Longrightarrow$ *Mapping.lookup* (*run* ($\uparrow\Delta_\times\ \delta_m$) $\iota_m\ w\ i$) $k = None$
  by (*transfer*; *insert product-run-None*)

lemma *product-abs-run-Some*:
  Mapping.lookup $\iota_m\ k = Some\ q_0 \Longrightarrow$ *Mapping.lookup* (*run* ($\uparrow\Delta_\times\ \delta_m$) ι_m

$w\ i)\ k = Some\ (run\ (\delta_m\ k)\ q_0\ w\ i)$
by (*transfer*; *insert product-run-Some*)

theorem *finite-reach-product-abs*:
assumes *finite* (*Mapping.keys* ι_m)
assumes $\bigwedge x.\ x \in (Mapping.keys\ \iota_m) \Longrightarrow finite\ (reach\ \Sigma\ (\delta_m\ x)\ (the\ (Mapping.lookup\ \iota_m\ x)))$
shows *finite* (*reach* Σ ($\uparrow\Delta_\times\ \delta_m$) ι_m)
using *assms* **by** (*transfer*; *blast intro*: *finite-reach-product*)

end

5 Mojmir Automata (Without Final States)

theory *Semi-Mojmir*
imports *Main Auxiliary/Preliminaries2 DTS*
begin

5.1 Definitions

locale *semi-mojmir-def* =
fixes
— Alphapet
Σ :: *'a set*
fixes
— Transition Function
δ :: (*'b*, *'a*) *DTS*
fixes
— Initial State
q_0 :: *'b*
fixes
— ω-Word
w :: *'a word*
begin

definition *sink* :: *'b* ⇒ *bool*
where
$sink\ q \equiv (q_0 \neq q) \wedge (\forall \nu \in \Sigma.\ \delta\ q\ \nu = q)$

declare *sink-def* [*code*]

fun *token-run* :: *nat* ⇒ *nat* ⇒ *'b*
where
$token\text{-}run\ x\ n = run\ \delta\ q_0\ (suffix\ x\ w)\ (n - x)$

fun *configuration* :: $'b \Rightarrow nat \Rightarrow nat\ set$
where
$configuration\ q\ n = \{x.\ x \leq n \wedge token\text{-}run\ x\ n = q\}$

fun *oldest-token* :: $'b \Rightarrow nat \Rightarrow nat\ option$
where
$oldest\text{-}token\ q\ n = (if\ configuration\ q\ n \neq \{\}\ then\ Some\ (Min\ (configuration\ q\ n))\ else\ None)$

fun *senior* :: $nat \Rightarrow nat \Rightarrow nat$
where
$senior\ x\ n = the\ (oldest\text{-}token\ (token\text{-}run\ x\ n)\ n)$

fun *older-seniors* :: $nat \Rightarrow nat \Rightarrow nat\ set$
where
$older\text{-}seniors\ x\ n = \{s.\ \exists\, y.\ s = senior\ y\ n \wedge s < senior\ x\ n \wedge \neg\ sink\ (token\text{-}run\ s\ n)\}$

fun *rank* :: $nat \Rightarrow nat \Rightarrow nat\ option$
where
$rank\ x\ n =$
$(if\ x \leq n \wedge \neg sink\ (token\text{-}run\ x\ n)\ then\ Some\ (card\ (older\text{-}seniors\ x\ n))\ else\ None)$

fun *senior-states* :: $'b \Rightarrow nat \Rightarrow 'b\ set$
where
$senior\text{-}states\ q\ n =$
$\{p.\ \exists\, x\ y.\ oldest\text{-}token\ p\ n = Some\ y \wedge oldest\text{-}token\ q\ n = Some\ x \wedge y < x \wedge \neg\ sink\ p\}$

fun *state-rank* :: $'b \Rightarrow nat \Rightarrow nat\ option$
where
$state\text{-}rank\ q\ n = (if\ configuration\ q\ n \neq \{\} \wedge \neg sink\ q\ then\ Some\ (card\ (senior\text{-}states\ q\ n))\ else\ None)$

definition *max-rank* :: nat
where
$max\text{-}rank = card\ (reach\ \Sigma\ \delta\ q_0 - \{q.\ sink\ q\})$

5.1.1 Iterative Computation of State-Ranks

fun *initial* :: $'b \Rightarrow nat\ option$
where

initial $q = (\textit{if } q = q_0 \textit{ then Some 0 else None})$

fun *pre-ranks* :: $('b \Rightarrow \textit{nat option}) \Rightarrow 'a \Rightarrow 'b \Rightarrow \textit{nat set}$
where
pre-ranks $r\ \nu\ q = \{i\ .\ \exists\, q'.\ r\ q' = \textit{Some } i \wedge q = \delta\ q'\ \nu\} \cup (\textit{if } q = q_0 \textit{ then } \{\textit{max-rank}\} \textit{ else } \{\})$

fun *step* :: $('b \Rightarrow \textit{nat option}) \Rightarrow 'a \Rightarrow ('b \Rightarrow \textit{nat option})$
where
step $r\ \nu\ q = ($
if
$\neg \textit{sink } q \wedge \textit{pre-ranks } r\ \nu\ q \neq \{\}$
then
Some $(\textit{card } \{q'.\ \neg \textit{sink } q' \wedge \textit{pre-ranks } r\ \nu\ q' \neq \{\} \wedge \textit{Min } (\textit{pre-ranks } r\ \nu\ q') < \textit{Min } (\textit{pre-ranks } r\ \nu\ q)\})$
else
None)

5.1.2 Properties of Tokens

definition *token-squats* :: $\textit{nat} \Rightarrow \textit{bool}$
where
token-squats $x = (\forall\, n.\ \neg \textit{sink } (\textit{token-run } x\ n))$

end

locale *semi-mojmir* = *semi-mojmir-def* +
assumes
— The alphabet is finite. Non-emptiness is derived from well-formed w
finite-Σ: *finite* Σ
assumes
— The set of reachable states is finite
finite-reach: *finite* $(\textit{reach } \Sigma\ \delta\ q_0)$
assumes
— w only contains letters from the alphabet
bounded-w: *range* $w \subseteq \Sigma$
begin

lemma *nonempty-*Σ: $\Sigma \neq \{\}$
using *bounded-w* **by** *blast*

lemma *bounded-w'*: $w\ i \in \Sigma$
using *bounded-w* **by** *blast*

— Naming Scheme:
This theory uses the following naming scheme to consistently name variables.
* Tokens: x, y, z * Time: n, m * Rank: i, j, k * States: p, q

lemma *sink-rev-step*:
  $\neg sink\ q \Longrightarrow q = \delta\ q'\ \nu \Longrightarrow \nu \in \Sigma \Longrightarrow \neg sink\ q'$
  $\neg sink\ q \Longrightarrow q = \delta\ q'\ (w\ i) \Longrightarrow \neg sink\ q'$
  using *bounded-w'* **by** (*force simp only*: *sink-def*)+

5.2 Token Run

lemma *token-stays-in-sink*:
  assumes *sink q*
  assumes *token-run x n = q*
  shows *token-run x (n + m) = q*
proof (*cases x ≤ n*)
  case *True*
    show *?thesis*
    proof (*induction m*)
      case *0*
        show *?case*
          using *assms*(*2*) **by** *simp*
    next
      case (*Suc m*)
        have *x ≤ n + m*
          using *True* **by** *simp*
        moreover
        have $\bigwedge x.\ w\ x \in \Sigma$
          using *bounded-w* **by** *auto*
        ultimately
        have $\bigwedge t.$ *token-run x (n + m) = q* $\Longrightarrow$ *token-run x (n + m + 1)*
= *q*
         using ‹*sink q*›[*unfolded sink-def*] *upt-add-eq-append*[*OF le0*, *of n* +
m 1]
          using *Suc-diff-le* **by** *simp*
        with *Suc* **show** *?case*
          by *simp*
    qed
qed (*insert assms*, *simp add*: *sink-def*)

lemma *token-is-not-in-sink*:
  token-run x n $\notin A \Longrightarrow$ *token-run x (Suc n)* $\in A \Longrightarrow \neg$*sink (token-run x
n)*
  by (*metis Suc-eq-plus1 token-stays-in-sink*)

lemma *token-run-intial-state*:
token-run x x = q_0
by *simp*

lemma *token-run-P*:
assumes ¬ *P* (*token-run x n*)
assumes *P* (*token-run x* (*Suc* (*n* + *m*)))
shows ∃ *m′* ≤ *m*. ¬ *P* (*token-run x* (*n* + *m′*)) ∧ *P* (*token-run x* (*Suc* (*n* + *m′*)))
using *assms* **by** (*induction m*) (*simp-all*, *metis add-Suc-right le-Suc-eq*)

lemma *token-run-merge-Suc*:
assumes *x* ≤ *n*
assumes *y* ≤ *n*
assumes *token-run x n* = *token-run y n*
shows *token-run x* (*Suc n*) = *token-run y* (*Suc n*)
proof −
have *run* δ q_0 (*suffix x w*) (*Suc* (*n* − *x*)) = *run* δ q_0 (*suffix y w*) (*Suc* (*n* − *y*))
using *assms* **by** *fastforce*
thus *?thesis*
using *Suc-diff-le assms*(*1*,*2*) **by** *force*
qed

lemma *token-run-merge*:
⟦*x* ≤ *n*; *y* ≤ *n*; *token-run x n* = *token-run y n*⟧ ⟹ *token-run x* (*n* + *m*) = *token-run y* (*n* + *m*)
using *token-run-merge-Suc*[*of x - y*] **by** (*induction m*) *auto*

lemma *token-run-mergepoint*:
assumes *x* < *y*
assumes *token-run x* (*y* + *n*) = *token-run y* (*y* + *n*)
obtains *m* **where** *x* ≤ (*Suc m*) **and** *y* ≤ (*Suc m*)
and *y* = *Suc m* ∨ *token-run x m* ≠ *token-run y m*
and *token-run x* (*Suc m*) = *token-run y* (*Suc m*)
using *assms* **by** (*induction n*)
((*metis add-0-iff le-Suc-eq le-add1 less-imp-Suc-add*),
(*metis add-Suc-right le-add1 less-or-eq-imp-le order-trans*))

5.2.1 Step Lemmas

lemma *token-run-step*:
assumes *x* ≤ *n*

assumes *token-run* $x\ n = q'$
assumes $q = \delta\ q'\ (w\ n)$
shows *token-run* x (*Suc* n) $= q$
using *assms* **unfolding** *token-run.simps Suc-diff-le*[*OF* ‹$x \leq n$›] **by** *force*

lemma *token-run-step'*:
$x \leq n \Longrightarrow$ *token-run* x (*Suc* n) $= \delta$ (*token-run* $x\ n$) ($w\ n$)
using *token-run-step* **by** *simp*

5.3 Configuration

5.3.1 Properties

lemma *configuration-distinct*:
$q \neq q' \Longrightarrow$ *configuration* $q\ n \cap$ *configuration* $q'\ n = \{\}$
by *auto*

lemma *configuration-finite*:
finite (*configuration* $q\ n$)
by *simp*

lemma *configuration-non-empty*:
$x \leq n \Longrightarrow$ *configuration* (*token-run* $x\ n$) $n \neq \{\}$
by *fastforce*

lemma *configuration-token*:
$x \leq n \Longrightarrow x \in$ *configuration* (*token-run* $x\ n$) n
by *fastforce*

lemmas *configuration-Max-in* = *Max-in*[*OF configuration-finite*]
lemmas *configuration-Min-in* = *Min-in*[*OF configuration-finite*]

5.3.2 Monotonicity

lemma *configuration-monotonic-Suc*:
$x \leq n \Longrightarrow$ *configuration* (*token-run* $x\ n$) $n \subseteq$ *configuration* (*token-run* x (*Suc* n)) (*Suc* n)
proof
fix y
assume $y \in$ *configuration* (*token-run* $x\ n$) n
hence $y \leq n$ **and** *token-run* $x\ n =$ *token-run* $y\ n$
by *simp-all*
moreover
assume $x \leq n$
ultimately

have *token-run* x (*Suc* n) = *token-run* y (*Suc* n)
using *token-run-merge-Suc* **by** *blast*
thus $y \in$ *configuration* (*token-run* x (*Suc* n)) (*Suc* n)
using *configuration-token* ‹$y \leq n$› **by** *simp*
qed

5.3.3 Pull-Up and Push-Down

lemma *pull-up-token-run-tokens*:
⟦$x \leq n$; $y \leq n$; *token-run* x n = *token-run* y n⟧ $\Longrightarrow \exists\, q.\ x \in$ *configuration* q n $\wedge$ $y \in$ *configuration* q n
by *force*

lemma *push-down-configuration-token-run*:
⟦$x \in$ *configuration* q n; $y \in$ *configuration* q n⟧ $\Longrightarrow x \leq n \wedge y \leq n \wedge$ *token-run* x n = *token-run* y n
by *simp*

5.3.4 Step Lemmas

lemma *configuration-step*:
$x \in$ *configuration* q' $n \Longrightarrow q = \delta\ q'\ (w\ n) \Longrightarrow x \in$ *configuration* q (*Suc* n)
using *Suc-diff-le* **by** *simp*

lemma *configuration-step-non-empty*:
configuration q' $n \neq \{\} \Longrightarrow q = \delta\ q'\ (w\ n) \Longrightarrow$ *configuration* q (*Suc* n) $\neq \{\}$
by (*blast dest*: *configuration-step*)

lemma *configuration-rev-step*$'$:
assumes $x \neq$ *Suc* n
assumes $x \in$ *configuration* q (*Suc* n)
obtains q' **where** $q = \delta\ q'\ (w\ n)$ **and** $x \in$ *configuration* q' n
using *assms Suc-diff-le* **by** *force*

lemma *configuration-rev-step*$''$:
assumes $x \in$ *configuration* q_0 (*Suc* n)
shows x = *Suc* n $\vee$ ($\exists\, q'.\ q_0 = \delta\ q'\ (w\ n) \wedge x \in$ *configuration* q' n)
using *assms configuration-rev-step*$'$ **by** *metis*

lemma *configuration-step-eq-q*$_0$:
configuration q_0 (*Suc* n) = {*Suc* n} $\cup \bigcup$ {*configuration* q' n | q'. $q_0 = \delta$ q' (w n)}
apply *rule* **using** *configuration-rev-step*$''$ **apply** *fast* **using** *configura-*

tion-step[*of* - - n q_0] **by** *fastforce*

lemma *configuration-rev-step*:
  assumes $q \neq q_0$
  assumes $x \in$ *configuration* q (*Suc* n)
  obtains q' **where** $q = \delta\ q'$ (w n) **and** $x \in$ *configuration* q' n
  using *configuration-rev-step'*[*OF* - *assms*(*2*)] *assms* **by** *fastforce*

lemma *configuration-step-eq*:
  assumes $q \neq q_0$
  shows *configuration* q (*Suc* n) $= \bigcup\{$*configuration* q' n $\mid q'.\ q = \delta\ q'$ (w n)$\}$
  using *configuration-rev-step*[*OF assms*, *of* - n] *configuration-step* **by** *auto*

lemma *configuration-step-eq-unified*:
  shows *configuration* q (*Suc* n) $= \bigcup\{$*configuration* q' n $\mid q'.\ q = \delta\ q'$ (w n)$\} \cup$ (*if* $q = q_0$ *then* $\{$*Suc* $n\}$ *else* $\{\}$)
  using *configuration-step-eq configuration-step-eq-*q_0 **by** *force*

5.4 Oldest Token

5.4.1 Properties

lemma *oldest-token-always-def*:
  $\exists\, i.\ i \leq x \wedge$ *oldest-token* (*token-run* x n) $n =$ *Some* i
proof (*cases* $x \leq n$)
  case *False*
    let *?q* = *token-run* x n
    from *False* **have** $n \in$ *configuration ?q* n **and** *configuration ?q* $n \neq \{\}$
      by *auto*
    then obtain i **where** $i \leq n$ **and** *oldest-token ?q* $n =$ *Some* i
      by (*metis Min.coboundedI oldest-token.simps configuration-finite*)
    moreover
    hence $i \leq x$
      using *False* **by** *linarith*
    ultimately
    show *?thesis*
      by *blast*
qed *fastforce*

lemma *oldest-token-bounded*:
  oldest-token q $n =$ *Some* $x \Longrightarrow x \leq n$
  by (*metis oldest-token.simps configuration-Min-in option.distinct*(*1*) *option.inject push-down-configuration-token-run*)

lemma *oldest-token-distinct*:
$q \neq q' \Longrightarrow$ *oldest-token* q n = *Some* $i \Longrightarrow$ *oldest-token* q' n = *Some* $j \Longrightarrow$ $i \neq j$
by (*metis configuration-Min-in configuration-distinct disjoint-iff-not-equal option.distinct*(*1*) *oldest-token.simps option.sel*)

lemma *oldest-token-equal*:
oldest-token q n = *Some* $i \Longrightarrow$ *oldest-token* q' n = *Some* $i \Longrightarrow q = q'$
using *oldest-token-distinct* **by** *blast*

5.4.2 Monotonicity

lemma *oldest-token-monotonic-Suc*:
assumes $x \leq n$
assumes *oldest-token* (*token-run* x n) n = *Some* i
assumes *oldest-token* (*token-run* x (*Suc* n)) (*Suc* n) = *Some* j
shows $i \geq j$
proof −
from *assms* **have** i = *Min* (*configuration* (*token-run* x n) n)
and j = *Min* (*configuration* (*token-run* x (*Suc* n)) (*Suc* n))
by (*metis oldest-token.elims option.discI option.sel*)+
thus *?thesis*
using *Min-antimono*[*OF configuration-monotonic-Suc*[*OF assms*(*1*)] *configuration-non-empty*[*OF assms*(*1*)] *configuration-finite*] **by** *blast*
qed

5.4.3 Pull-Up and Push-Down

lemma *push-down-oldest-token-configuration*:
oldest-token q n = *Some* $x \Longrightarrow x \in$ *configuration* q n
by (*metis configuration-Min-in oldest-token.simps option.distinct*(*2*) *option.inject*)

lemma *push-down-oldest-token-token-run*:
oldest-token q n = *Some* $x \Longrightarrow$ *token-run* x n = q
using *push-down-oldest-token-configuration configuration.simps* **by** *blast*

5.5 Senior Token

5.5.1 Properties

lemma *senior-le-token*:
senior x $n \leq x$
using *oldest-token-always-def*[*of* x n] **by** *fastforce*

lemma *senior-token-run*:
senior x n = senior y n ⟷ token-run x n = token-run y n
by (*metis oldest-token-always-def oldest-token-distinct option.sel senior.simps*)

The senior of a token is always in the same state

lemma *senior-same-state*:
token-run (*senior x n*) *n = token-run x n*
proof −
have *X*: {*t. t* ≤ *n* ∧ *token-run t n = token-run x n*} ≠ {}
by (*cases x* ≤ *n*) *auto*
show *?thesis*
using *Min-in*[*OF - X*] **by** *force*
qed

lemma *senior-senior*:
senior (*senior x n*) *n = senior x n*
using *senior-same-state senior-token-run* **by** *blast*

5.5.2 Monotonicity

lemma *senior-monotonic-Suc*:
x ≤ *n* ⟹ *senior x n* ≥ *senior x* (*Suc n*)
by (*metis oldest-token-always-def oldest-token-monotonic-Suc option.sel senior.simps*)

5.5.3 Pull-Up and Push-Down

lemma *pull-up-configuration-senior*:
⟦*x* ∈ *configuration q n*; *y* ∈ *configuration q n*⟧ ⟹ *senior x n = senior y n*
by *force*

lemma *push-down-senior-tokens*:
⟦*x* ≤ *n*; *y* ≤ *n*; *senior x n = senior y n*⟧ ⟹ ∃ *q. x* ∈ *configuration q n* ∧ *y* ∈ *configuration q n*
using *senior-token-run pull-up-token-run-tokens* **by** *blast*

5.6 Set of Older Seniors

5.6.1 Properties

lemma *older-seniors-cases-subseteq* [*case-names le ge*]:
assumes *older-seniors x n* ⊆ *older-seniors y n* ⟹ *P*
assumes *older-seniors x n* ⊇ *older-seniors y n* ⟹ *P*
shows *P* **using** *assms* **by** *fastforce*

lemma *older-seniors-cases-subset* [*case-names less equal greater*]:
  assumes *older-seniors x n* ⊂ *older-seniors y n* ⟹ *P*
  assumes *older-seniors x n* = *older-seniors y n* ⟹ *P*
  assumes *older-seniors x n* ⊃ *older-seniors y n* ⟹ *P*
  shows *P* **using** *assms older-seniors-cases-subseteq* **by** *blast*

lemma *older-seniors-finite*:
  finite (*older-seniors x n*)
  by *fastforce*

lemma *older-seniors-older*:
  y ∈ *older-seniors x n* ⟹ *y* < *x*
  using *less-le-trans*[*OF - senior-le-token*, *of y x n*] **by** *force*

lemma *older-seniors-senior-simp*:
  older-seniors (*senior x n*) *n* = *older-seniors x n*
  unfolding *older-seniors.simps senior-senior* **..**

lemma *older-seniors-not-self-referential*:
  senior x n ∉ *older-seniors x n*
  by *simp*

lemma *older-seniors-not-self-referential-2*:
  x ∉ *older-seniors x n*
  using *older-seniors-older older-seniors-not-self-referential less-not-refl* **by** *blast*

lemma *older-seniors-subset*:
  y ∈ *older-seniors x n* ⟹ *older-seniors y n* ⊂ *older-seniors x n*
  using *older-seniors-not-self-referential-2* **by** (*cases rule*: *older-seniors-cases-subset*) *blast+*

lemma *older-seniors-subset-2*:
  assumes ¬ *sink* (*token-run x n*)
  assumes *older-seniors x n* ⊂ *older-seniors y n*
  shows *senior x n* ∈ *older-seniors y n*
proof −
  have *senior x n* < *senior y n*
    using *assms*(*2*) **by** *fastforce*
  thus *?thesis*
    using *assms*(*1*)[*unfolded senior-same-state*[*symmetric*, *of x n*]]
    unfolding *older-seniors.simps* **by** *blast*
qed

lemmas *older-seniors-Max-in* = *Max-in*[*OF older-seniors-finite*]
lemmas *older-seniors-Min-in* = *Min-in*[*OF older-seniors-finite*]
lemmas *older-seniors-Max-coboundedI* = *Max.coboundedI*[*OF older-seniors-finite*]
lemmas *older-seniors-Min-coboundedI* = *Min.coboundedI*[*OF older-seniors-finite*]
lemmas *older-seniors-card-mono* = *card-mono*[*OF older-seniors-finite*]
lemmas *older-seniors-psubset-card-mono* = *psubset-card-mono*[*OF older-seniors-finite*]

lemma *older-seniors-recursive*:
  fixes *x n*
  defines *os* ≡ *older-seniors x n*
  assumes *os* ≠ {}
  shows *os* = {*Max os*} ∪ *older-seniors* (*Max os*) *n*
  (**is** *?lhs* = *?rhs*)
proof
  show *?lhs* ⊆ *?rhs*
  proof
    fix *x*
    assume *x* ∈ *?lhs*
    show *x* ∈ *?rhs*
    proof (*cases x* = *Max os*)
      case *False*
        hence *x* < *Max os*
       by (*metis older-seniors-Max-coboundedI os-def* ‹*x* ∈ *os*› *dual-order.order-iff-strict*)
        moreover
        obtain *y*′ **where** *Max os* = *senior y*′ *n*
          using *older-seniors-Max-in assms*(*2*)
          unfolding *os-def older-seniors.simps* **by** *blast*
        ultimately
        have *x* < *senior* (*Max os*) *n*
          using *senior-senior* **by** *presburger*
        moreover
          from ‹*x* ∈ *?lhs*› **obtain** *y* **where** *x* = *senior y n* **and** ¬ *sink*
(*token-run x n*)
          unfolding *os-def older-seniors.simps* **by** *blast*
        ultimately
        show *?thesis*
          unfolding *older-seniors.simps* **by** *blast*
    qed *blast*
  qed
next
  show *?lhs* ⊇ *?rhs*
    using *older-seniors-subset older-seniors-Max-in assms*(*2*)
    unfolding *os-def* **by** *blast*

qed

lemma *older-seniors-recursive-card*:
  fixes *x n*
  defines *os* ≡ *older-seniors x n*
  assumes *os* ≠ {}
  shows *card os* = *Suc* (*card* (*older-seniors* (*Max os*) *n*))
 by (*metis older-seniors-recursive assms Un-empty-left Un-insert-left card-insert-disjoint older-seniors-finite older-seniors-not-self-referential-2*)

lemma *older-seniors-card*:
  card (*older-seniors x n*) = *card* (*older-seniors y n*) ⟷ *older-seniors x n* = *older-seniors y n*
 by (*metis less-not-refl older-seniors-cases-subset older-seniors-psubset-card-mono*)

lemma *older-seniors-card-le*:
  card (*older-seniors x n*) < *card* (*older-seniors y n*) ⟷ *older-seniors x n* ⊂ *older-seniors y n*
 by (*metis card-mono card-psubset not-le older-seniors-cases-subseteq older-seniors-finite psubset-card-mono*)

lemma *older-seniors-card-less*:
  card (*older-seniors x n*) ≤ *card* (*older-seniors y n*) ⟷ *older-seniors x n* ⊆ *older-seniors y n*
  by (*metis not-le older-seniors-card-mono older-seniors-cases-subseteq older-seniors-psubset-card-mo subset-not-subset-eq*)

5.6.2 Monotonicity

lemma *older-seniors-monotonic-Suc*:
  assumes *x* ≤ *n*
  shows *older-seniors x n* ⊇ *older-seniors x* (*Suc n*)
proof
  fix *y*
  assume *y* ∈ *older-seniors x* (*Suc n*)
  then obtain *ox* **where** *y* = *senior ox* (*Suc n*)
    and *y* < *senior x* (*Suc n*)
    and ¬ *sink* (*token-run y* (*Suc n*))
    unfolding *older-seniors.simps* **by** *blast*

  hence *y* = *senior y n*
    using *senior-senior senior-le-token senior-monotonic-Suc assms*
    by (*metis add.commute add.left-commute dual-order.order-iff-strict linear not-add-less1 not-less le-iff-add*)

moreover
have *y* < *senior x n*
using *assms less-le-trans*[*OF* ‹*y* < *senior x* (*Suc n*)› *senior-monotonic-Suc*]
by *blast*
moreover
have ¬ *sink* (*token-run y n*)
using ‹¬ *sink* (*token-run y* (*Suc n*))› *token-stays-in-sink*
unfolding *Suc-eq-plus1* **by** *metis*

ultimately
show *y* ∈ *older-seniors x n*
unfolding *older-seniors.simps* **by** *blast*
qed

lemma *older-seniors-monotonic*:
$x \leq n \Longrightarrow$ *older-seniors x n* ⊇ *older-seniors x* (*n* + *m*)
by (*induction m*) (*simp*, *metis older-seniors-monotonic-Suc add-Suc-right dual-order.trans trans-le-add1*)

lemma *older-seniors-stable*:
$x \leq n \Longrightarrow$ *older-seniors x n* = *older-seniors x* (*n* + *m* + *m'*) ⟹ *older-seniors x n* = *older-seniors x* (*n* + *m*)
by (*induction m'*) (*simp*, *unfold set-eq-subset*, *metis dual-order.trans le-add1 older-seniors-monotonic*)

lemma *card-older-seniors-monotonic*:
$x \leq n \Longrightarrow$ *card* (*older-seniors x n*) ≥ *card* (*older-seniors x* (*n* + *m*))
using *older-seniors-monotonic older-seniors-card-mono* **by** *meson*

5.6.3 Pull-Up and Push-Down

lemma *pull-up-senior-older-seniors*:
senior x n = *senior y n* ⟹ *older-seniors x n* = *older-seniors y n*
unfolding *older-seniors.simps senior.simps senior-token-run* **by** *presburger*

lemma *pull-up-senior-older-seniors-less*:
senior x n < *senior y n* ⟹ *older-seniors x n* ⊆ *older-seniors y n*
by *force*

lemma *pull-up-senior-older-seniors-less-2*:
assumes ¬ *sink* (*token-run x n*)
assumes *senior x n* < *senior y n*
shows *older-seniors x n* ⊂ *older-seniors y n*

proof −
  from *assms* **have** *senior x n* ∈ *older-seniors y n*
    unfolding *senior-same-state*[*of x n*, *symmetric*] *older-seniors.simps* **by** *blast*
  thus *?thesis*
    using *older-seniors-not-self-referential pull-up-senior-older-seniors-less*[*OF assms*(*2*)] **by** *blast*
qed

lemma *pull-up-senior-older-seniors-le*:
  senior x n ≤ *senior y n* ⟹ *older-seniors x n* ⊆ *older-seniors y n*
  using *pull-up-senior-older-seniors pull-up-senior-older-seniors-less*
  unfolding *dual-order.order-iff-strict* **by** *blast*

lemma *push-down-older-seniors-senior*:
  assumes ¬ *sink* (*token-run x n*)
  assumes ¬ *sink* (*token-run y n*)
  assumes *older-seniors x n* = *older-seniors y n*
  shows *senior x n* = *senior y n*
  using *assms* **by** (*cases senior x n senior y n rule*: *linorder-cases*) (*fast dest*: *pull-up-senior-older-seniors-less-2*)+

5.6.4 Tower Lemma

lemma *older-seniors-tower''*:
  assumes $x \leq n$
  assumes $y \leq n$
  assumes ¬*sink* (*token-run x n*)
  assumes ¬*sink* (*token-run y n*)
  assumes *older-seniors x n* = *older-seniors x* (*Suc n*)
  assumes *older-seniors y n* ⊆ *older-seniors x n*
  shows *older-seniors y n* = *older-seniors y* (*Suc n*)
proof
  {
    fix *s*
    assume *s* ∈ *older-seniors y n* **and** *older-seniors y n* ⊂ *older-seniors x n*
    hence *s* ∈ *older-seniors x n*
      using *assms* **by** *blast*
    hence ¬*sink* (*token-run s* (*Suc n*)) **and** ∃ *z*. *s* = *senior z* (*Suc n*)
      unfolding *assms* **by** *simp*+
    moreover
    have *senior y n* ≤ *senior y* (*Suc n*)
    proof (*rule ccontr*)
      assume ¬*senior y n* ≤ *senior y* (*Suc n*)

```
      moreover
      have senior y n ≤ n
        by (metis assms(2) senior-le-token le-trans)
      ultimately
      have ∀ z. senior y n ≠ senior z (Suc n)
       using token-run-merge-Suc[unfolded senior-token-run[symmetric], OF
‹y ≤ n›]
        by (metis senior-senior le-refl)
      hence senior y n ∉ older-seniors x (Suc n)
        using assms by simp
      moreover
      have senior y n ∈ older-seniors x n
      using assms ‹older-seniors y n ⊂ older-seniors x n› older-seniors-subset-2
by meson
      ultimately
      show False
        unfolding assms ..
    qed
    hence s < senior y (Suc n)
      using ‹s ∈ older-seniors y n› by fastforce
    ultimately
    have s ∈ older-seniors y (Suc n)
      unfolding older-seniors.simps by blast
  }
  moreover
  {
    fix s
    assume s ∈ older-seniors y n and older-seniors y n = older-seniors x n
    moreover
    hence senior y n = senior x n
      using assms(3−4) push-down-older-seniors-senior by blast
    hence senior y (Suc n) = senior x (Suc n)
     using token-run-merge-Suc[OF assms(2,1)] unfolding senior-token-run
by blast
    ultimately
    have s ∈ older-seniors y (Suc n)
      by (metis assms(5) older-seniors-senior-simp)
  }
  ultimately
  show older-seniors y n ⊆ older-seniors y (Suc n)
    using assms by blast
qed (metis older-seniors-monotonic-Suc assms(2))

lemma older-seniors-tower′′2:
```

assumes $x \leq n$
assumes $y \leq n$
assumes ¬*sink* (*token-run* x (n + m))
assumes ¬*sink* (*token-run* y (n + m))
assumes *older-seniors* x n = *older-seniors* x (n + m)
assumes *older-seniors* y n ⊆ *older-seniors* x n
shows *older-seniors* y n = *older-seniors* y (n + m)
using *assms*
proof (*induction m arbitrary*: n)
case (*Suc m*)
have ¬*sink* (*token-run* x (n + m)) **and** ¬*sink* (*token-run* y (n + m))
using ‹¬*sink* (*token-run* x (n + *Suc m*))› ‹¬*sink* (*token-run* y (n + *Suc m*))›
using *token-stays-in-sink*[*of* - - n + m *1*]
unfolding *Suc-eq-plus1 add.assoc*[*symmetric*] **by** *metis*+
moreover
have *older-seniors* x n = *older-seniors* x (n + m)
using *Suc.prems*(*5*) *older-seniors-stable*[*OF* ‹$x \leq n$›]
unfolding *Suc-eq-plus1 add.assoc* **by** *blast*
moreover
hence *older-seniors* x (n + m) = *older-seniors* x (*Suc* (n + m))
unfolding *Suc.prems add-Suc-right* **..**
ultimately
have *older-seniors* y n = *older-seniors* y (n + m)
using *Suc* **by** *meson*
also
have ... = *older-seniors* y (*Suc* (n + m))
using *older-seniors-tower''*[*OF* - - ‹¬*sink* (*token-run* x (n + m))› ‹¬*sink* (*token-run* y (n + m))› ‹*older-seniors* x (n + m) = *older-seniors* x (*Suc* (n + m))›] *Suc*
by (*metis* ‹*older-seniors* x n = *older-seniors* x (n + m)› *add.commute add.left-commute calculation le-iff-add*)
finally
show *?case*
unfolding *add-Suc-right* **.**
qed *simp*

lemma *older-seniors-tower'*:
assumes $y \in$ *older-seniors* x n
assumes *older-seniors* x n = *older-seniors* x (*Suc* n)
shows *older-seniors* y n = *older-seniors* y (*Suc* n)
(**is** *?lhs* = *?rhs*)
using *assms*
proof (*induction card* (*older-seniors* x n) *arbitrary*: x y)

case *0*
  hence *older-seniors x n* = {}
    using *older-seniors-finite card-eq-0-iff* **by** *metis*
  thus *?case*
    using *0.prems* **by** *blast*
next
  case (*Suc c*)
  let *?os* = *older-seniors x n*
  have *?os* ≠ {}
    using *Suc.prems*(*1*) **by** *blast*

  hence *y* = *Max ?os* ∨ *y* ∈ *older-seniors* (*Max ?os*) *n*
    using *Suc.prems*(*1*) *older-seniors-recursive* **by** *blast*
  moreover
  have *older-seniors* (*Max ?os*) *n* = *older-seniors* (*Max ?os*) (*Suc n*)
  using *Suc.prems*(*2*) *older-seniors-recursive* ‹*?os* ≠ {}› *older-seniors-not-self-referential-2*
    by (*metis Un-empty-left Un-insert-left insert-ident*)
  moreover
  {
    fix *s*
    assume *s* ∈ *older-seniors* (*Max ?os*) *n*
    moreover
    from *Suc.hyps*(*2*) **have** *card* (*older-seniors* (*Max ?os*) *n*) = *c*
      unfolding *older-seniors-recursive-card*[*OF* ‹*?os* ≠ {}›] **by** *blast*
    ultimately
    have *older-seniors s n* = *older-seniors s* (*Suc n*)
      by (*metis Suc.hyps*(*1*) ‹*older-seniors* (*Max ?os*) *n* = *older-seniors* (*Max ?os*) (*Suc n*)›)
  }
  ultimately
  show *?case*
    by *blast*
qed

lemma *older-seniors-tower*:
  ⟦*x* ≤ *n*; *y* ∈ *older-seniors x n*; *older-seniors x n* = *older-seniors x* (*n* + *m*)⟧ ⟹ *older-seniors y n* = *older-seniors y* (*n* + *m*)
proof (*induction m*)
  case (*Suc m*)
  hence *older-seniors x n* = *older-seniors x* (*n* + *m*)
  using *older-seniors-monotonic older-seniors-monotonic-Suc subset-antisym*
    by (*metis Nat.add-0-right add.assoc add-Suc-shift trans-le-add1*)
  hence *older-seniors y n* = *older-seniors y* (*n* + *m*)
    using *Suc.IH*[*OF Suc.prems*(*1*,*2*)] **by** *blast*

also
have ... = *older-seniors y* (*n* + *Suc m*)
using *older-seniors-tower'*[*of y x n* + *m*] *Suc.prems* **unfolding** *add-Suc-right*
by (*metis* ‹*older-seniors x n* = *older-seniors x* (*n* + *m*)›)
finally
show *?case* **.**
qed *simp*

5.7 Rank

5.7.1 Properties

lemma *rank-None-before*:
$x > n \Longrightarrow$ *rank x n* = *None*
by *simp*

lemma *rank-None-Suc*:
assumes $x \leq n$
assumes *rank x n* = *None*
shows *rank x* (*Suc n*) = *None*
proof −
have *sink* (*token-run x n*)
using *assms* **by** (*metis option.distinct*(*1*) *rank.simps*)
hence *sink* (*token-run x* (*Suc n*))
using *token-stays-in-sink* **by** (*metis* (*erased*, *opaque-lifting*) *Suc-leD le-Suc-ex not-less-eq-eq*)
thus *?thesis*
by *simp*
qed

lemma *rank-Some-time*:
rank x n = *Some j* $\Longrightarrow x \leq n$
by (*metis option.distinct*(*1*) *rank.simps*)

lemma *rank-Some-sink*:
rank x n = *Some j* $\Longrightarrow$ ¬*sink* (*token-run x n*)
by *fastforce*

lemma *rank-Some-card*:
rank x n = *Some j* $\Longrightarrow$ *card* (*older-seniors x n*) = *j*
by (*metis option.distinct*(*1*) *option.inject rank.simps*)

lemma *rank-initial*:
∃ *i*. *rank x x* = *Some i*

```
  unfolding rank.simps sink-def by force

lemma rank-continuous:
  assumes rank x n = Some i
  assumes rank x (n + m) = Some j
  assumes m′ ≤ m
  shows ∃ k. rank x (n + m′) = Some k
  using assms
proof (induction m arbitrary: j m′)
  case (Suc m)
    thus ?case
    proof (cases m′ = Suc m)
      case False
        with Suc.prems have m′ ≤ m
          by linarith
        moreover
        obtain j′ where rank x (n + m) = Some j′
          using Suc.prems(1,2) rank-Some-time rank-None-Suc
          by (metis add-Suc-right add-lessD1 not-less rank.simps)
        ultimately
        show ?thesis
          using Suc.IH[OF Suc.prems(1)] by blast
    qed simp
qed simp

lemma rank-token-squats:
  token-squats x ⟹ x ≤ n ⟹ ∃ i. rank x n = Some i
  unfolding token-squats-def by simp

lemma rank-older-seniors-bounded:
  assumes y ∈ older-seniors x n
  assumes rank x n = Some j
  shows ∃ j′ < j. rank y n = Some j′
proof −
  from assms(1) have ¬sink (token-run y n)
    by simp
  moreover
  from assms have y ≤ n
   by (metis dual-order.trans linear not-less older-seniors-older option.distinct(1)
rank.simps)
  moreover
  have older-seniors y n ⊂ older-seniors x n
    using older-seniors-subset assms(1) by presburger
  hence card (older-seniors y n) < card (older-seniors x n)
```

by (*rule older-seniors-psubset-card-mono*)
ultimately
show *?thesis*
using *rank-Some-card*[*OF assms*(*2*)] *rank.simps* **by** *meson*
qed

5.7.2 Bounds

lemma *max-rank-lowerbound*:
0 < *max-rank*
proof −
obtain *a* **where** $a \in \Sigma$
using *nonempty-*Σ **by** *blast*
hence *range* (λ-. *a*) ⊆ Σ **and** q_0 = *run* δ q_0 (λ-. *a*) *0*
by *auto*
hence q_0 ∈ *reach* Σ δ q_0
unfolding *reach-def* **by** *blast*
thus *?thesis*
using *reach-card-0*[*OF nonempty-*Σ] *finite-reach max-rank-def sink-def*
by *force*
qed

lemma *older-seniors-card-bounded*:
assumes ¬*sink* (*token-run x n*) **and** *x* ≤ *n*
shows *card* (*older-seniors x n*) < *card* (*reach* Σ δ q_0 − {*q*. *sink q*})
(**is** *card ?S4* < *card ?S0*)
proof −
let *?S1* = {*token-run x n* | *x n*. *True*} − {*q*. *sink q*}
let *?S2* = (λ*q*. *the* (*oldest-token q n*)) ‘ *?S1*
let *?S3* = {*s*. ∃ *x*. *s* = *senior x n* ∧ ¬(*sink* (*token-run s n*))}

have *?S1* ⊆ *?S0*
unfolding *reach-def token-run.simps* **using** *bounded-w* **by** *fastforce*
hence *finite ?S1* **and** *C1*: *card ?S1* ≤ *card ?S0*
using *finite-reach card-mono finite-subset*
apply (*simp add*: *finite-subset*) **by** (*metis* ‹{*token-run x n* |*x n*. *True*} − *Collect sink* ⊆ *reach* Σ δ q_0 − *Collect sink*› *card-mono finite-Diff local.finite-reach*)
hence *finite ?S2* **and** *C2*: *card ?S2* ≤ *card ?S1*
using *finite-imageI card-image-le* **by** *blast+*
moreover
have *?S3* ⊆ *?S2*
proof
fix *s*

```
    assume s ∈ ?S3
    hence s = senior s n and ¬sink (token-run s n)
      using senior-senior by fastforce+
    thus s ∈ ?S2
      by auto
  qed
  ultimately
  have finite ?S3 and C3: card ?S3 ≤ card ?S2
    using card-mono finite-subset by blast+
  moreover
  have senior x n ∈ ?S3 and senior x n ∉ ?S4 and ?S4 ⊆ ?S3
    using assms older-seniors-not-self-referential senior-same-state by auto
  hence ?S4 ⊂ ?S3
    by blast
  ultimately
  have finite ?S4 and C4: card ?S4 < card ?S3
    using psubset-card-mono finite-subset by blast+
  show ?thesis
    using C1 C2 C3 C4 by linarith
qed

lemma rank-upper-bound:
  rank x n = Some i ⟹ i < max-rank
  using older-seniors-card-bounded unfolding max-rank-def
  by (fast dest: rank-Some-card rank-Some-time rank-Some-sink )

lemma rank-range:
  ∃ i. range (rank x) ⊆ {None} ∪ Some ` {0..<i}
proof
  {
    fix i-option
    assume i-option ∈ range (rank x)
    hence i-option ∈ {None} ∪ Some ` {0..<max-rank}
    proof (cases i-option)
      case (Some i)
        hence i ∈ {0..<max-rank}
          using ‹i-option ∈ range (rank x)› rank-upper-bound by force
        thus ?thesis
          using Some by blast
    qed blast
  }
  thus range (rank x) ⊆ ({None} ∪ Some ` {0..<max-rank}) ..
qed
```

5.7.3 Monotonicity

lemma *rank-monotonic*:
$[\![rank\ x\ n = Some\ i;\ rank\ x\ (n + m) = Some\ j]\!] \Longrightarrow i \geq j$
using *card-older-seniors-monotonic rank-Some-card rank-Some-time* **by** *metis*

5.7.4 Pull-Up and Push-Down

lemma *pull-up-senior-rank*:
$[\![x \leq n;\ y \leq n;\ senior\ x\ n = senior\ y\ n]\!] \Longrightarrow rank\ x\ n = rank\ y\ n$
by (*metis senior-token-run rank.simps pull-up-senior-older-seniors*)

lemma *pull-up-configuration-rank*:
$[\![x \in configuration\ q\ n;\ y \in configuration\ q\ n]\!] \Longrightarrow rank\ x\ n = rank\ y\ n$
by *force*

lemma *push-down-rank-older-seniors*:
$[\![rank\ x\ n = rank\ y\ n;\ rank\ x\ n = Some\ i]\!] \Longrightarrow older\text{-}seniors\ x\ n = older\text{-}seniors\ y\ n$
by (*metis older-seniors-card option.distinct*(*2*) *option.sel rank.simps*)

lemma *push-down-rank-senior*:
$[\![rank\ x\ n = rank\ y\ n;\ rank\ x\ n = Some\ i]\!] \Longrightarrow senior\ x\ n = senior\ y\ n$
by (*metis push-down-rank-older-seniors push-down-older-seniors-senior option.distinct*(*1*) *rank.elims*)

lemma *push-down-rank-tokens*:
$[\![rank\ x\ n = rank\ y\ n;\ rank\ x\ n = Some\ i]\!] \Longrightarrow (\exists\, q.\ x \in configuration\ q\ n \wedge y \in configuration\ q\ n)$
by (*metis push-down-senior-tokens rank-Some-time push-down-rank-senior*)

5.7.5 Pulled-Up Lemmas

lemma *rank-senior-senior*:
$x \leq n \Longrightarrow rank\ (senior\ x\ n)\ n = rank\ x\ n$
by (*metis le-iff-add add.commute add.left-commute pull-up-senior-rank senior-le-token senior-senior*)

5.7.6 Stable Rank

definition *stable-rank* :: $nat \Rightarrow nat \Rightarrow bool$
where
$stable\text{-}rank\ x\ i = (\forall_\infty n.\ rank\ x\ n = Some\ i)$

lemma *stable-rank-unique*:
assumes *stable-rank x i*
assumes *stable-rank x j*
shows $i = j$
proof −
from *assms* **obtain** *n m* **where** $\bigwedge n'.\ n' \geq n \Longrightarrow$ *rank x n′* = *Some i*
and $\bigwedge m'.\ m' \geq m \Longrightarrow$ *rank x m′* = *Some j*
unfolding *stable-rank-def MOST-nat-le* **by** *blast*
hence *rank x* $(n + m)$ = *Some i* **and** *rank x* $(n + m)$ = *Some j*
by (*metis add.commute le-add1*)+
thus *?thesis*
by *simp*
qed

lemma *stable-rank-equiv-token-squats*:
token-squats x = ($\exists$ *i*. *stable-rank x i*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
define *ranks* **where** *ranks* = $\{j \mid j\ n.$ *rank x n* = *Some j*$\}$
hence *ranks* $\subseteq$ {*0..<max-rank*} **and** *the* (*rank x x*) $\in$ *ranks*
using *rank-upper-bound rank-initial*[*of x*] **unfolding** *ranks-def* **by** *fast-force*+
hence *finite ranks* **and** *ranks* $\neq$ {}
using *finite-reach finite-atLeastAtMost infinite-super* **by** *fast*+

define *i* **where** *i* = *Min ranks*
obtain *n* **where** *rank x n* = *Some i*
using *Min-in*[*OF* ‹*finite ranks*› ‹*ranks* $\neq$ {}›]
unfolding *i-def ranks-def* **by** *blast*

have $\bigwedge j.\ j \in$ *ranks* $\Longrightarrow j \geq i$
using *Min-in*[*OF* ‹*finite ranks*› ‹*ranks* $\neq$ {}›] **unfolding** *i-def*
by (*metis Min.coboundedI* ‹*finite ranks*›)
hence $\bigwedge m\ j.$ *rank x* $(n + m)$ = *Some j* $\Longrightarrow j \geq i$
unfolding *ranks-def* **by** *blast*
moreover
have $\bigwedge m\ j.$ *rank x* $(n + m)$ = *Some j* $\Longrightarrow j \leq i$
using *rank-monotonic*[*OF* ‹*rank x n* = *Some i*›] **by** *blast*
moreover
have $\bigwedge m.\ \exists j.$ *rank x* $(n + m)$ = *Some j*
using *rank-token-squats*[*OF* ‹*?lhs*›] *rank-Some-time*[*OF* ‹*rank x n* = *Some i*›] **by** *simp*
ultimately

have $\bigwedge m.$ *rank x* $(n + m) =$ *Some i*
by (*metis le-antisym*)
thus *?rhs*
unfolding *stable-rank-def MOST-nat-le* **by** (*metis le-iff-add*)
next
assume *?rhs*
thus *?lhs*
unfolding *token-squats-def stable-rank-def MOST-nat-le*
by (*metis le-add2 rank-Some-sink token-stays-in-sink*)
qed

lemma *stable-rank-same-tokens*:
assumes *stable-rank x i*
assumes *stable-rank y j*
assumes $x \in$ *configuration q n*
assumes $y \in$ *configuration q n*
shows $i = j$
proof −
from *assms*(*1*) **obtain** *n-i* **where** *n-i* $\geq n$ **and** $\forall t \geq$ *n-i*. *rank x t* $=$ *Some i*
unfolding *stable-rank-def MOST-nat-le* **by** (*metis linear order-trans*)
moreover
from *assms*(*2*) **obtain** *n-j* **where** *n-j* $\geq n$ **and** $\forall t \geq$ *n-j*. *rank y t* $=$ *Some j*
unfolding *stable-rank-def MOST-nat-le* **by** (*metis linear order-trans*)
moreover
define *m* **where** $m =$ *max n-i n-j*
ultimately
have *rank x m* $=$ *Some i* **and** *rank y m* $=$ *Some j*
by (*metis max.bounded-iff order-refl*)+
moreover
have $m \geq n$
by (*metis* ‹$n \leq$ *n-j*› *le-trans max.cobounded2 m-def*)
have $\exists q'. x \in$ *configuration* q' $m \wedge y \in$ *configuration* q' *m*
using *push-down-configuration-token-run*[*OF assms*(*3*,*4*)]
using *token-run-merge*[*of x n y*]
using *pull-up-token-run-tokens*[*of x m y*]
using ‹$m \geq n$›[*unfolded le-iff-add*] **by** *force*
ultimately
show *?thesis*
using *pull-up-configuration-rank* **by** (*metis option.inject*)
qed

5.7.7 Tower Lemma

lemma *rank-tower*:
  assumes $i \le j$
  assumes *rank x n* = *Some j*
  assumes *rank x* (*n* + *m*) = *Some j*
  assumes *rank y n* = *Some i*
  shows *rank y* (*n* + *m*) = *Some i*
proof (*cases i j rule*: *linorder-cases*)
  case *less*
    {
      hence *card* (*older-seniors* (*senior y n*) *n*) < *card* (*older-seniors x n*)
        using *assms rank-Some-card senior-same-state* **by** *force*
      hence *senior y n* ∈ *older-seniors x n*
      by (*metis older-seniors-card-le rank-Some-sink assms*(*4*) *older-seniors-senior-simp older-seniors-subset-2*)
      moreover
      have *older-seniors x n* = *older-seniors x* (*n* + *m*)
      by (*metis assms*(*2*,*3*) *rank-Some-card rank-Some-time card-subset-eq*[*OF older-seniors-finite*] *older-seniors-monotonic*)
      ultimately
      have *older-seniors* (*senior y n*) *n* = *older-seniors* (*senior y n*) (*n* + *m*) **and** *senior y n* ∈ *older-seniors x* (*n* + *m*)
        using *older-seniors-tower rank-Some-time assms*(*2*) **by** *blast+*
    }
    moreover
    have *rank* (*senior y n*) *n* = *Some i*
      by (*metis assms*(*4*) *rank-Some-time rank-senior-senior*)
    ultimately
    have *rank* (*senior y n*) (*n* + *m*) = *Some i*
     by (*metis rank-older-seniors-bounded*[*OF - assms*(*3*)] *rank-Some-card*)
    moreover
    have *senior y n* ≤ *n*
      by (*metis* ‹*rank* (*senior y n*) *n* = *Some i*› *rank-Some-time*)
    hence *senior y n* ∈ *configuration* (*token-run y* (*n* + *m*)) (*n* + *m*)
    by (*metis* (*full-types*) *token-run-merge*[*OF - rank-Some-time*[*OF assms*(*4*)] *senior-same-state*] *configuration-token trans-le-add1*)
    ultimately
    show *?thesis*
        by (*metis pull-up-configuration-rank le-iff-add add.assoc assms*(*4*) *configuration-token rank-Some-time*)
next
  case *equal*
    hence $x \le n$ **and** $y \le n$ **and** *token-run x n* = *token-run y n*

using *assms(2−4) push-down-rank-tokens* **by** *force+*
moreover
hence *token-run x (n + m) = token-run y (n + m)*
using *token-run-merge* **by** *blast*
ultimately
show *?thesis*
by (*metis assms(3) equal rank-senior-senior senior-token-run le-iff-add add.assoc*)
qed (*insert* ‹$i \leq j$›, *linarith*)

lemma *stable-rank-alt-def*:
rank x n = Some j ∧ *stable-rank x j* ⟷ (∀ *m* ≥ *n*. *rank x m = Some j*)
(**is** *?rhs* ⟷ *?lhs*)
proof
assume *?rhs*
then obtain *m′* **where** ∀ *m* ≥ *m′*. *rank x m = Some j*
unfolding *stable-rank-def MOST-nat-le* **by** *blast*
moreover
hence *rank x n = Some j* **and** *rank x m′ = Some j*
using ‹*?rhs*› **by** *blast+*
{
fix *m*
assume $n \leq n + m$ **and** $n + m < m'$
then obtain *j′* **where** *rank x (n + m) = Some j′*
by (*metis* ‹*?rhs*› *stable-rank-equiv-token-squats rank-Some-time rank-token-squats trans-le-add1*)
moreover
hence $j' \leq j$
using ‹*rank x n = Some j*› *rank-monotonic* **by** *blast*
moreover
have $j \leq j'$
using ‹*rank x (n + m) = Some j′*› ‹*rank x m′ = Some j*› ‹$n + m < m'$› *rank-monotonic*
by (*metis add-Suc-right less-imp-Suc-add*)
ultimately
have *rank x (n + m) = Some j*
by *simp*
}
ultimately
show *?lhs*
by (*metis le-add-diff-inverse not-le*)
qed (*unfold stable-rank-def MOST-nat-le*, *blast*)

lemma *stable-rank-tower*:

assumes *j* ≤ *i*
assumes *rank x n* = *Some j*
assumes *rank y n* = *Some i*
assumes *stable-rank y i*
shows *stable-rank x j*
using *assms rank-tower*[*OF* ‹*j* ≤ *i*›] *stable-rank-alt-def*[*of y n i*]
unfolding *stable-rank-def*[*of x j*, *unfolded MOST-nat-le*] **by** (*metis le-Suc-ex*)

5.8 Senior States

lemma *senior-states-initial*:
senior-states q 0 = {}
by *simp*

lemma *senior-states-cases-subseteq* [*case-names le ge*]:
assumes *senior-states p n* ⊆ *senior-states q n* ⟹ *P*
assumes *senior-states p n* ⊇ *senior-states q n* ⟹ *P*
shows *P* **using** *assms* **by** *force*

lemma *senior-states-cases-subset* [*case-names less equal greater*]:
assumes *senior-states p n* ⊂ *senior-states q n* ⟹ *P*
assumes *senior-states p n* = *senior-states q n* ⟹ *P*
assumes *senior-states p n* ⊃ *senior-states q n* ⟹ *P*
shows *P* **using** *assms senior-states-cases-subseteq* **by** *blast*

lemma *senior-states-finite*:
finite (*senior-states q n*)
by *fastforce*

lemmas *senior-states-card-mono* = *card-mono*[*OF senior-states-finite*]
lemmas *senior-states-psubset-card-mono* = *psubset-card-mono*[*OF senior-states-finite*]

lemma *senior-states-card*:
card (*senior-states p n*) = *card* (*senior-states q n*) ⟷ *senior-states p n* = *senior-states q n*
by (*metis less-not-refl senior-states-cases-subset senior-states-psubset-card-mono*)

lemma *senior-states-card-le*:
card (*senior-states p n*) < *card* (*senior-states q n*) ⟷ *senior-states p n* ⊂ *senior-states q n*
by (*metis card-mono not-less senior-states-cases-subseteq senior-states-finite senior-states-psubset-card-mono subset-not-subset-eq*)

lemma *senior-states-card-less*:

card (*senior-states p n*) ≤ *card* (*senior-states q n*) ⟷ *senior-states p n* ⊆ *senior-states q n*
by (*metis card-mono card-seteq senior-states-cases-subseteq senior-states-finite*)

lemma *senior-states-older-seniors*:
(λ*y. token-run y n*) ' *older-seniors x n* = *senior-states* (*token-run x n*) *n*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = {*q*′. ∃ *ost ot. q*′ = *token-run ost n* ∧ *ost* = *senior ot n* ∧ *ost* < *senior x n* ∧ ¬ *sink q*′}
by *auto*
also
have … = {*q*′. ∃ *t ot. oldest-token q*′ *n* = *Some t* ∧ *t* = *senior ot n* ∧ *t* < *senior x n* ∧ ¬ *sink q*′}
unfolding *senior.simps* **by** (*metis* (*erased*, *opaque-lifting*) *oldest-token-always-def push-down-oldest-token-token-run option.sel*)
also
have … = {*q*′. ∃ *t. oldest-token q*′ *n* = *Some t* ∧ *t* < *senior x n* ∧ ¬ *sink q*′}
by *auto*
also
have … = *?rhs*
unfolding *senior-states.simps senior.simps* **by** (*metis* (*erased*, *opaque-lifting*) *oldest-token-always-def option.sel*)
finally
show *?lhs* = *?rhs*
.
qed

lemma *card-older-senior-senior-states*:
assumes *x* ∈ *configuration q n*
shows *card* (*older-seniors x n*) = *card* (*senior-states q n*)
(**is** *?lhs* = *?rhs*)
proof −
have *inj-on* (λ*t. token-run t n*) (*older-seniors x n*)
unfolding *inj-on-def* **using** *senior-same-state*
by (*fastforce simp del*: *token-run.simps*)
moreover
have *token-run x n* = *q*
using *assms* **by** *simp*
ultimately
show *?lhs* = *?rhs*
using *card-image*[*of* (λ*t. token-run t n*) *older-seniors x n*]
unfolding *senior-states-older-seniors* **by** *presburger*

qed

5.9 Rank of States

5.9.1 Alternative Definitions

lemma *state-rank-eq-rank*:
state-rank q n = (*case oldest-token q n of None* ⇒ *None* | *Some t* ⇒ *rank t n*)
(**is** *?lhs* = *?rhs*)
proof (*cases oldest-token q n*)
case (*None*)
thus *?thesis*
by (*metis not-Some-eq oldest-token.elims option.simps*(*4*) *state-rank.elims*)
next
case (*Some x*)
hence *?lhs* = (*if* ¬*sink q then Some* (*card* (*older-seniors x n*)) *else None*)
by (*metis emptyE push-down-oldest-token-configuration*[*OF Some*] *card-older-senior-senior-states state-rank.simps*)
also
have ... = *rank x n*
using *oldest-token-bounded*[*OF Some*] *push-down-oldest-token-token-run*[*OF Some*] **by** *auto*
also
have ... = *?rhs*
using *Some* **by** *force*
finally
show *?thesis* **.**
qed

lemma *state-rank-eq-rank-SOME*:
state-rank q n = (*if configuration q n* ≠ {} *then rank* (*SOME x*. *x* ∈ *configuration q n*) *n else None*)
proof (*cases oldest-token q n*)
case (*Some x*)
thus *?thesis*
unfolding *state-rank-eq-rank Some option.simps*(*5*)
by (*metis Some ex-in-conv pull-up-configuration-rank push-down-oldest-token-configuration someI-ex*)
qed (*unfold state-rank-eq-rank*; *metis not-Some-eq oldest-token.elims option.simps*(*4*))

lemma *rank-eq-state-rank*:
$x \le n \Longrightarrow$ *rank x n* = *state-rank* (*token-run x n*) *n*

unfolding *state-rank-eq-rank-SOME*[*of token-run x n*]
by (*metis all-not-in-conv configuration-token pull-up-configuration-rank someI-ex*)

5.9.2 Pull-Up and Push-Down

lemma *pull-up-configuration-state-rank*:
configuration q n = {} ⟹ *state-rank q n* = *None*
by *force*

lemma *push-down-state-rank-tokens*:
state-rank q n = *Some i* ⟹ *configuration q n* ≠ {}
by (*metis not-Some-eq state-rank.elims*)

lemma *push-down-state-rank-configuration-None*:
state-rank q n = *None* ⟹ ¬*sink q* ⟹ *configuration q n* = {}
unfolding *state-rank.simps* **by** (*metis option.distinct*(*1*))

lemma *push-down-state-rank-oldest-token*:
state-rank q n = *Some i* ⟹ ∃ *x*. *oldest-token q n* = *Some x*
by (*metis oldest-token.elims state-rank.elims*)

lemma *push-down-state-rank-token-run*:
state-rank q n = *Some i* ⟹ ∃ *x*. *token-run x n* = *q* ∧ *x* ≤ *n*
by (*blast dest*: *push-down-state-rank-oldest-token push-down-oldest-token-token-run oldest-token-bounded*)

5.9.3 Properties

lemma *state-rank-distinct*:
assumes *distinct*: *p* ≠ *q*
assumes *ranked-1*: *state-rank p n* = *Some i*
assumes *ranked-2*: *state-rank q n* = *Some j*
shows *i* ≠ *j*
proof
assume *i* = *j*
obtain *x y* **where** *x* ∈ *configuration p n* **and** *y* ∈ *configuration q n*
using *assms push-down-state-rank-tokens* **by** *blast*
hence *rank x n* = *Some i* **and** *rank y n* = *Some j*
using *assms pull-up-configuration-rank* **unfolding** *state-rank-eq-rank-SOME*
by (*metis all-not-in-conv someI-ex*)+
hence *x* ∈ *configuration q n*
using ‹*y* ∈ *configuration q n*› *push-down-rank-tokens*
unfolding ‹*i* = *j*› **by** *auto*

hence $p = q$
  using ‹$x \in$ *configuration p n*› **by** *fastforce*
thus *False*
  using *distinct* **by** *blast*
qed

lemma *state-rank-initial-state*:
  obtains i **where** *state-rank* q_0 n = *Some* i
  unfolding *state-rank.simps sink-def* **by** *fastforce*

lemma *state-rank-sink*:
  sink q $\Longrightarrow$ *state-rank q n* = *None*
  by *simp*

lemma *state-rank-upper-bound*:
  state-rank q n = *Some i* $\Longrightarrow$ $i <$ *max-rank*
  by (*metis option.simps*(*5*) *rank-upper-bound push-down-state-rank-oldest-token state-rank-eq-rank*)

lemma *state-rank-range*:
  state-rank q n $\in$ {*None*} $\cup$ *Some* ‘ {*0..<max-rank*}
  by (*cases state-rank q n*) (*simp add*: *state-rank-upper-bound*[*of q n*])+

lemma *state-rank-None*:
  $\neg$*sink q* $\Longrightarrow$ *state-rank q n* = *None* $\longleftrightarrow$ *oldest-token q n* = *None*
  by *simp*

lemma *state-rank-Some*:
  $\neg$*sink q* $\Longrightarrow$ ($\exists$ *i*. *state-rank q n* = *Some i*) $\longleftrightarrow$ ($\exists$ *j*. *oldest-token q n* = *Some j*)
  by *simp*

lemma *state-rank-oldest-token*:
  assumes *state-rank p n* = *Some i*
  assumes *state-rank q n* = *Some j*
  assumes *oldest-token p n* = *Some x*
  assumes *oldest-token q n* = *Some y*
  shows $i < j \longleftrightarrow x < y$
proof –
  have *configuration p n* $\neq$ {} **and** *configuration q n* $\neq$ {}
    using *assms*(*3*,*4*) **by** (*metis oldest-token.simps option.distinct*(*1*))+
  moreover
  have $\neg$*sink p* **and** $\neg$*sink q*
    using *assms*(*1*,*2*) *state-rank-sink* **by** *auto*

ultimately
have *i-def*: $i = card\ (senior\text{-}states\ p\ n)$ **and** *j-def*: $j = card\ (senior\text{-}states\ q\ n)$
using *assms*(*1*,*2*) *option.sel* **by** *simp-all*
hence $i < j \longleftrightarrow senior\text{-}states\ p\ n \subset senior\text{-}states\ q\ n$
using *senior-states-card-le* **by** *presburger*
also
with *assms*(*3*,*4*) **have** $\ldots \longleftrightarrow x < y$
proof (*cases rule*: *senior-states-cases-subset*[*of p n q*])
case *equal*
thus *?thesis*
using *assms state-rank-distinct i-def j-def*
by (*metis less-irrefl option.sel*)
qed *auto*
ultimately
show *?thesis*
by *meson*
qed

lemma *state-rank-oldest-token-le*:
assumes $state\text{-}rank\ p\ n = Some\ i$
assumes $state\text{-}rank\ q\ n = Some\ j$
assumes $oldest\text{-}token\ p\ n = Some\ x$
assumes $oldest\text{-}token\ q\ n = Some\ y$
shows $i \leq j \longleftrightarrow x \leq y$
using *state-rank-oldest-token*[*OF assms*] *assms state-rank-distinct oldest-token-equal*
by (*cases* $x = y$) ((*metis option.sel order-refl*), (*metis le-eq-less-or-eq option.inject*))

lemma *state-rank-in-function-set*:
shows $(\lambda q.\ state\text{-}rank\ q\ t) \in \{f.\ (\forall x.\ x \notin reach\ \Sigma\ \delta\ q_0 \longrightarrow f\ x = None) \wedge$
$(\forall x.\ x \in reach\ \Sigma\ \delta\ q_0 \longrightarrow f\ x \in \{None\} \cup Some\ `\ \{0..<max\text{-}rank\})\}$
proof –
{
fix *x*
assume $x \notin reach\ \Sigma\ \delta\ q_0$
hence $\bigwedge token.\ x \neq token\text{-}run\ token\ t$
unfolding *reach-def token-run.simps* **using** *bounded-w* **by** *fastforce*
hence $state\text{-}rank\ x\ t = None$
using *pull-up-configuration-state-rank* **by** *auto*
}
with *state-rank-range* **show** *?thesis*

by *blast*
qed

5.10 Step Function

fun *pre-oldest-tokens* :: $'b \Rightarrow nat \Rightarrow nat\ set$
where
pre-oldest-tokens $q\ n = \{x.\ \exists\, q'.\ oldest\text{-}token\ q'\ n = Some\ x \wedge q = \delta\ q'$
$(w\ n)\} \cup (if\ q = q_0\ then\ \{Suc\ n\}\ else\ \{\})$

lemma *pre-oldest-configuration-range*:
pre-oldest-tokens $q\ n \subseteq \{0..Suc\ n\}$
proof −
have $\{x.\ \exists\, q'.\ oldest\text{-}token\ q'\ n = Some\ x \wedge q = \delta\ q'\ (w\ n)\} \subseteq \{0..n\}$
(**is** *?lhs* ⊆ *?rhs*)
proof
fix x
assume $x \in$ *?lhs*
then obtain q' **where** *oldest-token* $q'\ n = Some\ x$
by *blast*
thus $x \in$ *?rhs*
unfolding *atLeastAtMost-iff* **using** *oldest-token-bounded*[*of* $q'\ n\ x$] **by**
blast
qed
thus *?thesis*
by (*cases* $q = q_0$) *fastforce+*
qed

lemma *pre-oldest-configuration-finite*:
finite (*pre-oldest-tokens* $q\ n$)
using *pre-oldest-configuration-range finite-atLeastAtMost* **by** (*rule finite-subset*)

lemmas *pre-oldest-configuration-Min-in* = *Min-in*[*OF pre-oldest-configuration-finite*]

lemma *pre-oldest-configuration-obtain*:
assumes $x \in$ *pre-oldest-tokens* $q\ n - \{Suc\ n\}$
obtains q' **where** *oldest-token* $q'\ n = Some\ x$ **and** $q = \delta\ q'\ (w\ n)$
using *assms* **by** (*cases* $q = q_0$, *auto*)

lemma *pre-oldest-configuration-element*:
assumes *oldest-token* $q'\ n = Some\ ot$
assumes $q = \delta\ q'\ (w\ n)$
shows $ot \in$ *pre-oldest-tokens* $q\ n$
proof

show *ot* ∈ {*ot*. ∃ *q'*. *oldest-token q' n* = *Some ot* ∧ *q* = δ *q'* (*w n*)}
(**is** - ∈ *?A*)
using *assms* **by** *blast*
show *?A* ⊆ *pre-oldest-tokens q n*
by *simp*
qed

lemma *pre-oldest-configuration-initial-state*:
Suc n ∈ *pre-oldest-tokens q n* ⟹ *q* = q_0
using *oldest-token-bounded*[*of - n Suc n*]
by (*cases q* = q_0) *auto*

lemma *pre-oldest-configuration-initial-state-2*:
q = q_0 ⟹ *Suc n* ∈ *pre-oldest-tokens q n*
by *fastforce*

lemma *pre-oldest-configuration-tokens*:
pre-oldest-tokens q n ≠ {} ⟷ *configuration q* (*Suc n*) ≠ {}
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
then obtain *ot* **where** *ot-def*: *ot* ∈ *pre-oldest-tokens q n*
by *blast*
thus *?rhs*
proof (*cases ot* = *Suc n*)
case *True*
thus *?thesis*
using *pre-oldest-configuration-initial-state configuration-non-empty*[*of Suc n Suc n*] ‹*ot* ∈ *pre-oldest-tokens q n*› **unfolding** *token-run-intial-state* **by** *blast*
next
case *False*
then obtain *q'* **where** *oldest-token q' n* = *Some ot* **and** *q* = δ *q'* (*w n*)
using *ot-def pre-oldest-configuration-obtain* **by** *blast*
moreover
hence *configuration q' n* ≠ {}
by (*metis oldest-token.simps option.distinct*(*2*))
ultimately
show *?rhs*
by (*elim configuration-step-non-empty*)
qed
next
assume *?rhs*

then obtain *token* **where** *token* ∈ *configuration q* (*Suc n*) **and** *token* ≤ *Suc n* **and** *token-run token* (*Suc n*) = *q*
by *auto*
moreover
{
assume *token* ≤ *n*
then obtain *q′* **where** *token-run token n* = *q′* **and** *q* = *δ q′* (*w n*)
using ‹*token-run token* (*Suc n*) = *q*› **unfolding** *token-run.simps* *Suc-diff-le*[*OF* ‹*token* ≤ *n*›] **by** *fastforce*
then obtain *ot* **where** *oldest-token q′ n* = *Some ot*
using *oldest-token-always-def* **by** *blast*
with ‹*q* = *δ q′* (*w n*)› **have** *?lhs*
using *pre-oldest-configuration-element* **by** *blast*
}
ultimately
show *?lhs*
using *pre-oldest-configuration-initial-state-2* **by** *fastforce*
qed

lemma *oldest-token-rec*:
oldest-token q (*Suc n*) = (*if pre-oldest-tokens q n* ≠ {} *then Some* (*Min* (*pre-oldest-tokens q n*)) *else None*)
proof (*cases oldest-token q* (*Suc n*))
case (*Some ot*)
moreover
hence *ot* ∈ *configuration q* (*Suc n*)
by (*rule push-down-oldest-token-configuration*)
hence *configuration q* (*Suc n*) ≠ {}
by *blast*
hence *pre-oldest-tokens q n* ≠ {}
unfolding *pre-oldest-configuration-tokens* **.**
let *?ot* = *Min* (*pre-oldest-tokens q n*)
{
{
{
assume *ot* < *Suc n*
hence *ot* ≠ *Suc n*
by *blast*
then obtain *q′* **where** *ot* ∈ *configuration q′ n* **and** *q* = *δ q′* (*w n*)
using *configuration-rev-step′* ‹*ot* ∈ *configuration q* (*Suc n*)› **by** *metis*
{
fix *token*
assume *token* ∈ *configuration q′ n*

hence *token* ∈ *configuration q* (*Suc n*)
using ‹$q = \delta\ q'$ (*w n*)› **by** (*rule configuration-step*)
hence *ot* ≤ *token*
using *Some* **by** (*metis Min.coboundedI* ‹*configuration q* (*Suc n*) ≠ {}› *configuration-finite oldest-token.simps option.inject*)
}
hence *Min* (*configuration* q' *n*) = *ot*
by (*metis Min-eqI* ‹*ot* ∈ *configuration* q' *n*› *configuration-finite*)
hence *oldest-token* q' *n* = *Some ot*
using ‹*ot* ∈ *configuration* q' *n*› **unfolding** *oldest-token.simps* **by** *auto*
hence *ot* ∈ *pre-oldest-tokens q n*
using ‹$q = \delta\ q'$ (*w n*)› **by** (*rule pre-oldest-configuration-element*)
}
moreover
{
assume *ot* = *Suc n*
moreover
hence $q = q_0$
using *Some* **by** (*metis push-down-oldest-token-token-run token-run-intial-state*)
ultimately
have *ot* ∈ *pre-oldest-tokens q n*
by *simp*
}
ultimately
have *ot* ∈ *pre-oldest-tokens q n*
using *Some*[*THEN oldest-token-bounded*] **by** *linarith*
}
moreover
{
fix ot' q'
assume *oldest-token* q' *n* = *Some* ot' **and** $q = \delta\ q'$ (*w n*)
moreover
hence ot' ∈ *configuration q* (*Suc n*)
using *push-down-oldest-token-configuration configuration-step* **by** *blast*
hence *ot* ≤ ot'
using *Some* **by** (*metis Min.coboundedI* ‹*configuration q* (*Suc n*) ≠ {}› *configuration-finite oldest-token.simps option.inject*)
}
hence ⋀*y*. *y* ∈ *pre-oldest-tokens q n* − {*Suc n*} ⟹ *ot* ≤ *y*
using *pre-oldest-configuration-obtain* **by** *metis*
hence ⋀*y*. *y* ∈ *pre-oldest-tokens q n* ⟹ *ot* ≤ *y*

using *Some*[*THEN oldest-token-bounded*] **by** *force*
ultimately
have *?ot* = *ot*
using *Min-eqI*[*OF pre-oldest-configuration-finite*, *of q n ot*] **by** *fast*
}
ultimately
show *?thesis*
unfolding *pre-oldest-configuration-tokens oldest-token.simps*
by (*metis* ‹*configuration q* (*Suc n*) $\neq$ {}›)
qed (*unfold pre-oldest-configuration-tokens oldest-token.simps*, *metis option.distinct*(*2*))

lemma *pre-ranks-range*:
pre-ranks (λ*q*. *state-rank q n*) ν *q* $\subseteq$ {*0..max-rank*}
proof −
have {*i* | *q' i*. *state-rank q' n* = *Some i* $\wedge$ *q* = δ *q'* ν} $\subseteq$ {*0..max-rank*}
using *state-rank-upper-bound* **by** *fastforce*
thus *?thesis*
by *auto*
qed

lemma *pre-ranks-finite*:
finite (*pre-ranks* (λ*q*. *state-rank q n*) ν *q*)
using *pre-ranks-range finite-atLeastAtMost* **by** (*rule finite-subset*)

lemmas *pre-ranks-Min-in* = *Min-in*[*OF pre-ranks-finite*]

lemma *pre-ranks-state-obtain*:
assumes r_q $\in$ *pre-ranks r* ν *q* − {*max-rank*}
obtains *q'* **where** *r q'* = *Some* r_q **and** *q* = δ *q'* ν
using *assms* **by** (*cases q* = q_0, *auto*)

lemma *pre-ranks-element*:
assumes *state-rank q' n* = *Some r*
assumes *q* = δ *q'* (*w n*)
shows *r* $\in$ *pre-ranks* (λ*q*. *state-rank q n*) (*w n*) *q*
proof
show *r* $\in$ {*i*. $\exists$ *q'*. (λ*q*. *state-rank q n*) *q'* = *Some i* $\wedge$ *q* = δ *q'* (*w n*)}
(**is** - $\in$ *?A*)
using *assms* **by** *blast*
show *?A* $\subseteq$ *pre-ranks* (λ*q*. *state-rank q n*) (*w n*) *q*
by *simp*
qed

lemma *pre-ranks-initial-state*:

max-rank $\in$ *pre-ranks* $(\lambda q.\ \textit{state-rank}\ q\ n)\ \nu\ q \Longrightarrow q = q_0$
using *state-rank-upper-bound* **by** (*cases* $q = q_0$) *auto*

lemma *pre-ranks-initial-state-2*:
$q = q_0 \Longrightarrow$ *max-rank* $\in$ *pre-ranks* $r\ \nu\ q$
by *fastforce*

lemma *pre-ranks-tokens*:
assumes $\neg$*sink* q
shows *pre-ranks* $(\lambda q.\ \textit{state-rank}\ q\ n)\ (w\ n)\ q \neq \{\} \longleftrightarrow$ *configuration* q $(\textit{Suc}\ n) \neq \{\}$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
thus *?rhs*
proof (*cases* $q \neq q_0$)
case *True*
hence $\{i.\ \exists\, q'.\ \textit{state-rank}\ q'\ n = \textit{Some}\ i \wedge q = \delta\ q'\ (w\ n)\} \neq \{\}$
using ‹*?lhs*› **by** *simp*
then obtain q' **where** *state-rank* q' $n \neq$ *None* **and** $q = \delta\ q'\ (w\ n)$
by *blast*
moreover
hence *configuration* q' $n \neq \{\}$
unfolding *state-rank.simps* **by** *meson*
ultimately
show *?rhs*
by (*elim configuration-step-non-empty*)
qed *auto*
next
assume *?rhs*
then obtain *token* **where** *token* $\in$ *configuration* q $(\textit{Suc}\ n)$ **and** *token* $\leq$ *Suc* n **and** *token-run token* $(\textit{Suc}\ n) = q$
by *auto*
moreover
{
assume *token* $\leq n$
then obtain q' **where** *token-run token* $n = q'$ **and** $q = \delta\ q'\ (w\ n)$
using ‹*token-run token* $(\textit{Suc}\ n) = q$› **unfolding** *token-run.simps* *Suc-diff-le*[*OF* ‹*token* $\leq n$›] **by** *fastforce*
hence $\neg$*sink* q'
using ‹$\neg$*sink* q› *sink-rev-step bounded-w* **by** *blast*
then obtain r **where** *state-rank* q' $n =$ *Some* r
using ‹$\neg$*sink* q› *configuration-non-empty*[*OF* ‹*token* $\leq n$›] **unfolding** ‹*token-run token* $n = q'$› **by** *simp*

```
    with ‹q = δ q' (w n)› have ?lhs
      using pre-ranks-element by blast
  }
  ultimately
  show ?lhs
    by fastforce
qed

lemma pre-ranks-pre-oldest-token-Min-state-special:
  assumes ¬sink q
  assumes configuration q (Suc n) ≠ {}
  shows Min (pre-ranks (λq. state-rank q n) (w n) q) = max-rank ⟷ Min
(pre-oldest-tokens q n) = Suc n
  (is ?lhs ⟷ ?rhs)
proof
  from assms have pre-oldest-tokens q n ≠ {}
    and pre-ranks (λq. state-rank q n) (w n) q ≠ {}
    using pre-ranks-tokens pre-oldest-configuration-tokens by simp-all

  {
    assume ?lhs
    have q = q0
      apply (rule ccontr)
        using state-rank-upper-bound pre-ranks-Min-in[OF ‹pre-ranks (λq.
state-rank q n) (w n) q ≠ {}›] ‹?lhs›
      by auto
    moreover
    {
      fix q'
      assume q = δ q' (w n)
      hence ¬sink q'
        using ‹¬sink q› bounded-w unfolding sink-def
        using calculation by blast
      {
        fix i
        assume state-rank q' n = Some i
        hence False
          using ‹q = δ q' (w n)›
        using Min.coboundedI[OF pre-ranks-finite, of - n (w n) q]
        unfolding ‹?lhs› using state-rank-upper-bound[of q' n] by fastforce
      }
      hence state-rank q' n = None
        by fastforce
      hence oldest-token q' n = None
```

```
    using ‹¬sink q'› by (metis state-rank-None)
  }
  hence {ot. ∃ q'. oldest-token q' n = Some ot ∧ q = δ q' (w n)} = {}
    by fastforce
  ultimately
  show ?rhs
    by auto
}

{
  assume ?rhs
  {
    fix q'
    assume q = δ q' (w n)
    have state-rank q' n = None
    proof (cases oldest-token q' n)
      case (Some t)
        hence t ≤ n
          using oldest-token-bounded[of q' n] by blast
        moreover
        have Suc n ≤ t
          using ‹q = δ q' (w n)›
         using Min.coboundedI[OF pre-oldest-configuration-finite, of - q n]
          unfolding ‹?rhs› using ‹oldest-token q' n = Some t› by auto
        ultimately
        have False
          by linarith
        thus ?thesis
          ..
    qed (unfold state-rank-eq-rank, auto)
  }
  hence X: {i. ∃ q'.  (λq. state-rank q n) q' = Some i ∧ q = δ q' (w n)}
= {}
    by fastforce

  have q = q₀
    apply (rule ccontr)
    using ‹pre-ranks (λq. state-rank q n) (w n) q ≠ {}›
    unfolding pre-ranks.simps X by simp
  hence pre-ranks (λq. state-rank q n) (w n) q = {max-rank}
    unfolding pre-ranks.simps X by force
  thus ?lhs
    by fastforce
}
```

qed

lemma *pre-ranks-pre-oldest-token-Min-state*:
  assumes $\neg$*sink q*
  assumes $q = \delta\ q'\ (w\ n)$
  assumes *configuration q* (*Suc n*) $\neq$ {}
  defines *min-r* $\equiv$ *Min* (*pre-ranks* (λq. *state-rank q n*) (*w n*) *q*)
  defines *min-ot* $\equiv$ *Min* (*pre-oldest-tokens q n*)
  shows *state-rank* q' *n* = *Some min-r* $\longleftrightarrow$ *oldest-token* q' *n* = *Some min-ot*
  (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
  from *assms* **have** *pre-oldest-tokens q n* $\neq$ {} **and** $\neg$*sink* q'
    and *pre-ranks* (λq. *state-rank q n*) (*w n*) *q* $\neq$ {}
     using *pre-ranks-tokens pre-oldest-configuration-tokens bounded-w* **unfolding** *sink-def*
    by (*simp-all*, *metis rangeI subset-iff*)

  {
    assume *?lhs*
    thus *?rhs*
    proof (*cases min-r max-rank rule*: *linorder-cases*)
      case *less*
        then obtain *ot* **where** *oldest-token* q' *n* = *Some ot*
          by (*metis push-down-state-rank-oldest-token* ‹*?lhs*›)
        moreover
        {
          {
            fix q'' ot''
            assume $q = \delta\ q''\ (w\ n)$
            assume *oldest-token* q'' *n* = *Some* ot''
            moreover
            have $\neg$*sink* q''
              using ‹$q = \delta\ q''\ (w\ n)$› *assms* **unfolding** *sink-def*
              by (*metis rangeI subset-eq bounded-w*)
            then obtain r'' **where** *state-rank* q'' *n* = *Some* r''
            using ‹*oldest-token* q'' *n* = *Some* ot''› **by** (*metis state-rank-Some*)
            moreover
            hence $r'' \in$ *pre-ranks* (λq. *state-rank q n*) (*w n*) *q*
              using ‹$q = \delta\ q''\ (w\ n)$› **unfolding** *pre-ranks.simps* **by** *blast*
            then have *min-r* $\leq r''$
             unfolding *min-r-def* **by** (*metis Min.coboundedI pre-ranks-finite*)
            ultimately
            have *ot* $\leq$ ot''
               using *state-rank-oldest-token-le*[*OF* ‹*?lhs*› - ‹*oldest-token* q' *n*

$= Some\ ot$›] **by** *blast*
}
hence $\bigwedge x.\ x \in \{ot.\ \exists\, q'.\ oldest\text{-}token\ q'\ n = Some\ ot \wedge q = \delta\ q'\ (w\ n)\} \Longrightarrow ot \le x$
by *blast*
moreover
have $ot \le Suc\ n$
using *oldest-token-bounded*[*OF* ‹$oldest\text{-}token\ q'\ n = Some\ ot$›] **by** *simp*
ultimately
have $\bigwedge x.\ x \in pre\text{-}oldest\text{-}tokens\ q\ n \Longrightarrow ot \le x$
unfolding *pre-oldest-tokens.simps* **apply** (*cases* $q_0 = q$) **apply** *auto* **done**
hence $ot \le min\text{-}ot$
unfolding *min-ot-def*
unfolding *Min-ge-iff*[*OF pre-oldest-configuration-finite* ‹$pre\text{-}oldest\text{-}tokens\ q\ n \neq \{\}$›, *of ot*]
by *simp*
}
moreover
have $ot \ge min\text{-}ot$
using *Min.coboundedI*[*OF pre-oldest-configuration-finite*] *pre-oldest-configuration-element*
unfolding *min-ot-def* **by** (*metis assms*(*2*) *calculation*(*1*))
ultimately
show *?thesis*
by *simp*
qed (*insert not-less*, *blast intro*: *state-rank-upper-bound less-imp-le-nat*)+
}

{
assume *?rhs*
thus *?lhs*
proof (*cases min-ot Suc n rule*: *linorder-cases*)
case *less*
then obtain *r* **where** $state\text{-}rank\ q'\ n = Some\ r$
using ‹*?rhs*› ‹$\neg sink\ q'$› **by** (*metis state-rank-Some*)
moreover
{
{
fix r''
assume $r'' \in pre\text{-}ranks\ (\lambda q.\ state\text{-}rank\ q\ n)\ (w\ n)\ q - \{max\text{-}rank\}$
then obtain q'' **where** $state\text{-}rank\ q''\ n = Some\ r''$
and $q = \delta\ q''\ (w\ n)$
using *pre-ranks-state-obtain* **by** *blast*

```
          moreover
          then obtain ot'' where oldest-token q'' n = Some ot''
            using push-down-state-rank-oldest-token by fastforce
          moreover
          hence min-ot ≤ ot''
                using ‹q = δ q'' (w n)› pre-oldest-configuration-element
Min.coboundedI pre-oldest-configuration-finite
            unfolding min-ot-def by metis
          ultimately
          have r ≤ r''
            using state-rank-oldest-token-le[OF ‹state-rank q' n = Some r›
- ‹?rhs›] by blast
        }
        moreover
        have r ≤ max-rank
          using state-rank-upper-bound[OF ‹state-rank q' n = Some r›] by
linarith
        ultimately
        have ⋀x. x ∈ pre-ranks (λq. state-rank q n) (w n) q ⟹ r ≤ x
            unfolding pre-ranks.simps apply (cases q0 = q) apply auto
done
        hence r ≤ min-r
            unfolding min-r-def Min-ge-iff[OF pre-ranks-finite ‹pre-ranks
(λq. state-rank q n) (w n) q ≠ {}›]
          by simp
      }
      moreover
      have r ≥ min-r
        using Min.coboundedI[OF pre-ranks-finite] pre-ranks-element
        unfolding min-r-def by (metis assms(2) calculation(1))
      ultimately
      show ?thesis
        by simp
    qed (insert not-less, blast intro: oldest-token-bounded Suc-lessD)+
  }
qed

lemma Min-pre-ranks-pre-oldest-tokens:
  fixes n
  defines r ≡ (λq. state-rank q n)
  assumes configuration p (Suc n) ≠ {}
      and configuration q (Suc n) ≠ {}
  assumes ¬sink q
      and ¬sink p
```

shows *Min* (*pre-ranks r* (*w n*) *p*) < *Min* (*pre-ranks r* (*w n*) *q*) ⟷ *Min* (*pre-oldest-tokens p n*) < *Min* (*pre-oldest-tokens q n*)
(**is** *?lhs* ⟷ *?rhs*)
proof
have *pre-ranks-Min*: ⋀*x ν*. (*x* < *Min* (*pre-ranks r* (*w n*) *q*)) = (∀ *a* ∈ *pre-ranks r* (*w n*) *q*. *x* < *a*)
using *assms pre-ranks-finite Min.bounded-iff pre-ranks-tokens* **by** *simp*
have *pre-oldest-configuration-Min*: ⋀*x*. (*x* < *Min* (*pre-oldest-tokens q n*)) = (∀ *a*∈*pre-oldest-tokens q n*. *x* < *a*)
using *assms pre-oldest-configuration-finite Min.bounded-iff pre-oldest-configuration-tokens* **by** *simp*
have ⋀*x*. *w x* ∈ Σ
using *bounded-w* **by** *auto*

{
let *?min-i* = *Min* (*pre-ranks r* (*w n*) *p*)
let *?min-j* = *Min* (*pre-ranks r* (*w n*) *q*)

assume *?lhs*

have *?min-i* ∈ *pre-ranks r* (*w n*) *p* **and** *?min-j* ∈ *pre-ranks r* (*w n*) *q*
using *Min-in*[*OF pre-ranks-finite*] *assms pre-ranks-tokens* **by** *presburger+*
hence *?min-i* ≤ *max-rank* **and** *?min-j* ≤ *max-rank*
using *pre-ranks-range atLeastAtMost-iff* **unfolding** *r-def* **by** *blast+*
with ‹*?lhs*› **have** *?min-i* ≠ *max-rank*
by *linarith*
then obtain *p′ i′* **where** *i′* = *?min-i* **and** *r p′* = *Some i′* **and** *p* = *δ p′* (*w n*)
using ‹*?min-i* ∈ *pre-ranks r* (*w n*) *p*› **apply** (*cases p* = q_0) **apply** *auto*[*1*] **by** *fastforce*
then obtain *ot′* **where** *oldest-token p′ n* = *Some ot′*
unfolding *assms* **by** (*metis push-down-state-rank-oldest-token*)
have *state-rank p′ n* = *Some ?min-i*
using ‹*i′* = *?min-i*› ‹*r p′* = *Some i′*› **unfolding** *assms* **by** *simp*
hence *ot′* = *Min* (*pre-oldest-tokens p n*)
using *pre-ranks-pre-oldest-token-Min-state*[*OF* ‹¬*sink p*› ‹*p* = *δ p′* (*w n*)› ‹*configuration p* (*Suc n*) ≠ {}›] ‹*oldest-token p′ n* = *Some ot′*›
unfolding *r-def* **by** (*metis option.inject*)
moreover
have *ot′* < *Suc n*
proof (*cases ot′ Suc n rule*: *linorder-cases*)
case *equal*
hence *?min-i* = *max-rank*

using *pre-ranks-pre-oldest-token-Min-state-special*[*of p n*, *OF* ‹¬*sink p*› ‹*configuration p* (*Suc n*) $\neq$ {}›] *assms*
unfolding ‹ot' = *Min* (*pre-oldest-tokens p n*)› **by** *simp*
thus *?thesis*
using ‹*?min-i* $\neq$ *max-rank*› **by** *simp*
next
case *greater*
moreover
have $ot' \in \{0..Suc\ n\}$
using ‹*oldest-token* p' *n* = *Some* ot'›[*THEN oldest-token-bounded*] **by** *fastforce*
ultimately
show *?thesis*
by *simp*
qed *simp*
moreover
{
fix ot_q
assume $ot_q \in$ *pre-oldest-tokens q n* − {*Suc n*}
then obtain q' **where** *oldest-token* q' *n* = *Some* ot_q **and** $q = \delta\ q'$ (*w n*)
using *pre-oldest-configuration-obtain* **by** *blast*
moreover
hence ¬*sink* q'
using ‹¬*sink q*› ‹$\bigwedge x.$ *w x* $\in \Sigma$› **unfolding** *sink-def* **by** *auto*
then obtain r_q **where** *state-rank* q' *n* = *Some* r_q
unfolding *assms state-rank.simps* **using** ‹*oldest-token* q' *n* = *Some* ot_q›
by (*metis oldest-token.simps option.distinct*(*2*))
moreover
hence $r_q \in$ *pre-ranks r* (*w n*) *q*
using ‹$q = \delta\ q'$ (*w n*)›
unfolding *pre-ranks.simps assms* **by** *blast*
hence *?min-j* $\leq r_q$
using *Min.coboundedI*[*OF pre-ranks-finite*] **unfolding** *assms* **by** *blast*
hence *?min-i* $< r_q$
using ‹*?lhs*› **by** *linarith*
hence $ot' < ot_q$
using *state-rank-oldest-token*[*OF* ‹*state-rank* p' *n* = *Some ?min-i*› ‹*state-rank* q' *n* = *Some* r_q› ‹*oldest-token* p' *n* = *Some* ot'› ‹*oldest-token* q' *n* = *Some* ot_q›]
unfolding *assms* **by** *simp*
}
ultimately

show *?rhs*
  using *pre-oldest-configuration-Min* **by** *blast*
}

{
  define *ot-p* **where** *ot-p* = *Min* (*pre-oldest-tokens p n*)
  define *ot-q* **where** *ot-q* = *Min* (*pre-oldest-tokens q n*)
  assume *?rhs*
  hence *ot-p* < *ot-q*
    unfolding *ot-p-def ot-q-def* **.**

  have *oldest-token p* (*Suc n*) = *Some ot-p* **and** *oldest-token q* (*Suc n*) = *Some ot-q*
    unfolding *ot-p-def ot-q-def oldest-token-rec pre-oldest-configuration-tokens* **by** (*metis assms*)+

  define *min-r*$_p$ **where** *min-r*$_p$ = *Min* (*pre-ranks r* (*w n*) *p*)
  hence *min-r*$_p$ ∈ *pre-ranks r* (*w n*) *p*
    using *pre-ranks-Min-in assms pre-ranks-tokens* **by** *simp*
  hence ∗: *min-r*$_p$ < *max-rank*
  proof (*cases min-r*$_p$ *max-rank rule*: *linorder-cases*)
    case *equal*
      hence *ot-p* = *Suc n*
        using *pre-ranks-pre-oldest-token-Min-state-special*[*of p n*, *OF* ‹*configuration p* (*Suc n*) ≠ {}›] *assms*
        unfolding *ot-p-def min-r*$_p$*-def* **by** *simp*
      moreover
      have *Min* (*pre-oldest-tokens q n*) ∈ *pre-oldest-tokens q n*
      using *Min-in*[*OF pre-oldest-configuration-finite*] *assms pre-oldest-configuration-tokens* **by** *presburger*
      hence *ot-q* ∈ {*0..Suc n*}
        using *pre-oldest-configuration-range*[*of q n*]
        unfolding *ot-q-def* **by** *blast*
      hence *ot-q* ≤ *Suc n*
        by *simp*
      ultimately
      show *?thesis*
        using ‹*ot-p* < *ot-q*› **by** *simp*
  next
    case *greater*
      moreover
      have *min-r*$_p$ ∈ {*0..max-rank*}
        using *pre-ranks-range* ‹*min-r*$_p$ ∈ *pre-ranks r* (*w n*) *p*›

unfolding *r-def* **..**
ultimately
show *?thesis*
by *simp*
qed *simp*
moreover
from ∗ **have** $min\text{-}r_p \in pre\text{-}ranks\ r\ (w\ n)\ p - \{max\text{-}rank\}$
using ‹$min\text{-}r_p \in pre\text{-}ranks\ r\ (w\ n)\ p$› **by** *simp*
then obtain p' **where** $r\ p' = Some\ min\text{-}r_p$ **and** $p = \delta\ p'\ (w\ n)$
using *pre-ranks-state-obtain* **by** *blast*
hence $oldest\text{-}token\ p'\ n = Some\ ot\text{-}p$
using *pre-ranks-pre-oldest-token-Min-state*[*OF* ‹¬*sink p*› ‹$p = \delta\ p'\ (w\ n)$› ‹*configuration p* (*Suc n*) ≠ {}›]
unfolding *r-def*[*symmetric*] $min\text{-}r_p\text{-}def$[*symmetric*] *ot-p-def*[*symmetric*] **by** (*metis r-def*)
{
fix r_q
assume $r_q \in pre\text{-}ranks\ r\ (w\ n)\ q - \{max\text{-}rank\}$
then obtain q' **where** q': $r\ q' = Some\ r_q\ \ q = \delta\ q'\ (w\ n)$
using *pre-ranks-state-obtain* **by** *blast*
moreover
from q' **obtain** $ot\text{-}q'$ **where** $ot\text{-}q'$: $oldest\text{-}token\ q'\ n = Some\ ot\text{-}q'$
unfolding *assms* **by** (*metis push-down-state-rank-oldest-token*)
moreover
from $ot\text{-}q'$ **have** $ot\text{-}q' \in pre\text{-}oldest\text{-}tokens\ q\ n$
using ‹$q = \delta\ q'\ (w\ n)$›
unfolding *pre-oldest-tokens.simps* **by** *blast*
hence $ot\text{-}q \le ot\text{-}q'$
unfolding *ot-q-def*
by (*rule Min.coboundedI*[*OF pre-oldest-configuration-finite*])
hence $ot\text{-}p < ot\text{-}q'$
using ‹$ot\text{-}p < ot\text{-}q$› **by** *linarith*
ultimately
have $min\text{-}r_p < r_q$
using *state-rank-oldest-token* ‹$r\ p' = Some\ min\text{-}r_p$› ‹$oldest\text{-}token\ p'\ n = Some\ ot\text{-}p$›
unfolding *assms* **by** *blast*
}
ultimately
show *?lhs*
using *pre-ranks-Min* **unfolding** $min\text{-}r_p\text{-}def$ **by** *blast*
}
qed

5.10.1 Definition of initial and step

```
lemma state-rank-initial:
  state-rank q 0 = initial q
  using state-rank-initial-state by force

lemma state-rank-step:
  state-rank q (Suc n) = step (λq. state-rank q n) (w n) q
  (is ?lhs = ?rhs)
proof (cases sink q)
  case False
    {
      assume configuration q (Suc n) = {}
      hence ?thesis
        using False pull-up-configuration-state-rank pre-ranks-tokens
        unfolding step.simps by presburger
    }
    moreover
    {
      assume configuration q (Suc n) ≠ {}
      hence ?lhs = Some (card (senior-states q (Suc n)))
        using False unfolding state-rank.simps by presburger
      also
      have ... = ?rhs
      proof −
        let ?r = λq. state-rank q n
        have {q'. ¬sink q' ∧ pre-ranks ?r (w n) q' ≠ {} ∧ Min (pre-ranks ?r
(w n) q') < Min (pre-ranks ?r (w n) q)} = senior-states q (Suc n)
          (is ?S = ?S')
        proof (rule set-eqI)
          fix q'
          have q' ∈ ?S ⟷ ¬sink q' ∧ configuration q' (Suc n) ≠ {} ∧ Min
(pre-ranks ?r (w n) q') < Min (pre-ranks ?r (w n) q)
            using pre-ranks-tokens by blast
          also
            have ... ⟷ ¬sink q' ∧ configuration q' (Suc n) ≠ {} ∧ Min
(pre-oldest-tokens q' n) < Min (pre-oldest-tokens q n)
        by (metis ‹configuration q (Suc n) ≠ {}› ‹¬sink q› Min-pre-ranks-pre-oldest-tokens)
          also
          have ... ⟷ ¬sink q' ∧ (∃ x y. oldest-token q' (Suc n) = Some y
∧ oldest-token q (Suc n) = Some x ∧ y < x)
        unfolding oldest-token-rec by (metis pre-oldest-configuration-tokens
‹configuration q (Suc n) ≠ {}› option.distinct(2) option.sel)
          finally
```

show $q' \in ?S \longleftrightarrow q' \in ?S'$
  unfolding *senior-states.simps* **by** *blast*
qed
thus *?thesis*
  using ‹¬*sink q*› ‹*configuration q* (*Suc n*) ≠ {}›
 unfolding *step.simps pre-ranks-tokens*[*OF* ‹¬*sink q*›] **by** *presburger*
qed
finally
have *?thesis* **.**
}
ultimately
show *?thesis*
  by *blast*
qed *auto*

lemma *state-rank-step-foldl*:
  (λq. *state-rank q n*) = *foldl step initial* (*map w* [*0*..<*n*])
  by (*induction n*) (*unfold state-rank-initial state-rank-step*, *simp-all*)

end

end

6 Mojmir Automata

theory *Mojmir*
  imports *Main Semi-Mojmir*
begin

6.1 Definitions

locale *mojmir-def* = *semi-mojmir-def* +
  fixes
    — Final States
    F :: *'b set*
begin

definition *token-succeeds* :: *nat* ⇒ *bool*
where
  token-succeeds x = (∃ *n*. *token-run x n* ∈ *F*)

definition *token-fails* :: *nat* ⇒ *bool*
where
  token-fails x = (∃ *n*. *sink* (*token-run x n*) ∧ *token-run x n* ∉ *F*)

definition *accept* :: *bool* (‹$accept_M$›)
where
accept ⟷ ($\forall_\infty x$. *token-succeeds* x)

definition *fail* :: *nat set*
where
fail = {x. *token-fails* x}

definition *merge* :: *nat* ⇒ (*nat* × *nat*) *set*
where
merge i = {(x, y) | x y n j. $j < i$
∧ (*token-run* x n ≠ *token-run* y n ∧ *rank* y n ≠ *None* ∨ y = *Suc* n)
∧ *token-run* x (*Suc* n) = *token-run* y (*Suc* n)
∧ *token-run* x (*Suc* n) ∉ F
∧ *rank* x n = *Some* j}

definition *succeed* :: *nat* ⇒ *nat set*
where
succeed i = {x. ∃ n. *rank* x n = *Some* i
∧ *token-run* x n ∉ $F - \{q_0\}$
∧ *token-run* x (*Suc* n) ∈ F}

definition *smallest-accepting-rank* :: *nat option*
where
smallest-accepting-rank ≡ (*if accept then*
Some (*LEAST* i. *finite fail* ∧ *finite* (*merge* i) ∧ *infinite* (*succeed* i)) *else None*)

definition *fail-t* :: *nat set*
where
fail-t = {n. ∃ q q'. *state-rank* q n ≠ *None* ∧ $q' = \delta\ q\ (w\ n)$ ∧ $q' \notin F$ ∧ *sink* q'}

definition *merge-t* :: *nat* ⇒ *nat set*
where
merge-t i = {n. ∃ q q' j. *state-rank* q n = *Some* j ∧ $j < i$ ∧ $q' = \delta\ q\ (w\ n)$ ∧ $q' \notin F$ ∧
((∃ q''. $q'' \neq q$ ∧ $q' = \delta\ q''\ (w\ n)$ ∧ *state-rank* q'' n ≠ *None*) ∨ $q' = q_0$)}

definition *succeed-t* :: *nat* ⇒ *nat set*
where
succeed-t i = {n. ∃ q. *state-rank* q n = *Some* i ∧ $q \notin F - \{q_0\}$ ∧ $\delta\ q\ (w\ n) \in F$}

fun $\mathcal{S}$
where
$\mathcal{S}$ *n* = *F* ∪ {*q*. (∃ *j* ≥ *the smallest-accepting-rank*. *state-rank q n* = *Some j*)}

end

locale *mojmir* = *semi-mojmir* + *mojmir-def* +
assumes
— All states reachable from final states are also final
wellformed-F: $\bigwedge q\ \nu.\ q \in F \Longrightarrow \delta\ q\ \nu \in F$
begin

lemma *token-stays-in-final-states*:
token-run x n ∈ *F* ⟹ *token-run x* (*n* + *m*) ∈ *F*
proof (*induction m*)
case (*Suc m*)
thus *?case*
proof (*cases n* + *m* < *x*)
case *False*
hence *n* + *m* ≥ *x*
by *arith*
then obtain *j* **where** *n* + *m* = *x* + *j*
using *le-Suc-ex* **by** *blast*
hence δ (*token-run x* (*n* + *m*)) (*suffix x w j*) = *token-run x* (*n* + (*Suc m*))
unfolding *suffix-def* **by** *fastforce*
thus *?thesis*
using *wellformed-F Suc suffix-nth* **by** (*metis* (*no-types*, *opaque-lifting*))
qed *fastforce*
qed *simp*

lemma *token-run-enter-final-states*:
assumes *token-run x n* ∈ *F*
shows ∃ *m* ≥ *x*. *token-run x m* ∉ *F* − {q_0} ∧ *token-run x* (*Suc m*) ∈ *F*
proof (*cases x* ≤ *n*)
case *True*
then obtain *n′* **where** *token-run x* (*x* + *n′*) ∈ *F*
using *assms* **by** *force*
hence ∃ *m*. *token-run x* (*x* + *m*) ∉ *F* − {q_0} ∧ *token-run x* (*x* + *Suc m*) ∈ *F*
by (*induction n′*) ((*metis* (*erased*, *opaque-lifting*) *token-stays-in-final-states token-run-intial-state Diff-iff Nat.add-0-right Suc-eq-plus1 insertCI*), *blast*)

thus *?thesis*
by (*metis add-Suc-right le-add1*)
next
case *False*
hence $token\text{-}run\ x\ x \notin F - \{q_0\}$ **and** $token\text{-}run\ x\ (Suc\ x) \in F$
using *assms wellformed-F* **by** *simp-all*
thus *?thesis*
by *blast*
qed

6.2 Token Properties

6.2.1 Alternative Definitions

lemma *token-succeeds-alt-def*:
$token\text{-}succeeds\ x = (\forall_\infty n.\ token\text{-}run\ x\ n \in F)$
unfolding *token-succeeds-def MOST-nat-le le-iff-add*
using *token-stays-in-final-states* **by** *blast*

lemma *token-fails-alt-def*:
$token\text{-}fails\ x = (\forall_\infty n.\ sink\ (token\text{-}run\ x\ n) \wedge token\text{-}run\ x\ n \notin F)$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *n* **where** *sink* (*token-run x n*) **and** $token\text{-}run\ x\ n \notin F$
using *token-fails-def* **by** *blast*
hence $\forall m \geq n.\ sink\ (token\text{-}run\ x\ m)$ **and** $\forall m \geq n.\ token\text{-}run\ x\ m \notin F$
using *token-stays-in-sink* **unfolding** *le-iff-add* **by** *auto*
thus *?rhs*
unfolding *MOST-nat-le* **by** *blast*
qed (*unfold MOST-nat-le token-fails-def*, *blast*)

lemma *token-fails-alt-def-2*:
$token\text{-}fails\ x \longleftrightarrow \neg token\text{-}succeeds\ x \wedge \neg token\text{-}squats\ x$
by (*metis add.commute token-fails-def token-squats-def token-stays-in-final-states token-stays-in-sink token-succeeds-def*)

6.2.2 Properties

lemma *token-succeeds-run-merge*:
$x \leq n \Longrightarrow y \leq n \Longrightarrow token\text{-}run\ x\ n = token\text{-}run\ y\ n \Longrightarrow token\text{-}succeeds\ x \Longrightarrow token\text{-}succeeds\ y$
using *token-run-merge token-stays-in-final-states add.commute* **unfolding** *token-succeeds-def* **by** *metis*

lemma *token-squats-run-merge*:
$x \le n \Longrightarrow y \le n \Longrightarrow$ *token-run* x n = *token-run* y $n \Longrightarrow$ *token-squats* $x \Longrightarrow$ *token-squats* y
using *token-run-merge token-stays-in-sink add.commute* **unfolding** *token-squats-def* **by** *metis*

6.2.3 Pulled-Up Lemmas

lemma *configuration-token-succeeds*:
$[\![x \in$ *configuration* q n; $y \in$ *configuration* q $n]\!] \Longrightarrow$ *token-succeeds* x = *token-succeeds* y
using *token-succeeds-run-merge push-down-configuration-token-run* **by** *meson*

lemma *configuration-token-squats*:
$[\![x \in$ *configuration* q n; $y \in$ *configuration* q $n]\!] \Longrightarrow$ *token-squats* x = *token-squats* y
using *token-squats-run-merge push-down-configuration-token-run* **by** *meson*

6.3 Mojmir Acceptance

lemma *Mojmir-reject*:
$\neg$ *accept* $\longleftrightarrow$ ($\exists_\infty x$. $\neg$*token-succeeds* x)
unfolding *accept-def Alm-all-def* **by** *blast*

lemma *mojmir-accept-alt-def*:
accept $\longleftrightarrow$ *finite* $\{x.\ \neg$*token-succeeds* $x\}$
using *Inf-many-def Mojmir-reject* **by** *blast*

lemma *mojmir-accept-initial*:
$q_0 \in F \Longrightarrow$ *accept*
unfolding *accept-def MOST-nat-le token-succeeds-def*
using *token-run-intial-state* **by** *metis*

6.4 Equivalent Acceptance Conditions

6.4.1 Token-Based Definitions

lemma *merge-token-succeeds*:
assumes $(x, y) \in$ *merge* i
shows *token-succeeds* $x \longleftrightarrow$ *token-succeeds* y
proof −
obtain n j j' **where** *token-run* x (*Suc* n) = *token-run* y (*Suc* n)
and *rank* x n = *Some* j **and** *rank* y n = *Some* $j' \vee y$ = *Suc* n

using *assms* **unfolding** *merge-def* **by** *blast*
hence $x \leq Suc\ n$ **and** $y \leq Suc\ n$
using *rank-Some-time le-Suc-eq* **by** *blast+*
then obtain q **where** $x \in configuration\ q\ (Suc\ n)$ **and** $y \in configuration\ q\ (Suc\ n)$
using ‹*token-run x* (*Suc n*) = *token-run y* (*Suc n*)› *pull-up-token-run-tokens* **by** *blast*
thus *?thesis*
using *configuration-token-succeeds* **by** *blast*
qed

lemma *merge-subset*:
$i \leq j \Longrightarrow merge\ i \subseteq merge\ j$
proof
assume $i \leq j$
fix p
assume $p \in merge\ i$
then obtain $x\ y\ n\ k$ **where** $p = (x, y)$ **and** $k < i$ **and** $token\text{-}run\ x\ n \neq token\text{-}run\ y\ n \wedge rank\ y\ n \neq None \vee y = Suc\ n$
and $token\text{-}run\ x\ (Suc\ n) = token\text{-}run\ y\ (Suc\ n)$ **and** $token\text{-}run\ x\ (Suc\ n) \notin F$ **and** $rank\ x\ n = Some\ k$
unfolding *merge-def* **by** *blast*
moreover
hence $k < j$
using ‹$i \leq j$› **by** *simp*
ultimately
have $(x, y) \in merge\ j$
unfolding *merge-def* **by** *blast*
thus $p \in merge\ j$
using ‹$p = (x, y)$› **by** *simp*
qed

lemma *merge-finite*:
$i \leq j \Longrightarrow finite\ (merge\ j) \Longrightarrow finite\ (merge\ i)$
using *merge-subset* **by** (*blast intro*: *rev-finite-subset*)

lemma *merge-finite′*:
$i < j \Longrightarrow finite\ (merge\ j) \Longrightarrow finite\ (merge\ i)$
using *merge-finite*[*of i j*] **by** *force*

lemma *succeed-membership*:
$token\text{-}succeeds\ x \longleftrightarrow (\exists\, i.\ x \in succeed\ i)$
(**is** *?lhs* ⟷ *?rhs*)
proof

assume *?lhs*
then obtain *m* **where** *token-run x m* $\in F$
unfolding *token-succeeds-alt-def MOST-nat-le* **by** *blast*
then obtain *n* **where** *1*: *token-run x n* $\notin F - \{q_0\}$
and *2*: *token-run x* (*Suc n*) $\in F$ **and** $x \leq n$
using *token-run-enter-final-states* **by** *blast*
moreover
hence ¬*sink* (*token-run x n*)
proof (*cases token-run x n* $\neq q_0$)
case *True*
hence *token-run x n* $\notin F$
using ‹*token-run x n* $\notin F - \{q_0\}$› **by** *blast*
thus *?thesis*
using ‹*token-run x* (*Suc n*) $\in F$› *token-stays-in-sink* **unfolding** *Suc-eq-plus1* **by** *metis*
qed (*simp add*: *sink-def*)
then obtain *i* **where** *rank x n* = *Some i*
using ‹$x \leq n$› **by** *fastforce*
ultimately
show *?rhs*
unfolding *succeed-def* **by** *blast*
qed (*unfold token-succeeds-def succeed-def*, *blast*)

lemma *stable-rank-succeed*:
assumes *infinite* (*succeed i*)
and $x \in$ *succeed i*
and $q_0 \notin F$
shows ¬*stable-rank x i*
proof
assume *stable-rank x i*
then obtain *n* **where** $\forall n' \geq n$. *rank x n'* = *Some i*
unfolding *stable-rank-def MOST-nat-le* **by** *rule*

from *assms*(*2*) **obtain** *m* **where** *token-run x m* $\notin F$
and *token-run x* (*Suc m*) $\in F$
and *rank x m* = *Some i*
using *assms*(*3*) **unfolding** *succeed-def* **by** *force*

obtain *y* **where** $y >$ *max n m* **and** $y \in$ *succeed i*
using *assms*(*1*) **unfolding** *infinite-nat-iff-unbounded* **by** *blast*

then obtain *m'* **where** *token-run y m'* $\notin F$
and *token-run y* (*Suc m'*) $\in F$
and *rank y m'* = *Some i*

using *assms*(*3*) **unfolding** *succeed-def* **by** *force*

moreover

— token has still rank i at m'
have $m' \geq n$
using *rank-Some-time*[*OF* ‹*rank y m'* = *Some i*›] ‹$y > max\ n\ m$› **by** *force*
hence *rank x m'* = *Some i*
using ‹$\forall n' \geq n$. *rank x n'* = *Some i*› **by** *blast*

moreover

— but x and y are not in the same state
have $m' \geq Suc\ m$
using *rank-Some-time*[*OF* ‹*rank y m'* = *Some i*›] ‹$y > max\ n\ m$› **by** *force*
hence *token-run x m'* $\in F$
using *token-stays-in-final-states*[*OF* ‹*token-run x* (*Suc m*) $\in F$›]
unfolding *le-iff-add* **by** *fast*
with ‹*token-run y m'* $\notin F$ › **have** *token-run y m'* $\neq$ *token-run x m'*
by *metis*

ultimately

show *False*
using *push-down-rank-tokens* **by** *force*
qed

lemma *stable-rank-bounded*:
assumes *stable*: *stable-rank x j*
assumes *inf*: *infinite* (*succeed i*)
assumes $q_0 \notin F$
shows $j < i$
proof −
from *stable* **obtain** *m* **where** $\forall m' \geq m$. *rank x m'* = *Some j*
unfolding *stable-rank-def MOST-nat-le* **by** *rule*
from *inf* **obtain** *y* **where** $y \geq m$ **and** $y \in$ *succeed i*
unfolding *infinite-nat-iff-unbounded-le* **by** *meson*
then obtain *n* **where** *rank y n* = *Some i*
unfolding *succeed-def MOST-nat-le* **by** *blast*

moreover

hence $n \geq y$
  by (*rule rank-Some-time*)
hence *rank x n* = *Some j*
  using ‹$\forall m' \geq m$. *rank x m'* = *Some j*› ‹$y \geq m$› **by** *fastforce*

ultimately

— In the case $i \leq j$, the token y has also to stabilise with i at n.
have $i \leq j \Longrightarrow$ *stable-rank y i*
  using *stable* **by** (*blast intro*: *stable-rank-tower*)
thus $j < i$
  using *stable-rank-succeed*[*OF inf* ‹$y \in$ *succeed i*› ‹$q_0 \notin F$›] **by** *linarith*
qed

— Relation to Mojmir Acceptance

lemma *mojmir-accept-token-set-def1*:
  assumes *accept*
  shows $\exists i <$ *max-rank*. *finite fail* $\wedge$ *finite* (*merge i*) $\wedge$ *infinite* (*succeed i*) $\wedge$ ($\forall j < i$. *finite* (*succeed j*))
proof (*rule*+)
  define *i* **where** i = (*LEAST k*. *infinite* (*succeed k*))

  from *assms* **have** *infinite* {*t*. *token-succeeds t*}
    unfolding *mojmir-accept-alt-def* **by** *force*

  moreover

  have {*x*. *token-succeeds x*} = $\bigcup$ {*succeed i* | *i*. $i <$ *max-rank*}
    (**is** *?lhs* = *?rhs*)
  proof −
    have *?lhs* = $\bigcup$ {*succeed i* | *i*. *True*}
      using *succeed-membership* **by** *blast*
    also
    have ... = *?rhs*
    proof
      show ... $\subseteq$ *?rhs*
      proof
        fix *x*
        assume $x \in \bigcup$ {*succeed i* |*i*. *True*}
        then obtain *i* **where** $x \in$ *succeed i*
          by *blast*
        moreover
        — Obtain upper bound for succeed ranks

have $\bigwedge u.\ u \geq$ *max-rank* $\Longrightarrow$ *succeed* $u = \{\}$
unfolding *succeed-def* **using** *rank-upper-bound* **by** *fastforce*
ultimately
show $x \in \bigcup\{$*succeed* $i\ |i.\ i <$ *max-rank*$\}$
by (*cases* $i <$ *max-rank*) (*blast*, *simp*)
qed
qed *blast*
finally
show *?thesis* **.**
qed

ultimately

have $\exists j.$ *infinite* (*succeed* j)
by *force*
hence *infinite* (*succeed* i) **and** $\bigwedge j.\ j < i \Longrightarrow$ *finite* (*succeed* j)
unfolding *i-def* **by** (*metis LeastI-ex*, *metis not-less-Least*)
hence *fin-succeed-ranks*: *finite* ($\bigcup\{$*succeed* $j\ |\ j.\ j < i\}$)
by *auto*

— i is bounded by *max-rank*
{
obtain x **where** $x \in$ *succeed* i
using ‹*infinite* (*succeed* i)› **by** *fastforce*
then obtain n **where** *rank* $x\ n$ = *Some* i
unfolding *succeed-def* **by** *blast*
thus $i <$ *max-rank*
by (*rule rank-upper-bound*)
}

define S **where** $S = \{(x, y).$ *token-succeeds* $x \wedge$ *token-succeeds* $y\}$

have *finite* (*merge* $i \cap S$)
proof (*rule finite-product*)
{
fix $x\ y$
assume $(x, y) \in$ (*merge* $i \cap S$)

then obtain $n\ k\ k''$ **where** $k < i$
and *rank* $x\ n$ = *Some* k
and *rank* $y\ n$ = *Some* $k'' \vee y$ = *Suc* n
and *token-run* x (*Suc* n) $\notin F$
and *token-run* x (*Suc* n) = *token-run* y (*Suc* n)
and *token-succeeds* x

unfolding *merge-def S-def* **by** *fast*

then obtain m **where** *token-run* x (*Suc* n + m) $\notin F$
and *token-run* x (*Suc* (*Suc* n + m)) $\in F$
by (*metis Suc-eq-plus1 add.commute token-run-P*[*of* $\lambda q.\ q \in F$] *token-stays-in-final-states token-succeeds-def*)

moreover

have $x \leq$ *Suc* n **and** $y \leq$ *Suc* n **and** $x \leq$ *Suc* $n + m$ **and** $y \leq$ *Suc* n + m
using *rank-Some-time* ‹*rank* x n = *Some* k› ‹*rank* y n = *Some* $k'' \vee$ y = *Suc* n› **by** *fastforce+*

hence *token-run* y (*Suc* n + m) $\notin F$ **and** *token-run* y (*Suc* (*Suc* n + m)) $\in F$
using ‹*token-run* x (*Suc* n + m) $\notin F$› ‹*token-run* x (*Suc* (*Suc* n + m)) $\in F$› ‹*token-run* x (*Suc* n) = *token-run* y (*Suc* n)›
using *token-run-merge token-run-merge-Suc* **by** *metis+*

moreover

have ¬*sink* (*token-run* x (*Suc* n + m))
using ‹*token-run* x (*Suc* n + m) $\notin F$› ‹*token-run* x (*Suc*(*Suc* n + m)) $\in F$›
using *token-is-not-in-sink* **by** *blast*

— Obtain rank used to enter final
obtain k' **where** *rank* x (*Suc* n + m) = *Some* k'
using ‹¬*sink* (*token-run* x (*Suc* n + m))› ‹$x \leq$ *Suc* n + m› **by** *fastforce*

moreover

hence *rank* y (*Suc* n + m) = *Some* k'
by (*metis* ‹$x \leq$ *Suc* n + m› ‹$y \leq$ *Suc* n + m› *token-run-merge* ‹$x \leq$ *Suc* n› ‹$y \leq$ *Suc* n›
‹*token-run* x (*Suc* n) = *token-run* y (*Suc* n)› *pull-up-token-run-tokens*
pull-up-configuration-rank[*of* x - *Suc* n + m y])

moreover

— Rank used to enter final states is strictly bounded by i
have $k' < i$

using ‹*rank x* (*Suc n* + *m*) = *Some k′*› *rank-monotonic*[*OF* ‹*rank x n* = *Some k*›] ‹*k* < *i*›
unfolding *add-Suc-shift* **by** *fastforce*

ultimately

have $x \in \bigcup\{succeed\ j \mid j.\ j < i\}$ **and** $y \in \bigcup\{succeed\ j \mid j.\ j < i\}$
unfolding *succeed-def* **by** *blast+*
}
hence *fst* ‘ (*merge i* ∩ *S*) ⊆ ⋃{*succeed j* | *j*. *j* < *i*} **and** *snd* ‘ (*merge i* ∩ *S*) ⊆ ⋃{*succeed j* | *j*. *j* < *i*}
by *force+*
thus *finite* (*fst* ‘ (*merge i* ∩ *S*)) **and** *finite* (*snd* ‘ (*merge i* ∩ *S*))
using *finite-subset*[*OF* - *fin-succeed-ranks*] **by** *meson+*
qed

moreover

have *finite* (*merge i* ∩ (*UNIV* − *S*))
proof −
obtain *l* **where** *l-def*: ∀ *x* ≥ *l*. *token-succeeds x*
using *assms* **unfolding** *accept-def MOST-nat-le* **by** *blast*
{
fix *x y*
assume (*x*, *y*) ∈ *merge i* ∩ (*UNIV* − *S*)
hence ¬*token-succeeds x* ∨ ¬*token-succeeds y*
unfolding *S-def* **by** *simp*
hence ¬*token-succeeds x* ∧ ¬*token-succeeds y*
using *merge-token-succeeds* ‹(*x*, *y*) ∈ *merge i* ∩ (*UNIV* − *S*)› **by** *blast*
hence *x* < *l* **and** *y* < *l*
by (*metis l-def le-eq-less-or-eq linear*)+
}
hence *merge i* ∩ (*UNIV* − *S*) ⊆ {*0..l*} × {*0..l*}
by *fastforce*
thus *?thesis*
using *finite-subset* **by** *blast*
qed

ultimately

have *finite* (*merge i*)
by (*metis Int-Diff Un-Diff-Int finite-UnI inf-top-right*)
moreover

have *finite fail*
by (*metis assms mojmir-accept-alt-def fail-def token-fails-alt-def-2 infinite-nat-iff-unbounded-le mem-Collect-eq*)
ultimately
show *finite fail* ∧ *finite* (*merge i*) ∧ *infinite* (*succeed i*) ∧ (∀ *j* < *i*. *finite* (*succeed j*))
using ‹*infinite* (*succeed i*)› ‹⋀*j*. *j* < *i* ⟹ *finite* (*succeed j*)› **by** *blast*
qed

lemma *mojmir-accept-token-set-def2*:
assumes *finite fail*
and *finite* (*merge i*)
and *infinite* (*succeed i*)
shows *accept*
proof (*rule ccontr*, *cases* $q_0 \notin F$)
case *True*
assume ¬ *accept*
moreover
have *finite* {*x*. ¬*token-succeeds x* ∧ ¬*token-squats x*}
using ‹*finite fail*› **unfolding** *fail-def token-fails-alt-def-2*[*symmetric*] **.**
moreover
have *X*: {*x*. ¬ *token-succeeds x*} = {*x*. ¬ *token-succeeds x* ∧ *token-squats x*} ∪ {*x*. ¬ *token-succeeds x* ∧ ¬ *token-squats x*}
by *blast*
ultimately
have *inf*: *infinite* {*x*. ¬*token-succeeds x* ∧ *token-squats x*}
unfolding *mojmir-accept-alt-def X* **by** *blast*

— Obtain j, where j is the rank used by infinitely many configuration stabilising and not succeeding
have {*x*. ¬*token-succeeds x* ∧ *token-squats x*} = {*x*. ∃ *j* < *i*. ¬*token-succeeds x* ∧ *token-squats x* ∧ *stable-rank x j*}
using *stable-rank-bounded* ‹*infinite* (*succeed i*)› ‹$q_0 \notin F$›
unfolding *stable-rank-equiv-token-squats* **by** *metis*
also
have ... = ⋃ {{*x*. ¬*token-succeeds x* ∧ *token-squats x* ∧ *stable-rank x j*} | *j*. *j* < *i*}
by *blast*
finally
obtain *j* **where** *j* < *i* **and** *infinite* {*t*. ¬*token-succeeds t* ∧ *token-squats t* ∧ *stable-rank t j*}
(**is** *infinite ?S*)
using *inf* **by** *force*

— Obtain such a token x

then obtain *x* **where** ¬*token-succeeds x* **and** *token-squats x* **and** *stable-rank x j*

unfolding *infinite-nat-iff-unbounded-le* **by** *blast*

then obtain *n* **where** $\forall\, m \geq n.$ *rank x m* = *Some j*

unfolding *stable-rank-def MOST-nat-le* **by** *blast*

— All configuration with same stable rank are bought at some n with rank smaller i

have $\{(x,\ y) \mid y.\ y > n \wedge$ *stable-rank y j*$\} \subseteq$ *merge i*

(**is** *?lhs* ⊆ *?rhs*)

proof

fix *p*

assume *p* ∈ *?lhs*

then obtain *y* **where** $p = (x,\ y)$ **and** $y > n$ **and** *stable-rank y j*

by *blast*

hence $x < y$ **and** $x \neq y$

using *rank-Some-time* ‹∀ *n'*≥*n*. *rank x n'* = *Some j*› **by** *fastforce+*

moreover

— Obtain a time n" where x and y have the same rank

obtain *n''* **where** *rank x n''* = *Some j* **and** *rank y n''* = *Some j*

using ‹∀ *n'*≥*n*. *rank x n'* = *Some j*› ‹*stable-rank y j*›

unfolding *stable-rank-def MOST-nat-le* **by** (*metis add.commute le-add2*)

hence *token-run x n''* = *token-run y n''* **and** $y \leq n''$

using *push-down-rank-tokens rank-Some-time*[*OF* ‹*rank y n''* = *Some j*›] **by** *simp-all*

— Obtain the time n' where x merges y and proof all necessary properties

then obtain *n'* **where** *token-run x n'* ≠ *token-run y n'* ∨ *y* = *Suc n'*

and *token-run x* (*Suc n'*) = *token-run y* (*Suc n'*) **and** *y* ≤ *Suc n'*

using *token-run-mergepoint*[*OF* ‹*x* < *y*›] *le-add-diff-inverse* **by** *metis*

moreover

hence (∃ *j'*. *rank y n'* = *Some j'*) ∨ *y* = *Suc n'*

using ‹*stable-rank y j*› *stable-rank-equiv-token-squats rank-token-squats*

unfolding *le-Suc-eq* **by** *blast*

moreover

have *rank x n'* = *Some j*

using ‹∀ $n' \geq n$. *rank x n'* = *Some j*› ‹$y \leq$ *Suc n'*› ‹$y > n$› **by** *fastforce*

moreover

have *token-run x* (*Suc n'*) $\notin F$
using ‹¬ *token-succeeds x*› *token-succeeds-def* **by** *blast*

ultimately
show $p \in$ *?rhs*
unfolding *merge-def* ‹$p = (x, y)$›
using ‹$j < i$› **by** *blast*
qed

moreover

— However, x merges infinitely many configuration
hence *infinite* {(x, y) | y. y > n ∧ *stable-rank y j*}
(**is** *infinite ?S'*)
proof –
{
{
fix *y*
assume *stable-rank y j* **and** $y > n$
then obtain n' **where** *rank y n'* = *Some j*
unfolding *stable-rank-def MOST-nat-le* **by** *blast*
moreover
hence $y \leq n'$
by (*rule rank-Some-time*)
hence $n' > n$
using ‹$y > n$› **by** *arith*
hence *rank x n'* = *Some j*
using ‹∀ $n' \geq n$. *rank x n'* = *Some j*› **by** *simp*
ultimately
have ¬*token-succeeds y*
by (*metis* ‹¬*token-succeeds x*› *configuration-token-succeeds push-down-rank-tokens*)
}
hence {y | y. y > n ∧ *stable-rank y j*} = {y | y. *token-squats y* ∧ ¬*token-succeeds y* ∧ *stable-rank y j* ∧ y > n}
(**is** - = *?S''*)
using *stable-rank-equiv-token-squats* **by** *blast*
moreover
have *finite* {y | y. *token-squats y* ∧ ¬*token-succeeds y* ∧ *stable-rank y j* ∧ y ≤ n}

(**is** *finite ?S'''*)
  by *simp*
moreover
have *?S = ?S'' ∪ ?S'''*
  by *auto*
ultimately
have *infinite {y | y. y > n ∧ stable-rank y j}*
  using ‹*infinite ?S*› **by** *simp*
}
moreover
have *{x} × {y. y > n ∧ stable-rank y j} = ?S'*
  by *auto*
ultimately
show *?thesis*
  by (*metis empty-iff finite-cartesian-productD2 singletonI*)
qed

ultimately

have *infinite* (*merge i*)
  by (*rule infinite-super*)
with ‹*finite* (*merge i*)› **show** *False*
  by *blast*
qed (*blast intro*: *mojmir-accept-initial*)

theorem *mojmir-accept-iff-token-set-accept*:
accept ⟷ (∃ *i* < *max-rank*. *finite fail* ∧ *finite* (*merge i*) ∧ *infinite* (*succeed i*))
using *mojmir-accept-token-set-def1 mojmir-accept-token-set-def2* **by** *blast*

theorem *mojmir-accept-iff-token-set-accept2*:
accept ⟷ (∃ *i* < *max-rank*. *finite fail* ∧ *finite* (*merge i*) ∧ *infinite* (*succeed i*) ∧ (∀ *j* < *i*. *finite* (*merge j*) ∧ *finite* (*succeed j*)))
using *mojmir-accept-token-set-def1 mojmir-accept-token-set-def2 merge-finite'* **by** *blast*

6.4.2 Time-Based Definitions

lemma *finite-monotonic-image*:
fixes *A B* :: *nat set*
assumes $\bigwedge i.\ i \in A \Longrightarrow i \leq f\ i$
assumes *f ‘ A = B*
shows *finite A* ⟷ *finite B*
proof

assume *finite B*
thus *finite A*
proof (*cases* $B \neq \{\}$)
case *True*
hence $\bigwedge i.\ i \in A \Longrightarrow i \leq$ *Max B*
by (*metis assms Max-ge-iff* ‹*finite B*› *imageI*)
thus *finite A*
unfolding *finite-nat-set-iff-bounded-le* **by** *blast*
qed (*metis assms*(*2*) *image-is-empty*)
qed (*metis assms*(*2*) *finite-imageI*)

lemma *finite-monotonic-image-pairs*:
fixes A :: (*nat* × *nat*) *set*
fixes B :: *nat set*
assumes $\bigwedge i.\ i \in A \Longrightarrow (fst\ i) \leq f\ i + c$
assumes $\bigwedge i.\ i \in A \Longrightarrow (snd\ i) \leq f\ i + d$
assumes f ‘ $A = B$
shows *finite A* $\longleftrightarrow$ *finite B*
proof
assume *finite B*
thus *finite A*
proof (*cases* $B \neq \{\}$)
case *True*
hence $\bigwedge i.\ i \in A \Longrightarrow fst\ i \leq Max\ B + c \wedge snd\ i \leq Max\ B + d$
by (*metis assms Max-ge-iff* ‹*finite B*› *imageI le-diff-conv*)
thus *finite A*
using *finite-product*[*of A*] **unfolding** *finite-nat-set-iff-bounded-le* **by** *blast*
qed (*metis assms*(*3*) *finite.emptyI image-is-empty*)
qed (*metis assms*(*3*) *finite-imageI*)

lemma *token-time-finite-rule*:
fixes *A B* :: *nat set*
assumes *unique*: $\bigwedge x\ y\ z.\ P\ x\ y \Longrightarrow P\ x\ z \Longrightarrow y = z$
and *existsA*: $\bigwedge x.\ x \in A \Longrightarrow (\exists\, y.\ P\ x\ y)$
and *existsB*: $\bigwedge y.\ y \in B \Longrightarrow (\exists\, x.\ P\ x\ y)$
and *inA*: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow x \in A$
and *inB*: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow y \in B$
and *mono*: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow x \leq y$
shows *finite A* $\longleftrightarrow$ *finite B*
proof (*rule finite-monotonic-image*)
let *?f* = (λx. *if* $x \in A$ *then The* (*P x*) *else undefined*)

{

```
    fix x
    assume x ∈ A
    then obtain y where P x y and y = ?f x
      using existsA the-equality unique by metis
    thus x ≤ ?f x
      using mono by blast
  }

  {
    fix y
    have y ∈ ?f ‘ A ⟷ (∃ x. x ∈ A ∧ y = The (P x))
      unfolding image-def by force
    also
    have ... ⟷ (∃ x. P x y)
      by (metis inA existsA unique the-equality)
    also
    have ... ⟷ y ∈ B
      using inB existsB by blast
    finally
    have y ∈ ?f ‘ A ⟷ y ∈ B
      .
  }
  thus ?f ‘ A = B
    by blast
qed

lemma token-time-finite-pair-rule:
  fixes A :: (nat × nat) set
  fixes B :: nat set
  assumes unique: ⋀x y z. P x y ⟹ P x z ⟹ y = z
      and existsA: ⋀x. x ∈ A ⟹ (∃ y. P x y)
      and existsB: ⋀y. y ∈ B ⟹ (∃ x. P x y)
      and inA:     ⋀x y. P x y ⟹ x ∈ A
      and inB:     ⋀x y. P x y ⟹ y ∈ B
      and mono:    ⋀x y. P x y ⟹ fst x ≤ y + c ∧ snd x ≤ y + d
  shows finite A ⟷ finite B
proof (rule finite-monotonic-image-pairs)
  let ?f = (λx. if x ∈ A then The (P x) else undefined)

  {
    fix x
    assume x ∈ A
    then obtain y where P x y and y = ?f x
      using existsA the-equality unique by metis
```

```
    thus fst x ≤ ?f x + c and snd x ≤ ?f x + d
      using mono by blast+
  }

  {
    fix y
    have y ∈ ?f ‘ A ⟷ (∃ x. x ∈ A ∧ y = The (P x))
      unfolding image-def by force
    also
    have ... ⟷ (∃ x. P x y)
      by (metis inA existsA unique the-equality)
    also
    have ... ⟷ y ∈ B
      using inB existsB by blast
    finally
    have y ∈ ?f ‘ A ⟷ y ∈ B
      .
  }
  thus ?f ‘ A = B
    by blast
qed
```

— Correspondence Between Token- and Time-Based Definitions

```
lemma fail-t-inclusion:
  assumes x ≤ n
  assumes ¬sink (token-run x n)
  assumes sink (token-run x (Suc n))
  assumes token-run x (Suc n) ∉ F
  shows n ∈ fail-t
proof −
  define q q′ where q = token-run x n and q′ = token-run x (Suc n)
  hence ∗: ¬sink q sink q′ and q′ ∉ F
    using assms by blast+
  moreover
  from ∗ have ∗∗: state-rank q n ≠ None
     unfolding q-def by (metis oldest-token-always-def option.distinct(1)
state-rank-None)
  moreover
  from ∗∗ have q′ = δ q (w n)
    unfolding q-def q′-def using assms(1) token-run-step′ by blast
  ultimately
  show n ∈ fail-t
    unfolding fail-t-def by blast
```

qed

lemma *merge-t-inclusion*:
assumes $x \leq n$
assumes $(\exists j'.\ \textit{token-run}\ x\ n \neq \textit{token-run}\ y\ n \wedge y \leq n \wedge \textit{state-rank}\ (\textit{token-run}\ y\ n)\ n = \textit{Some}\ j') \vee y = \textit{Suc}\ n$
assumes $\textit{token-run}\ x\ (\textit{Suc}\ n) = \textit{token-run}\ y\ (\textit{Suc}\ n)$
assumes $\textit{token-run}\ x\ (\textit{Suc}\ n) \notin F$
assumes $\textit{state-rank}\ (\textit{token-run}\ x\ n)\ n = \textit{Some}\ j$
assumes $j < i$
shows $n \in \textit{merge-t}\ i$
proof −
define $q\ q'\ q''$
where $q = \textit{token-run}\ x\ n$
and $q' = \textit{token-run}\ x\ (\textit{Suc}\ n)$
and $q'' = \textit{token-run}\ y\ n$
have $y \leq \textit{Suc}\ n$
using *assms(2)* **by** *linarith*
hence $(q' = \delta\ q''\ (w\ n) \wedge \textit{state-rank}\ q''\ n \neq \textit{None} \wedge q'' \neq q) \vee q' = q_0$
unfolding *q-def q'-def q''-def* **using** *assms(2−3)*
by (*cases y = Suc n*) ((*metis token-run-intial-state*), (*metis option.distinct(1) token-run-step*))
moreover
have $\textit{state-rank}\ q\ n = \textit{Some}\ j \wedge j < i \wedge q' = \delta\ q\ (w\ n) \wedge q' \notin F$
unfolding *q-def q'-def* **using** *token-run-step[OF assms(1)] assms(4−6)* **by** *blast*
ultimately
show $n \in \textit{merge-t}\ i$
unfolding *merge-t-def* **by** *blast*
qed

lemma *succeed-t-inclusion*:
assumes $\textit{rank}\ x\ n = \textit{Some}\ i$
assumes $\textit{token-run}\ x\ n \notin F - \{q_0\}$
assumes $\textit{token-run}\ x\ (\textit{Suc}\ n) \in F$
shows $n \in \textit{succeed-t}\ i$
proof −
define q **where** $q = \textit{token-run}\ x\ n$
hence $\textit{state-rank}\ q\ n = \textit{Some}\ i$ **and** $q \notin F - \{q_0\}$ **and** $\delta\ q\ (w\ n) \in F$
using *token-run-step' rank-Some-time[OF assms(1)] assms rank-eq-state-rank* **by** *auto*
thus $n \in \textit{succeed-t}\ i$
unfolding *succeed-t-def* **by** *blast*
qed

lemma *finite-fail-t*:
finite fail = *finite fail-t*
proof (*rule token-time-finite-rule*)
let *?P* = ($\lambda x\ n.\ x \leq n$
$\wedge \neg$*sink* (*token-run x n*)
$\wedge$ *sink* (*token-run x* (*Suc n*))
$\wedge$ *token-run x* (*Suc n*) $\notin F$)

{
fix *x*
have $\neg$*sink* (*token-run x x*)
unfolding *sink-def* **by** *simp*

assume $x \in$ *fail*
hence *token-fails x*
unfolding *fail-def* **..**
moreover
then obtain y'' **where** *sink* (*token-run x* (*Suc* ($x + y''$)))
unfolding *token-fails-alt-def MOST-nat*
using ‹¬ *sink* (*token-run x x*)› *less-add-Suc2* **by** *blast*
then obtain y' **where** $\neg$*sink* (*token-run x* ($x + y'$)) **and** *sink* (*token-run x* (*Suc* ($x + y'$)))
using *token-run-P*[*of* $\lambda q.$ *sink q*, *OF* ‹¬*sink* (*token-run x x*)›] **by** *blast*
ultimately
show $\exists y.$ *?P x y*
using *token-fails-alt-def-2 token-succeeds-def* **by** (*metis le-add1*)
}

{
fix *y*
assume $y \in$ *fail-t*
then obtain *q q' i* **where** *state-rank q y* = *Some i* **and** $q' = \delta\ q\ (w\ y)$ **and** $q' \notin F$ **and** *sink* q'
unfolding *fail-t-def* **by** *blast*
moreover
then obtain *x* **where** *token-run x y* = *q* **and** $x \leq y$
by (*blast dest*: *push-down-state-rank-token-run*)
moreover
hence *token-run x* (*Suc y*) = q'
using *token-run-step*[*OF* - - ‹$q' = \delta\ q\ (w\ y)$›] **by** *blast*
ultimately
show $\exists x.$ *?P x y*
by (*metis option.distinct*(*1*) *state-rank-sink*)

}

{
fix $x\ y$
assume *?P* $x\ y$
thus $x \in$ *fail* and $x \leq y$ and $y \in$ *fail-t*
unfolding *fail-def* using *token-fails-def fail-t-inclusion* by *blast+*
}

— Uniqueness
{
fix $x\ y\ z$
assume *?P* $x\ y$ and *?P* $x\ z$
from ‹*?P* $x\ y$› have ¬*sink* (*token-run* $x\ y$) and *sink* (*token-run* x (*Suc* y))
by *blast+*
moreover
from ‹*?P* $x\ z$› have ¬*sink* (*token-run* $x\ z$) and *sink* (*token-run* x (*Suc* z))
by *blast+*
ultimately
show $y = z$
using *token-stays-in-sink*
by (*cases* $y\ z$ *rule*: *linorder-cases*, *simp-all*)
(*metis* (*no-types*, *lifting*) *Suc-leI le-add-diff-inverse*)+
}
qed

lemma *finite-succeed-t'*:
assumes $q_0 \notin F$
shows *finite* (*succeed* i) = *finite* (*succeed-t* i)
proof (*rule token-time-finite-rule*)
let *?P* = ($\lambda x\ n.\ x \leq n$
∧ *state-rank* (*token-run* $x\ n$) n = *Some* i
∧ (*token-run* $x\ n$) $\notin F - \{q_0\}$
∧ (*token-run* x (*Suc* n)) $\in F$)

{
fix x
assume $x \in$ *succeed* i
then obtain y where *token-run* $x\ y \notin F - \{q_0\}$ and *token-run* x (*Suc* y) $\in F$ and *rank* $x\ y$ = *Some* i
unfolding *succeed-def* by *force*
moreover

hence *rank* (*senior x y*) *y* = *Some i*
using *rank-Some-time*[*THEN rank-senior-senior*] **by** *presburger*
hence *state-rank* (*token-run x y*) *y* = *Some i*
unfolding *state-rank-eq-rank senior.simps* **by** (*metis oldest-token-always-def option.sel option.simps*(*5*))
ultimately
show $\exists y.$ *?P x y*
using *rank-Some-time* **by** *blast*
}

{
fix *y*
assume $y \in$ *succeed-t i*
then obtain *q* **where** *state-rank q y* = *Some i* **and** $q \notin F - \{q_0\}$ **and** $(\delta\ q\ (w\ y)) \in F$
unfolding *succeed-t-def* **by** *blast*
moreover
then obtain *x* **where** *q* = *token-run x y* **and** $x \le y$
by (*metis oldest-token-bounded push-down-oldest-token-token-run push-down-state-rank-oldest-tok*
moreover
hence *token-run x* (*Suc y*) $\in F$
using *token-run-step* ‹$(\delta\ q\ (w\ y)) \in F$› **by** *simp*
ultimately
show $\exists x.$ *?P x y*
by *meson*
}

{
fix *x y*
assume *?P x y*
thus $x \le y$ **and** $x \in$ *succeed i* **and** $y \in$ *succeed-t i*
unfolding *succeed-def* **using** *rank-eq-state-rank*[*of x y*] *succeed-t-inclusion*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)+
}

— Uniqueness
{
fix *x y z*
assume *?P x y* **and** *?P x z*
from ‹*?P x y*› **have** *token-run x y* $\notin F$ **and** *token-run x* (*Suc y*) $\in F$
using ‹$q_0 \notin F$› **by** *auto*
moreover
from ‹*?P x z*› **have** *token-run x z* $\notin F$ **and** *token-run x* (*Suc z*) $\in F$
using ‹$q_0 \notin F$› **by** *auto*

ultimately
show $y = z$
using *token-stays-in-final-states*
by (*cases y z rule*: *linorder-cases*, *simp-all*)
(*metis le-Suc-ex less-Suc-eq-le not-le*)+
}
qed

lemma *initial-in-F-token-run*:
assumes $q_0 \in F$
shows *token-run x y* $\in F$
using *assms token-stays-in-final-states*[*of - 0*] **by** *fastforce*

lemma *finite-succeed-t''*:
assumes $q_0 \in F$
shows *finite* (*succeed i*) = *finite* (*succeed-t i*)
(**is** *?lhs* = *?rhs*)
proof
have *succeed-t i* = {*n*. *state-rank* q_0 *n* = *Some i*}
unfolding *succeed-t-def* **using** *initial-in-F-token-run assms wellformed-F* **by** *auto*
also
have ... = {*n*. *rank n n* = *Some i*}
unfolding *rank-eq-state-rank*[*OF order-refl*] *token-run-intial-state* **..**
finally
have *succeed-t-alt-def*: *succeed-t i* = {*n*. *rank n n* = *Some i* $\wedge$ *token-run n n* = q_0}
by *simp*

have *succeed-alt-def*: *succeed i* = {*x*. $\exists$ *n*. *rank x n* = *Some i* $\wedge$ *token-run x n* = q_0}
unfolding *succeed-def* **using** *initial-in-F-token-run*[*OF assms*] **by** *auto*

{
assume *?lhs*
moreover
have *succeed-t i* $\subseteq$ *succeed i*
unfolding *succeed-t-alt-def succeed-alt-def* **by** *blast*
ultimately
show *?rhs*
by (*rule rev-finite-subset*)
}

{

assume *?rhs*
then obtain U **where** *U-def*: $\bigwedge x.\ x \in$ *succeed-t* $i \Longrightarrow U \geq x$
unfolding *finite-nat-set-iff-bounded-le* **by** *blast*
{
fix x
assume $x \in$ *succeed* i
then obtain n **where** *rank* $x\ n =$ *Some* i **and** *token-run* $x\ n = q_0$
unfolding *succeed-alt-def* **by** *blast*
moreover
hence $x \leq n$
by (*blast dest*: *rank-Some-time*)
moreover
hence *rank* $n\ n =$ *Some* i
using ‹*rank* $x\ n =$ *Some* i› ‹*token-run* $x\ n = q_0$›
by (*metis order-refl token-run-intial-state*[*of* n] *pull-up-token-run-tokens pull-up-configuration-rank*)
hence $n \in$ *succeed-t* i
unfolding *succeed-t-alt-def* **by** *simp*
ultimately
have $U \geq x$
using *U-def* **by** *fastforce*
}
thus *?lhs*
unfolding *finite-nat-set-iff-bounded-le* **by** *blast*
}
qed

lemma *finite-succeed-t*:
finite (*succeed* i) = *finite* (*succeed-t* i)
using *finite-succeed-t′ finite-succeed-t″* **by** *blast*

lemma *finite-merge-t*:
finite (*merge* i) = *finite* (*merge-t* i)
proof (*rule token-time-finite-pair-rule*)
let $?P = (\lambda(x,\ y)\ n.\ \exists j.\ x \leq n$
$\wedge$ (($\exists j'$. *token-run* $x\ n \neq$ *token-run* $y\ n \wedge y \leq n \wedge$ *state-rank* (*token-run* $y\ n$) $n =$ *Some* j') $\vee\ y =$ *Suc* n)
$\wedge$ *token-run* x (*Suc* n) = *token-run* y (*Suc* n)
$\wedge$ *token-run* x (*Suc* n) $\notin F$
$\wedge$ *state-rank* (*token-run* $x\ n$) $n =$ *Some* j
$\wedge\ j < i$)

{
fix x

assume $x \in merge\ i$
then obtain $t\ t'\ n\ j$ **where** *1*: $x = (t,\ t')$
and *3*: $(\exists j'.\ token\text{-}run\ t\ n \neq token\text{-}run\ t'\ n \wedge rank\ t'\ n = Some\ j') \vee t' = Suc\ n$
and *4*: $token\text{-}run\ t\ (Suc\ n) = token\text{-}run\ t'\ (Suc\ n)$
and *5*: $token\text{-}run\ t\ (Suc\ n) \notin F$
and *6*: $rank\ t\ n = Some\ j$
and *7*: $j < i$
unfolding *merge-def* **by** *blast*
moreover
hence *8*: $t \leq n$ **and** *9*: $t' \leq Suc\ n$
using *rank-Some-time le-Suc-eq* **by** *blast+*
moreover
hence *10*: $state\text{-}rank\ (token\text{-}run\ t\ n)\ n = Some\ j$
using ‹$rank\ t\ n = Some\ j$› *rank-eq-state-rank* **by** *metis*
ultimately
show $\exists y.\ ?P\ x\ y$
proof $(cases\ t' = Suc\ n)$
case *False*
hence $t' \leq n$
using ‹$t' \leq Suc\ n$› **by** *simp*
with *1 3 4 5 7 8 10* **show** *?thesis*
unfolding *rank-eq-state-rank*[*OF* ‹$t' \leq n$›] **by** *blast*
qed *blast*
}

{
fix y
assume $y \in merge\text{-}t\ i$
then obtain $q\ q'\ j$ **where** *1*: $state\text{-}rank\ q\ y = Some\ j$
and *2*: $j < i$
and *3*: $q' = \delta\ q\ (w\ y)$
and *4*: $q' \notin F$
and *5*: $(\exists q''.\ q' = \delta\ q''\ (w\ y) \wedge state\text{-}rank\ q''\ y \neq None \wedge q'' \neq q) \vee q' = q_0$
unfolding *merge-t-def* **by** *blast*

then obtain t **where** *6*: $q = token\text{-}run\ t\ y$ **and** *7*: $t \leq y$
using *push-down-state-rank-token-run* **by** *metis*
hence *8*: $q' = token\text{-}run\ t\ (Suc\ y)$
unfolding ‹$q' = \delta\ q\ (w\ y)$› **using** *token-run-step* **by** *simp*

{
assume $q' = q_0$

hence *token-run t* (*Suc y*) = *token-run* (*Suc y*) (*Suc y*)
  unfolding *8* **by** *simp*
moreover
then obtain *x* **where** $x = (t, \mathit{Suc}\ y)$
  by *simp*
ultimately
have *?P x y*
  using *1 2 3 4 5 7* **unfolding** *6 8* **by** *force*
hence $\exists x.$ *?P x y*
  by *blast*
}
moreover
{
assume $q' \neq q_0$
then obtain q'' j' **where** *9*: $q' = \delta\ q''\ (w\ y)$
  and *state-rank* q'' *y* = *Some* j'
  and $q'' \neq q$
  using *5* **by** *blast*
moreover
then obtain t' **where** *12*: q'' = *token-run* t' *y* **and** $t' \leq y$
  by (*blast dest*: *push-down-state-rank-token-run*)
moreover
hence *token-run t* (*Suc y*) = *token-run* t' (*Suc y*)
  using *8 9 token-run-step* **by** *presburger*
moreover
have *token-run t y* ≠ *token-run* t' *y*
  using ‹$q'' \neq q$› **unfolding** *6 12* **..**
moreover
then obtain *x* **where** $x = (t, t')$
  by *simp*
ultimately
have *?P x y*
  using *1 2 4 7* **unfolding** *6 8* **by** *fast*
hence $\exists x.$ *?P x y*
  by *blast*
}
ultimately
show $\exists x.$ *?P x y*
  by *blast*
}

{
fix *x y*
assume *?P x y*

then obtain $t\ t'\ j$ **where** 1: $x = (t,\ t')$
  and 3: $t \leq y$
  and 4: $(\exists j'.\ token\text{-}run\ t\ y \neq token\text{-}run\ t'\ y \wedge t' \leq y \wedge state\text{-}rank$ $(token\text{-}run\ t'\ y)\ y = Some\ j') \vee t' = Suc\ y$
  and 5: $token\text{-}run\ t\ (Suc\ y) = token\text{-}run\ t'\ (Suc\ y)$
  and 6: $token\text{-}run\ t\ (Suc\ y) \notin F$
  and 7: $state\text{-}rank\ (token\text{-}run\ t\ y)\ y = Some\ j$
  and 8: $j < i$
  by *blast*

thus $x \in merge\ i$
proof $(cases\ t' = Suc\ y)$
  case *False*
    hence $t' \leq y$
      using 4 **by** *blast*
    thus *?thesis*
      using *1 3 4 5 6 7 8* **unfolding** *merge-def*
    unfolding *rank-eq-state-rank*[*OF* ‹$t' \leq y$›, *symmetric*] *rank-eq-state-rank*[*OF* ‹$t \leq y$›, *symmetric*]
      by *blast*
qed (*unfold rank-eq-state-rank*[*OF* ‹$t \leq y$›, *symmetric*] *merge-def*, *blast*)

show $y \in merge\text{-}t\ i$ **and** $fst\ x \leq y + 0 \wedge snd\ x \leq y + 1$
  using *merge-t-inclusion* ‹*?P x y*› **by** *force+*
}

— Uniqueness
{
fix $x\ y\ z$
assume *?P x y* **and** *?P x z*
then obtain $t\ t'$ **where** $x = (t,\ t')$
  by *blast*
from ‹*?P x y*›[*unfolded* ‹$x = (t,\ t')$›] **have** $y1$: $t \leq y$
  and $y2$: $(token\text{-}run\ t\ y \neq token\text{-}run\ t'\ y \wedge t' \leq y) \vee t' = Suc\ y$
  and $y3$: $token\text{-}run\ t\ (Suc\ y) = token\text{-}run\ t'\ (Suc\ y)$ **by** *blast+*
moreover
from ‹*?P x z*›[*unfolded* ‹$x = (t,\ t')$›] **have** $z1$: $t \leq z$
  and $z2$: $(token\text{-}run\ t\ z \neq token\text{-}run\ t'\ z \wedge t' \leq z) \vee t' = Suc\ z$
  and $z3$: $token\text{-}run\ t\ (Suc\ z) = token\text{-}run\ t'\ (Suc\ z)$ **by** *blast+*
moreover
have $y4$: $t' \leq Suc\ y$ **and** $z4$: $t' \leq Suc\ z$
  using *y2 z2* **by** *linarith+*
ultimately
show $y = z$

proof (*cases y z rule*: *linorder-cases*)
  case *less*
    then obtain *d* **where** *Suc y* + *d* = *z*
      by (*metis add-Suc-right add-Suc-shift less-imp-Suc-add*)
    thus *?thesis*
      using *y1 y2 z2 token-run-merge*[*OF* - *y4 y3*] **by** *auto*
next
  case *greater*
    then obtain *d* **where** *Suc z* + *d* = *y*
      by (*metis add-Suc-right add-Suc-shift less-imp-Suc-add*)
    thus *?thesis*
      using *z1 y2 z2 token-run-merge*[*OF* - *z4 z3*] **by** *auto*
qed
}
qed

6.4.3 Relation to Mojmir Acceptance

lemma *token-iff-time-accept*:
shows (*finite fail* ∧ *finite* (*merge i*) ∧ *infinite* (*succeed i*) ∧ (∀ *j* < *i*. *finite* (*succeed j*)))
= (*finite fail-t* ∧ *finite* (*merge-t i*) ∧ *infinite* (*succeed-t i*) ∧ (∀ *j* < *i*. *finite* (*succeed-t j*)))
unfolding *finite-fail-t finite-merge-t finite-succeed-t* **by** *simp*

6.5 Succeeding Tokens (Alternative Definition)

definition *stable-rank-at* :: *nat* ⇒ *nat* ⇒ *bool*
where
stable-rank-at x n ≡ ∃ *i*. ∀ *m* ≥ *n*. *rank x m* = *Some i*

lemma *stable-rank-at-ge*:
n ≤ *m* ⟹ *stable-rank-at x n* ⟹ *stable-rank-at x m*
unfolding *stable-rank-at-def* **by** *fastforce*

lemma *stable-rank-equiv*:
(∃ *i*. *stable-rank x i*) = (∃ *n*. *stable-rank-at x n*)
unfolding *stable-rank-def MOST-nat-le stable-rank-at-def* **by** *blast*

lemma *smallest-accepting-rank-properties*:
assumes *smallest-accepting-rank* = *Some i*
shows *accept finite fail finite* (*merge i*) *infinite* (*succeed i*) ∀ *j* < *i*. *finite* (*succeed j*) *i* < *max-rank*
proof –

from *assms* **show** *accept*
unfolding *smallest-accepting-rank-def* **using** *option.distinct(1)* **by** *metis*
then obtain i' **where** *finite fail* **and** *finite* (*merge* i') **and** *infinite* (*succeed* i')
and $\forall j < i'$. *finite* (*succeed* j) **and** $i' <$ *max-rank*
unfolding *mojmir-accept-iff-token-set-accept2* **by** *blast*
moreover
hence $\bigwedge y$. *finite fail* $\wedge$ *finite* (*merge* y) $\wedge$ *infinite* (*succeed* y) $\longrightarrow i' \leq y$
using *not-le* **by** *blast*
ultimately
have (*LEAST* i. *finite fail* $\wedge$ *finite* (*merge* i) $\wedge$ *infinite* (*succeed* i)) $= i'$
using *le-antisym* **unfolding** *Least-def* **by** (*blast dest*: *the-equality*[*of* - i'])
hence $i' = i$
using ‹*smallest-accepting-rank* = *Some* i› ‹*accept*› **unfolding** *smallest-accepting-rank-def* **by** *simp*
thus *finite fail* **and** *finite* (*merge* i) **and** *infinite* (*succeed* i)
and $\forall j < i$. *finite* (*succeed* j) **and** $i <$ *max-rank*
using ‹*finite fail*› ‹*finite* (*merge* i')› ‹*infinite* (*succeed* i')›
using ‹$\forall j < i'$. *finite* (*succeed* j)› ‹$i' <$ *max-rank*› **by** *simp+*
qed

lemma *token-smallest-accepting-rank*:
assumes *smallest-accepting-rank* = *Some* i
shows $\forall_\infty n. \forall x.$ *token-succeeds* $x \longleftrightarrow (x > n \vee (\exists j \geq i.$ *rank* $x\ n =$ *Some* $j) \vee$ *token-run* $x\ n \in F)$
proof –
from *assms* **have** *accept finite fail infinite* (*succeed* i) $\forall j < i$. *finite* (*succeed* j)
using *smallest-accepting-rank-properties* **by** *blast+*

then obtain n_1 **where** n_1*-def*: $\forall x \geq n_1$. *token-succeeds* x
unfolding *accept-def MOST-nat-le* **by** *blast*
define n_2 **where** $n_2 =$ *Suc* (*Max* (*fail-t* $\cup \bigcup\{$*succeed-t* $j \mid j.\ j < i\}$)) (**is** - = *Suc* (*Max ?S*))
define n_3 **where** $n_3 =$ *Max* ($\{$*LEAST* m. *stable-rank-at* $x\ m \mid x.\ x < n_1 \wedge$ *token-squats* $x\}$) (**is** - = *Max ?S'*)
define n **where** $n =$ *Max* $\{n_1, n_2, n_3\}$

have *finite ?S* **and** *finite ?S'*
using ‹*finite fail*› ‹$\forall j < i$. *finite* (*succeed* j)›
unfolding *finite-fail-t finite-succeed-t* **by** *fastforce+*

{

fix x
assume $x < n_1$ *token-squats* x
hence (*LEAST* m. *stable-rank-at* x m) $\in$ *?S'* (**is** *?m* $\in$ -)
by *blast*
hence *?m* $\leq n_3$
using *Max.coboundedI*[*OF* ‹*finite ?S'*›] n_3*-def* **by** *simp*
moreover
obtain k **where** *stable-rank* x k
using ‹$x < n_1$› ‹*token-squats* x› *stable-rank-equiv-token-squats* **by** *blast*
hence *stable-rank-at* x *?m*
by (*metis stable-rank-equiv LeastI*)
ultimately
have *stable-rank-at* x n_3
by (*rule stable-rank-at-ge*)
hence $\exists\, i.\ \forall\, m' \geq n.$ *rank* x m' = *Some* i
unfolding *n-def stable-rank-at-def* **by** *fastforce*
}
note *Stable* = *this*

have $\bigwedge m\ j.\ j < i \Longrightarrow m \in$ *succeed-t* $j \Longrightarrow m < n$
using *Max.coboundedI*[*OF* ‹*finite ?S*›] **unfolding** *n-def* n_2*-def* **by** *fastforce*
hence *Succeed*: $\bigwedge m\ j\ x.\ n \leq m \Longrightarrow$ *token-run* x $m \notin F - \{q_0\} \Longrightarrow$ *token-run* x (*Suc* m) $\in F \Longrightarrow$ *rank* x m = *Some* $j \Longrightarrow i \leq j$
by (*metis not-le succeed-t-inclusion*)

have $\bigwedge m.\ m \in$ *fail-t* $\Longrightarrow m < n$
using *Max.coboundedI*[*OF* ‹*finite ?S*›] **unfolding** *n-def* n_2*-def* **by** *fastforce*
hence *Fail*: $\bigwedge m\ x.\ n \leq m \Longrightarrow x \leq m \Longrightarrow$ *sink* (*token-run* x m) $\vee$ ¬*sink* (*token-run* x (*Suc* m)) $\vee$ ¬*token-run* x (*Suc* m) $\notin F$
using *fail-t-inclusion* **by** *fastforce*

{
fix m x
assume $m \geq n$ $m \geq x$
moreover
{
assume *token-succeeds* x *token-run* x $m \notin F$
then obtain m' **where** $x \leq m'$ **and** *token-run* x $m' \notin F - \{q_0\}$ **and** *token-run* x (*Suc* m') $\in F$
using *token-run-enter-final-states* **unfolding** *token-succeeds-def* **by** *meson*
moreover

hence $\neg$*sink* (*token-run x m'*)
by (*metis Diff-empty Diff-insert0* ‹*token-run x m* $\notin$ *F*› *initial-in-F-token-run token-is-not-in-sink*)
ultimately
obtain j' **where** *rank x m'* = *Some j'*
by *simp*
moreover
have $m \leq m'$
by (*metis* ‹*token-run x m* $\notin$ *F*› *token-stays-in-final-states*[*OF* ‹*token-run x* (*Suc m'*) $\in$ *F*›] *add-Suc-right add-Suc-shift less-imp-Suc-add not-le*)
moreover
hence $m' \geq n$
using ‹$x \leq m$› ‹$m \geq n$› **by** *simp*
hence $j' \geq i$
using *Succeed*[*OF* - ‹*token-run x m'* $\notin F - \{q_0\}$› ‹*token-run x* (*Suc m'*) $\in$ *F*› ‹*rank x m'* = *Some j'*›] **by** *blast*
moreover
obtain *k* **where** *rank x x* = *Some k*
using *rank-initial*[*of x*] **by** *blast*
ultimately
obtain *j* **where** *rank x m* = *Some j*
by (*metis rank-continuous*[*OF* ‹*rank x x* = *Some k*›, *of m'* − *x*] ‹$x \leq m'$› ‹$x \leq m$› *diff-le-mono le-add-diff-inverse*)
hence $\exists j \geq i$. *rank x m* = *Some j*
using *rank-monotonic* ‹*rank x m'* = *Some j'*› ‹$j' \geq i$› ‹$m \leq m'$›[*THEN le-Suc-ex*]
by (*blast dest*: *le-Suc-ex trans-le-add1*)
}
moreover
{
assume $\neg$*token-succeeds x*
hence $\bigwedge m$. *token-run x m* $\notin$ *F*
unfolding *token-succeeds-def* **by** *blast*
moreover
have $\neg(\exists j \geq i$. *rank x m* = *Some j*)
proof (*cases token-squats x*)
case *True*
— The token already stabilised
have $x < n_1$
using ‹$\neg$*token-succeeds x*› n_1*-def* **by** (*metis not-le*)
then obtain *k* **where** $\forall m' \geq n$. *rank x m'* = *Some k*
using *Stable*[*OF* - *True*] **by** *blast*
moreover
hence *stable-rank x k*

unfolding *stable-rank-def MOST-nat-le* **by** *blast*
moreover
have $q_0 \notin F$
by (*metis* ‹$\bigwedge m$. *token-run x m* $\notin F$› *initial-in-F-token-run*)
ultimately
— Hence the rank is smaller than i
have $k < i$ **and** *rank x m* = *Some k*
using *stable-rank-bounded* ‹*infinite* (*succeed i*)› ‹$n \leq m$› **by** *blast+*
thus *?thesis*
by *simp*
next
case *False*
— Then token is already in a sink
have *sink* (*token-run x m*)
proof (*rule ccontr*)
assume ¬*sink* (*token-run x m*)
moreover
obtain m' **where** $m < m'$ **and** *sink* (*token-run x* m')
by (*metis False token-squats-def le-add2 not-le not-less-eq-eq token-stays-in-sink*)
ultimately
obtain m'' **where** $m \leq m''$ **and** ¬*sink* (*token-run x* m'') **and** *sink* (*token-run x* (*Suc* m''))
by (*metis le-add1 less-imp-Suc-add token-run-P*)
thus *False*
by (*metis Fail* ‹$\bigwedge m$. *token-run x m* $\notin F$› ‹$n \leq m$› ‹$x \leq m$› *le-trans*)
qed
— Hence there is no rank
thus *?thesis*
by *simp*
qed
ultimately
have $\neg(\exists j \geq i.$ *rank x m* = *Some j*$)$ $\wedge$ *token-run x m* $\notin F$
by *blast*
}
ultimately
have $(\exists j \geq i.$ *rank x m* = *Some j*$)$ $\vee$ *token-run x m* $\in F \longleftrightarrow$ *token-succeeds x*
by (*cases token-succeeds x*) (*blast*, *simp*)
}
moreover
— By definition of n all tokens $\bigwedge x.\ n \leq x$ succeed
have $\bigwedge m\ x.\ m \geq n \Longrightarrow \neg x \leq m \Longrightarrow$ *token-succeeds x*

using *n-def* n_1*-def* **by** *force*
ultimately
show *?thesis*
unfolding *MOST-nat-le not-le*[*symmetric*] **by** *blast*
qed

lemma *succeeding-states*:
assumes *smallest-accepting-rank* = *Some i*
shows $\forall_\infty n. \forall q.$ ((∃ *x* ∈ *configuration q n. token-succeeds x*) ⟶ *q* ∈ $\mathcal{S}$ *n*) ∧ (*q* ∈ $\mathcal{S}$ *n* ⟶ (∀ *x* ∈ *configuration q n. token-succeeds x*))
proof −
obtain *n* **where** *n-def*: ⋀*m x. m* ≥ *n* ⟹ *token-succeeds x* = (*x* > *m* ∨ (∃ *j* ≥ *i. rank x m* = *Some j*) ∨ *token-run x m* ∈ *F*)
using *token-smallest-accepting-rank*[*OF assms*] **unfolding** *MOST-nat-le* **by** *auto*
{
fix *m q*
assume *m* ≥ *n q* ∉ *F* ∃ *x* ∈ *configuration q m. token-succeeds x*
moreover
then obtain *x* **where** *token-run x m* = *q* **and** *x* ≤ *m* **and** *token-succeeds x*
by *auto*
ultimately
have ∃ *j* ≥ *i. rank x m* = *Some j*
using *n-def* **by** *simp*
hence ∃ *j* ≥ *i. state-rank q m* = *Some j*
using *rank-eq-state-rank* ‹*x* ≤ *m*› ‹*token-run x m* = *q*› **by** *metis*
}
moreover
{
fix *m q x*
assume *m* ≥ *n x* ∈ *configuration q m*
hence *x* ≤ *m* **and** *token-run x m* = *q*
by *simp+*
moreover
assume *q* ∈ $\mathcal{S}$ *m*
hence (∃ *j* ≥ *i. state-rank q m* = *Some j*) ∨ *q* ∈ *F*
using *assms* **by** *fastforce*
ultimately
have (∃ *j* ≥ *i. rank x m* = *Some j*) ∨ *q* ∈ *F*
using *rank-eq-state-rank* **by** *presburger*
hence *token-succeeds x*
unfolding *n-def*[*OF* ‹*m* ≥ *n*›] ‹*token-run x m* = *q*› **by** *presburger*
}

ultimately
show *?thesis*
unfolding *MOST-nat-le* $\mathcal{S}$*.simps assms option.sel* **by** *blast*
qed

end

end

7 (Generalized) Rabin Automata

theory *Rabin*
imports *Main DTS*
begin

type-synonym $('a, 'b)$ *rabin-pair* = $(('a, 'b)$ *transition set* $\times$ $('a, 'b)$ *transition set*$)$
type-synonym $('a, 'b)$ *generalized-rabin-pair* = $(('a, 'b)$ *transition set* $\times$ $('a, 'b)$ *transition set set*$)$

type-synonym $('a, 'b)$ *rabin-condition* = $('a, 'b)$ *rabin-pair set*
type-synonym $('a, 'b)$ *generalized-rabin-condition* = $('a, 'b)$ *generalized-rabin-pair set*

type-synonym $('a, 'b)$ *rabin-automaton* = $('a, 'b)$ *DTS* $\times$ $'a$ $\times$ $('a, 'b)$ *rabin-condition*
type-synonym $('a, 'b)$ *generalized-rabin-automaton* = $('a, 'b)$ *DTS* $\times$ $'a$ $\times$ $('a, 'b)$ *generalized-rabin-condition*

definition *accepting-pair*$_R$ :: $('a, 'b)$ *DTS* $\Rightarrow$ $'a$ $\Rightarrow$ $('a, 'b)$ *rabin-pair* $\Rightarrow$ $'b$ *word* $\Rightarrow$ *bool*
where
accepting-pair$_R$ δ q_0 P w $\equiv$ *limit* (*run*$_t$ δ q_0 w) $\cap$ *fst* P = {} $\wedge$ *limit* (*run*$_t$ δ q_0 w) $\cap$ *snd* P $\neq$ {}

definition *accept*$_R$ :: $('a, 'b)$ *rabin-automaton* $\Rightarrow$ $'b$ *word* $\Rightarrow$ *bool*
where
accept$_R$ R w $\equiv$ ($\exists$ P $\in$ (*snd* (*snd* R)). *accepting-pair*$_R$ (*fst* R) (*fst* (*snd* R)) P w)

definition *accepting-pair*$_{GR}$ :: $('a, 'b)$ *DTS* $\Rightarrow$ $'a$ $\Rightarrow$ $('a, 'b)$ *generalized-rabin-pair* $\Rightarrow$ $'b$ *word* $\Rightarrow$ *bool*
where

$accepting\text{-}pair_{GR}\ \delta\ q_0\ P\ w \equiv limit\ (run_t\ \delta\ q_0\ w) \cap fst\ P = \{\} \wedge (\forall I \in snd\ P.\ limit\ (run_t\ \delta\ q_0\ w) \cap I \neq \{\})$

definition $accept_{GR}$ :: $('a,\ 'b)$ *generalized-rabin-automaton* $\Rightarrow$ $'b$ *word* $\Rightarrow$ *bool*
where
$accept_{GR}\ R\ w \equiv (\exists (Fin,\ Inf) \in (snd\ (snd\ R)).\ accepting\text{-}pair_{GR}\ (fst\ R)\ (fst\ (snd\ R))\ (Fin,\ Inf)\ w)$

declare $accepting\text{-}pair_R\text{-}def[simp]$
declare $accepting\text{-}pair_{GR}\text{-}def[simp]$

lemma $accepting\text{-}pair_R\text{-}simp[simp]$:
$accepting\text{-}pair_R\ \delta\ q_0\ (F,I)\ w \equiv limit\ (run_t\ \delta\ q_0\ w) \cap F = \{\} \wedge limit\ (run_t\ \delta\ q_0\ w) \cap I \neq \{\}$
by *simp*

lemma $accepting\text{-}pair_{GR}\text{-}simp[simp]$:
$accepting\text{-}pair_{GR}\ \delta\ q_0\ (F,\ \mathcal{I})\ w \equiv limit\ (run_t\ \delta\ q_0\ w) \cap F = \{\} \wedge (\forall I \in \mathcal{I}.\ limit\ (run_t\ \delta\ q_0\ w) \cap I \neq \{\})$
by *simp*

lemma $accept_R\text{-}simp[simp]$:
$accept_R\ (\delta,\ q_0,\ \alpha)\ w = (\exists (Fin,\ Inf) \in \alpha.\ limit\ (run_t\ \delta\ q_0\ w) \cap Fin = \{\} \wedge limit\ (run_t\ \delta\ q_0\ w) \cap Inf \neq \{\})$
by (*unfold* $accept_R$*-def* $accepting\text{-}pair_R$*-def case-prod-unfold fst-conv snd-conv*; *rule*)

lemma $accept_{GR}\text{-}simp[simp]$:
$accept_{GR}\ (\delta,\ q_0,\ \alpha)\ w \longleftrightarrow (\exists (Fin,\ Inf) \in \alpha.\ limit\ (run_t\ \delta\ q_0\ w) \cap Fin = \{\} \wedge (\forall I \in Inf.\ limit\ (run_t\ \delta\ q_0\ w) \cap I \neq \{\}))$
by (*unfold* $accept_{GR}$*-def* $accepting\text{-}pair_{GR}$*-def case-prod-unfold fst-conv snd-conv*; *rule*)

lemma $accept_{GR}\text{-}simp2$:
$accept_{GR}\ (\delta,\ q_0,\ \alpha)\ w \longleftrightarrow (\exists P \in \alpha.\ accepting\text{-}pair_{GR}\ \delta\ q_0\ P\ w)$
by (*unfold* $accept_{GR}$*-def* $accepting\text{-}pair_{GR}$*-def case-prod-unfold fst-conv snd-conv*; *rule*)

type-synonym $('a,\ 'b)$ *LTS* = $('a,\ 'b)$ *transition set*

definition *LTS-is-inf-run* :: $('q,'a)$ *LTS* $\Rightarrow$ $'a$ *word* $\Rightarrow$ $'q$ *word* $\Rightarrow$ *bool*
where
$LTS\text{-}is\text{-}inf\text{-}run\ \Delta\ w\ r \longleftrightarrow (\forall i.\ (r\ i,\ w\ i,\ r\ (Suc\ i)) \in \Delta)$

fun $accept_R$-*LTS* :: (('a, 'b) *LTS* × 'a × ('a, 'b) *rabin-condition*) ⇒ 'b *word* ⇒ *bool*
where
$accept_R$-*LTS* (δ, q_0, α) w ⟷ (∃ (*Fin*, *Inf*) ∈ α. ∃ r.
LTS-is-inf-run δ w r ∧ r *0* = q_0
∧ *limit* (λi. (r i, w i, r (*Suc* i))) ∩ *Fin* = {}
∧ *limit* (λi. (r i, w i, r (*Suc* i))) ∩ *Inf* ≠ {})

definition *accepting-pair*$_{GR}$-*LTS* :: ('a, 'b) *LTS* ⇒ 'a ⇒ ('a, 'b) *generalized-rabin-pair* ⇒ 'b *word* ⇒ *bool*
where
accepting-pair$_{GR}$-*LTS* δ q_0 P w ≡ ∃ r. *LTS-is-inf-run* δ w r ∧ r *0* = q_0
∧ *limit* (λi. (r i, w i, r (*Suc* i))) ∩ *fst* P = {}
∧ (∀ I ∈ *snd* P. *limit* (λi. (r i, w i, r (*Suc* i))) ∩ I ≠ {})

fun $accept_{GR}$-*LTS* :: (('a, 'b) *LTS* × 'a × ('a, 'b) *generalized-rabin-condition*) ⇒ 'b *word* ⇒ *bool*
where
$accept_{GR}$-*LTS* (δ, q_0, α) w = (∃ (*Fin*, *Inf*) ∈ α. *accepting-pair*$_{GR}$-*LTS* δ q_0 (*Fin*, *Inf*) w)

lemma $accept_{GR}$-*LTS-E*:
assumes $accept_{GR}$-*LTS* R w
obtains F I **where** (F, I) ∈ *snd* (*snd* R)
and *accepting-pair*$_{GR}$-*LTS* (*fst* R) (*fst* (*snd* R)) (F, I) w
proof −
obtain δ q_0 α **where** R = (δ, q_0, α)
using *prod-cases3* **by** *blast*
show (⋀F I. (F, I) ∈ *snd* (*snd* R) ⟹ *accepting-pair*$_{GR}$-*LTS* (*fst* R) (*fst* (*snd* R)) (F, I) w ⟹ *thesis*) ⟹ *thesis*
using *assms* **unfolding** ‹R = (δ, q_0, α)› **by** *auto*
qed

lemma $accept_{GR}$-*LTS-I*:
accepting-pair$_{GR}$-*LTS* δ q_0 (F, $\mathcal{I}$) w ⟹ (F, $\mathcal{I}$) ∈ α ⟹ $accept_{GR}$-*LTS* (δ, q_0, α) w
by *auto*

lemma $accept_{GR}$-*I*:
accepting-pair$_{GR}$ δ q_0 (F, $\mathcal{I}$) w ⟹ (F, $\mathcal{I}$) ∈ α ⟹ $accept_{GR}$ (δ, q_0, α) w
by *auto*

lemma *transfer-accept*:

accepting-pair$_R$ δ q_0 (F, I) $w \longleftrightarrow$ *accepting-pair*$_{GR}$ δ q_0 $(F, \{I\})$ w
accept$_R$ (δ, q_0, α) $w \longleftrightarrow$ *accept*$_{GR}$ $(\delta, q_0, (\lambda(F, I). (F, \{I\}))$ ' $\alpha)$ w
by (*simp add*: *case-prod-unfold*)+

7.1 Restriction Lemmas

lemma *accepting-pair*$_{GR}$*-restrict*:
assumes *range* $w \subseteq \Sigma$
shows *accepting-pair*$_{GR}$ δ q_0 $(F, \mathcal{I})$ w = *accepting-pair*$_{GR}$ δ q_0 $(F \cap$ *reach*$_t$ Σ δ q_0, $(\lambda I.$ $I \cap$ *reach*$_t$ Σ δ $q_0)$ ' $\mathcal{I})$ w
proof −
have *limit* (*run*$_t$ δ q_0 w) $\cap F = \{\} \longleftrightarrow$ *limit* (*run*$_t$ δ q_0 w) $\cap$ $(F \cap$ *reach*$_t$ Σ δ $q_0) = \{\}$
using *assms*[*THEN limit-subseteq-reach*(*2*), *of* δ q_0] **by** *auto*
moreover
have $(\forall I \in \mathcal{I}.$ *limit* (*run*$_t$ δ q_0 w) $\cap I \neq \{\}) = ((\forall I \in \{y. \exists x \in \mathcal{I}. y = x \cap$ *reach*$_t$ Σ δ $q_0\}.$ *limit* (*run*$_t$ δ q_0 w) $\cap I \neq \{\}))$
using *assms*[*THEN limit-subseteq-reach*(*2*), *of* δ q_0] **by** *auto*
ultimately
show *?thesis*
unfolding *accepting-pair*$_{GR}$*-simp image-def* **by** *meson*
qed

lemma *accept*$_{GR}$*-restrict*:
assumes *range* $w \subseteq \Sigma$
shows *accept*$_{GR}$ $(\delta, q_0, \{(f\ x, g\ x) \mid x. P\ x\})$ w = *accept*$_{GR}$ $(\delta, q_0, \{(f\ x \cap$ *reach*$_t$ Σ δ q_0, $(\lambda I.$ $I \cap$ *reach*$_t$ Σ δ $q_0)$ ' $g\ x) \mid x. P\ x\})$ w
apply (*simp only*: *accept*$_{GR}$*-simp*)
apply (*subst accepting-pair*$_{GR}$*-restrict*[*OF assms*, *simplified*])
apply *simp*
apply *standard*
apply (*metis* (*no-types*, *lifting*) *imageE*)
apply *fastforce*
done

lemma *accepting-pair*$_R$*-restrict*:
assumes *range* $w \subseteq \Sigma$
shows *accepting-pair*$_R$ δ q_0 (F, I) w = *accepting-pair*$_R$ δ q_0 $(F \cap$ *reach*$_t$ Σ δ q_0, $I \cap$ *reach*$_t$ Σ δ $q_0)$ w
by (*simp only*: *transfer-accept*; *subst accepting-pair*$_{GR}$*-restrict*[*OF assms*]; *simp*)

lemma *accept*$_R$*-restrict*:
assumes *range* $w \subseteq \Sigma$

shows $accept_R$ (δ, q_0, $\{(f\ x,\ g\ x) \mid x.\ P\ x\}$) w = $accept_R$ (δ, q_0, $\{(f\ x \cap reach_t\ \Sigma\ \delta\ q_0,\ g\ x \cap reach_t\ \Sigma\ \delta\ q_0) \mid x.\ P\ x\}$) w
by (*simp only*: $accept_R$-*simp*; *subst accepting-pair*$_R$-*restrict*[*OF assms*, *simplified*]; *auto*)

7.2 Abstraction Lemmas

lemma *accepting-pair*$_{GR}$-*abstract*:
assumes *finite* ($reach_t\ \Sigma\ \delta\ q_0$)
and *finite* ($reach_t\ \Sigma\ \delta'\ q_0'$)
assumes *range* $w \subseteq \Sigma$
assumes $run_t\ \delta\ q_0\ w = f\ o\ (run_t\ \delta'\ q_0'\ w)$
assumes $\bigwedge t.\ t \in reach_t\ \Sigma\ \delta'\ q_0' \Longrightarrow f\ t \in F \longleftrightarrow t \in F'$
assumes $\bigwedge t\ i.\ i \in \mathcal{I} \Longrightarrow t \in reach_t\ \Sigma\ \delta'\ q_0' \Longrightarrow f\ t \in I\ i \longleftrightarrow t \in I'\ i$
shows *accepting-pair*$_{GR}$ $\delta\ q_0\ (F,\ \{I\ i \mid i.\ i \in \mathcal{I}\})\ w \longleftrightarrow$ *accepting-pair*$_{GR}$ $\delta'\ q_0'\ (F',\ \{I'\ i \mid i.\ i \in \mathcal{I}\})\ w$
proof −
have *finite* (*range* ($run_t\ \delta\ q_0\ w$)) (**is** - (*range ?r*))
and *finite* (*range* ($run_t\ \delta'\ q_0'\ w$)) (**is** - (*range ?r'*))
using *assms*(*1*,*2*,*3*) *run-subseteq-reach*(*2*) **by** (*metis finite-subset*)+
then obtain *k* **where** *1*: *limit ?r* = *range* (*suffix k ?r*)
and *2*: *limit ?r'* = *range* (*suffix k ?r'*)
using *common-range-limit* **by** *blast*
have *X*: *limit* ($run_t\ \delta\ q_0\ w$) = *f* ‘ *limit* ($run_t\ \delta'\ q_0'\ w$)
unfolding *1 2 suffix-def* **by** (*auto simp add*: *assms*(*4*))
have *3*: (*limit ?r* $\cap$ *F* = {}) $\longleftrightarrow$ (*limit ?r'* $\cap$ *F'* = {})
and *4*: ($\forall\ i \in \mathcal{I}$. *limit ?r* $\cap$ *I i* $\neq$ {}) $\longleftrightarrow$ ($\forall\ i \in \mathcal{I}$. *limit ?r'* $\cap$ *I' i* $\neq$ {})
using *assms*(*5*,*6*) *limit-subseteq-reach*(*2*)[*OF assms*(*3*)] **by** (*unfold X*; *fastforce*)+
thus *?thesis*
unfolding *accepting-pair*$_{GR}$-*simp* **by** *blast*
qed

lemma *accepting-pair*$_R$-*abstract*:
assumes *finite* ($reach_t\ \Sigma\ \delta\ q_0$)
and *finite* ($reach_t\ \Sigma\ \delta'\ q_0'$)
assumes *range* $w \subseteq \Sigma$
assumes $run_t\ \delta\ q_0\ w = f\ o\ (run_t\ \delta'\ q_0'\ w)$
assumes $\bigwedge t.\ t \in reach_t\ \Sigma\ \delta'\ q_0' \Longrightarrow f\ t \in F \longleftrightarrow t \in F'$
assumes $\bigwedge t.\ t \in reach_t\ \Sigma\ \delta'\ q_0' \Longrightarrow f\ t \in I \longleftrightarrow t \in I'$
shows *accepting-pair*$_R$ $\delta\ q_0\ (F,\ I)\ w \longleftrightarrow$ *accepting-pair*$_R$ $\delta'\ q_0'\ (F',\ I')\ w$
using *accepting-pair*$_{GR}$-*abstract*[*OF assms*(*1*−*5*), *of UNIV* λ-. *I* λ-. *I'*]

assms(*6*) **by** *simp*

7.3 LTS Lemmas

lemma *accepting-pair*$_{GR}$*-LTS*:
 assumes *range w* $\subseteq \Sigma$
 shows *accepting-pair*$_{GR}$ δ q_0 $(F, \mathcal{I})$ *w* $\longleftrightarrow$ *accepting-pair*$_{GR}$*-LTS* (*reach*$_t$ Σ δ q_0) q_0 $(F, \mathcal{I})$ *w*
 (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
 {
  assume *?lhs*
  moreover
  have *LTS-is-inf-run* (*reach*$_t$ Σ δ q_0) *w* (*run* δ q_0 *w*)
   unfolding *LTS-is-inf-run-def reach*$_t$*-def* **using** *assms*(*1*) **by** *auto*
  moreover
  have *run* δ q_0 *w 0* = q_0
   by *simp*
  ultimately
  show *?rhs*
   unfolding *accept*$_{GR}$*-simp accept*$_{GR}$*-LTS.simps accepting-pair*$_{GR}$*-simp run*$_t$*.simps limit-def accepting-pair*$_{GR}$*-LTS-def snd-conv fst-conv* **by** *blast*
 }

 {
  assume *?rhs*
  then obtain *r* **where** *LTS-is-inf-run* (*reach*$_t$ Σ δ q_0) *w r*
   and *r 0* = q_0
   and *limit* ($\lambda i.$ (*r i*, *w i*, *r* (*Suc i*))) $\cap F = \{\}$
   and $\bigwedge I.\ I \in \mathcal{I} \Longrightarrow$ *limit* ($\lambda i.$ (*r i*, *w i*, *r* (*Suc i*))) $\cap I \neq \{\}$
   unfolding *accepting-pair*$_{GR}$*-LTS-def* **by** *auto*
  moreover
  {
   fix *i*
   from ‹*r 0* = q_0› ‹*LTS-is-inf-run* (*reach*$_t$ Σ δ q_0) *w r*› **have** *r i* = *run* δ q_0 *w i*
    by (*induction i*; *simp-all add*: *LTS-is-inf-run-def reach*$_t$*-def*) *metis*
  }
  hence *r* = *run* δ q_0 *w*
   by *blast*
  hence ($\lambda i.$ (*r i*, *w i*, *r* (*Suc i*))) = *run*$_t$ δ q_0 *w*
   by *auto*
  ultimately
  show *?lhs*

by *auto*
}
qed

lemma $accept_{GR}$*-LTS*:
assumes *range* $w \subseteq \Sigma$
shows $accept_{GR}$ (δ, q_0, α) $w \longleftrightarrow$ $accept_{GR}$*-LTS* $(reach_t\ \Sigma\ \delta\ q_0, q_0, \alpha)$ w

unfolding $accept_{GR}$*-def fst-conv snd-conv accepting-pair*$_{GR}$*-LTS*[*OF assms*]
by *simp*

lemma $accept_R$*-LTS*:
assumes *range* $w \subseteq \Sigma$
shows $accept_R$ (δ, q_0, α) $w \longleftrightarrow$ $accept_R$*-LTS* $(reach_t\ \Sigma\ \delta\ q_0, q_0, \alpha)$ w
unfolding *transfer-accept* $accept_{GR}$*-LTS.simps* $accept_R$*-LTS.simps* $accept_{GR}$*-LTS*[*OF assms*] *case-prod-unfold accepting-pair*$_{GR}$*-LTS-def* **by** *simp*

7.4 Combination Lemmas

lemma *combine-rabin-pairs*:
$(\bigwedge x.\ x \in I \Longrightarrow accepting\text{-}pair_R\ \delta\ q_0\ (f\ x,\ g\ x)\ w) \Longrightarrow accepting\text{-}pair_{GR}\ \delta\ q_0\ (\bigcup\{f\ x \mid x.\ x \in I\},\ \{g\ x \mid x.\ x \in I\})\ w$
by *auto*

lemma *combine-rabin-pairs-UNIV*:
$accepting\text{-}pair_R\ \delta\ q_0\ (fin,\ UNIV)\ w \Longrightarrow accepting\text{-}pair_{GR}\ \delta\ q_0\ (fin',\ inf')\ w \Longrightarrow accepting\text{-}pair_{GR}\ \delta\ q_0\ (fin \cup fin',\ inf')\ w$
by *auto*

end

8 Auxiliary List Facts

theory *List2*
imports *Main HOL−Library.Omega-Words-Fun List−Index.List-Index*
begin

8.1 remdups_fwd

fun *remdups-fwd-acc*
where
remdups-fwd-acc Acc [] = []
| *remdups-fwd-acc Acc* (*x*#*xs*) = (*if x* ∈ *Acc then* [] *else* [*x*]) @ *remdups-fwd-acc* (*insert x Acc*) *xs*

lemma *remdups-fwd-acc-append*[*simp*]:
remdups-fwd-acc Acc (*xs*@*ys*) = (*remdups-fwd-acc Acc xs*) @ (*remdups-fwd-acc* (*Acc* ∪ *set xs*) *ys*)
by (*induction xs arbitrary*: *Acc*) *simp*+

lemma *remdups-fwd-acc-set*[*simp*]:
set (*remdups-fwd-acc Acc xs*) = *set xs* − *Acc*
by (*induction xs arbitrary*: *Acc*) *force*+

lemma *remdups-fwd-acc-distinct*:
distinct (*remdups-fwd-acc Acc xs*)
by (*induction xs arbitrary*: *Acc rule*: *rev-induct*) *simp*+

lemma *remdups-fwd-acc-empty*:
set xs ⊆ *Acc* ⟷ *remdups-fwd-acc Acc xs* = []
by (*metis remdups-fwd-acc-set set-empty Diff-eq-empty-iff Diff-eq-empty-iff*)

lemma *remdups-fwd-acc-drop*:
set ys ⊆ *Acc* ∪ *set xs* ⟹ *remdups-fwd-acc Acc* (*xs* @ *ys* @ *zs*) = *remdups-fwd-acc Acc* (*xs* @ *zs*)
by (*simp add*: *remdups-fwd-acc-empty sup.absorb1*)

lemma *remdups-fwd-acc-filter*:
remdups-fwd-acc Acc (*filter P xs*) = *filter P* (*remdups-fwd-acc Acc xs*)
by (*induction xs rule*: *rev-induct*) *simp*+

fun *remdups-fwd*
where
remdups-fwd xs = *remdups-fwd-acc* {} *xs*

lemma *remdups-fwd-eq*:
remdups-fwd xs = (*rev o remdups o rev*) *xs*
by (*induction xs rule*: *rev-induct*) *simp*+

lemma *remdups-fwd-set*[*simp*]:
set (*remdups-fwd xs*) = *set xs*
by *simp*

lemma *remdups-fwd-distinct*:
distinct (*remdups-fwd xs*)
using *remdups-fwd-acc-distinct* **by** *simp*

lemma *remdups-fwd-filter*:

remdups-fwd (*filter P xs*) = *filter P* (*remdups-fwd xs*)
using *remdups-fwd-acc-filter* **by** *simp*

8.2 Split Lemmas

lemma *map-splitE*:
assumes *map f xs* = *ys* @ *zs*
obtains *us vs* **where** *xs* = *us* @ *vs* **and** *map f us* = *ys* **and** *map f vs* = *zs*
by (*insert assms*; *induction ys arbitrary*: *xs*)
(*simp-all add*: *map-eq-Cons-conv*, *metis append-Cons*)

lemma *filter-split′*:
filter P xs = *ys* @ *zs* ⟹ ∃ *us vs*. *xs* = *us* @ *vs* ∧ *filter P us* = *ys* ∧ *filter P vs* = *zs*
proof (*induction ys arbitrary*: *zs xs rule*: *rev-induct*)
case (*snoc y ys*)
obtain *us vs* **where** *xs* = *us* @ *vs* **and** *filter P us* = *ys* **and** *filter P vs* = *y* # *zs*
using *snoc*(*1*)[*OF snoc*(*2*)[*unfolded append-assoc*]] **by** *auto*
moreover
then obtain *vs′ vs″* **where** *vs* = *vs′* @ *y* # *vs″* **and** *y* ∉ *set vs′* **and** (∀ *u*∈*set vs′*. ¬ *P u*) **and** *filter P vs″* = *zs* **and** *P y*
unfolding *filter-eq-Cons-iff* **by** *blast*
ultimately
have *xs* = (*us* @ *vs′* @ [*y*]) @ *vs″* **and** *filter P* (*us* @ *vs′* @ [*y*]) = *ys* @ [*y*] **and** *filter P* (*vs″*) = *zs*
unfolding *filter-append* **by** *auto*
thus *?case*
by *blast*
qed *fastforce*

lemma *filter-splitE*:
assumes *filter P xs* = *ys* @ *zs*
obtains *us vs* **where** *xs* = *us* @ *vs* **and** *filter P us* = *ys* **and** *filter P vs* = *zs*
using *filter-split′*[*OF assms*] **by** *blast*

lemma *filter-map-splitE*:
assumes *filter P* (*map f xs*) = *ys* @ *zs*
obtains *us vs* **where** *xs* = *us* @ *vs* **and** *filter P* (*map f us*) = *ys* **and** *filter P* (*map f vs*) = *zs*
using *assms* **by** (*fastforce elim*: *filter-splitE map-splitE*)

lemma *filter-map-split-iff*:
filter P (*map f xs*) = *ys* @ *zs* ⟷ (∃ *us vs*. *xs* = *us* @ *vs* ∧ *filter P* (*map f us*) = *ys* ∧ *filter P* (*map f vs*) = *zs*)
by (*fastforce elim*: *filter-map-splitE*)

lemma *list-empty-prefix*:
xs @ *y* # *zs* = *y* # *us* ⟹ *y* ∉ *set xs* ⟹ *xs* = []
by (*metis hd-append2 list.sel*(*1*) *list.set-sel*(*1*))

lemma *remdups-fwd-split*:
remdups-fwd-acc Acc xs = *ys* @ *zs* ⟹ ∃ *us vs*. *xs* = *us* @ *vs* ∧ *remdups-fwd-acc Acc us* = *ys* ∧ *remdups-fwd-acc* (*Acc* ∪ *set ys*) *vs* = *zs*
proof (*induction ys arbitrary*: *zs rule*: *rev-induct*)
  case (*snoc y ys*)
    then obtain *us vs* **where** *xs* = *us* @ *vs*
      and *remdups-fwd-acc Acc us* = *ys*
      and *remdups-fwd-acc* (*Acc* ∪ *set ys*) *vs* = *y* # *zs*
      by *fastforce*
    moreover
    hence *y* ∈ *set vs* **and** *y* ∉ *Acc* ∪ *set ys*
      using *remdups-fwd-acc-set*[*of Acc* ∪ *set ys vs*] **by** *auto*
    moreover
    then obtain *vs′ vs″* **where** *vs* = *vs′* @ *y* # *vs″* **and** *y* ∉ *set vs′*
      using *split-list-first* **by** *metis*
    moreover
    hence *remdups-fwd-acc* (*Acc* ∪ *set ys*) *vs′* = []
      using ‹*remdups-fwd-acc* (*Acc* ∪ *set ys*) *vs* = *y* # *zs*› ‹*y* ∉ *Acc* ∪ *set ys*›
      by (*force intro*: *list-empty-prefix*)
    ultimately
    have *xs* = (*us* @ *vs′* @ [*y*]) @ *vs″*
      and *remdups-fwd-acc Acc* (*us* @ *vs′* @ [*y*]) = *ys* @ [*y*]
      and *remdups-fwd-acc* (*Acc* ∪ *set* (*ys* @ [*y*])) *vs″* = *zs*
      by (*auto simp add*: *remdups-fwd-acc-empty sup.absorb1*)
    thus *?case*
      by *blast*
qed *force*

lemma *remdups-fwd-split-exact*:
  assumes *remdups-fwd-acc Acc xs* = *ys* @ *x* # *zs*
  shows ∃ *us vs*. *xs* = *us* @ *x* # *vs* ∧ *x* ∉ *Acc* ∧ *x* ∉ *set ys* ∧ *remdups-fwd-acc Acc us* = *ys* ∧ *remdups-fwd-acc* (*Acc* ∪ *set ys* ∪ {*x*}) *vs* = *zs*
proof −
  obtain *us vs* **where** *xs* = *us* @ *vs* **and** *remdups-fwd-acc Acc us* = *ys* **and**

remdups-fwd-acc (*Acc* ∪ *set ys*) *vs* = *x* # *zs*
    using *assms* **by** (*blast dest*: *remdups-fwd-split*)
  moreover
  hence *x* ∈ *set vs* **and** *x* ∉ *Acc* ∪ *set ys*
   using *remdups-fwd-acc-set*[*of Acc* ∪ *set ys*] **by** (*fastforce*, *metis* (*no-types*) *Diff-iff list.set-intros*(*1*))
  moreover
  then obtain *vs′ vs″* **where** *vs* = *vs′* @ *x* # *vs″* **and** *x* ∉ *set vs′*
    by (*blast dest*: *split-list-first*)
  moreover
  hence *set vs′* ⊆ *Acc* ∪ *set ys*
    using ‹*remdups-fwd-acc* (*Acc* ∪ *set ys*) *vs* = *x* # *zs*› ‹*x* ∉ *Acc* ∪ *set ys*›

    unfolding *remdups-fwd-acc-empty* **by** (*fastforce intro*: *list-empty-prefix*)
  moreover
  hence *remdups-fwd-acc* (*Acc* ∪ *set ys*) *vs′* = []
    using *remdups-fwd-acc-empty* **by** *blast*
  ultimately
  have *xs* = (*us* @ *vs′*) @ *x* # *vs″*
    and *remdups-fwd-acc Acc* (*us* @ *vs′*) = *ys*
    and *remdups-fwd-acc* (*Acc* ∪ *set ys* ∪ {*x*}) *vs″* = *zs*
    by (*fastforce dest*: *sup.absorb1*)+
  thus *?thesis*
    using ‹*x* ∉ *Acc* ∪ *set ys*› **by** *blast*
qed

lemma *remdups-fwd-split-exactE*:
  assumes *remdups-fwd-acc Acc xs* = *ys* @ *x* # *zs*
 obtains *us vs* **where** *xs* = *us* @ *x* # *vs* **and** *x* ∉ *set us* **and** *remdups-fwd-acc Acc us* = *ys* **and** *remdups-fwd-acc* (*Acc* ∪ *set ys* ∪ {*x*}) *vs* = *zs*
    using *remdups-fwd-split-exact*[*OF assms*] **by** *auto*

lemma *remdups-fwd-split-exact-iff*:
  remdups-fwd-acc Acc xs = *ys* @ *x* # *zs* ⟷
    (∃ *us vs*. *xs* = *us* @ *x* # *vs* ∧ *x* ∉ *Acc* ∧ *x* ∉ *set us* ∧ *remdups-fwd-acc Acc us* = *ys* ∧ *remdups-fwd-acc* (*Acc* ∪ *set ys* ∪ {*x*}) *vs* = *zs*)
  using *remdups-fwd-split-exact* **by** *fastforce*

lemma *sorted-pre*:
  (⋀*x y xs ys*. *zs* = *xs* @ [*x*, *y*] @ *ys* ⟹ *x* ≤ *y*) ⟹ *sorted zs*
apply (*induction zs*)
 apply *simp*
by (*metis append-Nil append-Cons list.exhaust sorted1 sorted2*)

lemma *sorted-list*:
  assumes $x \in set\ xs$ **and** $y \in set\ xs$
  assumes *sorted* (*map f xs*) **and** $f\ x < f\ y$
  shows $\exists xs'\ xs''\ xs'''.\ xs = xs'\ @\ x\ \#\ xs''\ @\ y\ \#\ xs'''$
proof −
  obtain *ys zs* **where** $xs = ys\ @\ y\ \#\ zs$ **and** $y \notin set\ ys$
    using *assms* **by** (*blast dest*: *split-list-first*)
  moreover
  hence *sorted* (*map f* ($y\ \#\ zs$))
    using ‹*sorted* (*map f xs*)› **by** (*simp add*: *sorted-append*)
  hence $\forall x \in set\ (map\ f\ (y\ \#\ zs)).\ f\ y \leq x$
    by *force*
  hence $\forall x \in set\ (y\ \#\ zs).\ f\ y \leq f\ x$
    by *auto*
  have $x \in set\ ys$
    apply (*rule ccontr*)
    using ‹$f\ x < f\ y$› ‹$x \in set\ xs$› ‹$\forall x \in set\ (y\ \#\ zs).\ f\ y \leq f\ x$› **unfolding** ‹$xs = ys\ @\ y\ \#\ zs$› *set-append* **by** *auto*
  then obtain $ys'\ zs'$ **where** $ys = ys'\ @\ x\ \#\ zs'$
    using *assms* **by** (*blast dest*: *split-list-first*)
  ultimately
  show *?thesis*
    by *auto*
qed

lemma *takeWhile-foo*:
  $x \notin set\ ys \Longrightarrow ys = takeWhile\ (\lambda y.\ y \neq x)\ (ys\ @\ x\ \#\ zs)$
  by (*metis* (*mono-tags*, *lifting*) *append-Nil2 takeWhile.simps*(*2*) *takeWhile-append2*)

lemma *takeWhile-split*:
  $x \in set\ xs \Longrightarrow y \in set\ (takeWhile\ (\lambda y.\ y \neq x)\ xs) \Longrightarrow \exists xs'\ xs''\ xs'''.\ xs = xs'\ @\ y\ \#\ xs''\ @\ x\ \#\ xs'''$
  using *split-list-first* **by** (*metis append-Cons append-assoc takeWhile-foo*)

lemma *takeWhile-distinct*:
  $distinct\ (xs'\ @\ x\ \#\ xs'') \Longrightarrow y \in set\ (takeWhile\ (\lambda y.\ y \neq x)\ (xs'\ @\ x\ \#\ xs'')) \longleftrightarrow y \in set\ xs'$
  by (*induction xs'*) *simp+*

lemma *finite-lists-length-eqE*:
  assumes *finite A*
  shows *finite* $\{xs.\ set\ xs = A \wedge length\ xs = n\}$
proof −
  have $\{xs.\ set\ xs = A \wedge length\ xs = n\} \subseteq \{xs.\ set\ xs \subseteq A \wedge length\ xs =$

n}
  by *blast*
 thus *?thesis*
   using *finite-lists-length-eq*[*OF assms*(*1*), *of n*] **using** *finite-subset* **by** *auto*
qed

lemma *finite-set2*:
 assumes *finite A*
 shows *finite* {*xs. set xs* = *A* ∧ *distinct xs*}
by(*blast intro*: *rev-finite-subset*[*OF finite-subset-distinct*[*OF assms*]])

lemma *set-list*:
 assumes *finite* (*set* ‘ *XS*)
 assumes ⋀*xs. xs* ∈ *XS* ⟹ *distinct xs*
 shows *finite XS*
proof −
 have *XS* ⊆ {*xs* | *xs. set xs* ∈ *set* ‘ *XS* ∧ *distinct xs*}
  using *assms* **by** *auto*
 moreover
 have *1*: {*xs* |*xs. set xs* ∈ *set* ‘ *XS* ∧ *distinct xs*} = ⋃{{*xs* | *xs. set xs* = *A* ∧ *distinct xs*} | *A. A* ∈ *set* ‘ *XS*}
  by *auto*
 have *finite* {*xs* |*xs. set xs* ∈ *set* ‘ *XS* ∧ *distinct xs*}
  using *finite-set2*[*OF finite-set*] *distinct-card assms*(*1*) **unfolding** *1* **by** *fastforce*
 ultimately
 show *?thesis*
  using *finite-subset* **by** *blast*
qed

lemma *set-foldl-append*:
 set (*foldl* (@) *i xs*) = *set i* ∪ ⋃{*set x* | *x. x* ∈ *set xs*}
 by (*induction xs arbitrary*: *i*) *auto*

8.3 Short-circuited Version of *foldl*

fun *foldl-break* :: ('*b* ⇒ '*a* ⇒ '*b*) ⇒ ('*b* ⇒ *bool*) ⇒ '*b* ⇒ '*a list* ⇒ '*b*
where
 foldl-break f s a [] = *a*
| *foldl-break f s a* (*x* # *xs*) = (*if s a then a else foldl-break f s* (*f a x*) *xs*)

lemma *foldl-break-append*:
 foldl-break f s a (*xs* @ *ys*) = (*if s* (*foldl-break f s a xs*) *then foldl-break f s*

a xs else (*foldl-break f s* (*foldl-break f s a xs*) *ys*))
by (*induction xs arbitrary*: *a*) (*cases ys*, *auto*)

8.4 Suffixes

fun *suffixes*
where
suffixes [] = []
| *suffixes* (*x*#*xs*) = (*suffixes xs*) @ [*x*#*xs*]

lemma *suffixes-append*:
suffixes (*xs* @ *ys*) = (*suffixes ys*) @ (*map* (λ*zs*. *zs* @ *ys*) (*suffixes xs*))
by (*induction xs*) *simp-all*

lemma *suffixes-alt-def*:
suffixes xs = *rev* (*prefix* (*length xs*) (λ*i*. *drop i xs*))
proof (*induction xs rule*: *rev-induct*)
case (*snoc x xs*)
show *?case*
by (*simp add*: *subsequence-def suffixes-append snoc rev-map*)
qed *simp*

end

9 Translation to Deterministic Transition-Based Rabin Automata

theory *Mojmir-Rabin*
imports *Main Mojmir Rabin Auxiliary/List2*
begin

locale *mojmir-to-rabin-def* = *mojmir-def*
begin

definition $fail_R$ :: ($'b \Rightarrow$ *nat option*, $'a$) *transition set*
where
$fail_R = \{(r, \nu, s) \mid r\ \nu\ s\ q\ q'.\ r\ q \neq None \wedge q' = \delta\ q\ \nu \wedge q' \notin F \wedge sink\ q'\}$

definition $succeed_R$ :: *nat* $\Rightarrow$ ($'b \Rightarrow$ *nat option*, $'a$) *transition set*
where
$succeed_R\ i = \{(r, \nu, s) \mid r\ \nu\ s\ q.\ r\ q = Some\ i \wedge q \notin F - \{q_0\} \wedge (\delta\ q\ \nu) \in F\}$

definition $merge_R$ :: $nat \Rightarrow ('b \Rightarrow nat\ option,\ 'a)\ transition\ set$
where
$merge_R\ i = \{(r,\ \nu,\ s) \mid r\ \nu\ s\ q\ q'\ j.\ r\ q = Some\ j \wedge j < i \wedge q' = \delta\ q\ \nu \wedge ((\exists\, q''.\ q' = \delta\ q''\ \nu \wedge r\ q'' \neq None \wedge q'' \neq q) \vee q' = q_0) \wedge q' \notin F\}$

abbreviation Q_R
where
$Q_R \equiv reach\ \Sigma\ step\ initial$

abbreviation $q_{\mathcal{R}}$
where
$q_{\mathcal{R}} \equiv initial$

abbreviation $\delta_{\mathcal{R}}$
where
$\delta_{\mathcal{R}} \equiv step$

abbreviation $Acc_{\mathcal{R}}$
where
$Acc_{\mathcal{R}}\ j \equiv (fail_R \cup merge_R\ j,\ succeed_R\ j)$

abbreviation $\mathcal{R}$
where
$\mathcal{R} \equiv (\delta_{\mathcal{R}},\ q_{\mathcal{R}},\ \{Acc_{\mathcal{R}}\ j \mid j.\ j < max\text{-}rank\})$

end

9.1 Well-formedness

lemma *function-set-finite*:
assumes *finite R*
assumes *finite A*
shows $finite\ \{f.\ (\forall\, x.\ x \notin R \longrightarrow f\ x = c) \wedge (\forall\, x.\ x \in R \longrightarrow f\ x \in A)\}$ (**is** *finite ?F*)
using *assms(1)*
proof (*induction R rule*: *finite-induct*)
case *empty*
have $\{f.\ (\forall\, x.\ x \in \{\} \longrightarrow f\ x \in A) \wedge (\forall\, x.\ x \notin \{\} \longrightarrow f\ x = c)\} \subseteq \{\lambda x.\ c\}$
by *auto*
thus *?case*
using *finite-subset* **by** *auto*
next

case (*insert r R*)
  let *?F* = {*f*. (∀ *x*. *x* ∉ *R* ∪ {*r*} ⟶ *f x* = *c*) ∧ (∀ *x*. *x* ∈ *R* ∪ {*r*} ⟶ *f x* ∈ *A*)}
  let *?F′* = {*f*. (∀ *x*. *x* ∉ *R* ⟶ *f x* = *c*) ∧ (∀ *x*. *x* ∈ *R* ⟶ *f x* ∈ *A*)}

  have *?F* ⊆ {*f*(*r* := *a*) | *f a*. *f* ∈ *?F′* ∧ *a* ∈ *A*} (**is** - ⊆ *?X*)
  proof
    fix *f*
    assume *f* ∈ *?F*
    hence *f*(*r* := *c*) ∈ *?F′* **and** *f r* ∈ *A*
      using *insert*(*2*) **by** (*simp*, *blast*)
    hence *f*(*r* := *c*, *r* := (*f r*)) ∈ *?X*
      by *blast*
    thus *f* ∈ *?X*
      by *simp*
  qed
  moreover
  have *finite* (*?F′* × *A*)
    using *assms*(*2*) *insert*(*3*) **by** *simp*
  have (λ(*f*,*a*). *f*(*r*:=*a*)) ‘ (*?F′* × *A*) = *?X*
    by *auto*
  hence *finite ?X*
    using *finite-imageI*[*OF* ‹*finite* (*?F′* × *A*)›, *of* (λ(*f*,*a*). *f*(*r*:=*a*))] **by** *presburger*
  ultimately
  have *finite ?F*
    by (*blast intro*: *finite-subset*)
  thus *?case*
    unfolding *insert-def* **by** *simp*
qed

lemma (**in** *semi-mojmir*) *wellformed-ℛ*:
  shows *finite* (*reach* Σ *step initial*)
proof (*rule finite-subset*)
  let *?R* = {*f*. (∀ *x*. *x* ∉ *reach* Σ δ q_0 ⟶ *f x* = *None*) ∧
    (∀ *x*. *x* ∈ *reach* Σ δ q_0 ⟶ *f x* ∈ {*None*} ∪ *Some* ‘ {*0*..<*max-rank*})}

  show *reach* Σ *step initial* ⊆ *?R*
  proof
    fix *x*
    assume *x* ∈ *reach* Σ *step initial*
    then obtain *w* **where** *x-def*: *x* = *foldl step initial w* **and** *set w* ⊆ Σ
      unfolding *reach-foldl-def*[*OF nonempty-*Σ] **by** *blast*
    obtain *a* **where** *a* ∈ Σ

using *nonempty-*Σ **by** *blast*
have *range* $(w \frown (\lambda i.\ a)) \subseteq \Sigma$
using ‹*set* $w \subseteq \Sigma$› ‹$a \in \Sigma$› **unfolding** *conc-def* **by** *auto*
then interpret $\mathfrak{H}$: *semi-mojmir* $\Sigma\ \delta\ q_0\ (w \frown (\lambda i.\ a))$
using *finite-reach finite-*Σ **by** (*unfold-locales*; *simp-all*)
have $x = (\lambda q.\ \mathfrak{H}$*.state-rank q* (*length w*))
unfolding $\mathfrak{H}$*.state-rank-step-foldl x-def* **using** *prefix-conc-length subsequence-def* **by** *metis*
thus $x \in$ *?R*
using $\mathfrak{H}$*.state-rank-in-function-set* **by** *meson*
qed

have *finite* ({*None*} $\cup$ *Some* ‘ {*0..<max-rank*})
by *blast*
thus *finite ?R*
using *finite-reach* **by** (*blast intro*: *function-set-finite*)
qed

locale *mojmir-to-rabin* = *mojmir* + *mojmir-to-rabin-def* **begin**

9.2 Correctness

lemma *imp-and-not-imp-eq*:
assumes $P \Longrightarrow Q$
assumes $\neg P \Longrightarrow \neg Q$
shows $P = Q$
using *assms* **by** *blast*

lemma *finite-limit-intersection*:
assumes *finite* (*range f*)
assumes $\bigwedge x$*::nat.* $x \in A \longleftrightarrow (f\ x) \in B$
shows *finite A* $\longleftrightarrow$ *limit f* $\cap B = \{\}$
proof (*rule imp-and-not-imp-eq*)
assume *finite A*
{
fix *x*
assume $x >$ *Max* ($A \cup \{0\}$)
moreover
have *finite* ($A \cup \{0\}$)
using ‹*finite A*› **by** *simp*
ultimately
have $x \notin A$
using *Max.coboundedI*
by (*metis insert-iff insert-is-Un not-le sup.commute*)

hence *f x* $\notin$ *B*
using *assms*(*2*) **by** *simp*
}
hence *f* ' {*Suc* (*Max* (*A* $\cup$ {*0*}))..} $\cap$ *B* = {}
by *auto*
thus *limit f* $\cap$ *B* = {}
using *limit-subset*[*of f*] **by** *blast*
next
assume *infinite A*
have *f* ' *A* $\subseteq$ *B*
unfolding *image-def* **using** *assms* **by** *auto*
moreover
have *finite* (*range f*)
using *assms*(*1*) **unfolding** *limit-def Inf-many-def* **by** *simp*
hence *finite* (*f* ' *A*)
by (*metis infinite-iff-countable-subset subset-UNIV subset-image-iff*)
then obtain *y* **where** *y* $\in$ *f* ' *A* **and** $\exists_\infty n.\ f\ n = y$
unfolding *Inf-many-def* **using** *pigeonhole-infinite*[*OF* ‹*infinite A*›] **by** *fast*
ultimately
show *limit f* $\cap$ *B* $\neq$ {}
using *limit-iff-frequent* **by** *fast*
qed

lemma *finite-range-run*:
defines *r* $\equiv$ $run_t\ \delta_{\mathcal{R}}\ q_{\mathcal{R}}\ w$
shows *finite* (*range r*)
proof −
{
fix *n*
have $\bigwedge n.$ *w n* $\in \Sigma$ **and** *set* (*map w* [*0*..<*n*]) $\subseteq \Sigma$ **and** *set* (*map w* [*0*..<*Suc n*]) $\subseteq \Sigma$
using *bounded-w* **by** *auto*
hence *r n* $\in Q_R \times \Sigma \times Q_R$
unfolding run_t*.simps run-foldl reach-foldl-def*[*OF nonempty-*Σ] *r-def*
unfolding *fst-conv comp-def snd-conv* **by** *blast*
}
hence *range r* $\subseteq Q_R \times \Sigma \times Q_R$
by *blast*
thus *finite* (*range r*)
using *finite-*Σ *wellformed-*$\mathcal{R}$
by (*blast dest*: *finite-subset*)
qed

theorem *mojmir-accept-iff-rabin-accept-rank*:
  shows (*finite* (*fail*) $\wedge$ *finite* (*merge i*) $\wedge$ *infinite* (*succeed i*))
    $\longleftrightarrow$ *accepting-pair*$_R$ $\delta_{\mathcal{R}}$ $q_{\mathcal{R}}$ ($Acc_{\mathcal{R}}$ *i*) *w*
  (**is** *?lhs* = *?rhs*)
proof
  define *r* **where** $r = run_t\ \delta_{\mathcal{R}}\ q_{\mathcal{R}}\ w$
  have *r-alt-def*: $\bigwedge i.\ r\ i = (\lambda q.\ state\text{-}rank\ q\ i,\ w\ i,\ \lambda q.\ state\text{-}rank\ q\ (Suc\ i))$
    using *r-def state-rank-step-foldl run-foldl* **by** *fastforce*

  have *F*: $\bigwedge x.\ x \in fail\text{-}t \longleftrightarrow (r\ x) \in fail_R$
    unfolding *fail-t-def fail*$_R$*-def r-alt-def* **by** *blast*
  have *B*: $\bigwedge x\ i.\ x \in merge\text{-}t\ i \longleftrightarrow (r\ x) \in merge_R\ i$
    unfolding *merge-t-def merge*$_R$*-def r-alt-def* **by** *blast*
  have *S*: $\bigwedge x\ i.\ x \in succeed\text{-}t\ i \longleftrightarrow (r\ x) \in succeed_R\ i$
    unfolding *succeed-t-def succeed*$_R$*-def r-alt-def* **by** *blast*

  have *finite* (*range r*)
    using *finite-range-run r-def* **by** *simp*
  note *finite-limit-rule* = *finite-limit-intersection*[*OF* ‹*finite* (*range r*)›]

  {
    assume *?lhs*
    hence *finite fail-t* **and** *finite* (*merge-t i*) **and** *infinite* (*succeed-t i*)
      unfolding *finite-fail-t finite-merge-t finite-succeed-t* **by** *blast+*

    have $limit\ r \cap fail_R = \{\}$
      by (*metis finite-limit-rule F* ‹*finite* (*fail-t*)›)
    moreover
    have $limit\ r \cap merge_R\ i = \{\}$
      by (*metis finite-limit-rule B* ‹*finite* (*merge-t i*)›)
    ultimately
    have $limit\ r \cap fst\ (Acc_{\mathcal{R}}\ i) = \{\}$
      by *auto*
    moreover
    have $limit\ r \cap succeed_R\ i \neq \{\}$
      by (*metis finite-limit-rule S* ‹*infinite* (*succeed-t i*)›)
    hence $limit\ r \cap snd\ (Acc_{\mathcal{R}}\ i) \neq \{\}$
      by *auto*
    ultimately
    show *?rhs*
      unfolding *r-def* **by** *simp*
  }

  {

assume *?rhs*
hence *limit r* $\cap$ $fail_R$ = {} **and** *limit r* $\cap$ $merge_R$ *i* = {} **and** *limit r* $\cap$ ($succeed_R$ *i*) $\neq$ {}
unfolding *r-def* **by** *auto*

have *finite fail-t*
by (*metis finite-limit-rule F* ‹*limit r* $\cap$ $fail_R$ = {}›)
moreover
have *finite* (*merge-t i*)
by (*metis finite-limit-rule B* ‹*limit r* $\cap$ $merge_R$ *i* = {}›)
moreover
have *infinite* (*succeed-t i*)
by (*metis finite-limit-rule S* ‹*limit r* $\cap$ ($succeed_R$ *i*) $\neq$ {}›)
ultimately
show *?lhs*
unfolding *finite-fail-t finite-merge-t finite-succeed-t* **by** *blast*
}
qed

theorem *mojmir-accept-iff-rabin-accept*:
accept $\longleftrightarrow$ $accept_R$ $\mathcal{R}$ *w*
unfolding *mojmir-accept-iff-token-set-accept mojmir-accept-iff-rabin-accept-rank*
by *auto*

definition *smallest-accepting-rank*$_{\mathcal{R}}$ :: *nat option*
where
smallest-accepting-rank$_{\mathcal{R}}$ $\equiv$ (*if* $accept_R$ $\mathcal{R}$ *w then*
Some (*LEAST i. accepting-pair*$_R$ $\delta_{\mathcal{R}}$ $q_{\mathcal{R}}$ ($Acc_{\mathcal{R}}$ *i*) *w*) *else None*)

theorem *Mojmir-rabin-smallest-accepting-rank*:
smallest-accepting-rank = *smallest-accepting-rank*$_{\mathcal{R}}$
by (*simp only*: *smallest-accepting-rank-def smallest-accepting-rank*$_{\mathcal{R}}$*-def*
mojmir-accept-iff-rabin-accept mojmir-accept-iff-rabin-accept-rank)

lemma *smallest-accepting-rank*$_{\mathcal{R}}$*-properties*:
smallest-accepting-rank$_{\mathcal{R}}$ = *Some i* $\Longrightarrow$ *accepting-pair*$_R$ $\delta_{\mathcal{R}}$ $q_{\mathcal{R}}$ ($Acc_{\mathcal{R}}$ *i*) *w*
by (*unfold Mojmir-rabin-smallest-accepting-rank*[*symmetric*] *mojmir-accept-iff-rabin-accept-rank*[*sy*
blast dest: *smallest-accepting-rank-properties*)

end

end

10 LTL (in Negation-Normal-Form, FGXU-Syntax)

theory *LTL-FGXU*
imports *Main HOL−Library.Omega-Words-Fun*
begin

Inspired/Based on schimpf/LTL

10.1 Syntax

datatype (*vars*: *'a*) *ltl* =
LTLTrue (‹*true*›)
| *LTLFalse* (‹*false*›)
| *LTLProp 'a* (‹*p'*(-*'*)›)
| *LTLPropNeg 'a* (‹*np'*(-*'*)› [*86*] *85*)
| *LTLAnd 'a ltl 'a ltl* (‹- *and* -› [*83*,*83*] *82*)
| *LTLOr 'a ltl 'a ltl* (‹- *or* -› [*82*,*82*] *81*)
| *LTLNext 'a ltl* (‹*X* -› [*88*] *87*)
| *LTLGlobal* (*theG*: *'a ltl*) (‹*G* -› [*85*] *84*)
| *LTLFinal 'a ltl* (‹*F* -› [*84*] *83*)
| *LTLUntil 'a ltl 'a ltl* (‹- *U* -› [*87*,*87*] *86*)

10.2 Semantics

fun *ltl-semantics* :: [*'a set word*, *'a ltl*] ⇒ *bool* (**infix** ‹⊨› *80*)
where
$w \models true = True$
| $w \models false = False$
| $w \models p(q) = (q \in w\ 0)$
| $w \models np(q) = (q \notin w\ 0)$
| $w \models \varphi\ and\ \psi = (w \models \varphi \wedge w \models \psi)$
| $w \models \varphi\ or\ \psi = (w \models \varphi \vee w \models \psi)$
| $w \models X\ \varphi = (suffix\ 1\ w \models \varphi)$
| $w \models G\ \varphi = (\forall k.\ suffix\ k\ w \models \varphi)$
| $w \models F\ \varphi = (\exists k.\ suffix\ k\ w \models \varphi)$
| $w \models \varphi\ U\ \psi = (\exists k.\ suffix\ k\ w \models \psi \wedge (\forall j < k.\ suffix\ j\ w \models \varphi))$

fun *ltl-prop-entailment* :: [*'a ltl set*, *'a ltl*] ⇒ *bool* (**infix** ‹$\models_P$› *80*)
where
$\mathcal{A} \models_P true = True$
| $\mathcal{A} \models_P false = False$
| $\mathcal{A} \models_P \varphi\ and\ \psi = (\mathcal{A} \models_P \varphi \wedge \mathcal{A} \models_P \psi)$
| $\mathcal{A} \models_P \varphi\ or\ \psi = (\mathcal{A} \models_P \varphi \vee \mathcal{A} \models_P \psi)$
| $\mathcal{A} \models_P \varphi = (\varphi \in \mathcal{A})$

10.2.1 Properties

lemma *LTL-G-one-step-unfolding*:
  $w \models G\ \varphi \longleftrightarrow (w \models \varphi \wedge w \models X\ (G\ \varphi))$
  (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
  assume *?lhs*
  hence $w \models \varphi$
    using *suffix-0*[*of w*] *ltl-semantics.simps*(*8*)[*of w* φ] **by** *metis*
  moreover
  from ‹*?lhs*› **have** $w \models X\ (G\ \varphi)$
    by *simp*
  ultimately
  show *?rhs* **by** *simp*
next
  assume *?rhs*
  hence $w \models X\ (G\ \varphi)$ **by** *simp*
  hence $\forall k.\ suffix\ (k + 1)\ w \models \varphi$ **by** *force*
  hence $\forall k > 0.\ suffix\ k\ w \models \varphi$
    by (*metis Suc-eq-plus1 gr0-implies-Suc*)
  moreover
  from ‹*?rhs*› **have** $(suffix\ 0\ w) \models \varphi$ **by** *simp*
  ultimately
  show *?lhs*
    using *neq0-conv ltl-semantics.simps*(*8*)[*of w* φ] **by** *blast*
qed

lemma *LTL-F-one-step-unfolding*:
  $w \models F\ \varphi \longleftrightarrow (w \models \varphi \vee w \models X\ (F\ \varphi))$
  (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
  assume *?lhs*
  then obtain *k* **where** $suffix\ k\ w \models \varphi$ **by** *fastforce*
  thus *?rhs* **by** (*cases k*) *auto*
next
  assume *?rhs*
  thus *?lhs*
   using *suffix-0*[*of w*] *suffix-suffix*[*of - 1 w*] **by** (*metis ltl-semantics.simps*(*7*)
ltl-semantics.simps(*9*))
qed

lemma *LTL-U-one-step-unfolding*:
  $w \models \varphi\ U\ \psi \longleftrightarrow (w \models \psi \vee (w \models \varphi \wedge w \models X\ (\varphi\ U\ \psi)))$
  (**is** *?lhs* $\longleftrightarrow$ *?rhs*)

proof
  assume *?lhs*
  then obtain k **where** *suffix k w* $\models \psi$ **and** $\forall j<k.$ *suffix j w* $\models \varphi$
    by *force*
  thus *?rhs*
    by (*cases k*) *auto*
next
  assume *?rhs*
  thus *?lhs*
  proof (*cases* $w \models \psi$)
    case *False*
      hence $w \models \varphi \wedge w \models X\ (\varphi\ U\ \psi)$
        using ‹*?rhs*› **by** *blast*
      obtain k **where** *suffix k* (*suffix 1 w*) $\models \psi$ **and** $\forall j<k.$ *suffix j* (*suffix 1 w*) $\models \varphi$
        using *False* ‹*?rhs*› **by** *force*
      moreover
      {
        fix j **assume** $j < 1 + k$
        hence *suffix j w* $\models \varphi$
          using ‹$w \models \varphi \wedge w \models X\ (\varphi\ U\ \psi)$› ‹$\forall j<k.$ *suffix j* (*suffix 1 w*) $\models \varphi$›[*unfolded suffix-suffix*]
          by (*cases j*) *simp+*
      }
      ultimately
      show *?thesis*
        by *auto*
  qed *force*
qed

lemma *LTL-GF-infinitely-many-suffixes*:
  $w \models G\ (F\ \varphi) = (\exists_\infty i.$ *suffix i w* $\models \varphi)$
  (**is** *?lhs* = *?rhs*)
proof
  let $?S = \{i \mid i\ j.$ *suffix* $(i + j)\ w \models \varphi\}$
  let $?S' = \{i + j \mid i\ j.$ *suffix* $(i + j)\ w \models \varphi\}$

  assume *?lhs*
  hence *infinite ?S*
    by *auto*
  moreover
  have $\forall s \in ?S.\ \exists s' \in ?S'.\ s \leq s'$
    by *fastforce*
  ultimately

have *infinite ?S′*
  using *infinite-nat-iff-unbounded-le le-trans* **by** *meson*
moreover
have $?S' = \{k \mid k.\ suffix\ k\ w \models \varphi\}$
  using *monoid-add-class.add.left-neutral* **by** *metis*
ultimately
have *infinite* $\{k \mid k.\ suffix\ k\ w \models \varphi\}$
  by *metis*
thus *?rhs* **unfolding** *Inf-many-def* **by** *force*

next
assume *?rhs*
{
  fix *i*
  from ‹*?rhs*› **obtain** *k* **where** $i \leq k$ **and** $suffix\ k\ w \models \varphi$
    using *INFM-nat-le*[*of* $\lambda n.\ suffix\ n\ w \models \varphi$] **by** *blast*
  then obtain *j* **where** $k = i + j$
    using *le-iff-add*[*of i k*] **by** *fast*
  hence $suffix\ j\ (suffix\ i\ w) \models \varphi$
    using ‹$suffix\ k\ w \models \varphi$› *suffix-suffix* **by** *fastforce*
  hence $suffix\ i\ w \models F\ \varphi$ **by** *auto*
}
thus *?lhs* **by** *auto*

qed

lemma *LTL-FG-almost-all-suffixes*:
  $w \models F\ G\ \varphi = (\forall_\infty i.\ suffix\ i\ w \models \varphi)$
  (**is** *?lhs* = *?rhs*)

proof
let $?S = \{k.\ \neg\ suffix\ k\ w \models \varphi\}$

assume *?lhs*
then obtain *i* **where** $suffix\ i\ w \models G\ \varphi$
  by *fastforce*
hence $\bigwedge j.\ j \geq i \Longrightarrow (suffix\ j\ w \models \varphi)$
  using *le-iff-add*[*of i*] **by** *auto*
hence $\bigwedge j.\ \neg suffix\ j\ w \models \varphi \Longrightarrow j < i$
  using *le-less-linear* **by** *blast*
hence $?S \subseteq \{k.\ k < i\}$
  by *blast*
hence *finite ?S*
  using *finite-subset* **by** *fast*
thus *?rhs*
  unfolding *Alm-all-def Inf-many-def* **by** *presburger*

next

assume *?rhs*
obtain S **where** *S-def*: $S = \{k.\ \neg$ *suffix* $k\ w \models \varphi\}$ **by** *blast*
hence *finite* S
using ‹*?rhs*› **unfolding** *Alm-all-def Inf-many-def* **by** *fast*
then obtain i **where** $i = Max\ S$ **by** *blast*
{
fix j
assume $i < j$
hence $j \notin S$
using ‹$i = Max\ S$› *Max.coboundedI*[*OF* ‹*finite* S›] *less-le-not-le* **by** *blast*
hence *suffix* $j\ w \models \varphi$ **using** *S-def* **by** *fast*
}
hence $\forall j > i.$ (*suffix* $j\ w \models \varphi$) **by** *simp*
hence *suffix* (*Suc* i) $w \models G\ \varphi$ **by** *auto*
thus *?lhs*
using *ltl-semantics.simps*(*9*)[*of* $w\ G\ \varphi$] **by** *blast*
qed

lemma *LTL-FG-suffix*:
(*suffix* $i\ w$) $\models F\ (G\ \varphi) = w \models F\ (G\ \varphi)$
proof −
have ($\exists m.\ \forall n \geq m.$ *suffix* $n\ w \models \varphi$) = ($\exists m.\ \forall n \geq m.$ *suffix* n (*suffix* $i\ w$) $\models \varphi$) (**is** *?l* = *?r*)
proof
assume *?r*
then obtain m **where** $\forall n \geq m.$ *suffix* n (*suffix* $i\ w$) $\models \varphi$
by *blast*
hence $\forall n \geq i+m.$ *suffix* $n\ w \models \varphi$
unfolding *suffix-suffix* **by** (*metis le-iff-add add-leE add-le-cancel-left*)
thus *?l*
by *auto*
qed (*metis suffix-suffix trans-le-add2*)
thus *?thesis*
unfolding *LTL-FG-almost-all-suffixes MOST-nat-le* **..**
qed

lemma *LTL-GF-suffix*:
(*suffix* $i\ w$) $\models G\ (F\ \varphi) = w \models G\ (F\ \varphi)$
proof −
have ($\forall m.\ \exists n \geq m.$ *suffix* $n\ w \models \varphi$) = ($\forall m.\ \exists n \geq m.$ *suffix* n (*suffix* $i\ w$) $\models \varphi$) (**is** *?l* = *?r*)
proof
assume *?l*

thus *?r*
by (*metis suffix-suffix add-leE add-le-cancel-left le-Suc-ex*)
qed (*metis suffix-suffix trans-le-add2*)
thus *?thesis*
unfolding *LTL-GF-infinitely-many-suffixes INFM-nat-le* **..**
qed

lemma *LTL-suffix-G*:
$w \models G\ \varphi \Longrightarrow$ *suffix* $i\ w \models G\ \varphi$
using *suffix-0 suffix-suffix* **by** (*induction i*) *simp-all*

lemma *LTL-prop-entailment-monotonI*[*intro*]:
$S \models_P \varphi \Longrightarrow S \subseteq S' \Longrightarrow S' \models_P \varphi$
by (*induction rule*: *ltl-prop-entailment.induct*) *auto*

lemma *ltl-models-equiv-prop-entailment*:
$w \models \varphi = \{\chi.\ w \models \chi\} \models_P \varphi$
by (*induction* φ) *auto*

10.2.2 Limit Behaviour of the G-operator

abbreviation *Only-G*
where
Only-G $S \equiv \forall x \in S.\ \exists y.\ x = G\ y$

lemma *ltl-G-stabilize*:
assumes *finite* $\mathcal{G}$
assumes *Only-G* $\mathcal{G}$
obtains i **where** $\bigwedge\chi\ j.\ \chi \in \mathcal{G} \Longrightarrow$ *suffix* $i\ w \models \chi =$ *suffix* $(i + j)\ w \models \chi$
proof −
have *finite* $\mathcal{G} \Longrightarrow$ *Only-G* $\mathcal{G} \Longrightarrow \exists i.\ \forall \chi \in \mathcal{G}.\ \forall j.$ *suffix* $i\ w \models \chi =$ *suffix* $(i + j)\ w \models \chi$
proof (*induction rule*: *finite-induct*)
case (*insert* χ $\mathcal{G}$)
then obtain i_1 **where** $\bigwedge\chi\ j.\ \chi \in \mathcal{G} \Longrightarrow$ *suffix* $i_1\ w \models \chi =$ *suffix* $(i_1 + j)\ w \models \chi$
by *blast*
moreover
from *insert* **obtain** ψ **where** $\chi = G\ \psi$
by *blast*
have $\exists i.\ \forall j.$ *suffix* $i\ w \models G\ \psi =$ *suffix* $(i + j)\ w \models G\ \psi$
by (*metis LTL-suffix-G plus-nat.add-0 suffix-0 suffix-suffix*)
then obtain i_2 **where** $\bigwedge j.$ *suffix* $i_2\ w \models \chi =$ *suffix* $(i_2 + j)\ w \models \chi$
unfolding ‹$\chi = G\ \psi$› **by** *blast*

ultimately
have $\bigwedge\chi'\, j.\ \chi' \in \mathcal{G} \cup \{\chi\} \Longrightarrow$ *suffix* $(i_1 + i_2)\ w \models \chi' =$ *suffix* $(i_1 + i_2 + j)\ w \models \chi'$
unfolding *Un-iff singleton-iff* **by** (*metis add.commute add.left-commute*)
thus *?case*
by *blast*
qed *simp*
with *assms* **obtain** i **where** $\bigwedge\chi\ j.\ \chi \in \mathcal{G} \Longrightarrow$ *suffix* $i\ w \models \chi =$ *suffix* $(i + j)\ w \models \chi$
by *blast*
thus *?thesis*
using *that* **by** *blast*
qed

lemma *ltl-G-stabilize-property*:
assumes *finite* $\mathcal{G}$
assumes *Only-G* $\mathcal{G}$
assumes $\bigwedge\chi\ j.\ \chi \in \mathcal{G} \Longrightarrow$ *suffix* $i\ w \models \chi =$ *suffix* $(i + j)\ w \models \chi$
assumes $G\ \psi \in \mathcal{G} \cap \{\chi.\ w \models F\ \chi\}$
shows *suffix* $i\ w \models G\ \psi$
proof −
obtain j **where** *suffix* $j\ w \models G\ \psi$
using *assms* **by** *fastforce*
thus *suffix* $i\ w \models G\ \psi$
by (*cases* $i \le j$, *insert assms*, *unfold le-iff-add*, *blast*,
metis (*erased*, *lifting*) *LTL-suffix-G* ‹*suffix* $j\ w \models G\ \psi$› *le-add-diff-inverse nat-le-linear suffix-suffix*)
qed

10.3 Subformulae

10.3.1 Propositions

fun *propos* :: $'a$ *ltl* $\Rightarrow 'a$ *ltl set*
where
propos true = {}
| *propos false* = {}
| *propos* $(\varphi$ *and* $\psi)$ = *propos* $\varphi \cup$ *propos* ψ
| *propos* $(\varphi$ *or* $\psi)$ = *propos* $\varphi \cup$ *propos* ψ
| *propos* $\varphi = \{\varphi\}$

fun *nested-propos* :: $'a$ *ltl* $\Rightarrow 'a$ *ltl set*
where
nested-propos true = {}

| *nested-propos false* = {}
| *nested-propos* (φ *and* ψ) = *nested-propos* $\varphi \cup$ *nested-propos* ψ
| *nested-propos* (φ *or* ψ) = *nested-propos* $\varphi \cup$ *nested-propos* ψ
| *nested-propos* ($F\ \varphi$) = $\{F\ \varphi\} \cup$ *nested-propos* φ
| *nested-propos* ($G\ \varphi$) = $\{G\ \varphi\} \cup$ *nested-propos* φ
| *nested-propos* ($X\ \varphi$) = $\{X\ \varphi\} \cup$ *nested-propos* φ
| *nested-propos* ($\varphi\ U\ \psi$) = $\{\varphi\ U\ \psi\} \cup$ *nested-propos* $\varphi \cup$ *nested-propos* ψ
| *nested-propos* $\varphi = \{\varphi\}$

lemma *finite-propos*:
finite (*propos* φ) *finite* (*nested-propos* φ)
by (*induction* φ) *simp+*

lemma *propos-subset*:
propos $\varphi \subseteq$ *nested-propos* φ
by (*induction* φ) *auto*

lemma *LTL-prop-entailment-restrict-to-propos*:
$S \models_P \varphi$ = ($S \cap$ *propos* φ) $\models_P \varphi$
by (*induction* φ) *auto*

lemma *propos-foldl*:
assumes $\bigwedge x\ y$. *propos* ($f\ x\ y$) = *propos* $x \cup$ *propos* y
shows $\bigcup\{$*propos* y $|y.\ y = i \vee y \in$ *set xs*$\}$ = *propos* (*foldl f i xs*)
proof (*induction xs rule*: *rev-induct*)
case (*snoc x xs*)
have $\bigcup\{$*propos* y $|y.\ y = i \vee y \in$ *set* (*xs*@[x])$\}$ = $\bigcup\{$*propos* y $|y.\ y = i \vee y \in$ *set xs*$\} \cup$ *propos* x
by *auto*
also
have ... = *propos* (*foldl f i xs*) $\cup$ *propos* x
using *snoc* **by** *auto*
also
have ... = *propos* (*foldl f i* (*xs*@[x]))
using *assms* **by** *simp*
finally
show *?case* **.**
qed *simp*

10.3.2 G-Subformulae

Notation for paper: mathdsG

fun *G-nested-propos* :: *'a ltl* $\Rightarrow$ *'a ltl set* (‹**G**›)

where
  G (*φ and ψ*) = **G** *φ* ∪ **G** *ψ*
| **G** (*φ or ψ*) = **G** *φ* ∪ **G** *ψ*
| **G** (*F φ*) = **G** *φ*
| **G** (*G φ*) = **G** *φ* ∪ {*G φ*}
| **G** (*X φ*) = **G** *φ*
| **G** (*φ U ψ*) = **G** *φ* ∪ **G** *ψ*
| **G** *φ* = {}

lemma *G-nested-finite*:
  finite (**G** *φ*)
  by (*induction φ*) *auto*

lemma *G-nested-propos-alt-def*:
  G *φ* = *nested-propos φ* ∩ {*ψ*. (∃ *x*. *ψ* = *G x*)}
  by (*induction φ*) *auto*

lemma *G-nested-propos-Only-G*:
  Only-G (**G** *φ*)
  unfolding *G-nested-propos-alt-def* **by** *blast*

lemma *G-not-in-G*:
  G φ ∉ **G** *φ*
proof −
  have ⋀*χ*. *χ* ∈ **G** *φ* ⟹ *size φ* ≥ *size χ*
    by (*induction φ*) *fastforce+*
  thus *?thesis*
    by *fastforce*
qed

lemma *G-subset-G*:
  ψ ∈ **G** *φ* ⟹ **G** *ψ* ⊆ **G** *φ*
  G ψ ∈ **G** *φ* ⟹ **G** *ψ* ⊆ **G** *φ*
  by (*induction φ*) *auto*

lemma *𝒢-properties*:
  assumes 𝒢 ⊆ **G** *φ*
  shows *𝒢-finite*: *finite* 𝒢 **and** *𝒢-elements*: *Only-G* 𝒢
   using *assms G-nested-finite G-nested-propos-alt-def* **by** (*blast dest*: *finite-subset*)+

10.4 Propositional Implication and Equivalence

definition *ltl-prop-implies* :: [*'a ltl*, *'a ltl*] ⇒ *bool* (**infix** ‹⟶$_P$› *75*)

where
$\varphi \longrightarrow_P \psi \equiv \forall \mathcal{A}.\ \mathcal{A} \models_P \varphi \longrightarrow \mathcal{A} \models_P \psi$

definition *ltl-prop-equiv* :: $['a\ ltl, 'a\ ltl] \Rightarrow bool$ (**infix** ‹$\equiv_P$› *75*)
where
$\varphi \equiv_P \psi \equiv \forall \mathcal{A}.\ \mathcal{A} \models_P \varphi \longleftrightarrow \mathcal{A} \models_P \psi$

lemma *ltl-prop-implies-equiv*:
$\varphi \longrightarrow_P \psi \wedge \psi \longrightarrow_P \varphi \longleftrightarrow \varphi \equiv_P \psi$
unfolding *ltl-prop-implies-def ltl-prop-equiv-def* **by** *meson*

lemma *ltl-prop-equiv-equivp*:
equivp ($\equiv_P$)
by (*blast intro*: *equivpI*[*of* ($\equiv_P$), *simplified transp-def symp-def reflp-def ltl-prop-equiv-def*])

lemma [*trans*]:
$\varphi \equiv_P \psi \Longrightarrow \psi \equiv_P \chi \Longrightarrow \varphi \equiv_P \chi$
using *ltl-prop-equiv-equivp*[*THEN equivp-transp*] **.**

10.4.1 Quotient Type for Propositional Equivalence

quotient-type *'a ltl-prop-equiv-quotient* = *'a ltl* / ($\equiv_P$)
morphisms *Rep Abs*
by (*simp add*: *ltl-prop-equiv-equivp*)

type-synonym *'a* ltl_P = *'a ltl-prop-equiv-quotient*

instantiation *ltl-prop-equiv-quotient* :: (*type*) *equal* **begin**

lift-definition *ltl-prop-equiv-quotient-eq-test* :: *'a* $ltl_P \Rightarrow$ *'a* $ltl_P \Rightarrow bool$ **is** $\lambda x\ y.\ x \equiv_P y$
by (*metis ltl-prop-equiv-quotient.abs-eq-iff*)

definition
eq: *equal-class.equal* $\equiv$ *ltl-prop-equiv-quotient-eq-test*

instance
by (*standard*; *simp add*: *eq ltl-prop-equiv-quotient-eq-test.rep-eq*, *metis Quotient-ltl-prop-equiv-quotient Quotient-rel-rep*)

end

lemma ltl_P*-abs-rep*: *Abs* (*Rep* φ) = φ

by (*meson Quotient3-abs-rep Quotient3-ltl-prop-equiv-quotient*)

lift-definition *ltl-prop-entails-abs* :: $'a\ ltl\ set \Rightarrow\ 'a\ ltl_P \Rightarrow bool$ (‹- ↑$\models_P$ -›)
is ($\models_P$)
by (*simp add*: *ltl-prop-equiv-def*)

lift-definition *ltl-prop-implies-abs* :: $'a\ ltl_P \Rightarrow\ 'a\ ltl_P \Rightarrow bool$ (‹- ↑$\longrightarrow_P$ -›)
is ($\longrightarrow_P$)
by (*simp add*: *ltl-prop-equiv-def ltl-prop-implies-def*)

10.4.2 Propositional Equivalence implies LTL Equivalence

lemma *ltl-prop-implication-implies-ltl-implication*:
$w \models \varphi \Longrightarrow \varphi \longrightarrow_P \psi \Longrightarrow w \models \psi$
using [[*unfold-abs-def* = *false*]]
unfolding *ltl-prop-implies-def ltl-models-equiv-prop-entailment* **by** *simp*

lemma *ltl-prop-equiv-implies-ltl-equiv*:
$\varphi \equiv_P \psi \Longrightarrow w \models \varphi = w \models \psi$
using *ltl-prop-implication-implies-ltl-implication ltl-prop-implies-equiv* **by** *blast*

10.5 Substitution

fun *subst* :: $'a\ ltl \Rightarrow ('a\ ltl \rightharpoonup\ 'a\ ltl) \Rightarrow\ 'a\ ltl$
where
subst true m = *true*
| *subst false m* = *false*
| *subst* (φ *and* ψ) *m* = *subst* φ *m and subst* ψ *m*
| *subst* (φ *or* ψ) *m* = *subst* φ *m or subst* ψ *m*
| *subst* φ *m* = (*case m* φ *of Some* φ' ⇒ φ' | *None* ⇒ φ)

Based on Uwe Schoening's Translation Lemma (Logic for CS, p. 54)

lemma *ltl-prop-equiv-subst-S*:
$S \models_P subst\ \varphi\ m = ((S - dom\ m) \cup \{\chi \mid \chi\ \chi'.\ \chi \in dom\ m \wedge m\ \chi = Some\ \chi' \wedge S \models_P \chi'\}) \models_P \varphi$
by (*induction* φ) (*auto split*: *option.split*)

lemma *subst-respects-ltl-prop-entailment*:
$\varphi \longrightarrow_P \psi \Longrightarrow subst\ \varphi\ m \longrightarrow_P subst\ \psi\ m$
$\varphi \equiv_P \psi \Longrightarrow subst\ \varphi\ m \equiv_P subst\ \psi\ m$
unfolding *ltl-prop-equiv-def ltl-prop-implies-def ltl-prop-equiv-subst-S* **by** *blast+*

lemma *subst-respects-ltl-prop-entailment-generalized*:
$(\bigwedge \mathcal{A}.\ (\bigwedge x.\ x \in S \Longrightarrow \mathcal{A} \models_P x) \Longrightarrow \mathcal{A} \models_P y) \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow \mathcal{A} \models_P \mathit{subst}\ x\ m) \Longrightarrow \mathcal{A} \models_P \mathit{subst}\ y\ m$
unfolding *ltl-prop-equiv-subst-S* **by** *simp*

lemma *decomposable-function-subst*:
$[\![f\ \mathit{true} = \mathit{true};\ f\ \mathit{false} = \mathit{false};\ \bigwedge \varphi\ \psi.\ f\ (\varphi\ \mathit{and}\ \psi) = f\ \varphi\ \mathit{and}\ f\ \psi;\ \bigwedge \varphi\ \psi.\ f\ (\varphi\ \mathit{or}\ \psi) = f\ \varphi\ \mathit{or}\ f\ \psi]\!] \Longrightarrow f\ \varphi = \mathit{subst}\ \varphi\ (\lambda\chi.\ \mathit{Some}\ (f\ \chi))$
by (*induction* φ) *auto*

10.6 Additional Operators

10.6.1 And

lemma *foldl-LTLAnd-prop-entailment*:
$S \models_P \mathit{foldl}\ \mathit{LTLAnd}\ i\ \mathit{xs} = (S \models_P i \wedge (\forall y \in \mathit{set}\ \mathit{xs}.\ S \models_P y))$
by (*induction xs arbitrary*: *i*) *auto*

fun *And* :: *'a ltl list* ⇒ *'a ltl*
where
And [] = *true*
| *And* (*x#xs*) = *foldl LTLAnd x xs*

lemma *And-prop-entailment*:
$S \models_P \mathit{And}\ \mathit{xs} = (\forall x \in \mathit{set}\ \mathit{xs}.\ S \models_P x)$
using *foldl-LTLAnd-prop-entailment* **by** (*cases xs*) *auto*

lemma *And-propos*:
$\mathit{propos}\ (\mathit{And}\ \mathit{xs}) = \bigcup \{\mathit{propos}\ x \mid x.\ x \in \mathit{set}\ \mathit{xs}\}$
proof (*cases xs*)
case *Nil*
thus *?thesis* **by** *simp*
next
case (*Cons x xs*)
thus *?thesis*
using *propos-foldl*[*of LTLAnd x*] **by** *auto*
qed

lemma *And-semantics*:
$w \models \mathit{And}\ \mathit{xs} = (\forall x \in \mathit{set}\ \mathit{xs}.\ w \models x)$
proof –
have *And-propos'*: $\bigwedge x.\ x \in \mathit{set}\ \mathit{xs} \Longrightarrow \mathit{propos}\ x \subseteq \mathit{propos}\ (\mathit{And}\ \mathit{xs})$
using *And-propos* **by** *blast*

have $w \models And\ xs = \{\chi.\ \chi \in propos\ (And\ xs) \wedge w \models \chi\} \models_P (And\ xs)$
using *ltl-models-equiv-prop-entailment LTL-prop-entailment-restrict-to-propos*
by *blast*
also
have $\ldots = (\forall x \in set\ xs.\ \{\chi.\ \chi \in propos\ (And\ xs) \wedge w \models \chi\} \models_P x)$
using *And-prop-entailment* **by** *auto*
also
have $\ldots = (\forall x \in set\ xs.\ \{\chi.\ \chi \in propos\ x \wedge w \models \chi\} \models_P x)$
using *LTL-prop-entailment-restrict-to-propos And-propos′* **by** *blast*
also
have $\ldots = (\forall x \in set\ xs.\ w \models x)$
using *ltl-models-equiv-prop-entailment LTL-prop-entailment-restrict-to-propos*
by *blast*
finally
show *?thesis* **.**
qed

lemma *And-append-syntactic*:
$xs \neq [] \Longrightarrow And\ (xs\ @\ ys) = And\ ((And\ xs)\#ys)$
by (*induction xs rule*: *list-nonempty-induct*) *simp+*

lemma *And-append-S*:
$S \models_P And\ (xs\ @\ ys) = S \models_P And\ xs\ and\ And\ ys$
using *And-prop-entailment*[*of S*] **by** *auto*

lemma *And-append*:
$And\ (xs\ @\ ys) \equiv_P And\ xs\ and\ And\ ys$
unfolding *ltl-prop-equiv-def* **using** *And-append-S* **by** *blast*

10.6.2 Lifted Variant

lift-definition *and-abs* :: $'a\ ltl_P \Rightarrow 'a\ ltl_P \Rightarrow 'a\ ltl_P$ (‹- ↑*and* -›) **is** $\lambda x\ y.\ x$ *and y*
unfolding *ltl-prop-equiv-def* **by** *simp*

fun *And-abs* :: $'a\ ltl_P\ list \Rightarrow 'a\ ltl_P$ (‹↑*And*›)
where
↑*And xs* = *foldl and-abs* (*Abs true*) *xs*

lemma *foldl-LTLAnd-prop-entailment-abs*:
$S \uparrow\models_P foldl\ and\text{-}abs\ i\ xs = (S \uparrow\models_P i \wedge (\forall y{\in}set\ xs.\ S \uparrow\models_P y))$
by (*induction xs arbitrary*: *i*)
(*simp-all add*: *and-abs-def ltl-prop-entails-abs.abs-eq*, *metis ltl-prop-entails-abs.rep-eq*)

lemma *And-prop-entailment-abs*:
  $S \uparrow\models_P \uparrow And\ xs = (\forall x \in set\ xs.\ S \uparrow\models_P x)$
  by (*simp add*: *foldl-LTLAnd-prop-entailment-abs ltl-prop-entails-abs.abs-eq*)

lemma *and-abs-conjunction*:
  $S \uparrow\models_P \varphi \uparrow and\ \psi \longleftrightarrow S \uparrow\models_P \varphi \wedge S \uparrow\models_P \psi$
  by (*metis and-abs.abs-eq* ltl_P*-abs-rep ltl-prop-entailment.simps*(*3*) *ltl-prop-entails-abs.abs-eq*)

10.6.3 Or

lemma *foldl-LTLOr-prop-entailment*:
  $S \models_P foldl\ LTLOr\ i\ xs = (S \models_P i \vee (\exists y \in set\ xs.\ S \models_P y))$
  by (*induction xs arbitrary*: *i*) *auto*

fun *Or* :: *'a ltl list* ⇒ *'a ltl*
where
  Or [] = *false*
| *Or* (*x#xs*) = *foldl LTLOr x xs*

lemma *Or-prop-entailment*:
  $S \models_P Or\ xs = (\exists x \in set\ xs.\ S \models_P x)$
  using *foldl-LTLOr-prop-entailment* **by** (*cases xs*) *auto*

lemma *Or-propos*:
  $propos\ (Or\ xs) = \bigcup\{propos\ x |\ x.\ x \in set\ xs\}$
proof (*cases xs*)
  case *Nil*
    thus *?thesis* **by** *simp*
next
  case (*Cons x xs*)
    thus *?thesis*
      using *propos-foldl*[*of LTLOr x*] **by** *auto*
qed

lemma *Or-semantics*:
  $w \models Or\ xs = (\exists x \in set\ xs.\ w \models x)$
proof –
  have *Or-propos'*: $\bigwedge x.\ x \in set\ xs \Longrightarrow propos\ x \subseteq propos\ (Or\ xs)$
    using *Or-propos* **by** *blast*

  have $w \models Or\ xs = \{\chi.\ \chi \in propos\ (Or\ xs) \wedge w \models \chi\} \models_P (Or\ xs)$
   using *ltl-models-equiv-prop-entailment LTL-prop-entailment-restrict-to-propos*
by *blast*
  also

have ... = ($\exists x \in$ *set xs*. $\{\chi.\ \chi \in$ *propos* (*Or xs*) $\wedge\ w \models \chi\} \models_P x$)
using *Or-prop-entailment* **by** *auto*
also
have ... = ($\exists x \in$ *set xs*. $\{\chi.\ \chi \in$ *propos* $x \wedge w \models \chi\} \models_P x$)
using *LTL-prop-entailment-restrict-to-propos Or-propos'* **by** *blast*
also
have ... = ($\exists x \in$ *set xs*. $w \models x$)
using *ltl-models-equiv-prop-entailment LTL-prop-entailment-restrict-to-propos*
by *blast*
finally
show *?thesis* **.**
qed

lemma *Or-append-syntactic*:
xs $\neq$ [] $\Longrightarrow$ *Or* (*xs* @ *ys*) = *Or* ((*Or xs*)#*ys*)
by (*induction xs rule*: *list-nonempty-induct*) *simp+*

lemma *Or-append-S*:
$S \models_P$ *Or* (*xs* @ *ys*) = $S \models_P$ *Or xs or Or ys*
using *Or-prop-entailment*[*of S*] **by** *auto*

lemma *Or-append*:
Or (*xs* @ *ys*) $\equiv_P$ *Or xs or Or ys*
unfolding *ltl-prop-equiv-def* **using** *Or-append-S* **by** *blast*

10.6.4 $eval_G$

fun $eval_G$
where
$eval_G$ *S* (φ *and* ψ) = $eval_G$ *S* φ *and* $eval_G$ *S* ψ
| $eval_G$ *S* (φ *or* ψ) = $eval_G$ *S* φ *or* $eval_G$ *S* ψ
| $eval_G$ *S* (*G* φ) = (*if G* $\varphi \in S$ *then true else false*)
| $eval_G$ *S* $\varphi = \varphi$

— Syntactic Properties

lemma $eval_G$*-And-map*:
$eval_G$ *S* (*And xs*) = *And* (*map* ($eval_G$ *S*) *xs*)
proof (*induction xs rule*: *rev-induct*)
case (*snoc x xs*)
thus *?case*
by (*cases xs*) *simp+*
qed *simp*

lemma $eval_G$-*Or-map*:
$eval_G\ S\ (Or\ xs) = Or\ (map\ (eval_G\ S)\ xs)$
proof (*induction xs rule*: *rev-induct*)
case (*snoc x xs*)
thus *?case*
by (*cases xs*) *simp+*
qed *simp*

lemma $eval_G$-*G-nested*:
$\mathbf{G}\ (eval_G\ \mathcal{G}\ \varphi) \subseteq \mathbf{G}\ \varphi$
by (*induction* φ) (*simp-all*, *blast+*)

lemma $eval_G$-*subst*:
$eval_G\ S\ \varphi = subst\ \varphi\ (\lambda\chi.\ Some\ (eval_G\ S\ \chi))$
by (*induction* φ) *simp-all*

— Semantic Properties

lemma $eval_G$-*prop-entailment*:
$S \models_P eval_G\ S\ \varphi \longleftrightarrow S \models_P \varphi$
by (*induction* φ, *auto*)

lemma $eval_G$-*respectfulness*:
$\varphi \longrightarrow_P \psi \Longrightarrow eval_G\ S\ \varphi \longrightarrow_P eval_G\ S\ \psi$
$\varphi \equiv_P \psi \Longrightarrow eval_G\ S\ \varphi \equiv_P eval_G\ S\ \psi$
using *subst-respects-ltl-prop-entailment* $eval_G$-*subst* **by** *metis+*

lemma $eval_G$-*respectfulness-generalized*:
$(\bigwedge\mathcal{A}.\ (\bigwedge x.\ x \in S \Longrightarrow \mathcal{A} \models_P x) \Longrightarrow \mathcal{A} \models_P y) \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow \mathcal{A} \models_P eval_G\ P\ x) \Longrightarrow \mathcal{A} \models_P eval_G\ P\ y$
using *subst-respects-ltl-prop-entailment-generalized*[*of* $S\ y\ \mathcal{A}$] $eval_G$-*subst*[*of* P] **by** *metis*

lift-definition $eval_G$-*abs* :: $'a\ ltl\ set \Rightarrow 'a\ ltl_P \Rightarrow 'a\ ltl_P$ (‹↑$eval_G$›) **is** $eval_G$
by (*insert* $eval_G$-*respectfulness*(*2*))

10.7 Finite Quotient Set

If we restrict formulas to a finite set of propositions, the set of quotients of these is finite

lemma *Rep-Abs-prop-entailment*[*simp*]:
$A \models_P Rep\ (Abs\ \varphi) = A \models_P \varphi$
using *Quotient3-ltl-prop-equiv-quotient*[*THEN rep-abs-rsp*]

by (*auto simp add*: *ltl-prop-equiv-def*)

fun *sat-models* :: $'a$ *ltl-prop-equiv-quotient* $\Rightarrow$ $'a$ *ltl set set*
where
sat-models a = $\{A.\ A \models_P Rep(a)\}$

lemma *sat-models-invariant*:
$A \in$ *sat-models* (*Abs* φ) = $A \models_P \varphi$
using *Rep-Abs-prop-entailment* **by** *auto*

lemma *sat-models-inj*:
inj sat-models
using *Quotient3-ltl-prop-equiv-quotient*[*THEN Quotient3-rel-rep*]
by (*auto simp add*: *ltl-prop-equiv-def inj-on-def*)

lemma *sat-models-finite-image*:
assumes *finite P*
shows *finite* (*sat-models* ‘ $\{Abs\ \varphi \mid \varphi.\ nested\text{-}propos\ \varphi \subseteq P\}$)
proof −
have *sat-models* (*Abs* φ) = $\{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\}$ (**is** *?lhs* = *?rhs*)
if *nested-propos* $\varphi \subseteq P$ **for** φ
proof
have $\bigwedge A\ B.\ A \in$ *sat-models* (*Abs* φ) $\Longrightarrow A \cup B \in$ *sat-models* (*Abs* φ)
unfolding *sat-models-invariant* **by** *blast*
moreover
have $\{A \mid A.\ A \subseteq P \wedge A \models_P \varphi\} \subseteq$ *sat-models* (*Abs* φ)
using *sat-models-invariant* **by** *fast*
ultimately
show *?rhs* $\subseteq$ *?lhs*
by *blast*
next
have *propos* $\varphi \subseteq P$
using *that propos-subset* **by** *blast*
have $A \in \{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\}$
if $A \in$ *sat-models* (*Abs* φ) **for** A
proof (*standard*, *goal-cases prems*)
case *prems*
then have $A \models_P \varphi$
using *that sat-models-invariant* **by** *blast*
then obtain $C\ D$ **where** $C = (A \cap P)$ **and** $D = A - P$ **and** $A = C \cup D$
by *blast*
then have $C \models_P \varphi$ **and** $C \subseteq P$ **and** $D \subseteq UNIV - P$

using ‹$A \models_P \varphi$› *LTL-prop-entailment-restrict-to-propos* ‹*propos* $\varphi \subseteq P$› **by** *blast+*
then have $C \cup D \in \{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\}$
by *blast*
thus *?case*
using ‹$A = C \cup D$› **by** *simp*
qed
thus $?lhs \subseteq ?rhs$
by *blast*
qed
hence *Equal*: $\{sat\text{-}models\ (Abs\ \varphi) \mid \varphi.\ nested\text{-}propos\ \varphi \subseteq P\} = \{\{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\} \mid \varphi.\ nested\text{-}propos\ \varphi \subseteq P\}$
by (*metis* (*lifting*, *no-types*))

have *Finite*: $finite\ \{\{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\} \mid \varphi.\ nested\text{-}propos\ \varphi \subseteq P\}$
proof −
let $?map = \lambda P\ S.\ \{A \cup B \mid A\ B.\ A \in S \wedge B \subseteq UNIV - P\}$
obtain S' **where** S'-*def*: $S' = \{\{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\} \mid \varphi.\ nested\text{-}propos\ \varphi \subseteq P\}$
by *blast*
obtain S **where** S-*def*: $S = \{\{A \mid A.\ A \subseteq P \wedge A \models_P \varphi\} \mid \varphi.\ nested\text{-}propos\ \varphi \subseteq P\}$
by *blast*

— Prove S and ?map applied to it is finite
hence $S \subseteq Pow\ (Pow\ P)$
by *blast*
hence *finite* S
using ‹*finite* P› *finite-Pow-iff infinite-super* **by** *fast*
hence $finite\ \{?map\ P\ A \mid A.\ A \in S\}$
by *fastforce*

— Prove that S’ can be embedded into S using ?map

have $S' \subseteq \{?map\ P\ A \mid A.\ A \in S\}$
proof
fix A
assume $A \in S'$
then obtain φ **where** $nested\text{-}propos\ \varphi \subseteq P$
and $A = \{A \cup B \mid A\ B.\ A \subseteq P \wedge A \models_P \varphi \wedge B \subseteq UNIV - P\}$
using S'-*def* **by** *blast*
then have $?map\ P\ \{A \mid A.\ A \subseteq P \wedge A \models_P \varphi\} = A$

by *simp*
moreover
have $\{A \mid A.\ A \subseteq P \wedge A \models_P \varphi\} \in S$
using ‹*nested-propos* $\varphi \subseteq P$› *S-def* **by** *auto*
ultimately
show $A \in \{?map\ P\ A \mid A.\ A \in S\}$
by *blast*
qed

show *?thesis*
using *rev-finite-subset*[*OF* ‹*finite* $\{?map\ P\ A \mid A.\ A \in S\}$› ‹$S' \subseteq \{?map\ P\ A \mid A.\ A \in S\}$›]
unfolding *S′-def* **.**
qed

have *Finite2*: *finite* $\{$*sat-models* $(Abs\ \varphi) \mid \varphi.$ *nested-propos* $\varphi \subseteq P\}$
unfolding *Equal* **using** *Finite* **by** *blast*
have *Equal2*: *sat-models* ‘ $\{Abs\ \varphi \mid \varphi.$ *nested-propos* $\varphi \subseteq P\} = \{$*sat-models* $(Abs\ \varphi) \mid \varphi.$ *nested-propos* $\varphi \subseteq P\}$
by *blast*

show *?thesis*
unfolding *Equal2* **using** *Finite2* **by** *blast*
qed

lemma *ltl-prop-equiv-quotient-restricted-to-P-finite*:
assumes *finite P*
shows *finite* $\{Abs\ \varphi \mid \varphi.$ *nested-propos* $\varphi \subseteq P\}$
proof −
have *inj-on sat-models* $\{Abs\ \varphi \mid \varphi.$ *nested-propos* $\varphi \subseteq P\}$
using *sat-models-inj subset-inj-on* **by** *auto*
thus *?thesis*
using *finite-imageD*[*OF sat-models-finite-image*[*OF assms*]] **by** *fast*
qed

locale *lift-ltl-transformer* =
fixes
$f :: 'a\ ltl \Rightarrow 'b \Rightarrow 'a\ ltl$
assumes
respectfulness: $\varphi \equiv_P \psi \Longrightarrow f\ \varphi\ \nu \equiv_P f\ \psi\ \nu$
assumes
nested-propos-bounded: *nested-propos* $(f\ \varphi\ \nu) \subseteq$ *nested-propos* φ
begin

lift-definition *f-abs* :: *'a ltl*$_P$ ⇒ *'b* ⇒ *'a ltl*$_P$ **is** *f*
  using *respectfulness* **.**

lift-definition *f-foldl-abs* :: *'a ltl*$_P$ ⇒ *'b list* ⇒ *'a ltl*$_P$ **is** *foldl f*
proof −
  fix φ ψ :: *'a ltl* **fix** *w* :: *'b list* **assume** $\varphi \equiv_P \psi$
  thus *foldl f* φ *w* $\equiv_P$ *foldl f* ψ *w*
    using *respectfulness* **by** (*induction w arbitrary*: φ ψ) *simp+*
qed

lemma *f-foldl-abs-alt-def*:
  f-foldl-abs (*Abs* φ) *w* = *foldl f-abs* (*Abs* φ) *w*
   by (*induction w rule*: *rev-induct*) (*unfold f-foldl-abs.abs-eq foldl.simps foldl-append*, (*metis f-abs.abs-eq*)+)

definition *abs-reach* :: *'a ltl-prop-equiv-quotient* ⇒ *'a ltl-prop-equiv-quotient set*
where
  abs-reach Φ = {*foldl f-abs* Φ *w* |*w*. *True*}

lemma *finite-abs-reach*:
  finite (*abs-reach* (*Abs* φ))
proof −
  {
    fix *w*
    have *nested-propos* (*foldl f* φ *w*) ⊆ *nested-propos* φ
     by (*induction w arbitrary*: φ) (*simp*, *metis foldl-Cons nested-propos-bounded subset-trans*)
  }
  hence *abs-reach* (*Abs* φ) ⊆ {*Abs* χ | χ. *nested-propos* χ ⊆ *nested-propos* φ}
     unfolding *abs-reach-def f-foldl-abs-alt-def*[*symmetric*] *f-foldl-abs.abs-eq* **by** *blast*
  thus *?thesis*
    using *ltl-prop-equiv-quotient-restricted-to-P-finite finite-propos*
    by (*blast dest*: *finite-subset*)
qed

end

end

11 af - Unfolding Functions

theory *af*
  imports *Main LTL-FGXU Auxiliary/List2*
begin

11.1 af

fun *af-letter* :: $'a\ ltl \Rightarrow 'a\ set \Rightarrow 'a\ ltl$
where
  af-letter true ν = *true*
| *af-letter false* ν = *false*
| *af-letter* $p(a)$ ν = (*if* $a \in \nu$ *then true else false*)
| *af-letter* $(np(a))$ ν = (*if* $a \notin \nu$ *then true else false*)
| *af-letter* (φ *and* ψ) ν = (*af-letter* φ ν) *and* (*af-letter* ψ ν)
| *af-letter* (φ *or* ψ) ν = (*af-letter* φ ν) *or* (*af-letter* ψ ν)
| *af-letter* (X φ) ν = φ
| *af-letter* (G φ) ν = (G φ) *and* (*af-letter* φ ν)
| *af-letter* (F φ) ν = (F φ) *or* (*af-letter* φ ν)
| *af-letter* (φ U ψ) ν = (φ U ψ *and* (*af-letter* φ ν)) *or* (*af-letter* ψ ν)

abbreviation *af* :: $'a\ ltl \Rightarrow 'a\ set\ list \Rightarrow 'a\ ltl$ (‹*af*›)
where
  af φ w $\equiv$ *foldl af-letter* φ w

lemma *af-decompose*:
  af (φ *and* ψ) w = (*af* φ w) *and* (*af* ψ w)
  af (φ *or* ψ) w = (*af* φ w) *or* (*af* ψ w)
  by (*induction w rule*: *rev-induct*) *simp-all*

lemma *af-simps*[*simp*]:
  af true w = *true*
  af false w = *false*
  af (X φ) ($x\#xs$) = *af* φ (xs)
  by (*induction w*) *simp-all*

lemma *af-F*:
  af (F φ) w = *Or* (F φ # *map* (*af* φ) (*suffixes w*))
proof (*induction w*)
  case (*Cons x xs*)
    have *af* (F φ) (x # xs) = *af* (*af-letter* (F φ) x) xs
      by *simp*
    also
    have ... = (*af* (F φ) xs) *or* (*af* (*af-letter* (φ) x) xs)

unfolding *af-decompose*[*symmetric*] **by** *simp*
finally
show *?case* **using** *Cons Or-append-syntactic* **by** *force*
qed *simp*

lemma *af-G*:
$af\ (G\ \varphi)\ w = And\ (G\ \varphi\ \#\ map\ (af\ \varphi)\ (suffixes\ w))$
proof (*induction w*)
case (*Cons x xs*)
have $af\ (G\ \varphi)\ (x\ \#\ xs) = af\ (af\text{-}letter\ (G\ \varphi)\ x)\ xs$
by *simp*
also
have $\ldots = (af\ (G\ \varphi)\ xs)\ and\ (af\ (af\text{-}letter\ (\varphi)\ x)\ xs)$
unfolding *af-decompose*[*symmetric*] **by** *simp*
finally
show *?case* **using** *Cons Or-append-syntactic* **by** *force*
qed *simp*

lemma *af-U*:
$af\ (\varphi\ U\ \psi)\ (x\#xs) = (af\ (\varphi\ U\ \psi)\ xs\ and\ af\ \varphi\ (x\#xs))\ or\ af\ \psi\ (x\#xs)$
by (*induction xs*) (*simp add*: *af-decompose*)+

lemma *af-respectfulness*:
$\varphi \longrightarrow_P \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \longrightarrow_P af\text{-}letter\ \psi\ \nu$
$\varphi \equiv_P \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \equiv_P af\text{-}letter\ \psi\ \nu$
proof −
{
fix φ
have $af\text{-}letter\ \varphi\ \nu = subst\ \varphi\ (\lambda\chi.\ Some\ (af\text{-}letter\ \chi\ \nu))$
by (*induction* φ) *auto*
}
thus $\varphi \longrightarrow_P \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \longrightarrow_P af\text{-}letter\ \psi\ \nu$
and $\varphi \equiv_P \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \equiv_P af\text{-}letter\ \psi\ \nu$
using *subst-respects-ltl-prop-entailment* **by** *metis*+
qed

lemma *af-respectfulness'*:
$\varphi \longrightarrow_P \psi \Longrightarrow af\ \varphi\ w \longrightarrow_P af\ \psi\ w$
$\varphi \equiv_P \psi \Longrightarrow af\ \varphi\ w \equiv_P af\ \psi\ w$
by (*induction w arbitrary*: $\varphi\ \psi$) (*insert af-respectfulness*, *fastforce*+)

lemma *af-nested-propos*:
$nested\text{-}propos\ (af\text{-}letter\ \varphi\ \nu) \subseteq nested\text{-}propos\ \varphi$
by (*induction* φ) *auto*

11.2 af_G

fun *af-G-letter* :: $'a\ ltl \Rightarrow 'a\ set \Rightarrow 'a\ ltl$
where
$af\text{-}G\text{-}letter\ true\ \nu = true$
$|\ af\text{-}G\text{-}letter\ false\ \nu = false$
$|\ af\text{-}G\text{-}letter\ p(a)\ \nu = (if\ a \in \nu\ then\ true\ else\ false)$
$|\ af\text{-}G\text{-}letter\ (np(a))\ \nu\ = (if\ a \notin \nu\ then\ true\ else\ false)$
$|\ af\text{-}G\text{-}letter\ (\varphi\ and\ \psi)\ \nu = (af\text{-}G\text{-}letter\ \varphi\ \nu)\ and\ (af\text{-}G\text{-}letter\ \psi\ \nu)$
$|\ af\text{-}G\text{-}letter\ (\varphi\ or\ \psi)\ \nu = (af\text{-}G\text{-}letter\ \varphi\ \nu)\ or\ (af\text{-}G\text{-}letter\ \psi\ \nu)$
$|\ af\text{-}G\text{-}letter\ (X\ \varphi)\ \nu = \varphi$
$|\ af\text{-}G\text{-}letter\ (G\ \varphi)\ \nu = (G\ \varphi)$
$|\ af\text{-}G\text{-}letter\ (F\ \varphi)\ \nu = (F\ \varphi)\ or\ (af\text{-}G\text{-}letter\ \varphi\ \nu)$
$|\ af\text{-}G\text{-}letter\ (\varphi\ U\ \psi)\ \nu = (\varphi\ U\ \psi\ and\ (af\text{-}G\text{-}letter\ \varphi\ \nu))\ or\ (af\text{-}G\text{-}letter\ \psi$
$\nu)$

abbreviation af_G :: $'a\ ltl \Rightarrow 'a\ set\ list \Rightarrow 'a\ ltl$
where
$af_G\ \varphi\ w \equiv (foldl\ af\text{-}G\text{-}letter\ \varphi\ w)$

lemma af_G*-decompose*:
$af_G\ (\varphi\ and\ \psi)\ w = (af_G\ \varphi\ w)\ and\ (af_G\ \psi\ w)$
$af_G\ (\varphi\ or\ \psi)\ w = (af_G\ \varphi\ w)\ or\ (af_G\ \psi\ w)$
by (*induction w rule*: *rev-induct*) *simp-all*

lemma af_G*-simps*[*simp*]:
$af_G\ true\ w = true$
$af_G\ false\ w = false$
$af_G\ (G\ \varphi)\ w = G\ \varphi$
$af_G\ (X\ \varphi)\ (x\#xs) = af_G\ \varphi\ (xs)$
by (*induction w*) *simp-all*

lemma af_G*-F*:
$af_G\ (F\ \varphi)\ w = Or\ (F\ \varphi\ \#\ map\ (af_G\ \varphi)\ (suffixes\ w))$
proof (*induction w*)
case (*Cons x xs*)
have $af_G\ (F\ \varphi)\ (x\ \#\ xs) = af_G\ (af\text{-}G\text{-}letter\ (F\ \varphi)\ x)\ xs$
by *simp*
also
have $\ldots = (af_G\ (F\ \varphi)\ xs)\ or\ (af_G\ (af\text{-}G\text{-}letter\ (\varphi)\ x)\ xs)$
unfolding af_G*-decompose*[*symmetric*] **by** *simp*
finally
show *?case* **using** *Cons Or-append-syntactic* **by** *force*
qed *simp*

lemma af_G-U:
af_G (φ U ψ) ($x\#xs$) = (af_G (φ U ψ) xs *and* af_G φ ($x\#xs$)) *or* af_G ψ ($x\#xs$)
by (*simp add*: af_G-*decompose*)

lemma af_G-*subsequence*-U:
af_G (φ U ψ) (w [$0 \to Suc\ n$]) = (af_G (φ U ψ) (w [$1 \to Suc\ n$]) *and* af_G φ (w [$0 \to Suc\ n$])) *or* af_G ψ (w [$0 \to Suc\ n$])
proof –
have $\bigwedge n.\ w\ [0 \to Suc\ n] = w\ 0\ \#\ w\ [1 \to Suc\ n]$
using *subsequence-append*[*of w 1*] **by** (*simp add*: *subsequence-def upt-conv-Cons*)

thus *?thesis*
using af_G-U **by** *metis*
qed

lemma *af-G-respectfulness*:
$\varphi \longrightarrow_P \psi \Longrightarrow$ *af-G-letter* φ ν $\longrightarrow_P$ *af-G-letter* ψ ν
$\varphi \equiv_P \psi \Longrightarrow$ *af-G-letter* φ ν $\equiv_P$ *af-G-letter* ψ ν
proof –
{
fix φ
have *af-G-letter* φ ν = *subst* φ ($\lambda\chi$. *Some* (*af-G-letter* χ ν))
by (*induction* φ) *auto*
}
thus $\varphi \longrightarrow_P \psi \Longrightarrow$ *af-G-letter* φ ν $\longrightarrow_P$ *af-G-letter* ψ ν
and $\varphi \equiv_P \psi \Longrightarrow$ *af-G-letter* φ ν $\equiv_P$ *af-G-letter* ψ ν
using *subst-respects-ltl-prop-entailment* **by** *metis*+
qed

lemma *af-G-respectfulness*′:
$\varphi \longrightarrow_P \psi \Longrightarrow af_G\ \varphi\ w \longrightarrow_P af_G\ \psi\ w$
$\varphi \equiv_P \psi \Longrightarrow af_G\ \varphi\ w \equiv_P af_G\ \psi\ w$
by (*induction w arbitrary*: φ ψ) (*insert af-G-respectfulness*, *fastforce*+)

lemma *af-G-nested-propos*:
nested-propos (*af-G-letter* φ ν) $\subseteq$ *nested-propos* φ
by (*induction* φ) *auto*

lemma *af-G-letter-sat-core*:
Only-G $\mathcal{G} \Longrightarrow \mathcal{G} \models_P \varphi \Longrightarrow \mathcal{G} \models_P$ *af-G-letter* φ ν
by (*induction* φ) (*simp-all*, *blast*+)

lemma *af*$_G$*-sat-core*:
Only-G $\mathcal{G} \Longrightarrow \mathcal{G} \models_P \varphi \Longrightarrow \mathcal{G} \models_P$ *af*$_G$ φ *w*
using *af-G-letter-sat-core* **by** (*induction w rule*: *rev-induct*) (*simp-all*, *blast*)

lemma *af*$_G$*-sat-core-generalized*:
Only-G $\mathcal{G} \Longrightarrow i \leq j \Longrightarrow \mathcal{G} \models_P$ *af*$_G$ φ (w $[0 \rightarrow i]$) $\Longrightarrow \mathcal{G} \models_P$ *af*$_G$ φ (w $[0 \rightarrow j]$)
by (*metis af*$_G$*-sat-core foldl-append subsequence-append le-add-diff-inverse*)

lemma *af*$_G$*-eval*$_G$:
Only-G $\mathcal{G} \Longrightarrow \mathcal{G} \models_P$ *af*$_G$ (*eval*$_G$ $\mathcal{G}$ φ) $w \longleftrightarrow \mathcal{G} \models_P$ *eval*$_G$ $\mathcal{G}$ (*af*$_G$ φ w)
by (*induction* φ) (*simp-all add*: *eval*$_G$*-prop-entailment af*$_G$*-decompose*)

lemma *af*$_G$*-keeps-F-and-S*:
assumes $ys \neq []$
assumes $S \models_P$ *af*$_G$ φ *ys*
shows $S \models_P$ *af*$_G$ (F φ) (*xs* @ *ys*)
proof −
have *af*$_G$ φ *ys* $\in$ *set* (*map* (*af*$_G$ φ) (*suffixes* (*xs* @ *ys*)))
using *assms*(*1*) **unfolding** *suffixes-append map-append*
by (*induction ys rule*: *List.list-nonempty-induct*) *auto*
thus *?thesis*
unfolding *af*$_G$*-F Or-prop-entailment* **using** *assms*(*2*) **by** *force*
qed

11.3 G-Subformulae Simplification

lemma *G-af-simp*[*simp*]:
G (*af* φ *w*) = **G** φ
proof −
{ fix φ ν **have G** (*af-letter* φ ν) = **G** φ **by** (*induction* φ) *auto* **}**
thus *?thesis*
by (*induction w arbitrary*: φ *rule*: *rev-induct*) *fastforce+*
qed

lemma *G-af*$_G$*-simp*[*simp*]:
G (*af*$_G$ φ *w*) = **G** φ
proof −
{ fix φ ν **have G** (*af-G-letter* φ ν) = **G** φ **by** (*induction* φ) *auto* **}**
thus *?thesis*
by (*induction w arbitrary*: φ *rule*: *rev-induct*) *fastforce+*
qed

11.4 Relation between af and af_G

lemma *af-G-letter-free-F*:
  G φ = {} ⟹ **G** (*af-letter* φ ν) = {}
  G φ = {} ⟹ **G** (*af-G-letter* φ ν) = {}
  by (*induction* φ) *auto*

lemma *af-G-free*:
  assumes G φ = {}
  shows *af* φ *w* = af_G φ *w*
  using *assms*
proof (*induction w arbitrary*: φ)
  case (*Cons x xs*)
    hence *af* (*af-letter* φ *x*) *xs* = af_G (*af-letter* φ *x*) *xs*
      using *af-G-letter-free-F*[*OF Cons.prems*, *THEN Cons.IH*] **by** *blast*
    moreover
    have *af-letter* φ *x* = *af-G-letter* φ *x*
      using *Cons.prems* **by** (*induction* φ) *auto*
    ultimately
    show *?case*
      by *simp*
qed *simp*

lemma *af-equals-af$_G$-base-cases*:
  af true w = af_G *true w*
  af false w = af_G *false w*
  af p(*a*) *w* = af_G *p*(*a*) *w*
  af (*np*(*a*)) *w* = af_G (*np*(*a*)) *w*
  by (*auto intro*: *af-G-free*)

lemma *af-implies-af$_G$*:
  S $\models_P$ *af* φ *w* ⟹ *S* $\models_P$ af_G φ *w*
proof (*induction w arbitrary*: *S rule*: *rev-induct*)
  case (*snoc x xs*)
    hence *S* $\models_P$ *af-letter* (*af* φ *xs*) *x*
      by *simp*
    hence *S* $\models_P$ *af-letter* (af_G φ *xs*) *x*
      using *af-respectfulness*(*1*) *snoc.IH* **unfolding** *ltl-prop-implies-def* **by**
blast
    moreover
    {
      fix φ
      have $\bigwedge\nu$. *S* $\models_P$ *af-letter* φ ν ⟹ *S* $\models_P$ *af-G-letter* φ ν
        by (*induction* φ) *auto*

}
ultimately
show *?case*
using *snoc.prems foldl-append* **by** *simp*
qed *simp*

lemma *af-implies-af$_G$-2*:
$w \models af\ \varphi\ xs \Longrightarrow w \models af_G\ \varphi\ xs$
by (*metis ltl-prop-implication-implies-ltl-implication af-implies-af$_G$ ltl-prop-implies-def*)

lemma *af$_G$-implies-af-eval$_G$'*:
assumes $S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ w)$
assumes $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow S \models_P G\ \psi$
assumes $\bigwedge\psi\ i.\ G\ \psi \in \mathcal{G} \Longrightarrow i < length\ w \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi$
$(drop\ i\ w))$
shows $S \models_P af\ \varphi\ w$
using *assms*
proof (*induction* φ *arbitrary*: *w*)
case (*LTLGlobal* φ)
hence $G\ \varphi \in \mathcal{G}$
unfolding *af$_G$-simps eval$_G$.simps* **by** (*cases* $G\ \varphi \in \mathcal{G}$) *simp+*
hence $S \models_P G\ \varphi$
using *LTLGlobal* **by** *simp*
moreover
{
fix *x*
assume $x \in set\ (map\ (af\ \varphi)\ (suffixes\ w))$
then obtain w' **where** $x = af\ \varphi\ w'$ **and** $w' \in set\ (suffixes\ w)$
by *auto*
then obtain *i* **where** $w' = drop\ i\ w$ **and** $i < length\ w$
by (*auto simp add*: *suffixes-alt-def subsequence-def*)
hence $S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ w')$
using *LTLGlobal.prems* ‹$G\ \varphi \in \mathcal{G}$› **by** *simp*
hence $S \models_P x$
using *LTLGlobal(1)*[*OF* ‹$S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ w')$›] *LTLGlobal(3−4)*
drop-drop
unfolding ‹$x = af\ \varphi\ w'$› ‹$w' = drop\ i\ w$› **by** *simp*
}
ultimately
show *?case*
unfolding *af-G eval$_G$-And-map And-prop-entailment* **by** *simp*
next
case (*LTLFinal* φ)
then obtain *x* **where** *x-def*: $x \in set\ (F\ \varphi\ \#\ map\ (eval_G\ \mathcal{G}\ o\ af_G\ \varphi)$

(*suffixes w*))
and $S \models_P x$
unfolding *Or-prop-entailment* af_G-*F* $eval_G$-*Or-map* **by** *force*
hence $\exists\, y \in set\ (F\ \varphi\ \#\ map\ (af\ \varphi)\ (suffixes\ w)).\ S \models_P y$
proof (*cases* $x \neq F\ \varphi$)
case *True*
then obtain w' **where** $S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ w')$ **and** $w' \in set$ (*suffixes w*)
using *x-def* ‹$S \models_P x$› **by** *auto*
hence $\bigwedge\psi\ i.\ G\ \psi \in \mathcal{G} \Longrightarrow i < length\ w' \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi$ $(drop\ i\ w'))$
using *LTLFinal.prems* **by** (*auto simp add*: *suffixes-alt-def subsequence-def*)
moreover
have $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (G\ \psi)$
using *LTLFinal* **by** *simp*
ultimately
have $S \models_P af\ \varphi\ w'$
using *LTLFinal.IH*[*OF* ‹$S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ w')$›] **using** *assms*(*2*) **by** *blast*
thus *?thesis*
using ‹$w' \in set\ (suffixes\ w)$› **by** *auto*
qed *simp*
thus *?case*
unfolding *af-F Or-prop-entailment* $eval_G$-*Or-map* **by** *simp*
next
case (*LTLNext* φ)
thus *?case*
proof (*cases w*)
case (*Cons x xs*)
{
fix $\psi\ i$
assume $G\ \psi \in \mathcal{G}$ **and** $Suc\ i < length\ (x\#xs)$
hence $S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi\ (drop\ (Suc\ i)\ (x\#xs)))$
using *LTLNext.prems* **unfolding** *Cons* **by** *blast*
hence $S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi\ (drop\ i\ xs))$
by *simp*
}
hence $\bigwedge\psi\ i.\ G\ \psi \in \mathcal{G} \Longrightarrow i < length\ xs \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi$ $(drop\ i\ xs))$
by *simp*
thus *?thesis*
using *LTLNext Cons* **by** *simp*
qed *simp*

next
  case (*LTLUntil* φ ψ)
    thus *?case*
    proof (*induction w*)
      case (*Cons x xs*)
        {
          assume $S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi\ (x\ \#\ xs))$
          moreover
          have $\bigwedge\psi\ i.\ G\ \psi \in \mathcal{G} \Longrightarrow i < length\ (x\#xs) \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi\ (drop\ i\ (x\#xs)))$
            using *Cons* **by** *simp*
          ultimately
          have $S \models_P af\ \psi\ (x\ \#\ xs)$
            using *Cons.prems* **by** *blast*
          hence *?case*
            unfolding *af-U* **by** *simp*
        }
        moreover
        {
          assume $S \models_P eval_G\ \mathcal{G}\ (af_G\ (\varphi\ U\ \psi)\ xs)$ **and** $S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ (x\ \#\ xs))$
          moreover
          have $\bigwedge\psi\ i.\ G\ \psi \in \mathcal{G} \Longrightarrow i < length\ (x\#xs) \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi\ (drop\ i\ (x\#xs)))$
            using *Cons* **by** *simp*
          ultimately
          have $S \models_P af\ \varphi\ (x\ \#\ xs)$ **and** $S \models_P af\ (\varphi\ U\ \psi)\ xs$
            using *Cons* **by** (*blast*, *force*)
          hence *?case*
            unfolding *af-U* **by** *simp*
        }
        ultimately
        show *?case*
          using *Cons*(*4*) **unfolding** af_G*-U* **by** *auto*
    qed *simp*
next
  case (*LTLProp a*)
    thus *?case*
    proof (*cases w*)
      case (*Cons x xs*)
        thus *?thesis*
          using *LTLProp* **by** (*cases* $a \in x$) *simp+*
    qed *simp*
next

case (*LTLPropNeg a*)
thus *?case*
proof (*cases w*)
case (*Cons x xs*)
thus *?thesis*
using *LTLPropNeg* **by** (*cases* $a \in x$) *simp+*
qed *simp*
qed (*unfold af-equals-af*$_G$*-base-cases af*$_G$*-decompose af-decompose*, *auto*)

lemma *af*$_G$*-implies-af-eval*$_G$:
assumes $S \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ (w\ [0{\to}j]))$
assumes $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow S \models_P G\ \psi$
assumes $\bigwedge\psi\ i.\ G\ \psi \in \mathcal{G} \Longrightarrow i \le j \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (af_G\ \psi\ (w\ [i \to j]))$
shows $S \models_P af\ \varphi\ (w\ [0{\to}j])$
using *af*$_G$*-implies-af-eval*$_G$*'*[*OF assms*(*1*−*2*), *unfolded subsequence-length subsequence-drop*] *assms*(*3*) **by** *force*

11.5 Continuation

lemma *af-ltl-continuation*:
$(w \frown w') \models \varphi \longleftrightarrow w' \models af\ \varphi\ w$
proof (*induction w arbitrary*: $\varphi\ w'$)
case (*Cons x xs*)
have $((x \# xs) \frown w')\ 0 = x$
unfolding *conc-def nth-Cons-0* **by** *simp*
moreover
have *suffix 1* $((x \# xs) \frown w') = xs \frown w'$
unfolding *suffix-def conc-def* **by** *fastforce*
moreover
{
fix φ :: *'a ltl*
have $\bigwedge w.\ w \models \varphi \longleftrightarrow$ *suffix 1 w* $\models$ *af-letter* $\varphi\ (w\ 0)$
by (*induction* φ) ((*unfold LTL-F-one-step-unfolding LTL-G-one-step-unfolding LTL-U-one-step-unfolding*)?, *auto*)
}
ultimately
have $((x \# xs) \frown w') \models \varphi \longleftrightarrow (xs \frown w') \models$ *af-letter* $\varphi\ x$
by *metis*
also
have $\ldots \longleftrightarrow w' \models af\ \varphi\ (x\#xs)$
using *Cons.IH* **by** *simp*
finally
show *?case* **.**

qed *simp*

lemma *af-ltl-continuation-suffix*:
$w \models \varphi \longleftrightarrow$ *suffix i w* $\models$ *af* φ $(w[0 \to i])$
using *af-ltl-continuation prefix-suffix subsequence-def* **by** *metis*

lemma *af-G-ltl-continuation*:
$\forall \psi \in \mathbf{G}\ \varphi.\ w' \models \psi = (w \frown w') \models \psi \Longrightarrow (w \frown w') \models \varphi \longleftrightarrow w' \models af_G$
$\varphi\ w$
proof (*induction w arbitrary*: $w'\ \varphi$)
case (*Cons x xs*)
{
fix ψ :: *'a ltl* **fix** $w\ w'\ w''$
assume $w'' \models G\ \psi = ((w\ @\ w') \frown w'') \models G\ \psi$
hence $w'' \models G\ \psi = (w' \frown w'') \models G\ \psi$ **and** $(w' \frown w'') \models G\ \psi =$
$((w\ @\ w') \frown w'') \models G\ \psi$
by (*induction* w' *arbitrary*: w) (*metis LTL-suffix-G suffix-conc-length conc-conc*)+
}
note *G-stable* = *this*
have *A*: $\forall \psi{\in}\mathbf{G}\ (af_G\ \varphi\ [x]).\ w' \models \psi = (xs \frown w') \models \psi$
using *G-stable*(*1*)[*of* w' - $[x]$] *Cons.prems* **unfolding** *G-af*$_G$*-simp conc-conc append.simps* **unfolding** *G-nested-propos-alt-def* **by** *blast*
have *B*: $\forall \psi{\in}\mathbf{G}\ \varphi.\ ([x] \frown xs \frown w') \models \psi = (xs \frown w') \models \psi$
using *G-stable*(*2*)[*of* w' - $[x]$] *Cons.prems* **unfolding** *conc-conc append.simps* **unfolding** *G-nested-propos-alt-def* **by** *blast*
hence $([x] \frown xs \frown w') \models \varphi = (xs \frown w') \models af_G\ \varphi\ [x]$
proof (*induction* φ)
case (*LTLFinal* φ)
thus *?case*
unfolding *LTL-F-one-step-unfolding*
by (*auto simp add*: *suffix-conc-length*[*of* $[x]$, *simplified*])
next
case (*LTLUntil* $\varphi\ \psi$)
thus *?case*
unfolding *LTL-U-one-step-unfolding*
by (*auto simp add*: *suffix-conc-length*[*of* $[x]$, *simplified*])
qed (*auto simp add*: *conc-fst*[*of 0* $[x]$] *suffix-conc-length*[*of* $[x]$, *simplified*])
also
have ... $= w' \models af_G\ \varphi\ (x\ \#\ xs)$
using *Cons.IH*[*of* $af_G\ \varphi\ [x]\ w'$] *A* **by** *simp*
finally
show *?case* **unfolding** *conc-conc*

by *simp*
qed *simp*

lemma af_G*-ltl-continuation-suffix*:
$\forall\psi \in \mathbf{G}\ \varphi.\ w \models \psi = (suffix\ i\ w) \models \psi \Longrightarrow w \models \varphi \longleftrightarrow suffix\ i\ w \models af_G\ \varphi\ (w\ [0 \to i])$
by (*metis af-G-ltl-continuation*[*of* φ *suffix i w*] *prefix-suffix subsequence-def*)

11.6 Eager Unfolding *af* and af_G

fun *Unf* :: $'a\ ltl \Rightarrow\ 'a\ ltl$
where
$Unf\ (F\ \varphi) = F\ \varphi\ or\ Unf\ \varphi$
$|\ Unf\ (G\ \varphi) = G\ \varphi\ and\ Unf\ \varphi$
$|\ Unf\ (\varphi\ U\ \psi) = (\varphi\ U\ \psi\ and\ Unf\ \varphi)\ or\ Unf\ \psi$
$|\ Unf\ (\varphi\ and\ \psi) = Unf\ \varphi\ and\ Unf\ \psi$
$|\ Unf\ (\varphi\ or\ \psi) = Unf\ \varphi\ or\ Unf\ \psi$
$|\ Unf\ \varphi = \varphi$

fun Unf_G :: $'a\ ltl \Rightarrow\ 'a\ ltl$
where
$Unf_G\ (F\ \varphi) = F\ \varphi\ or\ Unf_G\ \varphi$
$|\ Unf_G\ (G\ \varphi) = G\ \varphi$
$|\ Unf_G\ (\varphi\ U\ \psi) = (\varphi\ U\ \psi\ and\ Unf_G\ \varphi)\ or\ Unf_G\ \psi$
$|\ Unf_G\ (\varphi\ and\ \psi) = Unf_G\ \varphi\ and\ Unf_G\ \psi$
$|\ Unf_G\ (\varphi\ or\ \psi) = Unf_G\ \varphi\ or\ Unf_G\ \psi$
$|\ Unf_G\ \varphi = \varphi$

fun *step* :: $'a\ ltl \Rightarrow\ 'a\ set \Rightarrow\ 'a\ ltl$
where
$step\ p(a)\ \nu = (if\ a \in \nu\ then\ true\ else\ false)$
$|\ step\ (np(a))\ \nu = (if\ a \notin \nu\ then\ true\ else\ false)$
$|\ step\ (X\ \varphi)\ \nu = \varphi$
$|\ step\ (\varphi\ and\ \psi)\ \nu = step\ \varphi\ \nu\ and\ step\ \psi\ \nu$
$|\ step\ (\varphi\ or\ \psi)\ \nu = step\ \varphi\ \nu\ or\ step\ \psi\ \nu$
$|\ step\ \varphi\ \nu = \varphi$

fun *af-letter-opt*
where
af-letter-opt $\varphi\ \nu = Unf\ (step\ \varphi\ \nu)$

fun *af-G-letter-opt*
where
af-G-letter-opt $\varphi\ \nu = Unf_G\ (step\ \varphi\ \nu)$

abbreviation *af-opt* :: $'a$ *ltl* ⇒ $'a$ *set list* ⇒ $'a$ *ltl* (‹$af_{\mathfrak{U}}$›)
where
$af_{\mathfrak{U}}$ φ *w* ≡ (*foldl af-letter-opt* φ *w*)

abbreviation *af-G-opt* :: $'a$ *ltl* ⇒ $'a$ *set list* ⇒ $'a$ *ltl* (‹$af_{G\mathfrak{U}}$›)
where
$af_{G\mathfrak{U}}$ φ *w* ≡ (*foldl af-G-letter-opt* φ *w*)

lemma *af-letter-alt-def*:
af-letter φ ν = *step* (*Unf* φ) ν
af-G-letter φ ν = *step* (Unf_G φ) ν
by (*induction* φ) *simp-all*

lemma *af-to-af-opt*:
Unf (*af* φ *w*) = $af_{\mathfrak{U}}$ (*Unf* φ) *w*
Unf_G (af_G φ *w*) = $af_{G\mathfrak{U}}$ (Unf_G φ) *w*
by (*induction w arbitrary*: φ)
(*simp-all add*: *af-letter-alt-def*)

lemma *af-equiv*:
af φ (*w* @ [ν]) = *step* ($af_{\mathfrak{U}}$ (*Unf* φ) *w*) ν
using *af-to-af-opt*(*1*) **by** (*metis af-letter-alt-def*(*1*) *foldl-Cons foldl-Nil foldl-append*)

lemma *af-equiv'*:
af φ (*w* [*0* → *Suc i*]) = *step* ($af_{\mathfrak{U}}$ (*Unf* φ) (*w* [*0* → *i*])) (*w i*)
using *af-equiv* **unfolding** *subsequence-def* **by** *auto*

11.7 Lifted Functions

lemma *respectfulness*:
$\varphi \longrightarrow_P \psi \Longrightarrow$ *af-letter-opt* φ ν $\longrightarrow_P$ *af-letter-opt* ψ ν
$\varphi \equiv_P \psi \Longrightarrow$ *af-letter-opt* φ ν $\equiv_P$ *af-letter-opt* ψ ν
$\varphi \longrightarrow_P \psi \Longrightarrow$ *af-G-letter-opt* φ ν $\longrightarrow_P$ *af-G-letter-opt* ψ ν
$\varphi \equiv_P \psi \Longrightarrow$ *af-G-letter-opt* φ ν $\equiv_P$ *af-G-letter-opt* ψ ν
$\varphi \longrightarrow_P \psi \Longrightarrow$ *step* φ ν $\longrightarrow_P$ *step* ψ ν
$\varphi \equiv_P \psi \Longrightarrow$ *step* φ ν $\equiv_P$ *step* ψ ν
$\varphi \longrightarrow_P \psi \Longrightarrow$ *Unf* φ $\longrightarrow_P$ *Unf* ψ
$\varphi \equiv_P \psi \Longrightarrow$ *Unf* φ $\equiv_P$ *Unf* ψ
$\varphi \longrightarrow_P \psi \Longrightarrow$ Unf_G φ $\longrightarrow_P$ Unf_G ψ
$\varphi \equiv_P \psi \Longrightarrow$ Unf_G φ $\equiv_P$ Unf_G ψ
using *decomposable-function-subst*[*of* $\lambda\chi$. *af-letter-opt* χ ν, *simplified*] *af-letter-opt.simps*

using *decomposable-function-subst*[*of* $\lambda\chi$. *af-G-letter-opt* χ ν, *simplified*] *af-G-letter-opt.simps*
using *decomposable-function-subst*[*of* $\lambda\chi$. *step* χ ν, *simplified*]
using *decomposable-function-subst*[*of Unf*, *simplified*]
using *decomposable-function-subst*[*of* Unf_G, *simplified*]
using *subst-respects-ltl-prop-entailment* **by** *metis+*

lemma *nested-propos*:
nested-propos (*step* φ ν) $\subseteq$ *nested-propos* φ
nested-propos (*Unf* φ) $\subseteq$ *nested-propos* φ
nested-propos (Unf_G φ) $\subseteq$ *nested-propos* φ
nested-propos (*af-letter-opt* φ ν) $\subseteq$ *nested-propos* φ
nested-propos (*af-G-letter-opt* φ ν) $\subseteq$ *nested-propos* φ
by (*induction* φ) *auto*

Lift functions and bind to new names

interpretation *af-abs*: *lift-ltl-transformer af-letter*
using *lift-ltl-transformer-def af-respectfulness af-nested-propos* **by** *blast*

definition *af-letter-abs* (‹↑*af*›)
where
↑*af* ≡ *af-abs.f-abs*

interpretation *af-G-abs*: *lift-ltl-transformer af-G-letter*
using *lift-ltl-transformer-def af-G-respectfulness af-G-nested-propos* **by** *blast*

definition *af-G-letter-abs* (‹$\uparrow af_G$›)
where
$\uparrow af_G$ ≡ *af-G-abs.f-abs*

interpretation *af-abs-opt*: *lift-ltl-transformer af-letter-opt*
using *lift-ltl-transformer-def respectfulness nested-propos* **by** *blast*

definition *af-letter-abs-opt* (‹$\uparrow af_{\mathfrak{U}}$›)
where
$\uparrow af_{\mathfrak{U}}$ ≡ *af-abs-opt.f-abs*

interpretation *af-G-abs-opt*: *lift-ltl-transformer af-G-letter-opt*
using *lift-ltl-transformer-def respectfulness nested-propos* **by** *blast*

definition *af-G-letter-abs-opt* (‹$\uparrow af_{G\mathfrak{U}}$›)
where
$\uparrow af_{G\mathfrak{U}}$ ≡ *af-G-abs-opt.f-abs*

lift-definition *step-abs* :: $'a\ ltl_P \Rightarrow 'a\ set \Rightarrow 'a\ ltl_P$ (‹↑*step*›) **is** *step*
by (*insert respectfulness*)

lift-definition *Unf-abs* :: $'a\ ltl_P \Rightarrow 'a\ ltl_P$ (‹↑*Unf*›) **is** *Unf*
by (*insert respectfulness*)

lift-definition Unf_G-*abs* :: $'a\ ltl_P \Rightarrow 'a\ ltl_P$ (‹↑Unf_G›) **is** Unf_G
by (*insert respectfulness*)

11.7.1 Properties

lemma *af-G-letter-opt-sat-core*:
Only-G $\mathcal{G} \Longrightarrow \mathcal{G} \models_P \varphi \Longrightarrow \mathcal{G} \models_P$ *af-G-letter-opt* $\varphi\ \nu$
by (*induction* φ) *auto*

lemma *af-G-letter-sat-core-lifted*:
Only-G $\mathcal{G} \Longrightarrow \mathcal{G} \models_P$ *Rep* $\varphi \Longrightarrow \mathcal{G} \models_P$ *Rep* (*af-G-letter-abs* $\varphi\ \nu$)
by (*metis af-G-letter-sat-core Quotient-ltl-prop-equiv-quotient*[*THEN Quotient-rep-abs*] *Quotient3-ltl-prop-equiv-quotient*[*THEN Quotient3-abs-rep*] *af-G-abs.f-abs.abs-eq ltl-prop-equiv-def af-G-letter-abs-def*)

lemma *af-G-letter-opt-sat-core-lifted*:
Only-G $\mathcal{G} \Longrightarrow \mathcal{G} \models_P$ *Rep* $\varphi \Longrightarrow \mathcal{G} \models_P$ *Rep* (↑$af_{G\mathfrak{U}}$ $\varphi\ \nu$)
unfolding *af-G-letter-abs-opt-def*
by (*metis af-G-letter-opt-sat-core Quotient-ltl-prop-equiv-quotient*[*THEN Quotient-rep-abs*] *Quotient3-ltl-prop-equiv-quotient*[*THEN Quotient3-abs-rep*] *af-G-abs-opt.f-abs.abs-eq ltl-prop-equiv-def*)

lemma *af-G-letter-abs-opt-split*:
↑Unf_G (↑*step* $\Phi\ \nu$) = ↑$af_{G\mathfrak{U}}$ $\Phi\ \nu$
unfolding *af-G-letter-abs-opt-def step-abs-def comp-def af-G-abs-opt.f-abs-def*

using *map-fun-apply* Unf_G-*abs.abs-eq af-G-letter-opt.simps* **by** *auto*

lemma *af-unfold*:
↑*af* = ($\lambda\varphi\ \nu$. ↑*step* (↑*Unf* φ) ν)
by (*metis Unf-abs-def af-abs.f-abs.abs-eq af-letter-abs-def af-letter-alt-def*(*1*) ltl_P-*abs-rep map-fun-apply step-abs.abs-eq*)

lemma *af-opt-unfold*:
↑$af_{\mathfrak{U}}$ = ($\lambda\varphi\ \nu$. ↑*Unf* (↑*step* $\varphi\ \nu$))
by (*metis* (*no-types*, *lifting*) *Quotient3-abs-rep Quotient3-ltl-prop-equiv-quotient Unf-abs.abs-eq af-abs-opt.f-abs.abs-eq af-letter-abs-opt-def af-letter-opt.elims*

id-apply map-fun-apply step-abs-def)

lemma *af-abs-equiv*:
  foldl ↑*af* ψ (*xs* @ [*x*]) = ↑*step* (*foldl* ↑$af_{\mathfrak{U}}$ (↑*Unf* ψ) *xs*) *x*
  unfolding *af-unfold af-opt-unfold* **by** (*induction xs arbitrary*: *x* ψ *rule*: *rev-induct*) *simp*+

lemma *Rep-Abs-equiv*:
  Rep (*Abs* φ) $\equiv_P$ φ
  using *Rep-Abs-prop-entailment* **unfolding** *ltl-prop-equiv-def* **by** *auto*

lemma *Rep-step*:
  Rep (↑*step* Φ ν) $\equiv_P$ *step* (*Rep* Φ) ν
  by (*metis Quotient3-abs-rep Quotient3-ltl-prop-equiv-quotient ltl-prop-equiv-quotient.abs-eq-iff step-abs.abs-eq*)

lemma *step*-$\mathcal{G}$:
  Only-G $\mathcal{G}$ $\Longrightarrow$ $\mathcal{G}$ $\models_P$ φ $\Longrightarrow$ $\mathcal{G}$ $\models_P$ *step* φ ν
  by (*induction* φ) *auto*

lemma Unf_G-$\mathcal{G}$:
  Only-G $\mathcal{G}$ $\Longrightarrow$ $\mathcal{G}$ $\models_P$ φ $\Longrightarrow$ $\mathcal{G}$ $\models_P$ Unf_G φ
  by (*induction* φ) *auto*

hide-fact (**open**) *respectfulness nested-propos*

end

12 Logical Characterization Theorems

theory *Logical-Characterization*
  imports *Main af Auxiliary/Preliminaries2*
begin

12.1 Eventually True G-Subformulae

fun $\mathcal{G}_{FG}$ :: *'a ltl* ⇒ *'a set word* ⇒ *'a ltl set*
where
  $\mathcal{G}_{FG}$ *true* *w* = {}
| $\mathcal{G}_{FG}$ (*false*) *w* = {}
| $\mathcal{G}_{FG}$ (*p*(*a*)) *w* = {}
| $\mathcal{G}_{FG}$ (*np*(*a*)) *w* = {}
| $\mathcal{G}_{FG}$ (φ_1 *and* φ_2) *w* = $\mathcal{G}_{FG}$ φ_1 *w* ∪ $\mathcal{G}_{FG}$ φ_2 *w*
| $\mathcal{G}_{FG}$ (φ_1 *or* φ_2) *w* = $\mathcal{G}_{FG}$ φ_1 *w* ∪ $\mathcal{G}_{FG}$ φ_2 *w*

| $\mathcal{G}_{FG}$ $(F\ \varphi)$ w = $\mathcal{G}_{FG}$ φ w
| $\mathcal{G}_{FG}$ $(G\ \varphi)$ w = (*if* $w \models F\ G\ \varphi$ *then* $\{G\ \varphi\} \cup \mathcal{G}_{FG}\ \varphi\ w$ *else* $\mathcal{G}_{FG}\ \varphi\ w$)
| $\mathcal{G}_{FG}$ $(X\ \varphi)$ w = $\mathcal{G}_{FG}$ φ w
| $\mathcal{G}_{FG}$ $(\varphi\ U\ \psi)$ w = $\mathcal{G}_{FG}\ \varphi\ w \cup \mathcal{G}_{FG}\ \psi\ w$

lemma $\mathcal{G}_{FG}$*-alt-def*:
$\mathcal{G}_{FG}\ \varphi\ w = \{G\ \psi \mid \psi.\ G\ \psi \in \mathbf{G}\ \varphi \wedge w \models F\ (G\ \psi)\}$
by (*induction* φ *arbitrary*: *w*) (*simp*; *blast*)+

lemma $\mathcal{G}_{FG}$*-Only-G*:
Only-G ($\mathcal{G}_{FG}$ φ w)
by (*induction* φ) *auto*

lemma $\mathcal{G}_{FG}$*-suffix*[*simp*]:
$\mathcal{G}_{FG}$ φ (*suffix i w*) = $\mathcal{G}_{FG}$ φ w
unfolding $\mathcal{G}_{FG}$*-alt-def LTL-FG-suffix* **..**

12.2 Eventually Provable and Almost All Eventually Provable

abbreviation $\mathfrak{P}$
where
$\mathfrak{P}\ \varphi\ \mathcal{G}\ w\ i \equiv \exists j.\ \mathcal{G} \models_P af_G\ \varphi\ (w\ [i \rightarrow j])$

definition *almost-all-eventually-provable* :: *'a ltl* ⇒ *'a ltl set* ⇒ *'a set word* ⇒ *bool* (‹$\mathfrak{P}_\infty$›)
where
$\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w \equiv \forall_\infty i.\ \mathfrak{P}\ \varphi\ \mathcal{G}\ w\ i$

12.2.1 Proof Rules

lemma *almost-all-eventually-provable-monotonI*[*intro*]:
$\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w \Longrightarrow \mathcal{G} \subseteq \mathcal{G}' \Longrightarrow \mathfrak{P}_\infty\ \varphi\ \mathcal{G}'\ w$
unfolding *almost-all-eventually-provable-def MOST-nat-le* **by** *blast*

lemma *almost-all-eventually-provable-restrict-to-G*:
$\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w \Longrightarrow$ *Only-G* $\mathcal{G} \Longrightarrow \mathfrak{P}_\infty\ \varphi\ (\mathcal{G} \cap \mathbf{G}\ \varphi)\ w$
proof –
assume *Only-G* $\mathcal{G}$ **and** $\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w$
moreover
hence $\bigwedge\varphi.\ \mathcal{G} \models_P \varphi = (\mathcal{G} \cap \mathbf{G}\ \varphi) \models_P \varphi$
using *LTL-prop-entailment-restrict-to-propos propos-subset*
unfolding *G-nested-propos-alt-def* **by** *blast*
ultimately

show *?thesis*
unfolding *almost-all-eventually-provable-def* **by** *force*
qed

fun *G-depth* :: *'a ltl* ⇒ *nat*
where
G-depth (*φ and ψ*) = *max* (*G-depth φ*) (*G-depth ψ*)
| *G-depth* (*φ or ψ*) = *max* (*G-depth φ*) (*G-depth ψ*)
| *G-depth* (*F φ*) = *G-depth φ*
| *G-depth* (*G φ*) = *G-depth φ* + *1*
| *G-depth* (*X φ*) = *G-depth φ*
| *G-depth* (*φ U ψ*) = *max* (*G-depth φ*) (*G-depth ψ*)
| *G-depth φ* = *0*

lemma *almost-all-eventually-provable-restrict-to-G-depth*:
assumes $\mathfrak{P}_\infty$ *φ* $\mathcal{G}$ *w*
assumes *Only-G* $\mathcal{G}$
shows $\mathfrak{P}_\infty$ *φ* ($\mathcal{G}$ ∩ {*ψ*. *G-depth ψ* ≤ *G-depth φ*}) *w*
proof −
{
fix *φ*
have $\mathcal{G} \models_P \varphi$ = ($\mathcal{G}$ ∩ {*ψ*. *G-depth ψ* ≤ *G-depth φ*}) $\models_P \varphi$
by (*induction φ*) (*insert* ‹*Only-G* $\mathcal{G}$›, *auto*)
}
note *Unfold1* = *this*

{
fix *w*
{
fix *φ ν*
have {*ψ*. *G-depth ψ* ≤ *G-depth* (*af-G-letter φ ν*)} = {*ψ*. *G-depth ψ* ≤ *G-depth φ*}
by (*induction φ*) (*unfold af-G-letter.simps G-depth.simps*, *simp-all*, (*metis le-max-iff-disj mem-Collect-eq*)+)
}
hence {*ψ*. *G-depth ψ* ≤ *G-depth* (af_G *φ w*)} = {*ψ*. *G-depth ψ* ≤ *G-depth* *φ*}
by (*induction w arbitrary*: *φ rule*: *rev-induct*) *fastforce*+
}
note *Unfold2* = *this*

from *assms*(*1*) **show** *?thesis*
unfolding *almost-all-eventually-provable-def Unfold1 Unfold2* **.**
qed

lemma *almost-all-eventually-provable-suffix*:
$\mathfrak{P}_\infty\ \varphi\ \mathcal{G}'\ w \Longrightarrow \mathfrak{P}_\infty\ \varphi\ \mathcal{G}'\ (suffix\ i\ w)$
unfolding *almost-all-eventually-provable-def MOST-nat-le*
by (*metis Nat.add-0-right subsequence-shift subsequence-prefix-suffix suffix-0 add.assoc diff-zero trans-le-add2*)

12.2.2 Threshold

The first index, such that the formula is eventually provable from this time on

fun *threshold* :: $'a\ ltl \Rightarrow 'a\ set\ \ word \Rightarrow 'a\ ltl\ set \Rightarrow nat\ option$
where
$threshold\ \varphi\ w\ \mathcal{G} = index\ (\lambda j.\ \mathfrak{P}\ \varphi\ \mathcal{G}\ w\ j)$

lemma *threshold-properties*:
$threshold\ \varphi\ w\ \mathcal{G} = Some\ i \Longrightarrow 0 < i \Longrightarrow \neg\ \mathcal{G} \models_P af_G\ \varphi\ (w\ [(i - 1) \rightarrow k])$
$threshold\ \varphi\ w\ \mathcal{G} = Some\ i \Longrightarrow j \geq i \Longrightarrow \exists\, k.\ \mathcal{G} \models_P af_G\ \varphi\ (w\ [j \rightarrow k])$
using *index-properties* **unfolding** *threshold.simps* **by** *blast+*

lemma *threshold-suffix*:
assumes $threshold\ \varphi\ w\ \mathcal{G} = Some\ k$
assumes $threshold\ \varphi\ (suffix\ i\ w)\ \mathcal{G} = Some\ k'$
shows $k \leq k' + i$
proof (*rule ccontr*)
assume $\neg k \leq k' + i$
hence $k > k' + i$
by *arith*
then obtain j **where** $k = k' + i + Suc\ j$
by (*metis Suc-diff-Suc le-Suc-eq le-add1 le-add-diff-inverse less-imp-Suc-add*)
hence $0 < k$ **and** $k' + i + Suc\ j - 1 = i + (k' + j)$
using ‹$k > k' + i$› **by** *arith+*
show *False*
using *threshold-properties*(*1*)[*OF assms*(*1*) ‹$0 < k$›] *threshold-properties*(*2*)[*OF assms*(*2*), *of* $k' + j$, *OF le-add1*]
unfolding *subsequence-shift* ‹$k = k' + i + Suc\ j$› ‹$k' + i + Suc\ j - 1 = i + (k' + j)$› **by** *blast*
qed

12.2.3 Relation to LTL semantics

lemma *ltl-implies-provable*:
$w \models \varphi \Longrightarrow \mathfrak{P}\ \varphi\ (\mathcal{G}_{FG}\ \varphi\ w)\ w\ 0$

proof (*induction* φ *arbitrary*: *w*)
  case (*LTLProp a*)
    hence {} $\models_P$ af_G (*p*(*a*)) (*w* [*0* → *1*])
      by (*simp add*: *subsequence-def*)
    thus *?case*
      by *blast*
next
  case (*LTLPropNeg a*)
    hence {} $\models_P$ af_G (*np*(*a*)) (*w* [*0* → *1*])
      by (*simp add*: *subsequence-def*)
    thus *?case*
      by *blast*
next
  case (*LTLAnd* φ_1 φ_2)
    obtain i_1 i_2 **where** ($\mathcal{G}_{FG}$ φ_1 *w*) $\models_P$ af_G φ_1 (*w* [*0* → i_1]) **and** ($\mathcal{G}_{FG}$ φ_2 *w*) $\models_P$ af_G φ_2 (*w* [*0* → i_2])
      using *LTLAnd* **unfolding** *ltl-semantics.simps* **by** *blast*
    have ($\mathcal{G}_{FG}$ φ_1 *w*) $\models_P$ af_G φ_1 (*w* [*0* → i_1 + i_2]) **and** ($\mathcal{G}_{FG}$ φ_2 *w*) $\models_P$ af_G φ_2 (*w* [*0* → i_2 + i_1])
      using af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G* - ‹($\mathcal{G}_{FG}$ φ_1 *w*) $\models_P$ af_G φ_1 (*w* [*0* → i_1])›]
      using af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G* - ‹($\mathcal{G}_{FG}$ φ_2 *w*) $\models_P$ af_G φ_2 (*w* [*0* → i_2])›]
      by *simp+*
    thus *?case*
      by (*simp only*: af_G*-decompose add.commute*) *auto*
next
  case (*LTLOr* φ_1 φ_2)
    thus *?case*
      unfolding af_G*-decompose* **by** (*cases w* $\models$ φ_1) *force+*
next
  case (*LTLNext* φ)
    obtain *i* **where** ($\mathcal{G}_{FG}$ φ *w*) $\models_P$ af_G φ (*suffix 1 w* [*0* → *i*])
      using *LTLNext*(*1*)[*OF LTLNext*(*2*)[*unfolded ltl-semantics.simps*]]
      unfolding $\mathcal{G}_{FG}$*-suffix* **by** *blast*
    hence ($\mathcal{G}_{FG}$ (*X* φ) *w*) $\models_P$ af_G (*X* φ) (*w* [*0* → *1* + *i*])
      unfolding *subsequence-shift subsequence-append* **by** (*simp add*: *subsequence-def*)
    thus *?case*
      by *blast*
next
  case (*LTLFinal* φ)
    then obtain *i* **where** *suffix i w* $\models$ φ
      by *auto*

then obtain *j* **where** $\mathcal{G}_{FG}$ φ *w* $\models_P$ af_G φ (*suffix i w* [*0* → *j*])
using *LTLFinal* $\mathcal{G}_{FG}$*-suffix* **by** *blast*
hence *A*: $\mathcal{G}_{FG}$ φ *w* $\models_P$ af_G φ (*suffix i w* [*0* → *Suc j*])
using af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G*, *of j Suc j*, *OF le-SucI*]
by *blast*
from af_G*-keeps-F-and-S*[*OF* - *A*] **have** $\mathcal{G}_{FG}$ φ *w* $\models_P$ af_G (*F* φ) (*w* [*0* → *Suc* (*i* + *j*)])
unfolding *subsequence-shift subsequence-append Suc-eq-plus1* **by** *simp*
thus *?case*
using $\mathcal{G}_{FG}$*.simps*(*7*) **by** *blast*
next
case (*LTLUntil* φ ψ)
then obtain *k* **where** *suffix k w* $\models$ ψ **and** $\forall$ *j* < *k*. *suffix j w* $\models$ φ
by *auto*
thus *?case*
proof (*induction k arbitrary*: *w*)
case *0*
then obtain *i* **where** $\mathcal{G}_{FG}$ ψ *w* $\models_P$ af_G ψ (*w* [*0* → *i*])
using *LTLUntil* **by** (*metis suffix-0*)
hence $\mathcal{G}_{FG}$ ψ *w* $\models_P$ af_G ψ (*w* [*0* → *Suc i*])
using af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G*, *of i Suc i*, *OF le-SucI*] **by** *auto*
hence $\mathcal{G}_{FG}$ (φ *U* ψ) *w* $\models_P$ af_G (φ *U* ψ) (*w* [*0* → *Suc i*])
unfolding af_G*-subsequence-U ltl-prop-entailment.simps* $\mathcal{G}_{FG}$*.simps*
by *blast*
thus *?case*
by *blast*
next
case (*Suc k*)
hence *w* $\models$ φ **and** *suffix k* (*suffix 1 w*) $\models$ ψ **and** $\forall$ *j*<*k*. *suffix j* (*suffix 1 w*) $\models$ φ
unfolding *suffix-0 suffix-suffix* **by** (*auto*, *metis Suc-less-eq*)+
then obtain *i* **where** *i-def*: $\mathcal{G}_{FG}$ (φ *U* ψ) *w* $\models_P$ af_G (φ *U* ψ) (*suffix 1 w* [*0* → *i*])
using *Suc*(*1*)[*of suffix 1 w*] **unfolding** *LTL-FG-suffix* $\mathcal{G}_{FG}$*-alt-def*
by *blast*
obtain *j* **where** *j-def*: $\mathcal{G}_{FG}$ φ *w* $\models_P$ af_G φ (*w* [*0* → *j*])
using *LTLUntil*(*1*)[*OF* ‹*w* $\models$ φ›] **by** *auto*
hence $\mathcal{G}_{FG}$ (φ *U* ψ) *w* $\models_P$ af_G φ (*w* [*0* → *j*])
by *auto*

hence $\mathcal{G}_{FG}$ (φ *U* ψ) *w* $\models_P$ af_G φ (*w* [*0* → *j* + (*i* + *1*)])
by (*blast intro*: af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G le-add1*])
moreover

have $1 + (i + j) = j + (i + 1)$
by *arith*
have $\mathcal{G}_{FG}$ $(\varphi\ U\ \psi)$ $w \models_P af_G$ $(\varphi\ U\ \psi)$ $(w\ [1 \to j + (i + 1)])$
using af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G le-add1 i-def*, *of j*]
unfolding *subsequence-shift* $\mathcal{G}_{FG}$*-suffix* ‹$1 + (i + j) = j + (i + 1)$› **by** *simp*
ultimately
have $\mathcal{G}_{FG}$ $(\varphi\ U\ \psi)$ $w \models_P af_G$ $(\varphi\ U\ \psi)$ $(w\ [1 \to Suc\ (j + i)])$ *and* $af_G\ \varphi$ $(w\ [0 \to Suc\ (j + i)])$
by *simp*
hence $\mathcal{G}_{FG}$ $(\varphi\ U\ \psi)$ $w \models_P af_G$ $(\varphi\ U\ \psi)$ $(w\ [0 \to Suc\ (j + i)])$
unfolding af_G*-subsequence-U ltl-prop-entailment.simps* **by** *blast*
thus *?case*
using af_G*-subsequence-U ltl-prop-entailment.simps* **by** *blast*
qed
qed *simp+*

lemma *ltl-implies-provable-almost-all*:
$w \models \varphi \Longrightarrow \forall_\infty i.\ \mathcal{G}_{FG}\ \varphi\ w \models_P af_G\ \varphi\ (w\ [0 \to i])$
using *ltl-implies-provable* af_G*-sat-core-generalized*[*OF* $\mathcal{G}_{FG}$*-Only-G*]
unfolding *MOST-nat-le* **by** *metis*

12.2.4 Closed Sets

abbreviation *closed*
where
closed $\mathcal{G}\ w \equiv$ *finite* $\mathcal{G} \land$ *Only-G* $\mathcal{G} \land (\forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow \mathfrak{P}_\infty\ \psi\ \mathcal{G}\ w)$

lemma *closed-FG*:
assumes *closed* $\mathcal{G}\ w$
assumes $G\ \psi \in \mathcal{G}$
shows $w \models F\ G\ \psi$
proof –
have *finite* $\mathcal{G}$ **and** *Only-G* $\mathcal{G}$ **and** $(\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow \mathfrak{P}_\infty\ \psi\ \mathcal{G}\ w)$
using *assms* **by** *simp+*
moreover
note ‹$G\ \psi \in \mathcal{G}$›
ultimately
show $w \models F\ G\ \psi$
proof (*induction arbitrary*: ψ *rule*: *finite-ranking-induct*[**where** f = *G-depth*])
case (*insert x* $\mathcal{G}$)

then obtain ψ' **where** $x = G\ \psi'$
by *auto*

{
fix ψ **assume** $G\ \psi \in insert\ x\ \mathcal{G}$ (**is** - $\in$ $?\mathcal{G}'$)
hence $\mathfrak{P}_\infty\ \psi\ (?\mathcal{G}' \cap \{\psi'.\ G\text{-}depth\ \psi' \leq G\text{-}depth\ \psi\})\ w$
using *insert*(*4−5*) **by** (*blast dest*: *almost-all-eventually-provable-restrict-to-G-depth*)
moreover
have $G\text{-}depth\ \psi < G\text{-}depth\ x$
using *insert*(*2*) ‹$G\ \psi \in insert\ x\ \mathcal{G}$› ‹$x = G\ \psi'$› **by** *force*
ultimately
have $\mathfrak{P}_\infty\ \psi\ \mathcal{G}\ w$
by *auto*
}
hence $\mathfrak{P}_\infty\ \psi'\ \mathcal{G}\ w$ **and** *closed* $\mathcal{G}\ w$
using *insert* ‹$x = G\ \psi'$› **by** *simp+*

have *Only-G* $\mathcal{G}$ **and** *Only-G* $(\mathcal{G} \cup \mathbf{G}\ \psi')$ **and** *finite* $(\mathcal{G} \cup \mathbf{G}\ \psi')$
using *G-nested-finite G-nested-propos-Only-G insert* **by** *blast+*
then obtain k_1 **where** *k1-def*: $\bigwedge\psi\ i.\ \psi \in \mathcal{G} \cup \mathbf{G}\ \psi' \Longrightarrow suffix\ k_1\ w$
$\models \psi = suffix\ (k_1 + i)\ w \models \psi$
by (*blast intro*: *ltl-G-stabilize*)

hence $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow w \models F\ (G\ \psi)$
using *insert* ‹*closed* $\mathcal{G}\ w$› **by** *simp*
then obtain k_2 **where** *k2-def*: $\forall\, i \geq k_2.\ \exists\, j.\ \mathfrak{P}\ \psi'\ \mathcal{G}\ w\ i$
using ‹$\mathfrak{P}_\infty\ \psi'\ \mathcal{G}\ w$› **unfolding** *almost-all-eventually-provable-def*
MOST-nat-le **by** *blast*

{
fix i
assume $i \geq max\ k_1\ k_2$
hence $i \geq k_1$ **and** $i \geq k_2$
by *simp+*
then obtain j' **where** $\mathcal{G} \models_P af_G\ \psi'\ (w\ [i \rightarrow j'])$
using *k2-def* **by** *blast*
then obtain j **where** $\mathcal{G} \models_P af_G\ \psi'\ (w\ [i \rightarrow i + j])$
by (*cases* $i \leq j'$) (*blast dest*: *le-Suc-ex*, *metis subsequence-empty*
le-add-diff-inverse nat-le-linear)
moreover
have $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow suffix\ k_1\ w \models G\ \psi$
using *ltl-G-stabilize-property*[*OF* ‹*finite* $(\mathcal{G} \cup \mathbf{G}\ \psi')$› ‹*Only-G* $(\mathcal{G} \cup$
$\mathbf{G}\ \psi')$› *k1-def*]
using ‹$\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow w \models F\ (G\ \psi)$› **by** *blast*
hence $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow suffix\ (i + j)\ w \models G\ \psi$

by (*metis* ‹$i \geq max\ k_1\ k_2$› *LTL-suffix-G suffix-suffix le-Suc-ex max.cobounded1*)
hence $\bigwedge\psi.\ \psi \in \mathcal{G} \Longrightarrow$ *suffix* $(i + j)\ w \models \psi$
using ‹*Only-G* $\mathcal{G}$› **by** *fast*
ultimately
have *Suffix*: *suffix* $(i + j)\ w \models af_G\ \psi'\ (w\ [i \rightarrow i + j])$
using *ltl-models-equiv-prop-entailment* **by** *blast*

obtain c **where** $i = k_1 + c$
using ‹$i \geq k_1$› **unfolding** *le-iff-add* **by** *blast*
hence *Stable*: $\forall\,\psi \in \mathbf{G}\ \psi'.$ *suffix* $i\ w \models \psi =$ *suffix* j (*suffix* $i\ w) \models \psi$
using *k1-def k1-def*[*of - c + j*] **unfolding** *suffix-suffix add.assoc*[*symmetric*] **by** *blast*
from *Suffix* **have** *suffix* $i\ w \models \psi'$
unfolding *suffix-suffix subsequence-shift* af_G*-ltl-continuation-suffix*[*OF Stable*] **by** *simp*
}
hence $w \models F\ G\ \psi'$
unfolding *MOST-nat-le LTL-FG-almost-all-suffixes* **by** *blast*
thus *?case*
using *insert* **using** ‹$\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow w \models F\ G\ \psi$› ‹$x = G\ \psi'$› **by** *auto*
qed *blast*
qed

lemma *closed-*$\mathcal{G}_{FG}$:
closed $(\mathcal{G}_{FG}\ \varphi\ w)\ w$
proof (*induction* φ)
case (*LTLGlobal* φ)
thus *?case*
proof (*cases* $w \models F\ G\ \varphi$)
case *True*
hence $\forall_\infty i.$ *suffix* $i\ w \models \varphi$
using *LTL-FG-almost-all-suffixes* **by** *blast*
then obtain i **where** $\forall\,j \geq i.$ *suffix* $j\ w \models \varphi$
unfolding *MOST-nat-le* **by** *blast*
{
fix k
assume $k \geq i$
hence *suffix* $k\ w \models \varphi$
using ‹$\forall\,j{\geq}i.$ *suffix* $j\ w \models \varphi$› **by** *blast*
hence $\mathfrak{P}\ \varphi\ \{G\ \psi\ |\psi.\ w \models F\ G\ \psi\}$ (*suffix* $k\ w$) *0*
using *LTL-FG-suffix*
by (*blast dest*: *ltl-implies-provable*[*unfolded* $\mathcal{G}_{FG}$*-alt-def*])

hence $\mathfrak{P}\ \varphi\ \{G\ \psi\ |\psi.\ w \models F\ G\ \psi\}\ w\ k$
unfolding *subsequence-shift* **by** *auto*
}
hence $\mathfrak{P}_\infty\ \varphi\ \{G\ \psi\ |\ \psi.\ w \models F\ G\ \psi\}\ w$
using *almost-all-eventually-provable-def*[*of* φ - *w*]
unfolding *MOST-nat-le* **by** *auto*
hence $\mathfrak{P}_\infty\ \varphi\ (\mathcal{G}_{FG}\ \varphi\ w)\ w$
unfolding $\mathcal{G}_{FG}$*-alt-def*
using *almost-all-eventually-provable-restrict-to-G* **by** *blast*
thus *?thesis*
using *LTLGlobal insert* **by** *auto*
qed *auto*
qed *auto*

12.2.5 Conjunction of Eventually Provable Formulas

definition $\mathcal{F}$
where
$\mathcal{F}\ \varphi\ w\ \mathcal{G}\ j = And\ (map\ (\lambda i.\ af_G\ \varphi\ (w\ [i \to j]))\ [the\ (threshold\ \varphi\ w\ \mathcal{G})$ $..< Suc\ j])$

lemma *almost-all-suffixes-model-*$\mathcal{F}$:
assumes *closed* $\mathcal{G}$ *w*
assumes $G\ \varphi \in \mathcal{G}$
shows $\forall_\infty j.$ *suffix j w* $\models eval_G\ \mathcal{G}\ (\mathcal{F}\ \varphi\ w\ \mathcal{G}\ j)$
proof −
have *Only-G* $\mathcal{G}$
using *assms*(*1*) **by** *simp*
hence $\mathcal{G} \subseteq \{\chi.\ w \models F\ \chi\}$ **and** $\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w$
using *closed-FG*[*OF assms*(*1*)] *assms* **by** *auto*
then obtain *k* **where** *threshold* φ *w* $\mathcal{G}$ = *Some k*
by (*simp add*: *almost-all-eventually-provable-def*)
hence *k-def*: *k* = *the* (*threshold* φ *w* $\mathcal{G}$)
by *simp*
moreover
have *finite* (**G** $\varphi \cup \mathcal{G}$) **and** *Only-G* (**G** $\varphi \cup \mathcal{G}$)
using *assms*(*1*) *G-nested-finite* **unfolding** *G-nested-propos-alt-def* **by** *auto*
then obtain *l* **where** *S*: $\bigwedge j\ \psi.\ \psi \in$ **G** $\varphi \cup \mathcal{G} \Longrightarrow$ *suffix l w* $\models \psi =$ *suffix* $(l + j)\ w \models \psi$
using *ltl-G-stabilize* **by** *metis*
hence $\mathcal{G}$*-sat*:$\bigwedge j\ \psi.\ G\ \psi \in \mathcal{G} \Longrightarrow$ *suffix* $(l + j)\ w \models G\ \psi$
using *ltl-G-stabilize-property* ‹$\mathcal{G} \subseteq \{\chi.\ w \models F\ \chi\}$› **by** *blast*
{

fix j
assume $l \leq j$
{
fix i
assume $k \leq i$ $i \leq j$
then obtain j' **where** $j = i + j'$
by (*blast dest*: *le-Suc-ex*)
hence $\exists j \geq i.\ \mathcal{G} \models_P af_G\ \varphi\ (w\ [i \to j])$
using ‹$\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w$› **unfolding** *almost-all-eventually-provable-def MOST-nat-le*
by (*metis* ‹$k \leq i$› ‹*threshold* φ w $\mathcal{G}$ = *Some* k› *threshold-properties*(*2*) *linear subsequence-empty*)
then obtain j'' **where** $\mathcal{G} \models_P af_G\ \varphi\ (w\ [i \to j''])$ **and** $i \leq j''$
by (*blast*)
have *suffix* j w $\models$ $eval_G\ \mathcal{G}\ (af_G\ \varphi\ (w\ [i \to j]))$
proof (*cases* $j'' \leq j$)
case *True*
hence $\mathcal{G} \models_P af_G\ \varphi\ (w\ [i \to j])$
using af_G-*sat-core-generalized*[*OF* ‹*Only-G* $\mathcal{G}$›, *of* - j' φ *suffix* i w] *le-Suc-ex*[*OF* ‹$i \leq j''$›] *le-Suc-ex*[*OF* ‹$j'' \leq j$›]
by (*metis add.right-neutral subsequence-shift* ‹$j = i + j'$› ‹$\mathcal{G} \models_P af_G\ \varphi\ (w\ [i \to j''])$› *nat-add-left-cancel-le*)
hence $\mathcal{G} \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ (w\ [i \to j]))$
unfolding $eval_G$-*prop-entailment* **.**
moreover
have $\mathcal{G} \subseteq \{\chi.\ suffix\ j\ w \models \chi\}$
using $\mathcal{G}$-*sat* ‹$l \leq j$› ‹*Only-G* $\mathcal{G}$› **by** (*fast dest*: *le-Suc-ex*)
ultimately
have $\{\chi.\ suffix\ j\ w \models \chi\} \models_P eval_G\ \mathcal{G}\ (af_G\ \varphi\ (w\ [i \to j]))$
by *blast*
thus *?thesis*
unfolding *ltl-models-equiv-prop-entailment*[*symmetric*] **by** *simp*
next
case *False*
hence $\mathcal{G} \models_P eval_G\ \mathcal{G}\ (af_G\ (af_G\ \varphi\ (w\ [i \to j]))\ (w\ [j \to j'']))$
unfolding *foldl-append*[*symmetric*] $eval_G$-*prop-entailment*
by (*metis le-iff-add* ‹$i \leq j$› *map-append upt-add-eq-append nat-le-linear subsequence-def* ‹$\mathcal{G} \models_P af_G\ \varphi\ (w\ [i \to j''])$›)
hence $\mathcal{G} \models_P af_G\ (eval_G\ \mathcal{G}\ (af_G\ \varphi\ (w\ [i \to j])))\ (w\ [j \to j''])$ (**is** $\mathcal{G} \models_P ?af_G$)
using af_G-$eval_G$[*OF* ‹*Only-G* $\mathcal{G}$›] **by** *blast*
moreover
have $l \leq j''$
using *False* ‹$l \leq j$› **by** *linarith*

hence $\mathcal{G} \subseteq \{\chi.\ \textit{suffix}\ j''\ w \models \chi\}$
using $\mathcal{G}$*-sat* ‹*Only-G* $\mathcal{G}$› **by** (*fast dest*: *le-Suc-ex*)
ultimately
have *suffix* $j''\ w \models$ *?af*$_G$
using *ltl-models-equiv-prop-entailment*[*symmetric*] **by** *blast*
moreover
{
have $\bigwedge\psi.\ \psi \in \mathbf{G}\ \varphi \cup \mathcal{G} \Longrightarrow \textit{suffix}\ j\ w \models \psi = \textit{suffix}\ j''\ w \models \psi$
using S ‹$l \leq j$› ‹$l \leq j''$› **by** (*metis le-add-diff-inverse*)
moreover
have G ($\textit{eval}_G\ \mathcal{G}\ (\textit{af}_G\ \varphi\ (w\ [i \rightarrow j]))) \subseteq$ **G** φ (**is** *?G* $\subseteq$ -)
using *eval*$_G$*-G-nested* **by** *force*
ultimately
have $\bigwedge\psi.\ \psi \in$ *?G* $\Longrightarrow \textit{suffix}\ j\ w \models \psi = \textit{suffix}\ j''\ w \models \psi$
by *auto*
}
ultimately
show *?thesis*
using *af*$_G$*-ltl-continuation-suffix*[*of eval*$_G$ $\mathcal{G}$ (*af*$_G$ φ (w [$i \rightarrow j$])) *suffix j w*, *unfolded suffix-suffix*]
by (*metis False le-Suc-ex nat-le-linear add-diff-cancel-left*$'$ *subsequence-prefix-suffix*)
qed
}
hence $\textit{suffix}\ j\ w \models \textit{And}\ (\textit{map}\ (\lambda i.\ \textit{eval}_G\ \mathcal{G}\ (\textit{af}_G\ \varphi\ (w\ [i \rightarrow j])))\ [k..{<}\textit{Suc}\ j])$
unfolding *And-semantics set-map set-upt image-def* **by** *force*
hence $\textit{suffix}\ j\ w \models \textit{eval}_G\ \mathcal{G}\ (\textit{And}\ (\textit{map}\ (\lambda i.\ \textit{af}_G\ \varphi\ (w\ [i \rightarrow j]))\ [k..{<}\textit{Suc}\ j]))$
unfolding *eval*$_G$*-And-map map-map comp-def* **.**
}
thus *?thesis*
unfolding $\mathcal{F}$*-def And-semantics MOST-nat-le k-def*[*symmetric*] **by** *meson*
qed

lemma *almost-all-commutative*$''$:
assumes *finite S*
assumes *Only-G S*
assumes $\bigwedge x.\ G\ x \in S \Longrightarrow \forall_\infty i.\ P\ x\ (i{::}\textit{nat})$
shows $\forall_\infty i.\ \forall x.\ G\ x \in S \longrightarrow P\ x\ i$
proof –
from *assms* **have** $(\bigwedge x.\ x \in S \Longrightarrow \forall_\infty i.\ P\ (\textit{theG}\ x)\ (i{::}\textit{nat}))$
by *fastforce*

with *assms*(*1*) **have** $\forall_\infty i.\ \forall x \in S.\ P\ (theG\ x)\ i$
  using *almost-all-commutative′* **by** *force*
thus *?thesis*
  using *assms*(*2*) **unfolding** *MOST-nat-le* **by** *force*
qed

lemma *almost-all-suffixes-model-$\mathcal{F}$-generalized*:
  assumes *closed* $\mathcal{G}$ *w*
  shows $\forall_\infty j.\ \forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow suffix\ j\ w \models eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)$
  using *almost-all-suffixes-model-$\mathcal{F}$*[*OF assms*] *almost-all-commutative″*[*of* $\mathcal{G}$] *assms* **by** *fast*

12.3 Technical Lemmas

lemma *threshold-suffix-2*:
  assumes *threshold* ψ *w* $\mathcal{G}'$ = *Some k*
  assumes $k \leq l$
  shows *threshold* ψ (*suffix l w*) $\mathcal{G}'$ = *Some 0*
proof –
  have $\mathfrak{P}_\infty\ \psi\ \mathcal{G}'\ w$
    using ‹*threshold* ψ *w* $\mathcal{G}'$ = *Some k*› *option.distinct*(*1*)
  unfolding *threshold.simps index.simps almost-all-eventually-provable-def* **by** *metis*
  hence $\mathfrak{P}_\infty\ \psi\ \mathcal{G}'$ (*suffix l w*)
    using *almost-all-eventually-provable-suffix* **by** *blast*
  moreover
  have $\forall i \geq k.\ \exists j.\ \mathcal{G}' \models_P af_G\ \psi\ (w\ [i \rightarrow j])$
    using *threshold-properties*(*2*)[*OF assms*(*1*)] **by** *blast*
  hence $\forall m.\ \exists j.\ \mathcal{G}' \models_P af_G\ \psi\ ((suffix\ l\ w)\ [m \rightarrow j])$
    unfolding *subsequence-shift* **using** ‹$k \leq l$› ‹$\forall i \geq k.\ \exists j.\ \mathcal{G}' \models_P af_G\ \psi\ (w\ [i \rightarrow j])$›
    by (*metis* (*mono-tags*, *opaque-lifting*) *leI less-imp-add-positive order-refl subsequence-empty trans-le-add1*)
  ultimately
  show *?thesis*
    by *simp*
qed

lemma *threshold-closed*:
  assumes *closed* $\mathcal{G}$ *w*
  shows $\exists k.\ \forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow threshold\ \psi\ (suffix\ k\ w)\ \mathcal{G} = Some\ 0$
proof –
  define *k* **where** $k = Max\ \{the\ (threshold\ \psi\ w\ \mathcal{G}) \mid \psi.\ G\ \psi \in \mathcal{G}\}$ (**is** - = *Max ?S*)

have *finite* $\mathcal{G}$ **and** *Only-G* $\mathcal{G}$ **and** $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow \mathfrak{P}_\infty\ \psi\ \mathcal{G}\ w$
using *assms* **by** *blast+*
hence $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow \exists\, k.\ threshold\ \psi\ w\ \mathcal{G} = Some\ k$
unfolding *almost-all-eventually-provable-def* **by** *simp*
moreover
have $?S = (\lambda x.\ the\ (threshold\ (theG\ x)\ w\ \mathcal{G}))$ ‘ $\mathcal{G}$
unfolding *image-def* **using** ‹*Only-G* $\mathcal{G}$› *ltl.sel(8)* **by** *metis*
hence *finite ?S*
using ‹*finite* $\mathcal{G}$› *finite-imageI* **by** *simp*
hence $\bigwedge\psi\ k'.\ G\ \psi \in \mathcal{G} \Longrightarrow threshold\ \psi\ w\ \mathcal{G} = Some\ k' \Longrightarrow k' \leq k$
by (*metis* (*mono-tags*, *lifting*) *CollectI Max-ge k-def option.sel*)
ultimately
have $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow threshold\ \psi\ (suffix\ k\ w)\ \mathcal{G} = Some\ 0$
using *threshold-suffix[of - w* $\mathcal{G}$ *- k 0] threshold-suffix-2* **by** *blast*
thus *?thesis*
by *blast*
qed

lemma $\mathcal{F}$*-drop*:
assumes $\mathfrak{P}_\infty\ \varphi\ \mathcal{G}'\ w$
assumes $S \models_P \mathcal{F}\ \varphi\ w\ \mathcal{G}'\ (i + j)$
shows $S \models_P \mathcal{F}\ \varphi\ (suffix\ i\ w)\ \mathcal{G}'\ j$
proof −
obtain $k\ k'$ **where** *k-def*: $threshold\ \varphi\ w\ \mathcal{G}' = Some\ k$ **and** k'*-def*: $threshold\ \varphi\ (suffix\ i\ w)\ \mathcal{G}' = Some\ k'$
using *assms almost-all-eventually-provable-suffix*
unfolding *threshold.simps index.simps almost-all-eventually-provable-def* **by** *fastforce*
hence *k-def-2*: $the\ (threshold\ \varphi\ w\ \mathcal{G}') = k$ **and** k'*-def-2*: $the\ (threshold\ \varphi\ (suffix\ i\ w)\ \mathcal{G}') = k'$
by *simp+*
moreover
hence $k \leq i + j \Longrightarrow S \models_P \varphi$
using ‹$S \models_P \mathcal{F}\ \varphi\ w\ \mathcal{G}'\ (i + j)$› **unfolding** $\mathcal{F}$*-def And-semantics And-prop-entailment* **by** (*simp add*: *subsequence-def*)
moreover
have $k' \leq j \Longrightarrow k \leq i + j$
using *k-def k'-def threshold-suffix* **by** *fastforce*
ultimately
have $the\ (threshold\ \varphi\ (suffix\ i\ w)\ \mathcal{G}') \leq j \Longrightarrow S \models_P \varphi$
by *blast*
moreover
{

fix *pos*
assume $k' \leq pos$ **and** $pos \leq j$
have $k \leq i + pos$
by (*metis threshold-suffix k-def k′-def* ‹$k' \leq pos$› *add.commute add-le-cancel-right order.trans*)
hence $(i + pos) \in set\ [k..<Suc\ (i + j)]$
using ‹$pos \leq j$› **by** *auto*
hence $af_G\ \varphi\ ((suffix\ i\ w)\ [pos \rightarrow j]) \in set\ (map\ (\lambda ia.\ af_G\ \varphi\ (subsequence\ w\ ia\ (i + j)))\ [k..<Suc\ (i + j)])$
unfolding *subsequence-shift set-map* **by** *blast*
hence $S \models_P af_G\ \varphi\ ((suffix\ i\ w)\ [pos \rightarrow j])$
using *assms(2)* **unfolding** *$\mathcal{F}$-def And-prop-entailment k-def-2* **by** (*cases* $k \leq i + j$) *auto*
}
ultimately
show *?thesis*
unfolding *$\mathcal{F}$-def And-prop-entailment k′-def-2* **by** *auto*
qed

12.4 Main Results

definition $accept_M$
where
$accept_M\ \varphi\ \mathcal{G}\ w \equiv (\forall_\infty j.\ \forall S.\ (\forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow S \models_P G\ \psi \wedge S \models_P eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)) \longrightarrow S \models_P af\ \varphi\ (w\ [0 \rightarrow j]))$

lemma *lemmaD*:
assumes $w \models \varphi$
assumes $\bigwedge\psi.\ G\ \psi \in \mathcal{G}_{FG}\ \varphi\ w \Longrightarrow threshold\ \psi\ w\ (\mathcal{G}_{FG}\ \varphi\ w) = Some\ 0$
shows $accept_M\ \varphi\ (\mathcal{G}_{FG}\ \varphi\ w)\ w$
proof –
obtain i **where** $\mathcal{G}_{FG}\ \varphi\ w \models_P af_G\ \varphi\ (w\ [0 \rightarrow i])$
using *ltl-implies-provable*[*OF* ‹$w \models \varphi$›] **by** *metis*
{
fix *S j*
assume *assm1*: $j \geq i$
assume *assm2*: $\bigwedge\psi.\ G\ \psi \in \mathcal{G}_{FG}\ \varphi\ w \Longrightarrow S \models_P G\ \psi \wedge S \models_P eval_G\ (\mathcal{G}_{FG}\ \varphi\ w)\ (\mathcal{F}\ \psi\ w\ (\mathcal{G}_{FG}\ \varphi\ w)\ j)$
moreover
{
have $\mathcal{G}_{FG}\ \varphi\ w \models_P af_G\ \varphi\ (w\ [0 \rightarrow j])$
using ‹$\mathcal{G}_{FG}\ \varphi\ w \models_P af_G\ \varphi\ (w\ [0 \rightarrow i])$› ‹$j \geq i$›
by (*metis af_G-sat-core-generalized $\mathcal{G}_{FG}$-Only-G*)
moreover

have $\mathcal{G}_{FG}\ \varphi\ w \subseteq S$
using *assm2* **unfolding** $\mathcal{G}_{FG}$*-alt-def* **by** *auto*
ultimately
have $S \models_P$ *eval*$_G$ $(\mathcal{G}_{FG}\ \varphi\ w)$ $(af_G\ \varphi\ (w\ [0 \rightarrow j]))$
using *eval*$_G$*-prop-entailment* **by** *blast*
}
moreover
{
fix ψ **assume** $G\ \psi \in \mathcal{G}_{FG}\ \varphi\ w$
hence *the* (*threshold* ψ w $(\mathcal{G}_{FG}\ \varphi\ w)) = 0$ **and** $S \models_P$ *eval*$_G$ $(\mathcal{G}_{FG}\ \varphi$ $w)$ $(\mathcal{F}\ \psi\ w\ (\mathcal{G}_{FG}\ \varphi\ w)\ j)$
using *assms assm2 option.sel* **by** *metis*+
hence $\bigwedge i.\ i \leq j \Longrightarrow S \models_P$ *eval*$_G$ $(\mathcal{G}_{FG}\ \varphi\ w)$ $(af_G\ \psi\ (w[i \rightarrow j]))$
unfolding $\mathcal{F}$*-def And-prop-entailment eval*$_G$*-And-map* **by** *auto*
}
ultimately
have $S \models_P af\ \varphi\ (w\ [0 \rightarrow j])$
using *af*$_G$*-implies-af-eval*$_G$[*of - -* φ] **by** *presburger*
}
thus *?thesis*
unfolding *accept*$_M$*-def MOST-nat-le* **by** *meson*
qed

theorem *ltl-FG-logical-characterization*:
$w \models F\ G\ \varphi \longleftrightarrow (\exists \mathcal{G} \subseteq \mathbf{G}\ (F\ G\ \varphi).\ G\ \varphi \in \mathcal{G} \wedge closed\ \mathcal{G}\ w)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs*
hence $G\ \varphi \in \mathcal{G}_{FG}\ (F\ G\ \varphi)\ w$ **and** $\mathcal{G}_{FG}\ (F\ G\ \varphi)\ w \subseteq \mathbf{G}\ (F\ G\ \varphi)$
unfolding $\mathcal{G}_{FG}$*-alt-def* **by** *auto*
thus *?rhs*
using *closed-*$\mathcal{G}_{FG}$ **by** *metis*
qed (*blast intro*: *closed-FG*)

theorem *ltl-logical-characterization*:
$w \models \varphi \longleftrightarrow (\exists \mathcal{G} \subseteq \mathbf{G}\ \varphi.\ accept_M\ \varphi\ \mathcal{G}\ w \wedge closed\ \mathcal{G}\ w)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs*

obtain k **where** *k-def*: $\bigwedge\psi.\ G\ \psi \in \mathcal{G}_{FG}\ \varphi\ w \Longrightarrow$ *threshold* ψ (*suffix* k $w)$ $(\mathcal{G}_{FG}\ \varphi\ w) =$ *Some 0*
using *threshold-closed*[*OF closed-*$\mathcal{G}_{FG}$] **by** *blast*

define w' **where** $w' = \mathit{suffix}\ k\ w$
define φ' **where** $\varphi' = \mathit{af}\ \varphi\ (w[\mathit{0} \rightarrow k])$

from ‹*?lhs*› **have** $w' \models \varphi'$
  unfolding *af-ltl-continuation-suffix*[*of* $w\ \varphi\ k$] w'*-def* φ'*-def* **.**
have *G-eq*: $\mathbf{G}\ \varphi' = \mathbf{G}\ \varphi$
  unfolding φ'*-def* *G-af-simp* **..**
have $\mathcal{G}$*-eq*: $\mathcal{G}_{FG}\ \varphi'\ w' = \mathcal{G}_{FG}\ \varphi\ w$
  unfolding $\mathcal{G}_{FG}$*-alt-def* w'*-def* φ'*-def* *G-af-simp* *LTL-FG-suffix* **..**
have φ'*-eq*: $\bigwedge j.\ \mathit{af}\ \varphi'\ (w'\ [\mathit{0} \rightarrow j]) = \mathit{af}\ \varphi\ (w\ [\mathit{0} \rightarrow k + j])$
  unfolding φ'*-def* w'*-def* *foldl-append*[*symmetric*] *subsequence-shift*
  unfolding *Nat.add-0-right* **by** (*metis subsequence-append*)

have $\mathit{accept}_M\ \varphi'\ (\mathcal{G}_{FG}\ \varphi'\ w')\ w'$
  using *lemmaD*[*OF* ‹$w' \models \varphi'$›] *k-def*
  unfolding $\mathcal{G}$*-eq* w'*-def*[*symmetric*] **by** *blast*

then obtain j' **where** j'*-def*: $\bigwedge j\ S.\ j \geq j' \Longrightarrow$
  $(\forall \psi.\ G\ \psi \in \mathcal{G}_{FG}\ \varphi'\ w' \longrightarrow S \models_P G\ \psi \wedge S \models_P \mathit{eval}_G\ (\mathcal{G}_{FG}\ \varphi'\ w')\ (\mathcal{F}$
$\psi\ w'\ (\mathcal{G}_{FG}\ \varphi'\ w')\ j)) \Longrightarrow S \models_P \mathit{af}\ \varphi'\ (w'\ [\mathit{0} \rightarrow j])$
  unfolding accept_M*-def* *MOST-nat-le* **by** *blast*

{
  fix $j\ S$
  let $?\mathit{af} = \mathit{af}\ \varphi\ (w[\mathit{0} \rightarrow k + j' + j])$
  assume $(\forall \psi.\ G\ \psi \in (\mathcal{G}_{FG}\ \varphi'\ w') \longrightarrow S \models_P G\ \psi \wedge S \models_P \mathit{eval}_G\ (\mathcal{G}_{FG}$
$\varphi'\ w')\ (\mathcal{F}\ \psi\ w\ (\mathcal{G}_{FG}\ \varphi'\ w')\ (k + j' + j)))$
  moreover
  {
    fix ψ
    assume $G\ \psi \in \mathcal{G}_{FG}\ \varphi'\ w'$ (**is** - $\in$ $?\mathcal{G}$)
    hence $\mathfrak{P}_\infty\ \psi\ ?\mathcal{G}\ w$
      unfolding $\mathcal{G}$*-eq* **using** *closed-*$\mathcal{G}_{FG}$ **by** *blast*
    have $\bigwedge S.\ S \models_P \mathit{eval}_G\ ?\mathcal{G}\ (\mathcal{F}\ \psi\ w\ ?\mathcal{G}\ (k + j' + j)) \Longrightarrow S \models_P \mathit{eval}_G$
$?\mathcal{G}\ (\mathcal{F}\ \psi\ w'\ ?\mathcal{G}\ (j' + j))$
    using $\mathcal{F}$*-drop*[*OF* ‹$\mathfrak{P}_\infty\ \psi\ (\mathcal{G}_{FG}\ \varphi'\ w')\ w$›, *of* - $k\ j' + j$] eval_G*-respectfulness*(*1*)[*unfolded*
ltl-prop-implies-def]
      unfolding *add.assoc* w'*-def* **by** *metis*
    moreover

assume $S \models_P eval_G$ *?*$\mathcal{G}$ $(\mathcal{F}\ \psi\ w$ *?*$\mathcal{G}$ $(k + j' + j))$
ultimately
have $S \models_P eval_G$ *?*$\mathcal{G}$ $(\mathcal{F}\ \psi\ w'$ *?*$\mathcal{G}$ $(j' + j))$
by *simp*
}
ultimately
have $S \models_P$ *?af*
using j'*-def* **unfolding** φ'*-eq add.assoc* **by** *simp*
}
hence $accept_M\ \varphi\ (\mathcal{G}_{FG}\ \varphi\ w)\ w$
unfolding $accept_M$*-def MOST-nat-le* $\mathcal{G}$*-eq* **by** (*metis le-Suc-ex*)
moreover
have $\mathcal{G}_{FG}\ \varphi\ w \subseteq \mathbf{G}\ \varphi$
unfolding $\mathcal{G}_{FG}$*-alt-def* **by** *auto*
ultimately
show *?rhs*
by (*metis closed-*$\mathcal{G}_{FG}$)
next
assume *?rhs*

then obtain $\mathcal{G}$ **where** $\mathcal{G}$*-prop*: $\mathcal{G} \subseteq \mathbf{G}\ \varphi$ *finite* $\mathcal{G}$ *Only-G* $\mathcal{G}$ $accept_M\ \varphi\ \mathcal{G}\ w$ *closed* $\mathcal{G}$ w
using $\mathcal{G}$*-elements* $\mathcal{G}$*-finite* **by** *blast*
then obtain i **where** $\bigwedge\chi\ j.\ \chi \in \mathcal{G} \Longrightarrow$ *suffix* $i\ w \models \chi =$ *suffix* $(i + j)\ w \models \chi$
using *ltl-G-stabilize* **by** *blast*
hence *i-def*: $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow$ *suffix* $i\ w \models G\ \psi$
using *ltl-G-stabilize-property*[*OF* ‹*finite* $\mathcal{G}$› ‹*Only-G* $\mathcal{G}$›] $\mathcal{G}$*-prop closed-FG*[*of* $\mathcal{G}$] **by** *blast*
obtain j **where** *j-def*: $\bigwedge j'\ S.\ j' \geq j \Longrightarrow$
$(\forall\psi.\ G\ \psi \in \mathcal{G} \longrightarrow S \models_P G\ \psi \wedge S \models_P eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j')) \longrightarrow S \models_P af\ \varphi\ (w\ [0 \rightarrow j'])$
using ‹$accept_M\ \varphi\ \mathcal{G}\ w$› **unfolding** $accept_M$*-def MOST-nat-le* **by** *presburger*
obtain j' **where** *lemma19*: $\bigwedge j\ \psi.\ j \geq j' \Longrightarrow G\ \psi \in \mathcal{G} \Longrightarrow$ *suffix* $j\ w \models eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)$
using *almost-all-suffixes-model-*$\mathcal{F}$*-generalized*[*OF* ‹*closed* $\mathcal{G}$ w›] **unfolding** *MOST-nat-le* **by** *blast*

define k **where** $k = max\ (max\ i\ j)\ j'$
define w' **where** $w' =$ *suffix* $k\ w$
define φ' **where** $\varphi' = af\ \varphi\ (w[0 \rightarrow k])$
define S **where** $S = \{\chi.\ w' \models \chi\}$

have $(\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow S \models_P G\ \psi \wedge S \models_P eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ k)) \Longrightarrow S \models_P \varphi'$
using *j-def*[*of k S*] **unfolding** *φ′-def k-def* **by** *fastforce*
moreover
{
fix ψ
assume $G\ \psi \in \mathcal{G}$
have $\bigwedge j.\ i \leq j \Longrightarrow$ *suffix i w* $\models G\ \psi \Longrightarrow$ *suffix j w* $\models G\ \psi$
by (*metis LTL-suffix-G le-Suc-ex suffix-suffix*)
hence $w' \models G\ \psi$
unfolding *w′-def k-def max-def*
using *i-def*[*OF* ‹$G\ \psi \in \mathcal{G}$›] **by** *simp*
moreover
have $w' \models eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ k)$
using *lemma19*[*OF -* ‹$G\ \psi \in \mathcal{G}$›, *of k*]
unfolding *w′-def k-def* **by** *fastforce*
ultimately
have $S \models_P G\ \psi$ **and** $S \models_P eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ k)$
unfolding *S-def ltl-models-equiv-prop-entailment*[*symmetric*] **by** *blast+*
}
ultimately
have $S \models_P \varphi'$
by *simp*
hence $w' \models \varphi'$
using *S-def ltl-models-equiv-prop-entailment* **by** *blast*
thus *?lhs*
using *w′-def φ′-def af-ltl-continuation-suffix* **by** *blast*
qed

end

13 Translation from LTL to (Deterministic Transitions-Based) Generalised Rabin Automata

theory *LTL-Rabin*
imports *Main Mojmir-Rabin Logical-Characterization*
begin

13.1 Preliminary Facts

lemma *run-af-G-letter-abs-eq-Abs-af-G-letter*:
run $\uparrow af_G$ (*Abs* φ) *w i* = *Abs* (*run af-G-letter* φ *w i*)

by (*induction i*) (*simp*, *metis af-G-abs.f-foldl-abs.abs-eq af-G-abs.f-foldl-abs-alt-def run-foldl af-G-letter-abs-def*)

lemma *finite-reach-af*:
finite (*reach* Σ ↑*af* (*Abs* φ))
proof (*cases* Σ ≠ {})
case *True*
thus *?thesis*
using *af-abs.finite-abs-reach* **unfolding** *af-abs.abs-reach-def reach-foldl-def*[*OF True*]
using *finite-subset*[*of* {*foldl* ↑*af* (*Abs* φ) *w* |*w*. *set w* ⊆ Σ} {*foldl* ↑*af*(*Abs* φ) *w* |*w*. *True*}]
unfolding *af-letter-abs-def*
by (*blast*)
qed (*simp add*: *reach-def*)

lemma *ltl-semi-mojmir*:
assumes *finite* Σ
assumes *range w* ⊆ Σ
shows *semi-mojmir* Σ ↑af_G (*Abs* ψ) *w*
proof
fix ψ
have *nonempty-*Σ: Σ ≠ {}
using *assms* **by** *auto*
show *finite* (*reach* Σ ↑af_G (*Abs* ψ)) (**is** *finite ?A*)
using *af-G-abs.finite-abs-reach finite-subset*[**where** *A* = *?A*, **where** *B* = *lift-ltl-transformer.abs-reach af-G-letter* (*Abs* ψ)]
unfolding *af-G-abs.abs-reach-def af-G-letter-abs-def reach-foldl-def*[*OF nonempty-*Σ] **by** *blast*
qed (*insert assms*, *auto*)

13.2 Single Secondary Automaton

locale *ltl-FG-to-rabin-def* =
fixes
Σ :: *'a set set*
fixes
φ :: *'a ltl*
fixes
𝒢 :: *'a ltl set*
fixes
w :: *'a set word*
begin

sublocale *mojmir-to-rabin-def* $\Sigma \uparrow af_G$ *Abs* φ *w* $\{q.\ \mathcal{G} \models_P Rep\ q\}$ **.**

— Import abbreviations from parent locale to simplify terms
abbreviation $\delta_R \equiv step$
abbreviation $q_R \equiv initial$
abbreviation $Acc_R\ j \equiv (fail_R \cup merge_R\ j,\ succeed_R\ j)$
abbreviation $max\text{-}rank_R \equiv max\text{-}rank$
abbreviation $smallest\text{-}accepting\text{-}rank_R \equiv smallest\text{-}accepting\text{-}rank$
abbreviation $accept_R{}' \equiv accept$
abbreviation $\mathcal{S}_R \equiv \mathcal{S}$

lemma *Rep-token-run-af*:
  $Rep\ (token\text{-}run\ x\ n) \equiv_P af_G\ \varphi\ (w\ [x \rightarrow n])$
proof –
  have $token\text{-}run\ x\ n = Abs\ (af_G\ \varphi\ ((suffix\ x\ w)\ [0 \rightarrow (n - x)]))$
   by (*simp add*: *subsequence-def run-foldl*; *metis af-G-abs.f-foldl-abs.abs-eq af-G-abs.f-foldl-abs-alt-def af-G-letter-abs-def*)
  hence $Rep\ (token\text{-}run\ x\ n) \equiv_P af_G\ \varphi\ ((suffix\ x\ w)\ [0 \rightarrow (n - x)])$
    using ltl_P*-abs-rep ltl-prop-equiv-quotient.abs-eq-iff* **by** *auto*
  thus *?thesis*
    unfolding *ltl-prop-equiv-def subsequence-shift* **by** (*cases* $x \leq n$; *simp add*: *subsequence-def*)
qed

end

locale *ltl-FG-to-rabin* = *ltl-FG-to-rabin-def* +
  assumes
    wellformed-$\mathcal{G}$: *Only-G* $\mathcal{G}$
  assumes
    bounded-w: *range* $w \subseteq \Sigma$
  assumes
    finite-Σ: *finite* Σ
begin

sublocale *mojmir-to-rabin* $\Sigma \uparrow af_G$ *Abs* φ *w* $\{q.\ \mathcal{G} \models_P Rep\ q\}$
proof
  show $\bigwedge q\ \nu.\ q \in \{q.\ \mathcal{G} \models_P Rep\ q\} \Longrightarrow \uparrow af_G q\ \nu \in \{q.\ \mathcal{G} \models_P Rep\ q\}$
    using *wellformed-*$\mathcal{G}$ *af-G-letter-sat-core-lifted* **by** *auto*
  have *nonempty-*Σ: $\Sigma \neq \{\}$
    using *bounded-w* **by** *blast*
  show *finite* (*reach* $\Sigma \uparrow af_G(Abs\ \varphi)$) (**is** *finite ?A*)
    using *af-G-abs.finite-abs-reach finite-subset*[**where** A = *?A*, **where** B = *lift-ltl-transformer.abs-reach af-G-letter* (*Abs* φ)]

unfolding *af-G-abs.abs-reach-def af-G-letter-abs-def reach-foldl-def*[*OF nonempty-Σ*] **by** *blast*
qed (*insert finite-Σ bounded-w*)

lemma *ltl-to-rabin-correct-exposed′*:
$\mathfrak{P}_\infty$ φ $\mathcal{G}$ $w \longleftrightarrow$ *accept*
proof −
{
fix i
have $(\exists j.\ \mathcal{G} \models_P af_G\ \varphi\ (map\ w\ [i + 0..<i + (j - i)])) = \mathfrak{P}\ \varphi\ \mathcal{G}\ w\ i$
by (*auto simp add*: *subsequence-def*, *metis add-diff-cancel-left′ le-Suc-ex nat-le-linear upt-conv-Nil*)
hence $(\exists j.\ \mathcal{G} \models_P af_G\ \varphi\ (w\ [i \rightarrow j])) \longleftrightarrow (\exists j.\ \mathcal{G} \models_P$ *run af-G-letter* φ (*suffix* i w) $(j{-}i))$
(**is** *?l* ⟷ -)
unfolding *run-foldl* **using** *subsequence-shift subsequence-def* **by** *metis*
also
have ... $\longleftrightarrow (\exists j.\ \mathcal{G} \models_P Rep\ (run\ \uparrow af_G(Abs\ \varphi)\ (suffix\ i\ w)\ (j{-}i)))$
using *Quotient3-ltl-prop-equiv-quotient*[*THEN Quotient3-rep-abs*]
unfolding *ltl-prop-equiv-def run-af-G-letter-abs-eq-Abs-af-G-letter* **by** *blast*
also
have ... $\longleftrightarrow (\exists j.\ token\text{-}run\ i\ j \in \{q.\ \mathcal{G} \models_P Rep\ q\})$
by *simp*
also
have ... ⟷ *token-succeeds i*
(**is** - ⟷ *?r*)
unfolding *token-succeeds-def* **by** *auto*
finally
have *?l* ⟷ *?r* **.**
}
thus *?thesis*
by (*simp only*: *almost-all-eventually-provable-def accept-def*)
qed

lemma *ltl-to-rabin-correct-exposed*:
$\mathfrak{P}_\infty\ \varphi\ \mathcal{G}\ w \longleftrightarrow accept_R\ (\delta_R,\ q_R,\ \{Acc_R\ i \mid i.\ i < max\text{-}rank_R\})\ w$
unfolding *ltl-to-rabin-correct-exposed′ mojmir-accept-iff-rabin-accept* **..**

— Import lemmas from parent locale to simplify assumptions
lemmas *max-rank-lowerbound* = *max-rank-lowerbound*
lemmas *state-rank-step-foldl* = *state-rank-step-foldl*
lemmas *smallest-accepting-rank-properties* = *smallest-accepting-rank-properties*

lemmas *wellformed-$\mathcal{R}$* = *wellformed-$\mathcal{R}$*

end

fun *ltl-to-rabin*
where
ltl-to-rabin Σ φ $\mathcal{G}$ = (*ltl-FG-to-rabin-def*.δ_R Σ φ, *ltl-FG-to-rabin-def*.q_R φ, {*ltl-FG-to-rabin-def*.Acc_R Σ φ $\mathcal{G}$ *i* | *i*. *i* < *ltl-FG-to-rabin-def*.*max-rank*$_R$ Σ φ})

context
fixes
Σ :: *'a set set*
assumes
finite-Σ: *finite* Σ
begin

lemma *ltl-to-rabin-correct*:
assumes *range* $w \subseteq \Sigma$
shows $w \models F\ G\ \varphi = (\exists \mathcal{G} \subseteq \mathbf{G}\ (G\ \varphi).\ G\ \varphi \in \mathcal{G} \wedge (\forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*))
proof –
have $\bigwedge\mathcal{G}\ \psi.\ \mathcal{G} \subseteq \mathbf{G}\ (G\ \varphi) \Longrightarrow G\ \psi \in \mathcal{G} \Longrightarrow (\mathfrak{P}_\infty\ \psi\ \mathcal{G}\ w \longleftrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*)
proof –
fix $\mathcal{G}$ ψ
assume $\mathcal{G} \subseteq \mathbf{G}\ (G\ \varphi)$ $G\ \psi \in \mathcal{G}$
then interpret *ltl-FG-to-rabin* Σ ψ $\mathcal{G}$
using *finite-Σ* *assms* *G-nested-propos-alt-def*
by (*unfold-locales*; *auto*)
show ($\mathfrak{P}_\infty$ ψ $\mathcal{G}$ *w* $\longleftrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*)
using *ltl-to-rabin-correct-exposed* **by** *simp*
qed
thus *?thesis*
using *$\mathcal{G}$-elements*[*of* - *G* φ] *$\mathcal{G}$-finite*[*of* - *G* φ]
unfolding *ltl-FG-logical-characterization* *G-nested-propos.simps*
by *meson*
qed

end

13.2.1 LTL-to-Mojmir Lemmas

lemma *$\mathcal{F}$-eq-$\mathcal{S}$*:

assumes *finite-Σ*: *finite* Σ
assumes *bounded-w*: *range* $w \subseteq \Sigma$
assumes *closed* $\mathcal{G}$ w
assumes $G\ \psi \in \mathcal{G}$
shows $\forall_\infty j.\ (\forall S.\ (S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j \land \mathcal{G} \subseteq S) \longleftrightarrow (\forall q.\ q \in (\textit{ltl-FG-to-rabin-def}.\mathcal{S}_R$ $\Sigma\ \psi\ \mathcal{G}\ w\ j) \longrightarrow S \models_P \textit{Rep}\ q))$
proof −
let $?F = \{q.\ \mathcal{G} \models_P \textit{Rep}\ q\}$

define *k* **where** $k = \textit{the}\ (\textit{threshold}\ \psi\ w\ \mathcal{G})$
hence $\textit{threshold}\ \psi\ w\ \mathcal{G} = \textit{Some}\ k$
using *assms* **unfolding** *threshold.simps index.simps almost-all-eventually-provable-def* **by** *simp*

have *Only-G* $\mathcal{G}$
using *assms G-nested-propos-alt-def* **by** *blast*
then interpret *ltl-FG-to-rabin* $\Sigma\ \psi\ \mathcal{G}\ w$
using *finite-Σ bounded-w* **by** (*unfold-locales*, *auto*)

have *accept*
using *ltl-to-rabin-correct-exposed′ assms* **by** *blast*
then obtain *i* **where** *smallest-accepting-rank* = *Some i*
unfolding *smallest-accepting-rank-def* **by** *force*

obtain n_1 **where** $\bigwedge m\ q.\ m \geq n_1 \Longrightarrow ((\exists x \in \textit{configuration}\ q\ m.\ \textit{token-succeeds}\ x) \longrightarrow q \in \mathcal{S}\ m) \land (q \in \mathcal{S}\ m \longrightarrow (\forall x \in \textit{configuration}\ q\ m.\ \textit{token-succeeds}\ x))$
using *succeeding-states*[*OF* ‹*smallest-accepting-rank* = *Some i*›] **unfolding** *MOST-nat-le* **by** *blast*

obtain n_2 **where** $\bigwedge x.\ x < k \Longrightarrow \textit{token-succeeds}\ x \Longrightarrow \textit{token-run}\ x\ n_2 \in\ ?F$
by (*induction k*) (*simp*, *metis token-stays-in-final-states add.commute le-neq-implies-less not-less not-less-eq token-succeeds-def*)

define *n* **where** $n = \textit{Max}\ \{n_1, n_2, k\}$

{
fix *m q*
assume $n \leq m$
hence $n_1 \leq m$
unfolding *n-def* **by** *simp*

hence $((\exists x \in$ *configuration q m. token-succeeds x*$) \longrightarrow q \in \mathcal{S}\ m) \wedge (q \in \mathcal{S}\ m \longrightarrow (\forall x \in$ *configuration q m. token-succeeds x*$))$
using ‹$\bigwedge m\ q.\ m \geq n_1 \Longrightarrow ((\exists x \in$ *configuration q m. token-succeeds x*$) \longrightarrow q \in \mathcal{S}\ m) \wedge (q \in \mathcal{S}\ m \longrightarrow (\forall x \in$ *configuration q m. token-succeeds x*$))$› **by** *blast*
}
hence *n-def-1*: $\bigwedge m\ q.\ m \geq n \Longrightarrow ((\exists x \in$ *configuration q m. token-succeeds x*$) \longrightarrow q \in \mathcal{S}\ m) \wedge (q \in \mathcal{S}\ m \longrightarrow (\forall x \in$ *configuration q m. token-succeeds x*$))$
by *presburger*
have *n-def-2*: $\bigwedge x\ m.\ x < k \Longrightarrow m \geq n \Longrightarrow$ *token-succeeds x* $\Longrightarrow$ *token-run x m* $\in$ *?F*
using ‹$\bigwedge x.\ x < k \Longrightarrow$ *token-succeeds x* $\Longrightarrow$ *token-run x* $n_2 \in$ *?F*› *Max.coboundedI*[*of* $\{n_1, n_2, k\}$]
using *token-stays-in-final-states*[*of* - n_2] *le-Suc-ex* **unfolding** *n-def* **by** *force*

{
fix *S m*
assume $n \leq m$
hence $k \leq m$ $n \leq$ *Suc m*
using *n-def* **by** *simp+*

{
assume $S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ m$ $\mathcal{G} \subseteq S$
hence $\bigwedge x.\ k \leq x \Longrightarrow x \leq$ *Suc m* $\Longrightarrow S \models_P af_G\ \psi\ (w\ [x \rightarrow m])$
unfolding *And-prop-entailment* $\mathcal{F}$*-def k-def*[*symmetric*] *subsequence-def*
using ‹$k \leq m$› **by** *auto*
fix *q* **assume** $q \in \mathcal{S}\ m$

have $S \models_P$ *Rep q*
proof (*cases* $q \in$ *?F*)
case *False*
moreover
from *False* **obtain** *j* **where** *state-rank q m = Some j* **and** $j \geq i$
using ‹$q \in \mathcal{S}\ m$› ‹*smallest-accepting-rank = Some i*› **by** *force*
then obtain *x* **where** *x*: $x \in$ *configuration q m token-run x m = q*
by *force*
moreover
from *x* **have** *token-succeeds x*
using *n-def-1*[*OF* ‹$n \leq m$›] ‹$q \in \mathcal{S}\ m$› **by** *blast*
ultimately
have $S \models_P af_G\ \psi\ (w\ [x \rightarrow m])$
using ‹$\bigwedge x.\ k \leq x \Longrightarrow x \leq$ *Suc m* $\Longrightarrow S \models_P af_G\ \psi\ (w\ [x \rightarrow$

$m]$)›[*of x*] *n-def-2*[*OF* - ‹$n \le m$›] **by** *fastforce*
thus *?thesis*
using *Rep-token-run-af* **unfolding** ‹*token-run x m* = *q*›[*symmetric*] *ltl-prop-equiv-def* **by** *simp*
qed (*insert* ‹$\mathcal{G} \subseteq S$›, *blast*)
}

moreover

{
assume $\bigwedge q.\ q \in \mathcal{S}\ m \Longrightarrow S \models_P Rep\ q$
hence $\bigwedge q.\ q \in\ ?F \Longrightarrow S \models_P Rep\ q$
by *simp*
have $\mathcal{G} \subseteq S$
proof
fix *x* **assume** $x \in \mathcal{G}$
with ‹*Only-G* $\mathcal{G}$› **show** $x \in S$
using ‹$\bigwedge q.\ q \in\ ?F \Longrightarrow S \models_P Rep\ q$›[*of Abs x*] **by** *auto*
qed

{
fix *x* **assume** $k \le x$ $x \le m$
define *q* **where** *q* = *token-run x m*

hence *token-succeeds x*
using *threshold-properties*[*OF* ‹*threshold* ψ *w* $\mathcal{G}$ = *Some k*›] ‹$x \ge k$› *Rep-token-run-af*
unfolding *token-succeeds-def ltl-prop-equiv-def* **by** *blast*
hence $q \in \mathcal{S}\ m$
using *n-def-1*[*OF* ‹$n \le m$›, *of q*] ‹$x \le m$›
unfolding *q-def configuration.simps* **by** *blast*
hence $S \models_P Rep\ q$
by (*rule* ‹$\bigwedge q.\ q \in \mathcal{S}\ m \Longrightarrow S \models_P Rep\ q$›)
hence $S \models_P af_G\ \psi\ (w\ [x \rightarrow m])$
using *Rep-token-run-af* **unfolding** *q-def ltl-prop-equiv-def* **by** *simp*
}
hence $\forall x \in (set\ (map\ (\lambda i.\ af_G\ \psi\ (w\ [i \rightarrow m]))\ [k..<Suc\ m])).\ S \models_P x$
unfolding *set-map set-upt* **by** *fastforce*
hence $S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ m$ **and** $\mathcal{G} \subseteq S$
unfolding $\mathcal{F}$*-def And-prop-entailment*[*of S*] *k-def*[*symmetric*]
using ‹$k \le m$› ‹$\mathcal{G} \subseteq S$› **by** *simp+*
}
ultimately

have $(S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ m \wedge \mathcal{G} \subseteq S) \longleftrightarrow (\forall\, q.\ q \in \mathcal{S}\ m \longrightarrow S \models_P Rep$ $q)$
by *blast*
}
thus *?thesis*
unfolding *MOST-nat-le* **by** *blast*
qed

lemma $\mathcal{F}$*-eq-*$\mathcal{S}$*-generalized*:
assumes *finite-*Σ: *finite* Σ
assumes *bounded-w*: *range* $w \subseteq \Sigma$
assumes *closed* $\mathcal{G}$ *w*
shows $\forall_\infty j.\ \forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow (\forall\, S.\ (S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j \wedge \mathcal{G} \subseteq S) \longleftrightarrow$ $(\forall\, q.\ q \in (\textit{ltl-FG-to-rabin-def}.\mathcal{S}_R\ \Sigma\ \psi\ \mathcal{G})\ w\ j \longrightarrow S \models_P Rep\ q))$
proof −
have *Only-G* $\mathcal{G}$ **and** *finite* $\mathcal{G}$
using *assms* **by** *simp+*
show *?thesis*
using *almost-all-commutative′′*[*OF* ‹*finite* $\mathcal{G}$› ‹*Only-G* $\mathcal{G}$›] $\mathcal{F}$*-eq-*$\mathcal{S}$[*OF* *assms*] **by** *simp*
qed

13.3 Product of Secondary Automata

context
fixes
Σ :: *′a set set*
begin

fun *product-initial-state* :: *′a set* ⇒ (*′a* ⇒ *′b*) ⇒ (*′a* ⇀ *′b*) (‹$\iota_\times$›)
where
$\iota_\times\ K\ q_m = (\lambda k.\ \textit{if}\ k \in K\ \textit{then Some}\ (q_m\ k)\ \textit{else None})$

fun *combine-pairs* :: ((*′a*, *′b*) *transition set* × (*′a*, *′b*) *transition set*) *set* ⇒ ((*′a*, *′b*) *transition set* × (*′a*, *′b*) *transition set set*)
where
combine-pairs P = ($\bigcup$(*fst* ‘ *P*), *snd* ‘ *P*)

fun *combine-pairs′* :: ((*′a ltl* ⇒ (*′a ltl-prop-equiv-quotient* ⇒ *nat option*) *option*, *′a set*) *transition set* × (*′a ltl* ⇒ (*′a ltl-prop-equiv-quotient* ⇒ *nat option*) *option*, *′a set*) *transition set*) *set* ⇒ ((*′a ltl* ⇒ (*′a ltl-prop-equiv-quotient* ⇒ *nat option*) *option*, *′a set*) *transition set* × (*′a ltl* ⇒ (*′a ltl-prop-equiv-quotient* ⇒ *nat option*) *option*, *′a set*) *transition set set*)
where

$combine\text{-}pairs'\ P = (\bigcup(fst\ `\ P),\ snd\ `\ P)$

lemma *combine-pairs-prop*:
$(\forall P \in \mathcal{P}.\ accepting\text{-}pair_R\ \delta\ q_0\ P\ w) = accepting\text{-}pair_{GR}\ \delta\ q_0\ (combine\text{-}pairs\ \mathcal{P})\ w$
by *auto*

lemma *combine-pairs2*:
$combine\text{-}pairs\ \mathcal{P} \in \alpha \Longrightarrow (\bigwedge P.\ P \in \mathcal{P} \Longrightarrow accepting\text{-}pair_R\ \delta\ q_0\ P\ w\ ) \Longrightarrow accept_{GR}\ (\delta,\ q_0,\ \alpha)\ w$
using *combine-pairs-prop*[*of* $\mathcal{P}\ \delta\ q_0\ w$] **by** *fastforce*

lemma *combine-pairs′-prop*:
$(\forall P \in \mathcal{P}.\ accepting\text{-}pair_R\ \delta\ q_0\ P\ w) = accepting\text{-}pair_{GR}\ \delta\ q_0\ (combine\text{-}pairs'\ \mathcal{P})\ w$
by *auto*

fun *ltl-FG-to-generalized-rabin* :: $'a\ ltl \Rightarrow ('a\ ltl \rightharpoonup 'a\ ltl_P \rightharpoonup nat,\ 'a\ set)\ generalized\text{-}rabin\text{-}automaton\ (\langle\mathcal{P}\rangle)$
where
$ltl\text{-}FG\text{-}to\text{-}generalized\text{-}rabin\ \varphi = ($
$\Delta_\times\ (\lambda\chi.\ ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def.\delta_R\ \Sigma\ (theG\ \chi)),$
$\iota_\times\ (\mathbf{G}\ (G\ \varphi))\ (\lambda\chi.\ ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def.q_R\ (theG\ \chi)),$
$\{combine\text{-}pairs'\ \{embed\text{-}pair\ \chi\ (ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def.Acc_R\ \Sigma\ (theG\ \chi)\ \mathcal{G}\ (\pi\ \chi))\ |\ \chi.\ \chi \in \mathcal{G}\}$
$|\ \mathcal{G}\ \pi.\ \mathcal{G} \subseteq \mathbf{G}\ (G\ \varphi) \wedge G\ \varphi \in \mathcal{G} \wedge (\forall\chi.\ \pi\ \chi < ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def.max\text{-}rank_R\ \Sigma\ (theG\ \chi))\})$

context
assumes
finite-Σ: *finite* Σ
begin

lemma *ltl-FG-to-generalized-rabin-wellformed*:
$finite\ (reach\ \Sigma\ (fst\ (\mathcal{P}\ \varphi))\ (fst\ (snd\ (\mathcal{P}\ \varphi))))$
proof $(cases\ \Sigma = \{\})$
case *False*
have $finite\ (reach\ \Sigma\ (\Delta_\times\ (\lambda\chi.\ ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def.\delta_R\ \Sigma\ (theG\ \chi)))\ (fst\ (snd\ (\mathcal{P}\ \varphi))))$
proof (*rule finite-reach-product*, *goal-cases*)
case *1*
show *?case*
using *G-nested-finite*(*1*) **by** (*auto simp add*: *dom-def LTL-Rabin.product-initial-state.simps*)

next
case (*2 x*)
hence *the* (*fst* (*snd* ($\mathcal{P}$ φ)) *x*) = *ltl-FG-to-rabin-def.q*$_R$ (*theG x*)
by (*auto simp add*: *LTL-Rabin.product-initial-state.simps*)
thus *?case*
using *ltl-FG-to-rabin.wellformed-*$\mathcal{R}$[*unfolded ltl-FG-to-rabin-def*, *of* {} - Σ *theG x*] *finite-*Σ *False* **by** *fastforce*
qed
thus *?thesis*
by *fastforce*
qed (*simp add*: *reach-def*)

theorem *ltl-FG-to-generalized-rabin-correct*:
assumes *range w* $\subseteq$ Σ
shows *w* $\models$ *F G* φ = *accept*$_{GR}$ ($\mathcal{P}$ φ) *w*
(**is** *?lhs* = *?rhs*)
proof
define *r* **where** *r* = *run*$_t$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) *w*

have [*intro*]: $\bigwedge$*i*. *w i* $\in$ Σ **and** $\Sigma \neq \{\}$
using *assms* **by** *auto*

{
let *?S* = (*reach* Σ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ)))) $\times$ Σ $\times$ (*reach* Σ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))))

have $\bigwedge$*n*. *r n* $\in$ *?S*
unfolding *run*$_t$*.simps run-foldl reach-foldl-def*[*OF* ‹$\Sigma \neq \{\}$›] *r-def* **by** *fastforce*
hence *range r* $\subseteq$ *?S* **and** *finite ?S*
using *ltl-FG-to-generalized-rabin-wellformed assms* ‹*finite* Σ› **by** (*blast*, *fast*)
}
hence *finite* (*range r*)
by (*blast dest*: *finite-subset*)

{
assume *?lhs*
then obtain $\mathcal{G}$ **where** $\mathcal{G}$ $\subseteq$ **G** (*G* φ) **and** *G* φ $\in$ $\mathcal{G}$ **and** $\forall\psi$. *G* ψ $\in$ $\mathcal{G}$ $\longrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*
unfolding *ltl-to-rabin-correct*[*OF* ‹*finite* Σ› ‹*range w* $\subseteq$ Σ›] **unfolding** *ltl-to-rabin.simps* **by** *auto*

note $\mathcal{G}$*-properties*[*OF* ‹$\mathcal{G}$ $\subseteq$ **G** (*G* φ)›]

hence *ltl-FG-to-rabin* Σ $\mathcal{G}$ *w*
using ‹*finite* Σ› ‹*range w* $\subseteq$ Σ› **unfolding** *ltl-FG-to-rabin-def* **by** *auto*

define π **where** π ψ =
(*if* $\psi \in \mathcal{G}$ *then the* (*ltl-FG-to-rabin-def.smallest-accepting-rank*$_R$ Σ (*theG* ψ) $\mathcal{G}$ *w*) *else 0*)
for ψ
let *?P'* = $\{\upharpoonright_\chi$ (*ltl-FG-to-rabin-def.Acc*$_R$ Σ (*theG* χ) $\mathcal{G}$ (π χ)) | χ. $\chi \in \mathcal{G}\}$

have $\forall$ *P* $\in$ *?P'*. *accepting-pair*$_R$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) *P w*
proof
fix *P*
assume *P* $\in$ *?P'*
then obtain χ **where** *P-def*: *P* = $\upharpoonright_\chi$ (*ltl-FG-to-rabin-def.Acc*$_R$ Σ (*theG* χ) $\mathcal{G}$ (π χ))
and $\chi \in \mathcal{G}$
by *blast*
hence $\exists \chi'$. χ = *G* χ'
using ‹$\mathcal{G} \subseteq$ **G** (*G* φ)› *G-nested-propos-alt-def* **by** *auto*

interpret *ltl-FG-to-rabin* Σ *theG* χ $\mathcal{G}$ *w*
by (*insert* ‹*ltl-FG-to-rabin* Σ $\mathcal{G}$ *w*›)

define r_χ **where** r_χ = *run*$_t$ $\delta_\mathcal{R}$ $q_\mathcal{R}$ *w*

moreover

have *accept* **and** *accept*$_R$ ($\delta_\mathcal{R}$, $q_\mathcal{R}$, {*Acc*$_\mathcal{R}$ *j* | *j*. *j* < *max-rank*}) *w*
using ‹$\chi \in \mathcal{G}$› ‹$\exists \chi'$. χ = *G* χ'› ‹$\forall \psi$. *G* $\psi \in \mathcal{G} \longrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*›
using *mojmir-accept-iff-rabin-accept* **by** *auto*

hence *smallest-accepting-rank*$_\mathcal{R}$ = *Some* (π χ)
unfolding π*-def smallest-accepting-rank-def Mojmir-rabin-smallest-accepting-rank*[*symmetric*]

using ‹$\chi \in \mathcal{G}$› **by** *simp*
hence *accepting-pair*$_R$ $\delta_\mathcal{R}$ $q_\mathcal{R}$ (*Acc*$_\mathcal{R}$ (π χ)) *w*
using ‹*accept*$_R$ ($\delta_\mathcal{R}$, $q_\mathcal{R}$, {*Acc*$_\mathcal{R}$ *j* | *j*. *j* < *max-rank*}) *w*› *LeastI*[*of* λi. *accepting-pair*$_R$ $\delta_\mathcal{R}$ $q_\mathcal{R}$ (*Acc*$_\mathcal{R}$ *i*) *w*]
by (*auto simp add*: *smallest-accepting-rank*$_\mathcal{R}$*-def*)

ultimately

have *limit* $r_\chi \cap$ *fst* (*Acc*$_\mathcal{R}$ (π χ)) = {} **and** *limit* $r_\chi \cap$ *snd* (*Acc*$_\mathcal{R}$ (π χ)) $\neq$ {}
by *simp+*

moreover

have *1*: ($\iota_\times$ (**G** (G φ)) ($\lambda\chi$. *ltl-FG-to-rabin-def.q*$_R$ (*theG* χ))) χ = *Some* $q_\mathcal{R}$
using ‹$\chi \in \mathcal{G}$› ‹$\mathcal{G} \subseteq$ **G** (G φ)› **by** (*simp add*: *LTL-Rabin.product-initial-state.simps subset-iff*)
have *2*: *finite* (*range* (*run*$_t$
($\Delta_\times$ ($\lambda\chi$. *ltl-FG-to-rabin-def.δ_R* Σ (*theG* χ)))
($\iota_\times$ (**G** (G φ)) ($\lambda\chi$. *ltl-FG-to-rabin-def.q*$_R$ (*theG* χ)))
w))
using ‹*finite* (*range* *r*)›[*unfolded r-def*] **by** *simp*

ultimately
have *limit r* $\cap$ *fst P* = {} **and** *limit r* $\cap$ *snd P* $\neq$ {}
using *product-run-embed-limit-finiteness*[*OF 1 2*]
unfolding *r-def* r_χ*-def P-def* **by** *auto*
thus *accepting-pair*$_R$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) *P w*
unfolding *P-def r-def* **by** *simp*
qed
hence *accepting-pair*$_{GR}$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) (*combine-pairs′ ?P′*) *w*
using *combine-pairs′-prop* **by** *blast*
moreover
{
fix ψ
assume $\psi \in \mathcal{G}$
hence $\exists\,\chi.\ \psi = G\ \chi$
using ‹$\mathcal{G} \subseteq$ **G** (G φ)› *G-nested-propos-alt-def* **by** *auto*

interpret *ltl-FG-to-rabin* Σ *theG* ψ $\mathcal{G}$ *w*
by (*insert* ‹*ltl-FG-to-rabin* Σ $\mathcal{G}$ *w*›)

have *accept*
using ‹$\psi \in \mathcal{G}$› ‹$\exists\,\chi.\ \psi = G\ \chi$› ‹$\forall\,\psi.\ G\ \psi \in \mathcal{G} \longrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*› *mojmir-accept-iff-rabin-accept* **by** *auto*
then obtain *i* **where** *smallest-accepting-rank* = *Some i*
unfolding *smallest-accepting-rank-def* **by** *fastforce*
hence π ψ < *max-rank*$_R$
using *smallest-accepting-rank-properties* π*-def* ‹$\psi \in \mathcal{G}$› **by** *auto*
}

hence $\bigwedge\chi.\ \pi\ \chi < $ *ltl-FG-to-rabin-def.max-rank*$_R$ Σ (*theG* χ)
unfolding π*-def* **using** *ltl-FG-to-rabin.max-rank-lowerbound*[*OF* ‹*ltl-FG-to-rabin* Σ $\mathcal{G}$ *w*›] **by** *force*
hence *combine-pairs'* *?P'* $\in$ *snd* (*snd* ($\mathcal{P}$ φ))
using ‹$\mathcal{G} \subseteq \mathbf{G}$ (G φ)› ‹G $\varphi \in \mathcal{G}$› **by** *auto*
ultimately
show *?rhs*
unfolding *accept*$_{GR}$*-simp2* *ltl-FG-to-generalized-rabin.simps* *fst-conv* *snd-conv* **by** *blast*
}

{
assume *?rhs*
then obtain $\mathcal{G}$ π P **where** P = *combine-pairs'* {$\rceil_\chi$ (*ltl-FG-to-rabin-def.Acc*$_R$ Σ (*theG* χ) $\mathcal{G}$ (π χ)) | χ. $\chi \in \mathcal{G}$} (**is** P = *combine-pairs'* *?P'*)
and *accepting-pair*$_{GR}$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) P w
and $\mathcal{G} \subseteq \mathbf{G}$ (G φ) **and** G $\varphi \in \mathcal{G}$ **and** $\bigwedge\chi.\ \pi\ \chi <$ *ltl-FG-to-rabin-def.max-rank*$_R$ Σ (*theG* χ)
unfolding *accept*$_{GR}$*-def* **by** *auto*
moreover
hence P'*-def*: $\bigwedge P.\ P \in$ *?P'* $\Longrightarrow$ *accepting-pair*$_R$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) P w
using *combine-pairs'-prop* **by** *meson*
note $\mathcal{G}$*-properties*[*OF* ‹$\mathcal{G} \subseteq \mathbf{G}$ (G φ)›]
hence *ltl-FG-to-rabin* Σ $\mathcal{G}$ w
using ‹*finite* Σ› ‹*range* $w \subseteq \Sigma$› **unfolding** *ltl-FG-to-rabin-def* **by** *auto*
have $\forall\psi.\ G\ \psi \in \mathcal{G} \longrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) w
proof (*rule+*)
fix ψ
assume G $\psi \in \mathcal{G}$
define χ **where** $\chi = G\ \psi$
define P **where** $P = \rceil_\chi$ (*ltl-FG-to-rabin-def.Acc*$_R$ Σ ψ $\mathcal{G}$ (π χ))
hence $\chi \in \mathcal{G}$ **and** *theG* $\chi = \psi$
using χ*-def* ‹G $\psi \in \mathcal{G}$› **by** *simp+*
hence $P \in$ *?P'*
unfolding *P-def* **by** *auto*
hence *accepting-pair*$_R$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) P w
using P'*-def* **by** *blast*

interpret *ltl-FG-to-rabin* Σ ψ $\mathcal{G}$ w
by (*insert* ‹*ltl-FG-to-rabin* Σ $\mathcal{G}$ *w*›)

define r_χ **where** r_χ = *run*$_t$ $\delta_\mathcal{R}$ $q_\mathcal{R}$ w

have *limit r* ∩ *fst P* = {} **and** *limit r* ∩ *snd P* ≠ {}
using ‹*accepting-pair*$_R$ (*fst* ($\mathcal{P}$ φ)) (*fst* (*snd* ($\mathcal{P}$ φ))) *P w*›
unfolding *r-def accepting-pair*$_R$*-def* **by** *metis+*

moreover

have *1*: ($\iota_\times$ (**G** (*G* φ)) ($\lambda\chi$. *ltl-FG-to-rabin-def*.q_R (*theG* χ))) (*G* ψ) = *Some* $q_\mathcal{R}$
using ‹*G* $\psi \in \mathcal{G}$› ‹$\mathcal{G} \subseteq$ **G** (*G* φ)› **by** (*auto simp add*: *LTL-Rabin.product-initial-state.simps subset-iff*)
have *2*: *finite* (*range* (*run*$_t$ ($\Delta_\times$ ($\lambda\chi$. *ltl-FG-to-rabin-def*.δ_R Σ (*theG* χ))) ($\iota_\times$ (**G** (*G* φ)) ($\lambda\chi$. *ltl-FG-to-rabin-def*.q_R (*theG* χ))) *w*))
using ‹*finite* (*range r*)›[*unfolded r-def*] **by** *simp*
have $\bigwedge S$. *limit r* ∩ ($\bigcup$ ($\upharpoonleft_\chi$ ‘ *S*)) = {} ⟷ *limit* r_χ ∩ *S* = {}
using *product-run-embed-limit-finiteness*[*OF 1 2*] **by** (*simp add*: *r-def* r_χ*-def* χ*-def*)

ultimately
have *limit* r_χ ∩ *fst* (*Acc*$_\mathcal{R}$ (π χ)) = {} **and** *limit* r_χ ∩ *snd* (*Acc*$_\mathcal{R}$ (π χ)) ≠ {}
unfolding *P-def fst-conv snd-conv embed-pair.simps* **by** *meson+*
hence *accepting-pair*$_R$ $\delta_\mathcal{R}$ $q_\mathcal{R}$ (*Acc*$_\mathcal{R}$ (π χ)) *w*
unfolding r_χ*-def* **by** *simp*
hence *accept*$_R$ ($\delta_\mathcal{R}$, $q_\mathcal{R}$, {*Acc*$_\mathcal{R}$ *j* | *j*. *j* < *max-rank*}) *w*
using ‹$\bigwedge\chi$. π χ < *ltl-FG-to-rabin-def*.*max-rank*$_R$ Σ (*theG* χ)› ‹*theG* χ = ψ›
unfolding *accept*$_R$*-simp accepting-pair*$_R$*-def fst-conv snd-conv* **by** *blast*
thus *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*
by *simp*
qed
ultimately
show *?lhs*
unfolding *ltl-to-rabin-correct*[*OF* ‹*finite* Σ› *assms*] **by** *auto*
}
qed

end

end

13.4 Automaton Template

locale *ltl-to-rabin-base-def* =

fixes
  δ :: *'a ltl*$_P$ $\Rightarrow$ *'a set* $\Rightarrow$ *'a ltl*$_P$
fixes
  δ_M :: *'a ltl*$_P$ $\Rightarrow$ *'a set* $\Rightarrow$ *'a ltl*$_P$
fixes
  q_0 :: *'a ltl* $\Rightarrow$ *'a ltl*$_P$
fixes
  q_{0M} :: *'a ltl* $\Rightarrow$ *'a ltl*$_P$
fixes
  M-fin :: (*'a ltl* $\rightharpoonup$ *nat*) $\Rightarrow$ (*'a ltl*$_P$ $\times$ (*'a ltl* $\rightharpoonup$ *'a ltl*$_P$ $\rightharpoonup$ *nat*), *'a set*) *transition set*
begin

— Transition Function and Initial State

fun *delta*
where
  delta $\Sigma = \delta \times \Delta_\times$ (*semi-mojmir-def.step* Σ δ_M *o* q_{0M} *o theG*)

fun *initial*
where
  initial φ = (q_0 φ, $\iota_\times$ (**G** φ) (*semi-mojmir-def.initial o* q_{0M} *o theG*))

— Acceptance Condition

definition *max-rank-of*
where
  max-rank-of Σ $\psi \equiv$ *semi-mojmir-def.max-rank* Σ δ_M (q_{0M} (*theG* ψ))

fun *Acc-fin*
where
  Acc-fin Σ π χ = $\bigcup$(*embed-transition-snd* ‘ $\bigcup$(*embed-transition* χ ‘ (*mojmir-to-rabin-def.fail*$_R$ Σ δ_M (q_{0M} (*theG* χ)) {*q*. *dom* π $\uparrow\models_P$ *q*} $\cup$ *mojmir-to-rabin-def.merge*$_R$ δ_M (q_{0M} (*theG* χ)) {*q*. *dom* π $\uparrow\models_P$ *q*} (*the* (π χ)))))

fun *Acc-inf*
where
  Acc-inf π χ = $\bigcup$(*embed-transition-snd* ‘ $\bigcup$(*embed-transition* χ ‘ (*mojmir-to-rabin-def.succeed*$_R$ δ_M (q_{0M} (*theG* χ)) {*q*. *dom* π $\uparrow\models_P$ *q*} (*the* (π χ)))))

abbreviation *Acc*
where

$Acc\ \Sigma\ \pi\ \chi \equiv (Acc\text{-}fin\ \Sigma\ \pi\ \chi,\ Acc\text{-}inf\ \pi\ \chi)$

fun *rabin-pairs* :: $'a\ set\ set \Rightarrow\ 'a\ ltl \Rightarrow ('a\ ltl_P \times ('a\ ltl \rightharpoonup\ 'a\ ltl_P \rightharpoonup nat),\ 'a\ set)$ *generalized-rabin-condition*
where
rabin-pairs $\Sigma\ \varphi = \{(M\text{-}fin\ \pi \cup \bigcup\{Acc\text{-}fin\ \Sigma\ \pi\ \chi \mid \chi.\ \chi \in dom\ \pi\},\ \{Acc\text{-}inf\ \pi\ \chi \mid \chi.\ \chi \in dom\ \pi\})$
$\mid \pi.\ dom\ \pi \subseteq \mathbf{G}\ \varphi \wedge (\forall \chi \in dom\ \pi.\ the\ (\pi\ \chi) < max\text{-}rank\text{-}of\ \Sigma\ \chi)\}$

fun *ltl-to-generalized-rabin* :: $'a\ set\ set \Rightarrow\ 'a\ ltl \Rightarrow ('a\ ltl_P \times ('a\ ltl \rightharpoonup\ 'a\ ltl_P \rightharpoonup nat),\ 'a\ set)$ *generalized-rabin-automaton* $(\langle\mathcal{A}\rangle)$
where
$\mathcal{A}\ \Sigma\ \varphi = (delta\ \Sigma,\ initial\ \varphi,\ rabin\text{-}pairs\ \Sigma\ \varphi)$

end

locale *ltl-to-rabin-base* = *ltl-to-rabin-base-def* +
fixes
Σ :: $'a\ set\ set$
fixes
w :: $'a\ set\ word$
assumes
finite-Σ: *finite* Σ
assumes
bounded-w: $range\ w \subseteq \Sigma$
assumes
M-fin-monotonic: $dom\ \pi = dom\ \pi' \Longrightarrow (\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow the\ (\pi\ \chi) \leq the\ (\pi'\ \chi)) \Longrightarrow M\text{-}fin\ \pi \subseteq M\text{-}fin\ \pi'$
assumes
finite-reach': $finite\ (reach\ \Sigma\ \delta\ (q_0\ \varphi))$
assumes
mojmir-to-rabin: $Only\text{-}G\ \mathcal{G} \Longrightarrow mojmir\text{-}to\text{-}rabin\ \Sigma\ \delta_M\ (q_{0M}\ \psi)\ w\ \{q.\ \mathcal{G} \uparrow\models_P q\}$
begin

lemma *semi-mojmir*:
$semi\text{-}mojmir\ \Sigma\ \delta_M\ (q_{0M}\ \psi)\ w$
using *mojmir-to-rabin*[*of* {}] **by** (*simp add*: *mojmir-to-rabin-def mojmir-def*)

lemma *finite-reach*:
$finite\ (reach\ \Sigma\ (delta\ \Sigma)\ (initial\ \varphi))$
apply $(cases\ \Sigma = \{\})$
apply (*simp add*: *reach-def*)

apply (*simp only*: *ltl-to-rabin-base-def.initial.simps ltl-to-rabin-base-def.delta.simps*)
apply (*rule finite-reach-simple-product*[*OF finite-reach′ finite-reach-product*])
apply (*insert mojmir-to-rabin*[*of* {}, *unfolded mojmir-to-rabin-def mojmir-def*])
apply (*auto simp add*: *dom-def intro*: *G-nested-finite semi-mojmir.wellformed-*$\mathcal{R}$)

done

lemma *run-limit-not-empty*:
limit (run_t (*delta* Σ) (*initial* φ) *w*) ≠ {}
by (*metis emptyE finite-*Σ *limit-nonemptyE finite-reach bounded-w* run_t*-finite*)

lemma *run-properties*:
fixes φ
defines $r \equiv run\ (delta\ \Sigma)\ (initial\ \varphi)\ w$
shows $fst\ (r\ i) = foldl\ \delta\ (q_0\ \varphi)\ (w\ [0 \rightarrow i])$
and $\bigwedge\chi\ q.\ \chi \in \mathbf{G}\ \varphi \Longrightarrow the\ (snd\ (r\ i)\ \chi)\ q = semi\text{-}mojmir\text{-}def.state\text{-}rank\ \Sigma\ \delta_M\ (q_{0M}\ (theG\ \chi))\ w\ q\ i$
proof −
have *sm*: $\bigwedge\psi.\ semi\text{-}mojmir\ \Sigma\ \delta_M\ (q_{0M}\ \psi)\ w$
using *mojmir-to-rabin*[*of* {}] **unfolding** *mojmir-to-rabin-def mojmir-def* **by** *simp*
have $r\ i = (foldl\ \delta\ (q_0\ \varphi)\ (w\ [0 \rightarrow i]),$
$\lambda\chi.\ if\ \chi \in \mathbf{G}\ \varphi\ then\ Some\ (\lambda\psi.\ foldl\ (semi\text{-}mojmir\text{-}def.step\ \Sigma\ \delta_M\ (q_{0M}\ (theG\ \chi)))\ (semi\text{-}mojmir\text{-}def.initial\ (q_{0M}\ (theG\ \chi)))\ (map\ w\ [0..< i])\ \psi)\ else\ None)$
proof (*induction i*)
case (*Suc i*)
show *?case*
unfolding *r-def run-foldl upt-Suc less-eq-nat.simps if-True map-append foldl-append*
unfolding *Suc*[*unfolded r-def run-foldl*] *subsequence-def* **by** *auto*
qed (*auto simp add*: *subsequence-def r-def*)
hence *state-run*: $r\ i = (foldl\ \delta\ (q_0\ \varphi)\ (w\ [0 \rightarrow i]),$
$\lambda\chi.\ if\ \chi \in \mathbf{G}\ \varphi\ then\ Some\ (\lambda\psi.\ semi\text{-}mojmir\text{-}def.state\text{-}rank\ \Sigma\ \delta_M\ (q_{0M}\ (theG\ \chi))\ w\ \psi\ i)\ else\ None)$
unfolding *semi-mojmir.state-rank-step-foldl*[*OF sm*] *r-def* **by** *simp*

show $fst\ (r\ i) = foldl\ \delta\ (q_0\ \varphi)\ (w\ [0 \rightarrow i])$
using *state-run* **by** *fastforce*
show $\bigwedge\chi\ q.\ \chi \in \mathbf{G}\ \varphi \Longrightarrow the\ (snd\ (r\ i)\ \chi)\ q = semi\text{-}mojmir\text{-}def.state\text{-}rank\ \Sigma\ \delta_M\ (q_{0M}\ (theG\ \chi))\ w\ q\ i$
unfolding *state-run* **by** *force*

qed

lemma $accept_{GR}$*-I*:
assumes $accept_{GR}$ ($\mathcal{A}$ Σ φ) *w*
obtains π **where** *dom* $\pi \subseteq$ **G** φ
and $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *the* (π χ) < *max-rank-of* Σ χ
and *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*M-fin* π, *UNIV*) *w*
and $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π χ) *w*
proof −
from *assms* **obtain** *P* **where** $P \in$ *rabin-pairs* Σ φ **and** *accepting-pair*$_{GR}$ (*delta* Σ) (*initial* φ) *P w*
unfolding $accept_{GR}$*-def ltl-to-generalized-rabin.simps fst-conv snd-conv* **by** *blast*
moreover
then obtain π **where** *dom* $\pi \subseteq$ **G** φ **and** $\forall\chi \in$ *dom* π. *the* (π χ) < *max-rank-of* Σ χ
and *P-def*: *P* = (*M-fin* π $\cup$ $\bigcup${*Acc-fin* Σ π χ | χ. $\chi \in$ *dom* π}, {*Acc-inf* π χ | χ. $\chi \in$ *dom* π})
by *auto*
have *limit* (*run*$_t$ (*delta* Σ) (*initial* φ) *w*) $\cap$ *UNIV* $\neq$ {}
using *run-limit-not-empty assms* **by** *simp*
ultimately
have *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*M-fin* π, *UNIV*) *w*
and $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π χ) *w*
unfolding *P-def accepting-pair*$_{GR}$*-simp accepting-pair*$_R$*-simp* **by** *blast*+

thus *?thesis*
using *that* ‹*dom* $\pi \subseteq$ **G** φ› ‹$\forall\chi \in$ *dom* π. *the* (π χ) < *max-rank-of* Σ χ› **by** *blast*
qed

context
fixes
φ :: *'a ltl*
begin

context
fixes
ψ :: *'a ltl*
fixes
π :: *'a ltl* ⇀ *nat*
assumes

$G\ \psi \in dom\ \pi$
assumes
$dom\ \pi \subseteq \mathbf{G}\ \varphi$
begin

interpretation $\mathfrak{M}$: *mojmir-to-rabin* $\Sigma\ \delta_M\ q_{0M}\ \psi\ w\ \{q.\ dom\ \pi \uparrow\models_P q\}$
by (*metis mojmir-to-rabin* ‹$dom\ \pi \subseteq \mathbf{G}\ \varphi$› $\mathcal{G}$*-elements*)

lemma *Acc-property*:
accepting-pair$_R$ (*delta* Σ) (*initial* φ) (*Acc* $\Sigma\ \pi$ ($G\ \psi$)) $w \longleftrightarrow$ *accepting-pair*$_R$ $\mathfrak{M}.\delta_{\mathcal{R}}$ $\mathfrak{M}.q_{\mathcal{R}}$ ($\mathfrak{M}.Acc_{\mathcal{R}}$ (*the* (π ($G\ \psi$)))) w
(**is** *?Acc* = *?Acc*$_{\mathcal{R}}$)
proof −
define $r\ r_\psi$ **where** $r = run_t$ (*delta* Σ) (*initial* φ) w **and** $r_\psi = run_t\ \mathfrak{M}.\delta_{\mathcal{R}}\ \mathfrak{M}.q_{\mathcal{R}}\ w$
hence *finite* (*range* r)
using run_t*-finite*[*OF finite-reach*] *bounded-w finite-*Σ
by (*blast dest*: *finite-subset*)

have $\bigwedge S.\ limit\ r_\psi \cap S = \{\} \longleftrightarrow limit\ r \cap \bigcup$(*embed-transition-snd* ‘ ($\bigcup$ ((*embed-transition* ($G\ \psi$)) ‘ S))) = {}
proof −
fix S
have *1*: *snd* (*initial* φ) ($G\ \psi$) = *Some* $\mathfrak{M}.q_{\mathcal{R}}$
using ‹$G\ \psi \in dom\ \pi$› ‹$dom\ \pi \subseteq \mathbf{G}\ \varphi$› **by** *auto*
have *2*: *finite* (*range* (run_t ($\Delta_\times$ (*semi-mojmir-def.step* $\Sigma\ \delta_M$ *o* q_{0M} *o* *theG*)) (*snd* (*initial* φ)) w))
using ‹*finite* (*range* r)› *r-def comp-apply*
by (*auto intro*: *product-run-finite-snd cong del*: *image-cong-simp*)
show *?thesis S*
unfolding *r-def* r_ψ*-def product-run-embed-limit-finiteness*[*OF 1 2*, *unfolded ltl.sel comp-def*, *symmetric*]
using *product-run-embed-limit-finiteness-snd*[*OF* ‹*finite* (*range r*)›[*unfolded r-def delta.simps initial.simps*]]
by (*auto simp del*: *simple-product.simps product.simps product-initial-state.simps simp add*: *comp-def cong del*: *SUP-cong-simp*)
qed
hence *limit* $r \cap$ *fst* (*Acc* $\Sigma\ \pi$ ($G\ \psi$)) = {} $\wedge$ *limit* $r \cap$ *snd* (*Acc* $\Sigma\ \pi$ ($G\ \psi$)) $\neq$ {}
$\longleftrightarrow$ *limit* $r_\psi \cap$ *fst* ($\mathfrak{M}.Acc_{\mathcal{R}}$ (*the* (π ($G\ \psi$)))) = {} $\wedge$ *limit* $r_\psi \cap$ *snd* ($\mathfrak{M}.Acc_{\mathcal{R}}$ (*the* (π ($G\ \psi$)))) $\neq$ {}
unfolding *fst-conv snd-conv* **by** *simp*
thus *?Acc* $\longleftrightarrow$ *?Acc*$_{\mathcal{R}}$
unfolding r_ψ*-def r-def accepting-pair*$_R$*-def* **by** *blast*

qed

lemma *Acc-to-rabin-accept*:
 $[\![$*accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π (G ψ)) *w*; *the* (π (G ψ)) $<$ $\mathfrak{M}$.*max-rank*$]\!]$ $\Longrightarrow$ *accept*$_R$ $\mathfrak{M}.\mathcal{R}$ *w*
 unfolding *Acc-property* **by** *auto*

lemma *Acc-to-mojmir-accept*:
 $[\![$*accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π (G ψ)) *w*; *the* (π (G ψ)) $<$ $\mathfrak{M}$.*max-rank*$]\!]$ $\Longrightarrow$ $\mathfrak{M}$.*accept*
 using *Acc-to-rabin-accept* **unfolding** $\mathfrak{M}$.*mojmir-accept-iff-rabin-accept* **by** *auto*

lemma *rabin-accept-to-Acc*:
 $[\![$*accept*$_R$ $\mathfrak{M}.\mathcal{R}$ *w*; π (G ψ) $=$ $\mathfrak{M}$.*smallest-accepting-rank*$]\!]$ $\Longrightarrow$ *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π (G ψ)) *w*
 unfolding *Acc-property* $\mathfrak{M}$.*Mojmir-rabin-smallest-accepting-rank*
 using $\mathfrak{M}$.*smallest-accepting-rank*$_{\mathcal{R}}$-*properties* $\mathfrak{M}$.*smallest-accepting-rank*$_{\mathcal{R}}$-*def*

 by (*metis* (*no-types*, *lifting*) *option.sel*)

lemma *mojmir-accept-to-Acc*:
 $[\![\mathfrak{M}$.*accept*; π (G ψ) $=$ $\mathfrak{M}$.*smallest-accepting-rank*$]\!]$ $\Longrightarrow$ *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π (G ψ)) *w*
 unfolding $\mathfrak{M}$.*mojmir-accept-iff-rabin-accept* **by** (*blast dest*: *rabin-accept-to-Acc*)

end

lemma *normalize-*π:
 assumes *dom-subset*: *dom* $\pi \subseteq$ **G** φ
 assumes $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *the* (π χ) $<$ *max-rank-of* Σ χ
 assumes *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*M-fin* π, *UNIV*) *w*
 assumes $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π χ) *w*
 obtains $\pi_{\mathcal{A}}$ **where** *dom* π $=$ *dom* $\pi_{\mathcal{A}}$
  and $\bigwedge\chi$. $\chi \in$ *dom* $\pi_{\mathcal{A}} \Longrightarrow \pi_{\mathcal{A}}$ χ $=$ *mojmir-def.smallest-accepting-rank* Σ δ_M (q_{0M} (*theG* χ)) *w* {*q*. *dom* $\pi_{\mathcal{A}} \uparrow\models_P q$}
  and *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*M-fin* $\pi_{\mathcal{A}}$, *UNIV*) *w*
  and $\bigwedge\chi$. $\chi \in$ *dom* $\pi_{\mathcal{A}} \Longrightarrow$ *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ $\pi_{\mathcal{A}}$ χ) *w*
proof −
 define $\mathcal{G}$ **where** $\mathcal{G}$ $=$ *dom* π
 note $\mathcal{G}$-*properties*[*OF dom-subset*]

define $\pi_{\mathcal{A}}$
where $\pi_{\mathcal{A}} = (\lambda\chi.$ *mojmir-def.smallest-accepting-rank* Σ δ_M $(q_{0M}$ *(theG* $\chi))$ *w* $\{q.$ *dom* π $\uparrow\models_P$ $q\})$ |' $\mathcal{G}$

moreover

{
fix χ **assume** $\chi \in$ *dom* π

interpret $\mathfrak{M}$: *mojmir-to-rabin* Σ δ_M q_{0M} *(theG* $\chi)$ *w* $\{q.$ *dom* π $\uparrow\models_P$ $q\}$
by (*metis mojmir-to-rabin* ‹*dom* $\pi \subseteq$ **G** φ› $\mathcal{G}$*-elements*)

from ‹$\chi \in$ *dom* π› **have** *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ π χ) *w*
using *assms*(*4*) **by** *blast*
hence *accepting-pair*$_R$ $\mathfrak{M}.\delta_{\mathcal{R}}$ $\mathfrak{M}.q_{\mathcal{R}}$ ($\mathfrak{M}.Acc_{\mathcal{R}}$ (*the* (π χ))) *w*
by (*metis* ‹$\chi \in$ *dom* π› *Acc-property*[*OF* - *dom-subset*] ‹*Only-G* (*dom* π)› *ltl.sel*(*8*))
moreover
hence *accept*$_R$ ($\mathfrak{M}.\delta_{\mathcal{R}}$, $\mathfrak{M}.q_{\mathcal{R}}$, $\{\mathfrak{M}.Acc_{\mathcal{R}}$ *j* | *j*. *j* < $\mathfrak{M}$.*max-rank*$\}$) *w*
using *assms*(*2*)[*OF* ‹$\chi \in$ *dom* π›] **unfolding** *max-rank-of-def* **by** *auto*
ultimately
have *the* ($\mathfrak{M}$.*smallest-accepting-rank*$_{\mathcal{R}}$) $\leq$ *the* (π χ) **and** $\mathfrak{M}$.*smallest-accepting-rank* $\neq$ *None*
using *Least-le*[*of* - *the* (π χ)] *assms*(*2*)[*OF* ‹$\chi \in$ *dom* π›] $\mathfrak{M}$.*mojmir-accept-iff-rabin-accept* *option.distinct*(*1*) $\mathfrak{M}$.*smallest-accepting-rank-def*
by (*simp add*: $\mathfrak{M}$.*smallest-accepting-rank*$_{\mathcal{R}}$*-def*)+
hence *the* ($\pi_{\mathcal{A}}$ χ) $\leq$ *the* (π χ) **and** $\chi \in$ *dom* $\pi_{\mathcal{A}}$
unfolding $\pi_{\mathcal{A}}$*-def* *dom-restrict* **using** *assms*(*2*) ‹$\chi \in$ *dom* π› **by** (*simp add*: $\mathfrak{M}$.*Mojmir-rabin-smallest-accepting-rank* $\mathcal{G}$*-def*, *subst dom-def*, *simp add*: $\mathcal{G}$*-def*)
}

hence *dom* π = *dom* $\pi_{\mathcal{A}}$
unfolding $\pi_{\mathcal{A}}$*-def* *dom-restrict* $\mathcal{G}$*-def* **by** *auto*

moreover

note $\mathcal{G}$*-properties*[*OF dom-subset*, *unfolded* ‹*dom* π = *dom* $\pi_{\mathcal{A}}$›]

have *M-fin* $\pi_{\mathcal{A}} \subseteq$ *M-fin* π
using ‹*dom* π = *dom* $\pi_{\mathcal{A}}$› **by** (*simp add*: *M-fin-monotonic* ‹$\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *the* ($\pi_{\mathcal{A}}$ χ) $\leq$ *the* (π χ)›)

hence *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*M-fin* $\pi_{\mathcal{A}}$, *UNIV*) *w*
using *assms* **unfolding** *accepting-pair*$_R$*-simp* **by** *blast*

moreover

— Goal 2
{
fix χ **assume** $\chi \in$ *dom* $\pi_{\mathcal{A}}$
hence $\chi = G$ (*theG* χ)
unfolding ‹*dom* π = *dom* $\pi_{\mathcal{A}}$›[*symmetric*] ‹*Only-G* (*dom* π)› **by** (*metis* ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)› ‹$\chi \in$ *dom* $\pi_{\mathcal{A}}$› *ltl.collapse*(*6*) *ltl.disc*(*58*))
moreover
hence *G* (*theG* χ) $\in$ *dom* $\pi_{\mathcal{A}}$
using ‹$\chi \in$ *dom* $\pi_{\mathcal{A}}$› **by** *simp*
moreover
hence *X*: *mojmir-def.accept* δ_M (q_{0M} (*theG* χ)) *w* {*q*. *dom* $\pi \uparrow\models_P q$}
using *assms*(*1*,*2*,*4*) ‹*dom* $\pi \subseteq$ **G** φ› *ltl.sel*(*8*) *Acc-to-mojmir-accept* ‹*dom* π = *dom* $\pi_{\mathcal{A}}$› **by** (*metis max-rank-of-def*)
have *Y*: $\pi_{\mathcal{A}}$ (*G theG* χ) = *mojmir-def.smallest-accepting-rank* Σ δ_M (q_{0M} (*theG* χ)) *w* {*q*. *dom* $\pi_{\mathcal{A}} \uparrow\models_P q$}
using ‹*G* (*theG* χ) $\in$ *dom* $\pi_{\mathcal{A}}$› ‹$\chi = G$ (*theG* χ)› $\pi_{\mathcal{A}}$*-def* ‹*dom* π = *dom* $\pi_{\mathcal{A}}$›[*symmetric*] **by** *simp*
ultimately
have *accepting-pair*$_R$ (*delta* Σ) (*initial* φ) (*Acc* Σ $\pi_{\mathcal{A}}$ χ) *w*
using *mojmir-accept-to-Acc*[*OF* ‹*G* (*theG* χ) $\in$ *dom* $\pi_{\mathcal{A}}$› ‹*dom* $\pi \subseteq$ **G** φ›[*unfolded* ‹*dom* π = *dom* $\pi_{\mathcal{A}}$›] *X*[*unfolded* ‹*dom* π = *dom* $\pi_{\mathcal{A}}$›] *Y*] **by** *simp*
}

ultimately

show *?thesis*
using *that*[*of* $\pi_{\mathcal{A}}$] *restrict-in* **unfolding** ‹*dom* π = *dom* $\pi_{\mathcal{A}}$› $\mathcal{G}$*-def*
by (*metis* (*no-types*, *lifting*))
qed

end

end

13.5 Generalized Deterministic Rabin Automaton

13.5.1 Definition

fun *M-fin* :: $('a\ ltl \rightharpoonup nat) \Rightarrow ('a\ ltl_P \times ('a\ ltl \rightharpoonup 'a\ ltl_P \rightharpoonup nat), 'a\ set)$ *transition set*
where
M-fin $\pi = \{((\varphi', m), \nu, p).$
$\neg(\forall S. (\forall \chi \in dom\ \pi.\ S \uparrow\models_P Abs\ \chi \wedge (\forall q. (\exists j \geq the\ (\pi\ \chi).\ the\ (m\ \chi)\ q = Some\ j) \longrightarrow S \uparrow\models_P \uparrow eval_G\ (dom\ \pi)\ q)) \longrightarrow S \uparrow\models_P \varphi')\}$

locale *ltl-to-rabin-af* = *ltl-to-rabin-base* $\uparrow af\ \uparrow af_G$ *Abs Abs M-fin* **begin**

abbreviation $\delta_{\mathcal{A}} \equiv$ *delta*
abbreviation $\iota_{\mathcal{A}} \equiv$ *initial*
abbreviation $Acc_{\mathcal{A}} \equiv Acc$
abbreviation $F_{\mathcal{A}} \equiv$ *rabin-pairs*
abbreviation $\mathcal{A} \equiv$ *ltl-to-generalized-rabin*

13.5.2 Correctness Theorem

theorem *ltl-to-generalized-rabin-correct*:
$w \models \varphi = accept_{GR}$ (*ltl-to-generalized-rabin* $\Sigma\ \varphi$) w
(**is** *?lhs* = *?rhs*)
proof
let $?\Delta = \delta_{\mathcal{A}}\ \Sigma$
let $?q_0 = \iota_{\mathcal{A}}\ \varphi$
let $?F = F_{\mathcal{A}}\ \Sigma\ \varphi$

— Preliminary facts needed by both proof directions
define r **where** $r = run_t\ ?\Delta\ ?q_0\ w$
have *r-alt-def'*: $\bigwedge i.\ fst\ (fst\ (r\ i)) = Abs\ (af\ \varphi\ (w\ [0 \rightarrow i]))$
using *run-properties*(*1*) **unfolding** *r-def* run_t*.simps fst-conv*
by (*metis af-abs.f-foldl-abs.abs-eq af-abs.f-foldl-abs-alt-def af-letter-abs-def*)

have *r-alt-def''*: $\bigwedge \chi\ i\ q.\ \chi \in \mathbf{G}\ \varphi \Longrightarrow the\ (snd\ (fst\ (r\ i))\ \chi)\ q =$ *semi-mojmir-def.state-rank* $\Sigma\ \uparrow af_G(Abs\ (theG\ \chi))\ w\ q\ i$
using *run-properties*(*2*) *r-def* **by** *force*
have φ'*-def*: $\bigwedge i.\ af\ \varphi\ (w\ [0 \rightarrow i]) \equiv_P Rep\ (fst\ (fst\ (r\ i)))$
by (*metis r-alt-def' Quotient3-ltl-prop-equiv-quotient ltl-prop-equiv-quotient.abs-eq-iff Quotient3-abs-rep*)

have *finite* (*range r*)
using run_t*-finite*[*OF finite-reach*] *bounded-w finite-*Σ
by (*simp add*: *r-def*)

— Assuming $w \models \varphi$ holds, we prove that $\mathcal{A}\ \Sigma\ \varphi$ accepts w
{
assume *?lhs*
then obtain $\mathcal{G}$ **where** $\mathcal{G} \subseteq \mathbf{G}\ \varphi$ **and** $accept_M\ \varphi\ \mathcal{G}\ w$ **and** *closed* $\mathcal{G}\ w$
unfolding *ltl-logical-characterization* **by** *blast*

note $\mathcal{G}$*-properties*[*OF* ‹$\mathcal{G} \subseteq \mathbf{G}\ \varphi$›]
hence *ltl-FG-to-rabin* $\Sigma\ \mathcal{G}\ w$
using *finite*-Σ *bounded-w* **unfolding** *ltl-FG-to-rabin-def* **by** *auto*

define π
where $\pi\ \chi = ($*if* $\chi \in \mathcal{G}$ *then* (*ltl-FG-to-rabin-def.smallest-accepting-rank*$_R$
Σ (*theG* χ) $\mathcal{G}\ w$) *else None*)
for χ

have $\mathfrak{M}$*-accept*: $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow$ *ltl-FG-to-rabin-def.accept*$_R{}'$ $\psi\ \mathcal{G}\ w$
using ‹*closed* $\mathcal{G}\ w$› ‹*ltl-FG-to-rabin* $\Sigma\ \mathcal{G}\ w$› *ltl-FG-to-rabin.ltl-to-rabin-correct-exposed′*
by *blast*
have $\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow accept_R$ (*ltl-to-rabin* $\Sigma\ \psi\ \mathcal{G}$) w
using ‹*closed* $\mathcal{G}\ w$› **unfolding** *ltl-FG-to-rabin.ltl-to-rabin-correct-exposed*[*OF*
‹*ltl-FG-to-rabin* $\Sigma\ \mathcal{G}\ w$›] **by** *simp*

{
fix ψ **assume** $G\ \psi \in \mathcal{G}$
interpret $\mathfrak{M}$: *ltl-FG-to-rabin* $\Sigma\ \psi\ \mathcal{G}\ w$
by (*insert* ‹*ltl-FG-to-rabin* $\Sigma\ \mathcal{G}\ w$›)
obtain i **where** $\mathfrak{M}$*.smallest-accepting-rank* = *Some* i
using $\mathfrak{M}$*-accept*[*OF* ‹$G\ \psi \in \mathcal{G}$›]
unfolding $\mathfrak{M}$*.smallest-accepting-rank-def* **by** *fastforce*
hence *the* $(\pi\ (G\ \psi)) <$ $\mathfrak{M}$*.max-rank* **and** $\pi\ (G\ \psi) \neq$ *None*
using $\mathfrak{M}$*.smallest-accepting-rank-properties* ‹$G\ \psi \in \mathcal{G}$›
unfolding π*-def* **by** *simp+*
}
hence $\mathcal{G} =$ *dom* π **and** $\bigwedge\chi.\ \chi \in \mathcal{G} \Longrightarrow$ *the* $(\pi\ \chi) <$ *ltl-FG-to-rabin-def.max-rank*$_R$
Σ (*theG* χ)
using ‹*Only-G* $\mathcal{G}$› π*-def* **unfolding** *dom-def* **by** *auto*

hence (*M-fin* $\pi \cup \bigcup\{$*Acc-fin* $\Sigma\ \pi\ \chi \mid \chi.\ \chi \in$ *dom* $\pi\}$, $\{$*Acc-inf* $\pi\ \chi \mid \chi.$
$\chi \in$ *dom* $\pi\}$) $\in$ *?F*
using ‹$\mathcal{G} \subseteq \mathbf{G}\ \varphi$› *max-rank-of-def* **by** *auto*

moreover

{
have *accepting-pair*$_R$ *?*Δ *?*q_0 (*M-fin* π, *UNIV*) *w*
proof −

obtain *i* **where** *i-def*:
$\bigwedge j.\ j \geq i \Longrightarrow \forall S.\ (\forall \psi.\ G\ \psi \in \mathcal{G} \longrightarrow S \models_P G\ \psi \wedge S \models_P eval_G\ \mathcal{G}$ $(\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)) \longrightarrow S \models_P af\ \varphi\ (w\ [0 \rightarrow j])$
using ‹*accept*$_M$ φ $\mathcal{G}$ *w*› **unfolding** *MOST-nat-le accept*$_M$*-def* **by** *blast*

obtain *i*′ **where** *i*′*-def*:
$\bigwedge j\ \psi\ S.\ j \geq i' \Longrightarrow G\ \psi \in \mathcal{G} \Longrightarrow (S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j \wedge \mathcal{G} \subseteq S) =$ $(\forall q.\ q \in$ *ltl-FG-to-rabin-def*.$\mathcal{S}_R\ \Sigma\ \psi\ \mathcal{G}\ w\ j \longrightarrow S \models_P Rep\ q)$
using $\mathcal{F}$*-eq-*$\mathcal{S}$*-generalized*[*OF finite-*Σ *bounded-w* ‹*closed* $\mathcal{G}$ *w*›] **unfolding** *MOST-nat-le* **by** *presburger*

have $\bigwedge j.\ j \geq max\ i\ i' \Longrightarrow r\ j \notin$ *M-fin* π
proof −
fix *j*
assume $j \geq max\ i\ i'$

let $?\varphi' = fst\ (fst\ (r\ j))$
let $?m = snd\ (fst\ (r\ j))$

{
fix *S*
assume $\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow S \uparrow\models_P Abs\ \chi$
hence *assm1*: $\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow S \models_P \chi$
using *ltl-prop-entails-abs.abs-eq* **by** *blast*
assume $\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow \forall q.\ (\exists j \geq the\ (\pi\ \chi).\ the\ (?m\ \chi)\ q =$ $Some\ j) \longrightarrow S \uparrow\models_P \uparrow eval_G\ \mathcal{G}\ q$
hence *assm2*: $\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow \forall q.\ (\exists j \geq the\ (\pi\ \chi).\ the\ (?m\ \chi)\ q$ $= Some\ j) \longrightarrow S \models_P eval_G\ \mathcal{G}\ (Rep\ q)$
unfolding *ltl-prop-entails-abs.rep-eq eval*$_G$*-abs-def* **by** *simp*

{
fix ψ
assume $G\ \psi \in \mathcal{G}$
hence $G\ \psi \in \mathbf{G}\ \varphi$ **and** $\mathcal{G} \subseteq S$
using ‹$\mathcal{G} \subseteq \mathbf{G}\ \varphi$› *assm1* ‹*Only-G* $\mathcal{G}$› **by** (*blast*, *force*)

interpret $\mathfrak{M}$: *ltl-FG-to-rabin* Σ ψ $\mathcal{G}$ *w*

by (*unfold-locales*; *insert* ‹*Only-G* $\mathcal{G}$› *finite-*Σ *bounded-w*; *blast*)

have $\bigwedge S.\ (\bigwedge q.\ q \in \mathfrak{M}.\mathcal{S}\ j \Longrightarrow S \models_P Rep\ q) \Longrightarrow S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j$
using i'*-def* ‹$G\ \psi \in \mathcal{G}$› ‹$j \geq max\ i\ i'$› *max.bounded-iff* **by** *metis*
hence $\bigwedge S.\ (\bigwedge q.\ q \in Rep\ `\ \mathfrak{M}.\mathcal{S}\ j \Longrightarrow S \models_P q) \Longrightarrow S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j$
by *simp*

moreover

have $\mathcal{S}$*-def*: $\mathfrak{M}.\mathcal{S}\ j = \{q.\ \mathcal{G} \models_P Rep\ q\} \cup \{q\ .\ \exists j'.\ the\ (\pi\ (G\ \psi)) \leq j' \wedge the\ (?m\ (G\ \psi))\ q = Some\ j'\}$
using *r-alt-def*$''$[*OF* ‹$G\ \psi \in \mathbf{G}\ \varphi$›, *unfolded ltl.sel*, *of j*] ‹$G\ \psi \in \mathcal{G}$› **by** (*simp add*: π*-def*)
have $\bigwedge q.\ \mathcal{G} \models_P Rep\ q \Longrightarrow S \models_P eval_G\ \mathcal{G}\ (Rep\ q)$
using ‹$\mathcal{G} \subseteq S$› $eval_G$*-prop-entailment* **by** *blast*
hence $\bigwedge q.\ q \in Rep\ `\ \mathfrak{M}.\mathcal{S}\ j \Longrightarrow S \models_P eval_G\ \mathcal{G}\ q$
using *assm2* ‹$G\ \psi \in \mathcal{G}$› **unfolding** $\mathcal{S}$*-def* **by** *auto*

ultimately
have $S \models_P eval_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)$
by (*rule* $eval_G$*-respectfulness-generalized*)
}
hence $S \models_P af\ \varphi\ (w\ [0 \to j])$
by (*metis max.bounded-iff i-def* ‹$j \geq max\ i\ i'$› ‹$\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow S \models_P \chi$›)
hence $S \models_P Rep\ ?\varphi'$
using φ'*-def ltl-prop-equiv-def* **by** *blast*
hence $S \uparrow\models_P ?\varphi'$
using *ltl-prop-entails-abs.rep-eq* **by** *blast*
}
thus $r\ j \notin M\text{-}fin\ \pi$
using ‹$\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow the\ (\pi\ \chi) < ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def.max\text{-}rank_R\ \Sigma\ (theG\ \chi)$› ‹$\mathcal{G} = dom\ \pi$› **by** *fastforce*
qed
hence *range* (*suffix* (*max i i*$'$) *r*) $\cap$ *M-fin* $\pi = \{\}$
unfolding *suffix-def* **by** (*blast intro*: *le-add1 elim*: *rangeE*)
hence *limit r* $\cap$ *M-fin* $\pi = \{\}$
using *limit-in-range-suffix*[*of r*] **by** *blast*
moreover
have *limit r* $\cap$ *UNIV* $\neq \{\}$

using ‹*finite* (*range r*)› **by** (*simp*, *metis empty-iff limit-nonemptyE*)

ultimately
show *?thesis*
unfolding *r-def accepting-pair*$_R$*-simp* **..**
qed

moreover

have $\bigwedge\chi.\ \chi \in \mathcal{G} \Longrightarrow$ *accepting-pair*$_R$ *?*Δ *?*q_0 (*Acc* Σ π χ) *w*
proof −
fix χ **assume** $\chi \in \mathcal{G}$
then obtain ψ **where** $\chi = G\ \psi$ **and** $G\ \psi \in \mathcal{G}$
using ‹*Only-G* $\mathcal{G}$› **by** *fastforce*
thus *?thesis* χ
using ‹$\bigwedge\psi.\ G\ \psi \in \mathcal{G} \Longrightarrow$ *accept*$_R$ (*ltl-to-rabin* Σ ψ $\mathcal{G}$) *w*›[*OF* ‹*G* $\psi \in \mathcal{G}$›]
using *rabin-accept-to-Acc*[*of* ψ π] ‹*G* $\psi \in \mathcal{G}$› ‹$\mathcal{G} \subseteq$ **G** φ› ‹$\chi \in \mathcal{G}$›
unfolding *ltl.sel* **unfolding** ‹$\chi = G\ \psi$› ‹$\mathcal{G}$ = *dom* π› **using** π*-def* ‹$\mathcal{G}$ = *dom* π› *ltl.sel*(*8*) **unfolding** *ltl-prop-entails-abs.rep-eq ltl-to-rabin.simps*
by (*metis* (*no-types*, *lifting*) *Collect-cong*)
qed
ultimately
have *accepting-pair*$_{GR}$ *?*Δ *?*q_0 (*M-fin* π $\cup$ $\bigcup${*Acc-fin* Σ π χ | χ. $\chi \in$ *dom* π}, {*Acc-inf* π χ | χ. $\chi \in$ *dom* π}) *w*
unfolding *accepting-pair*$_{GR}$*-def accepting-pair*$_R$*-def fst-conv snd-conv* ‹$\mathcal{G}$ = *dom* π› **by** *blast*
}
ultimately
show *?rhs*
unfolding *ltl-to-rabin-base-def.ltl-to-generalized-rabin.simps accept*$_{GR}$*-def fst-conv snd-conv* **by** *blast*
}

— Assuming $\mathcal{A}$ Σ φ accepts *w*, we prove that $w \models \varphi$ holds
{
assume *?rhs*
obtain π' **where** *0*: *dom* $\pi' \subseteq$ **G** φ
and *1*: $\bigwedge\chi.\ \chi \in$ *dom* $\pi' \Longrightarrow$ *the* (π' χ) < *ltl-FG-to-rabin-def.max-rank*$_R$ Σ (*theG* χ)
and *2*: *accepting-pair*$_R$ *?*Δ *?*q_0 (*M-fin* π', *UNIV*) *w*
and *3*: $\bigwedge\chi.\ \chi \in$ *dom* $\pi' \Longrightarrow$ *accepting-pair*$_R$ *?*Δ *?*q_0 (*Acc* Σ π' χ) *w*
using *accept*$_{GR}$*-I*[*OF* ‹*?rhs*›] **unfolding** *max-rank-of-def* **by** *blast*

define $\mathcal{G}$ **where** $\mathcal{G} = dom\ \pi'$
hence $\mathcal{G} \subseteq \mathbf{G}\ \varphi$
using ‹$dom\ \pi' \subseteq \mathbf{G}\ \varphi$› **by** *simp*

moreover

note $\mathcal{G}$*-properties*[*OF* ‹$dom\ \pi' \subseteq \mathbf{G}\ \varphi$›[*unfolded* $\mathcal{G}$*-def*[*symmetric*]]]
ultimately
have $\mathfrak{M}$*-Accept*: $\bigwedge\chi.\ \chi \in \mathcal{G} \Longrightarrow$ *ltl-FG-to-rabin-def.accept*$_R$*'* (*theG* χ) $\mathcal{G}$ *w*
using *Acc-to-mojmir-accept*[*OF - 0 3*, *of theG -*] *1*[*of G theG -*, *unfolded ltl.sel*] $\mathcal{G}$*-def*
unfolding *ltl-prop-entails-abs.rep-eq* **by** (*metis* (*no-types*) *ltl.sel*(*8*))

— Normalise π to the smallest accepting ranks
obtain π **where** $dom\ \pi' = dom\ \pi$
and $\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow \pi\ \chi =$ *ltl-FG-to-rabin-def.smallest-accepting-rank*$_R$ Σ (*theG* χ) (*dom* π) *w*
and *accepting-pair*$_R$ ($\delta_{\mathcal{A}}\ \Sigma$) ($\iota_{\mathcal{A}}\ \varphi$) (*M-fin* π, *UNIV*) *w*
and $\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow$ *accepting-pair*$_R$ ($\delta_{\mathcal{A}}\ \Sigma$) ($\iota_{\mathcal{A}}\ \varphi$) (*Acc* $\Sigma\ \pi\ \chi$) *w*
using *normalize-*π[*OF 0 - 2 3*] *1* **unfolding** *max-rank-of-def ltl-prop-entails-abs.rep-eq* **by** *blast*

have *ltl-FG-to-rabin* $\Sigma\ \mathcal{G}$ *w*
using *finite-*Σ *bounded-w* ‹*Only-G* $\mathcal{G}$› **unfolding** *ltl-FG-to-rabin-def* **by** *auto*

have *closed* $\mathcal{G}$ *w*
using $\mathfrak{M}$*-Accept* ‹*Only-G* $\mathcal{G}$› *ltl.sel*(*8*) ‹*finite* $\mathcal{G}$›
unfolding *ltl-FG-to-rabin.ltl-to-rabin-correct-exposed'*[*OF* ‹*ltl-FG-to-rabin* $\Sigma\ \mathcal{G}$ *w*›, *symmetric*] **by** *fastforce*

moreover

have *accept*$_M$ $\varphi\ \mathcal{G}$ *w*
proof –

obtain *i* **where** *i-def*: $\bigwedge j.\ j \geq i \Longrightarrow r\ j \notin$ *M-fin* π
using ‹*accepting-pair*$_R$ *?*Δ *?*q_0 (*M-fin* π, *UNIV*) *w*› *limit-inter-empty*[*OF* ‹*finite* (*range r*)›, *of M-fin* π]
unfolding *r-def*[*symmetric*] *MOST-nat-le accepting-pair*$_R$*-def* **by** *auto*

obtain i' **where** i'-*def*:
$\bigwedge j\ \psi\ S.\ j \geq i' \Longrightarrow G\ \psi \in \mathcal{G} \Longrightarrow (S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j \wedge \mathcal{G} \subseteq S) = (\forall\, q.\ q \in \mathit{ltl\text{-}FG\text{-}to\text{-}rabin\text{-}def}.\mathcal{S}_R\ \Sigma\ \psi\ \mathcal{G}\ w\ j \longrightarrow S \models_P \mathit{Rep}\ q)$
using $\mathcal{F}$-*eq*-$\mathcal{S}$-*generalized*[*OF finite*-Σ *bounded-w* ‹*closed* $\mathcal{G}$ *w*›] **unfolding** *MOST-nat-le* **by** *presburger*

{
fix $j\ S$
assume $j \geq \mathit{max}\ i\ i'$
hence $j \geq i$ **and** $j \geq i'$
by *simp+*
assume $\mathcal{G}$-*def*′: $\forall\,\psi.\ G\ \psi \in \mathcal{G} \longrightarrow S \models_P G\ \psi \wedge S \models_P \mathit{eval}_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)$

let $?\varphi' = \mathit{fst}\ (\mathit{fst}\ (r\ j))$
let $?m = \mathit{snd}\ (\mathit{fst}\ (r\ j))$

have $\bigwedge\chi.\ \chi \in \mathcal{G} \Longrightarrow S \models_P \chi$
using $\mathcal{G}$-*def*′ ‹$\mathcal{G} \subseteq \mathbf{G}\ \varphi$› **unfolding** *G-nested-propos-alt-def* **by** *auto*
moreover

{
fix χ
assume $\chi \in \mathcal{G}$
then obtain ψ **where** $\chi = G\ \psi$ **and** $G\ \psi \in \mathcal{G}$
using ‹*Only-G* $\mathcal{G}$› **by** *auto*
hence $G\ \psi \in \mathbf{G}\ \varphi$
using ‹$\mathcal{G} \subseteq \mathbf{G}\ \varphi$› **by** *blast*

interpret $\mathfrak{M}$: *ltl-FG-to-rabin* $\Sigma\ \psi\ \mathcal{G}\ w$
by (*insert* ‹*ltl-FG-to-rabin* $\Sigma\ \mathcal{G}\ w$›)

{
fix q
assume $q \in \mathfrak{M}.\mathcal{S}\ j$
hence $S \models_P \mathit{eval}_G\ \mathcal{G}\ (\mathcal{F}\ \psi\ w\ \mathcal{G}\ j)$
using $\mathcal{G}$-*def*′ ‹$G\ \psi \in \mathcal{G}$› **by** *simp*
moreover
have $S \supseteq \mathcal{G}$
using $\mathcal{G}$-*def*′ ‹*Only-G* $\mathcal{G}$› **by** *auto*
hence $\bigwedge x.\ x \in \mathcal{G} \Longrightarrow S \models_P \mathit{eval}_G\ \mathcal{G}\ x$

using ‹*Only-G* $\mathcal{G}$› ‹$S \supseteq \mathcal{G}$› **by** *fastforce*
moreover
{
fix S
assume $\bigwedge x.\ x \in \mathcal{G} \cup \{\mathcal{F}\ \psi\ w\ \mathcal{G}\ j\} \Longrightarrow S \models_P x$
hence $\mathcal{G} \subseteq S$ **and** $S \models_P \mathcal{F}\ \psi\ w\ \mathcal{G}\ j$
using ‹*Only-G* $\mathcal{G}$› **by** *fastforce+*
hence $S \models_P$ *Rep* q
using ‹$q \in$ *ltl-FG-to-rabin-def*.$\mathcal{S}_R$ $\Sigma\ \psi\ \mathcal{G}\ w\ j$›
using i'*-def*[*OF* ‹$j \geq i'$› ‹$G\ \psi \in \mathcal{G}$›] **by** *blast*
}
ultimately
have $S \models_P$ *eval*$_G$ $\mathcal{G}$ (*Rep* q)
using *eval*$_G$*-respectfulness-generalized*[*of* $\mathcal{G} \cup \{\mathcal{F}\ \psi\ w\ \mathcal{G}\ j\}$ *Rep*
q S $\mathcal{G}$]
by *blast*
}
moreover
have $\mathfrak{M}.\mathcal{S}\ j = \{q.\ \mathcal{G} \models_P$ *Rep* $q\} \cup \{q\ .\ \exists j'.$ *the* $\mathfrak{M}$.*smallest-accepting-rank*
$\leq j' \wedge$ *the* (*?m* ($G\ \psi$)) q = *Some* $j'\}$
unfolding $\mathfrak{M}.\mathcal{S}$.*simps* **using** *run-properties*(*2*)[*OF* ‹$G\ \psi \in \mathbf{G}\ \varphi$›]
r-def **by** *simp*
ultimately
have $\bigwedge q\ j.\ j \geq$ *the* ($\pi\ \chi$) $\Longrightarrow$ *the* (*?m* χ) q = *Some* $j \Longrightarrow S \models_P$
eval$_G$ $\mathcal{G}$ (*Rep* q)
using ‹$\chi \in \mathcal{G}$›[*unfolded* $\mathcal{G}$*-def* ‹*dom* π' = *dom* π›]
unfolding ‹$\chi = G\ \psi$› ‹$\bigwedge \chi.\ \chi \in$ *dom* $\pi \Longrightarrow \pi\ \chi$ = *ltl-FG-to-rabin-def*.*smallest-accepting-rank*$_R$
Σ (*theG* χ) (*dom* π) w›[*OF* ‹$\chi \in \mathcal{G}$›[*unfolded* $\mathcal{G}$*-def* ‹*dom* π' = *dom* π›],
unfolded ‹$\chi = G\ \psi$›] *ltl.sel*(*8*)
unfolding ‹$\mathcal{G} \equiv$ *dom* π'›[*symmetric*] ‹*dom* π' = *dom* π›[*symmetric*]
by *blast*
}
moreover

have ($\bigwedge \chi.\ \chi \in \mathcal{G} \Longrightarrow S \models_P \chi \wedge (\forall q.\ \forall j' \geq$ *the* ($\pi\ \chi$). *the* (*?m* χ)
q = *Some* $j' \longrightarrow S \models_P$ *eval*$_G$ $\mathcal{G}$ (*Rep* q))) $\Longrightarrow S \models_P$ *Rep* *?*φ'
apply (*insert* *i-def*[*OF* ‹$j \geq i$›])
apply (*simp add*: *eval*$_G$*-abs-def* *ltl-prop-entails-abs.rep-eq* *case-prod-beta*
option.case-eq-if)
apply (*unfold* ‹$\mathcal{G} \equiv$ *dom* π'›[*symmetric*] ‹*dom* π' = *dom* π›[*symmetric*])
apply *meson*
done

ultimately

have $S \models_P$ *Rep* $?\varphi'$
by *fast*
hence $S \models_P$ *af* φ $(w\ [0 \rightarrow j])$
using φ'*-def ltl-prop-equiv-def* **by** *blast*
}
thus $accept_M\ \varphi\ \mathcal{G}\ w$
unfolding $accept_M$*-def MOST-nat-le* **by** *blast*
qed

ultimately
show *?lhs*
using ‹$\mathcal{G} \subseteq \mathbf{G}\ \varphi$› *ltl-logical-characterization* **by** *blast*
}
qed

end

fun *ltl-to-generalized-rabin-af*
where
ltl-to-generalized-rabin-af $\Sigma\ \varphi$ = *ltl-to-rabin-base-def.ltl-to-generalized-rabin* ↑*af* ↑af_G *Abs Abs M-fin* $\Sigma\ \varphi$

lemma *ltl-to-generalized-rabin-af-wellformed*:
finite $\Sigma \Longrightarrow$ *range* $w \subseteq \Sigma \Longrightarrow$ *ltl-to-rabin-af* Σ w
apply (*unfold-locales*)
apply (*auto simp add*: *af-G-letter-sat-core-lifted ltl-prop-entails-abs.rep-eq intro*: *finite-reach-af*)
apply (*meson le-trans ltl-semi-mojmir*[*unfolded semi-mojmir-def*])+
done

theorem *ltl-to-generalized-rabin-af-correct*:
assumes *finite* Σ
assumes *range* $w \subseteq \Sigma$
shows $w \models \varphi = accept_{GR}$ (*ltl-to-generalized-rabin-af* $\Sigma\ \varphi$) w
using *ltl-to-generalized-rabin-af-wellformed*[*OF assms*, *THEN ltl-to-rabin-af.ltl-to-generalized-rabin-*
by *simp*

thm *ltl-to-generalized-rabin-af-correct ltl-FG-to-generalized-rabin-correct*

end

14 Eager Unfolding Optimisation

theory *LTL-Rabin-Unfold-Opt*
  imports *Main LTL-Rabin*
begin

14.1 Preliminary Facts

lemma *finite-reach-af-opt*:
  finite (*reach* Σ ↑*af*$_{\mathfrak{U}}$ (*Abs* φ))
proof (*cases* Σ ≠ {})
  case *True*
    thus *?thesis*
      using *af-abs-opt.finite-abs-reach* **unfolding** *af-abs-opt.abs-reach-def reach-foldl-def*[*OF True*]
      using *finite-subset*[*of* {*foldl* ↑*af*$_{\mathfrak{U}}$ (*Abs* φ) *w* |*w*. *set w* ⊆ Σ} {*foldl* ↑*af*$_{\mathfrak{U}}$ (*Abs* φ) *w* |*w*. *True*}]
      unfolding *af-letter-abs-opt-def*
      by *blast*
qed (*simp add*: *reach-def*)

lemma *finite-reach-af-G-opt*:
  finite (*reach* Σ ↑*af*$_{G\mathfrak{U}}$ (*Abs* φ))
proof (*cases* Σ ≠ {})
  case *True*
    thus *?thesis*
    using *af-G-abs-opt.finite-abs-reach* **unfolding** *af-G-abs-opt.abs-reach-def reach-foldl-def*[*OF True*]
      using *finite-subset*[*of* {*foldl* ↑*af*$_{G\mathfrak{U}}$ (*Abs* φ) *w* |*w*. *set w* ⊆ Σ} {*foldl* ↑*af*$_{G\mathfrak{U}}$ (*Abs* φ) *w* |*w*. *True*}]
      unfolding *af-G-letter-abs-opt-def*
      by *blast*
qed (*simp add*: *reach-def*)

lemma *wellformed-mojmir-opt*:
  assumes *Only-G* $\mathcal{G}$
  assumes *finite* Σ
  assumes *range w* ⊆ Σ
  shows *mojmir* Σ ↑*af*$_{G\mathfrak{U}}$ (*Abs* φ) *w* {*q*. $\mathcal{G}$ $\models_P$ *Rep q*}
proof –
  have ∀ *q* ν. *q* ∈ {*q*. $\mathcal{G}$ $\models_P$ *Rep q*} ⟶ *af-G-letter-abs-opt q* ν ∈ {*q*. $\mathcal{G}$ $\models_P$ *Rep q*}
    using ‹*Only-G* $\mathcal{G}$› *af-G-letter-opt-sat-core-lifted* **by** *auto*
  thus *?thesis*

using *finite-reach-af-G-opt assms* **by** (*unfold-locales*; *auto*)
qed

locale *ltl-FG-to-rabin-opt-def* =
fixes
Σ :: *'a set set*
fixes
φ :: *'a ltl*
fixes
$\mathcal{G}$ :: *'a ltl set*
fixes
w :: *'a set word*
begin

sublocale *mojmir-to-rabin-def* Σ $\uparrow af_{G\mathfrak{U}}$ *Abs* (*Unf*$_G$ φ) *w* $\{q.\ \mathcal{G} \models_P Rep\ q\}$
.

end

locale *ltl-FG-to-rabin-opt* = *ltl-FG-to-rabin-opt-def* +
assumes
wellformed-$\mathcal{G}$: *Only-G* $\mathcal{G}$
assumes
bounded-w: *range w* $\subseteq \Sigma$
assumes
finite-Σ: *finite* Σ
begin

sublocale *mojmir-to-rabin* Σ $\uparrow af_{G\mathfrak{U}}$ *Abs* (*Unf*$_G$ φ) *w* $\{q.\ \mathcal{G} \models_P Rep\ q\}$
proof
show $\bigwedge q\ \nu.\ q \in \{q.\ \mathcal{G} \models_P Rep\ q\} \Longrightarrow \uparrow af_{G\mathfrak{U}}\ q\ \nu \in \{q.\ \mathcal{G} \models_P Rep\ q\}$
using *wellformed-*$\mathcal{G}$ *af-G-letter-opt-sat-core-lifted* **by** *auto*
have *nonempty-*Σ: $\Sigma \neq \{\}$
using *bounded-w* **by** *blast*
show *finite* (*reach* Σ $\uparrow af_{G\mathfrak{U}}$ (*Abs* (*Unf*$_G$ φ))) (**is** *finite ?A*)
using *finite-reach-af-G-opt wellformed-*$\mathcal{G}$ **by** *blast*
qed (*insert finite-*Σ *bounded-w*)

end

14.2 Equivalences between the standard and the eager Mojmir construction

context

fixes
Σ :: *'a set set*
fixes
φ :: *'a ltl*
fixes
$\mathcal{G}$ :: *'a ltl set*
fixes
w :: *'a set word*
assumes
context-assms: *Only-G* $\mathcal{G}$ *finite* Σ *range* $w \subseteq \Sigma$
begin

— Create an interpretation of the mojmir locale for the standard construction
interpretation $\mathfrak{M}$: *ltl-FG-to-rabin* Σ φ $\mathcal{G}$ *w*
by (*unfold-locales*; *insert context-assms*; *auto*)

— Create an interpretation of the mojmir locale for the optimised construction
interpretation $\mathfrak{U}$: *ltl-FG-to-rabin-opt* Σ φ $\mathcal{G}$ *w*
by (*unfold-locales*; *insert context-assms*; *auto*)

lemma *unfold-token-run-eq*:
assumes $x \leq n$
shows $\mathfrak{M}$.*token-run* *x* (*Suc n*) = ↑*step* ($\mathfrak{U}$.*token-run x n*) (*w n*)
(**is** *?lhs* = *?rhs*)
proof −
have $x + (n - x) = n$ **and** $x + (Suc\ n - x) = Suc\ n$
using *assms* **by** *arith+*
have $w\ [x \rightarrow Suc\ n] = w\ [x \rightarrow n]\ @\ [w\ n]$
unfolding *upt-Suc subsequence-def* **using** *assms* **by** *simp*

have $af_G\ \varphi\ (w\ [x \rightarrow Suc\ n]) = step\ (af_{G\mathfrak{U}}\ (Unf_G\ \varphi)\ (w\ [x \rightarrow n]))\ (w\ n)$
(**is** *?l* = *?r*)
unfolding *af-to-af-opt*[*symmetric*] ‹$w\ [x \rightarrow Suc\ n] = w\ [x \rightarrow n]\ @\ [w\ n]$› *foldl-append*
using *af-letter-alt-def* **by** *auto*
moreover
have *?lhs* = *Abs ?l*
unfolding $\mathfrak{M}$.*token-run.simps run-foldl*
using *subsequence-shift* ‹$x + (Suc\ n - x) = Suc\ n$› *Nat.add-0-right subsequence-def*
by (*metis af-G-abs.f-foldl-abs-alt-def af-G-abs.f-foldl-abs.abs-eq af-G-letter-abs-def*)

moreover
have *Abs ?r* = *?rhs*
unfolding $\mathfrak{U}$.*token-run.simps run-foldl subsequence-def*[*symmetric*]
unfolding *subsequence-shift* ‹$x + (n - x) = n$› *Nat.add-0-right af-G-letter-abs-opt-def*
unfolding *af-G-abs-opt.f-foldl-abs-alt-def*[*unfolded af-G-abs-opt.f-foldl-abs.abs-eq*, *symmetric*]
by (*simp add*: *step-abs.abs-eq*)
ultimately
show *?lhs* = *?rhs*
by *presburger*
qed

lemma *unfold-token-succeeds-eq*:
$\mathfrak{M}$.*token-succeeds* x = $\mathfrak{U}$.*token-succeeds* x
proof
assume $\mathfrak{M}$.*token-succeeds* x

then obtain n **where** $\bigwedge m.\ m > n \Longrightarrow \mathfrak{M}.\textit{token-run}\ x\ m \in \{q.\ \mathcal{G} \models_P \textit{Rep}\ q\}$
unfolding $\mathfrak{M}$.*token-succeeds-alt-def MOST-nat* **by** *blast*
then obtain n **where** $\mathfrak{M}.\textit{token-run}\ x\ (\textit{Suc}\ n) \in \{q.\ \mathcal{G} \models_P \textit{Rep}\ q\}$ **and** $x \leq n$
by (*cases* $x \leq n$) *auto*

hence *1*: $\mathcal{G} \models_P \textit{Rep}\ (\textit{step-abs}\ (\mathfrak{U}.\textit{token-run}\ x\ n)\ (w\ n))$
using *unfold-token-run-eq* **by** *fastforce*
moreover
have $\textit{Suc}\ n - x = \textit{Suc}\ (n - x)$ **and** $x + (n - x) = n$
using ‹$x \leq n$› **by** *arith+*
ultimately
have $\mathfrak{U}.\textit{token-run}\ x\ (\textit{Suc}\ n) = \textit{Unf}_G\textit{-abs}\ (\textit{step-abs}\ (\mathfrak{U}.\textit{token-run}\ x\ n)\ (w\ n))$
unfolding *af-G-letter-abs-opt-split* **by** *simp*

hence $\mathcal{G} \models_P \textit{Rep}\ (\mathfrak{U}.\textit{token-run}\ x\ (\textit{Suc}\ n))$
using *1* $\textit{Unf}_G$-$\mathcal{G}$[*OF* ‹*Only-G* $\mathcal{G}$›] **by** (*simp add*: *Rep-Abs-equiv* $\textit{Unf}_G$*-abs-def*)
thus $\mathfrak{U}$.*token-succeeds* x
unfolding $\mathfrak{U}$.*token-succeeds-def* **by** *blast*
next
assume $\mathfrak{U}$.*token-succeeds* x

then obtain n **where** $\bigwedge m.\ m > n \Longrightarrow \mathfrak{U}.\textit{token-run}\ x\ m \in \{q.\ \mathcal{G} \models_P \textit{Rep}\ q\}$
unfolding $\mathfrak{U}$.*token-succeeds-alt-def MOST-nat* **by** *blast*

then obtain n **where** $\mathfrak{U}$.*token-run x n* $\in \{q.\ \mathcal{G} \models_P Rep\ q\}$ **and** $x \le n$
by (*cases* $x \le n$) (*fastforce*, *auto*)

hence $\mathcal{G} \models_P$ *Rep* (*step-abs* ($\mathfrak{U}$.*token-run x n*) (*w n*))
using *step-*$\mathcal{G}$[*OF* ‹*Only-G* $\mathcal{G}$›] *Rep-step*[*unfolded ltl-prop-equiv-def*] **by** *blast*
thus $\mathfrak{M}$.*token-succeeds x*
unfolding $\mathfrak{M}$.*token-succeeds-def unfold-token-run-eq*[*OF* ‹$x \le n$›, *symmetric*] **by** *blast*
qed

lemma *unfold-accept-eq*:
$\mathfrak{M}$.*accept* = $\mathfrak{U}$.*accept*
unfolding $\mathfrak{M}$.*accept-def* $\mathfrak{U}$.*accept-def MOST-nat-le unfold-token-succeeds-eq*
..

lemma *unfold-*$\mathcal{S}$*-eq*:
assumes $\mathfrak{M}$.*accept*
shows $\forall_\infty n.\ \mathfrak{M}.\mathcal{S}\ (Suc\ n) = (\lambda q.\ step\text{-}abs\ q\ (w\ n))\ `\ (\mathfrak{U}.\mathcal{S}\ n) \cup \{Abs\ \varphi\} \cup \{q.\ \mathcal{G} \models_P Rep\ q\}$
proof –
— Obtain lower bounds for proof
obtain $i_{\mathfrak{M}}$ **where** $i_{\mathfrak{M}}$*-def*: $\mathfrak{M}$.*smallest-accepting-rank* = *Some* $i_{\mathfrak{M}}$
using *assms* **unfolding** $\mathfrak{M}$.*smallest-accepting-rank-def* **by** *simp*
obtain $n_{\mathfrak{M}}$ **where** $n_{\mathfrak{M}}$*-def*: $\bigwedge x\ m.\ m \ge n_{\mathfrak{M}} \Longrightarrow \mathfrak{M}.token\text{-}succeeds\ x = (m < x \vee (\exists j \ge i_{\mathfrak{M}}.\ \mathfrak{M}.rank\ x\ m = Some\ j) \vee \mathfrak{M}.token\text{-}run\ x\ m \in \{q.\ \mathcal{G} \models_P Rep\ q\})$
using $\mathfrak{M}$.*token-smallest-accepting-rank*[*OF* $i_{\mathfrak{M}}$*-def*] **unfolding** *MOST-nat-le* **by** *metis*

have $\mathfrak{U}$.*accept*
using *assms unfold-accept-eq* **by** *simp*
obtain $i_{\mathfrak{U}}$ **where** $i_{\mathfrak{U}}$*-def*: $\mathfrak{U}$.*smallest-accepting-rank* = *Some* $i_{\mathfrak{U}}$
using ‹$\mathfrak{U}$.*accept*› **unfolding** $\mathfrak{U}$.*smallest-accepting-rank-def* **by** *simp*
obtain $n_{\mathfrak{U}}$ **where** $n_{\mathfrak{U}}$*-def*: $\bigwedge x\ m.\ m \ge n_{\mathfrak{U}} \Longrightarrow \mathfrak{U}.token\text{-}succeeds\ x = (m < x \vee (\exists j \ge i_{\mathfrak{U}}.\ \mathfrak{U}.rank\ x\ m = Some\ j) \vee \mathfrak{U}.token\text{-}run\ x\ m \in \{q.\ \mathcal{G} \models_P Rep\ q\})$
using $\mathfrak{U}$.*token-smallest-accepting-rank*[*OF* $i_{\mathfrak{U}}$*-def*] **unfolding** *MOST-nat-le* **by** *metis*

show *?thesis*
proof (*unfold MOST-nat-le*, *rule*, *rule*, *rule*)
fix *m*
assume $m \ge max\ n_{\mathfrak{M}}\ n_{\mathfrak{U}}$

hence $m \geq n_{\mathfrak{M}}$ **and** $m \geq n_{\mathfrak{U}}$ **and** $Suc\ m \geq n_{\mathfrak{M}}$
by *simp+*

— Using the properties of $n_{\mathfrak{M}}$ and $n_{\mathfrak{U}}$ and the lemma $\mathfrak{M}$.*token-succeeds ?x* = $\mathfrak{U}$.*token-succeeds ?x*, we prove that the behaviour of x in $\mathfrak{M}$ and $\mathfrak{U}$ is similar in regards to creation time, accepting rank or final states.

hence *token-trans*: $\bigwedge x.\ Suc\ m < x \vee (\exists j \geq i_{\mathfrak{M}}.\ \mathfrak{M}.rank\ x\ (Suc\ m) = Some\ j) \vee \mathfrak{M}.token\text{-}run\ x\ (Suc\ m) \in \{q.\ \mathcal{G} \models_P Rep\ q\}$
$\longleftrightarrow m < x \vee (\exists j \geq i_{\mathfrak{U}}.\ \mathfrak{U}.rank\ x\ m = Some\ j) \vee \mathfrak{U}.token\text{-}run\ x\ m \in \{q.\ \mathcal{G} \models_P Rep\ q\}$
using $n_{\mathfrak{M}}$*-def* $n_{\mathfrak{U}}$*-def* **unfolding** *unfold-token-succeeds-eq* **by** *presburger*

show $\mathfrak{M}.\mathcal{S}\ (Suc\ m) = (\lambda q.\ step\text{-}abs\ q\ (w\ m))$ ' $(\mathfrak{U}.\mathcal{S}\ m) \cup \{Abs\ \varphi\} \cup \{q.\ \mathcal{G} \models_P Rep\ q\}$ (**is** *?lhs = ?rhs*)
proof
{
fix q **assume** $\exists j \geq i_{\mathfrak{M}}.\ \mathfrak{M}.state\text{-}rank\ q\ (Suc\ m) = Some\ j$
moreover
then obtain x **where** *q-def*: $q = \mathfrak{M}.token\text{-}run\ x\ (Suc\ m)$ **and** $x \leq Suc\ m$
using $\mathfrak{M}$.*push-down-state-rank-tokens* **by** *fastforce*
ultimately
have $\exists j \geq i_{\mathfrak{M}}.\ \mathfrak{M}.rank\ x\ (Suc\ m) = Some\ j$
using $\mathfrak{M}$.*rank-eq-state-rank* **by** *metis*
hence *token-cases*: $(\exists j \geq i_{\mathfrak{U}}.\ \mathfrak{U}.rank\ x\ m = Some\ j) \vee \mathfrak{U}.token\text{-}run\ x\ m \in \{q.\ \mathcal{G} \models_P Rep\ q\} \vee x = Suc\ m$
using *token-trans*[*of x*] $\mathfrak{M}$.*rank-Some-time* **by** *fastforce*
have $q \in$ *?rhs*
proof (*cases* $x \neq Suc\ m$)
case *True*
hence $x \leq m$
using ‹$x \leq Suc\ m$› **by** *arith*
have $\mathfrak{U}.token\text{-}run\ x\ m \in \{q.\ \mathcal{G} \models_P Rep\ q\} \Longrightarrow \mathcal{G} \models_P Rep\ q$
unfolding ‹$q = \mathfrak{M}.token\text{-}run\ x\ (Suc\ m)$› *unfold-token-run-eq*[*OF* ‹$x \leq m$›]
using *Rep-step*[*unfolded ltl-prop-equiv-def*] *step-*$\mathcal{G}$[*OF* ‹*Only-G* $\mathcal{G}$›] **by** *blast*
moreover
{
assume $\exists j \geq i_{\mathfrak{U}}.\ \mathfrak{U}.rank\ x\ m = Some\ j$
moreover
define q' **where** $q' = \mathfrak{U}.token\text{-}run\ x\ m$
ultimately
have $\exists j \geq i_{\mathfrak{U}}.\ \mathfrak{U}.state\text{-}rank\ q'\ m = Some\ j$
unfolding $\mathfrak{U}$.*rank-eq-state-rank*[*OF* ‹$x \leq m$›] q'*-def* **by** *blast*

hence $q' \in \mathfrak{U}.\mathcal{S}\ m$
using *assms* $i_{\mathfrak{U}}$*-def* **by** *simp*
moreover
have $q = \textit{step-abs}\ q'\ (w\ m)$
unfolding *q-def q'-def unfold-token-run-eq*[*OF* ‹$x \le m$›] **..**
ultimately
have $q \in (\lambda q.\ \textit{step-abs}\ q\ (w\ m))$ ‘ $(\mathfrak{U}.\mathcal{S}\ m)$
by *blast*
}
ultimately
show *?thesis*
using *token-cases True* **by** *blast*
qed (*simp add*: *q-def*)
}
thus *?lhs* $\subseteq$ *?rhs*
unfolding $\mathfrak{M}.\mathcal{S}$*.simps* $i_{\mathfrak{M}}$*-def option.sel* **by** *blast*
next
{
fix *q*
assume $q \in (\lambda q.\ \textit{step-abs}\ q\ (w\ m))$ ‘ $(\mathfrak{U}.\mathcal{S}\ m)$
then obtain q' **where** *q-def*: $q = \textit{step-abs}\ q'\ (w\ m)$ **and** $q' \in \mathfrak{U}.\mathcal{S}\ m$
by *blast*
hence $q \in$ *?lhs*
proof (*cases* $\mathcal{G} \models_P \textit{Rep}\ q'$)
case *False*
hence $\exists j \ge i_{\mathfrak{U}}.\ \mathfrak{U}.\textit{state-rank}\ q'\ m = \textit{Some}\ j$
using ‹$q' \in \mathfrak{U}.\mathcal{S}\ m$› **unfolding** $\mathfrak{U}.\mathcal{S}$*.simps* $i_{\mathfrak{U}}$*-def option.sel* **by** *blast*
moreover
then obtain x **where** *q'-def*: $q' = \mathfrak{U}.\textit{token-run}\ x\ m$ **and** $x \le m$ **and** $x \le \textit{Suc}\ m$
using $\mathfrak{U}$*.push-down-state-rank-tokens* **by** *force*
ultimately
have $\exists j \ge i_{\mathfrak{U}}.\ \mathfrak{U}.\textit{rank}\ x\ m = \textit{Some}\ j$
unfolding $\mathfrak{U}$*.rank-eq-state-rank*[*OF* ‹$x \le m$›] *q'-def* **by** *blast*
hence $(\exists j \ge i_{\mathfrak{M}}.\ \mathfrak{M}.\textit{rank}\ x\ (\textit{Suc}\ m) = \textit{Some}\ j) \vee \mathfrak{M}.\textit{token-run}\ x\ (\textit{Suc}\ m) \in \{q.\ \mathcal{G} \models_P \textit{Rep}\ q\}$
using *token-trans*[*of x*] $\mathfrak{U}$*.rank-Some-time* **by** *fastforce*
moreover
have $\mathfrak{M}.\textit{token-run}\ x\ (\textit{Suc}\ m) = q$
unfolding *q-def q'-def unfold-token-run-eq*[*OF* ‹$x \le m$›] **..**
ultimately
have $(\exists j \ge i_{\mathfrak{M}}.\ \mathfrak{M}.\textit{state-rank}\ q\ (\textit{Suc}\ m) = \textit{Some}\ j) \vee q \in \{q.\ \mathcal{G} \models_P \textit{Rep}\ q\}$

using $\mathfrak{M}$.*rank-eq-state-rank*[*OF* ‹$x \le$ *Suc m*›] **by** *metis*
thus *?thesis*
unfolding $\mathfrak{M}.\mathcal{S}$.*simps option.sel* $i_{\mathfrak{M}}$-*def* **by** *blast*
qed (*insert step-*$\mathcal{G}$[*OF* ‹*Only-G* $\mathcal{G}$›, *of Rep q'*], *unfold q-def Rep-step*[*unfolded ltl-prop-equiv-def*, *rule-format*, *symmetric*], *auto*)
}
moreover
have ($\exists j \ge i_{\mathfrak{M}}$. $\mathfrak{M}$.*rank* (*Suc m*) (*Suc m*) = *Some j*) $\vee$ $\mathcal{G} \models_P$ *Rep* (*Abs* φ)
using *token-trans*[*of Suc m*] **by** *simp*
hence *Abs* $\varphi \in$ *?lhs*
using $i_{\mathfrak{M}}$-*def* $\mathfrak{M}$.*rank-eq-state-rank*[*OF order-refl*] **by** (*cases* $\mathcal{G} \models_P$ *Rep* (*Abs* φ)) *simp+*
ultimately
show *?lhs* $\supseteq$ *?rhs*
unfolding $\mathfrak{M}.\mathcal{S}$.*simps* **by** *blast*
qed
qed
qed

end

14.3 Automaton Definition

fun $M_{\mathfrak{U}}$-*fin* :: (*'a ltl* $\rightharpoonup$ *nat*) $\Rightarrow$ (*'a* ltl_P $\times$ (*'a ltl* $\rightharpoonup$ *'a* ltl_P $\rightharpoonup$ *nat*), *'a set*) *transition set*
where
$M_{\mathfrak{U}}$-*fin* π = {((φ', *m*), ν, *p*). $\neg$($\forall S$. ($\forall \chi \in$ (*dom* π). $S \uparrow\models_P$ *Abs* χ $\wedge$ $S \uparrow\models_P \uparrow eval_G$ (*dom* π) (*Abs* (*theG* χ)) $\wedge$ ($\forall q$. ($\exists j \ge$ *the* (π χ). *the* (*m* χ) *q* = *Some j*) $\longrightarrow$ $S \uparrow\models_P \uparrow eval_G$ (*dom* π) ($\uparrow$*step q* ν))) $\longrightarrow$ $S \uparrow\models_P$ ($\uparrow$*step* φ' ν))}

locale *ltl-to-rabin-af-unf* = *ltl-to-rabin-base* $\uparrow af_{\mathfrak{U}}$ $\uparrow af_{G\mathfrak{U}}$ *Abs o Unf Abs o* Unf_G $M_{\mathfrak{U}}$-*fin* **begin**

abbreviation $\delta_{\mathfrak{U}}$ $\equiv$ *delta*
abbreviation $\iota_{\mathfrak{U}}$ $\equiv$ *initial*
abbreviation $Acc_{\mathfrak{U}}$-*fin* $\equiv$ *Acc-fin*
abbreviation $Acc_{\mathfrak{U}}$-*inf* $\equiv$ *Acc-inf*
abbreviation $F_{\mathfrak{U}}$ $\equiv$ *rabin-pairs*
abbreviation $Acc_{\mathfrak{U}}$ $\equiv$ *Acc*
abbreviation $\mathcal{A}_{\mathfrak{U}}$ $\equiv$ *ltl-to-generalized-rabin*

14.4 Properties

14.5 Correctness Theorem

lemma *unfold-optimisation-correct-M*:
assumes *dom* $\pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$
assumes *dom* $\pi_{\mathfrak{U}}$ = *dom* $\pi_{\mathcal{A}}$
assumes $\bigwedge\chi.\ \chi \in$ *dom* $\pi_{\mathcal{A}} \Longrightarrow \pi_{\mathcal{A}}\ \chi$ = *mojmir-def.smallest-accepting-rank* Σ $\uparrow af_G$ (*Abs* (*theG* χ)) *w* {*q*. *dom* $\pi_{\mathcal{A}}$ $\uparrow\models_P$ *q*}
assumes $\bigwedge\chi.\ \chi \in$ *dom* $\pi_{\mathfrak{U}} \Longrightarrow \pi_{\mathfrak{U}}\ \chi$ = *mojmir-def.smallest-accepting-rank* Σ *af-G-letter-abs-opt* (*Abs* (Unf_G (*theG* χ))) *w* {*q*. *dom* $\pi_{\mathfrak{U}}$ $\uparrow\models_P$ *q*}
shows *accepting-pair*$_R$ (*ltl-to-rabin-af*.$\delta_{\mathcal{A}}$ Σ) (*ltl-to-rabin-af*.$\iota_{\mathcal{A}}$ φ) (*M-fin* $\pi_{\mathcal{A}}$, *UNIV*) *w* $\longleftrightarrow$ *accepting-pair*$_R$ ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) ($M_{\mathfrak{U}}$-*fin* $\pi_{\mathfrak{U}}$, *UNIV*) *w*
proof –
— Preliminary Facts
note $\mathcal{G}$-*properties*[*OF* ‹*dom* $\pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$›]

interpret $\mathcal{A}$: *ltl-to-rabin-af*
using *ltl-to-generalized-rabin-af-wellformed bounded-w finite*-Σ **by** *auto*

— Define constants for both runs
define $r_{\mathcal{A}}$ $r_{\mathfrak{U}}$
where $r_{\mathcal{A}}$ = run_t (*ltl-to-rabin-af*.$\delta_{\mathcal{A}}$ Σ) (*ltl-to-rabin-af*.$\iota_{\mathcal{A}}$ φ) *w*
and $r_{\mathfrak{U}}$ = run_t ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) *w*
hence *finite* (*range* $r_{\mathcal{A}}$) **and** *finite* (*range* $r_{\mathfrak{U}}$)
using run_t-*finite*[*OF* $\mathcal{A}$.*finite-reach*] run_t-*finite*[*OF finite-reach*] *bounded-w finite*-Σ **by** *simp+*

— Prove that the limit of both runs behave the same in respect to the M acceptance condition
have *limit* $r_{\mathcal{A}}$ $\cap$ *M-fin* $\pi_{\mathcal{A}}$ = {} $\longleftrightarrow$ *limit* $r_{\mathfrak{U}}$ $\cap$ $M_{\mathfrak{U}}$-*fin* $\pi_{\mathfrak{U}}$ = {}
proof –
have *ltl-FG-to-rabin* Σ (*dom* $\pi_{\mathcal{A}}$) *w*
by (*unfold-locales*; *insert* $\mathcal{G}$-*elements*[*OF* ‹*dom* $\pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$›] *finite*-$\Sigma$ *bounded-w*)
hence *X*: $\bigwedge\chi.\ \chi \in$ *dom* $\pi_{\mathcal{A}} \Longrightarrow$ *mojmir-def.accept* $\uparrow af_G$ (*Abs* (*theG* χ)) *w* {*q*. *dom* $\pi_{\mathcal{A}}$ $\models_P$ *Rep q*}
by (*metis assms*(*3*)[*unfolded ltl-prop-entails-abs.rep-eq*] *ltl-FG-to-rabin.smallest-accepting-rank-pr* *domD*)
have $\forall_\infty i.\ \forall\chi \in$ *dom* $\pi_{\mathcal{A}}$. *mojmir-def*.$\mathcal{S}$ Σ $\uparrow af_G$ (*Abs* (*theG* χ)) *w* {*q*. *dom* $\pi_{\mathcal{A}}$ $\models_P$ *Rep q*} (*Suc i*)
= (λq. *step-abs q* (*w i*)) ‘ (*mojmir-def*.$\mathcal{S}$ Σ $\uparrow af_{G\mathfrak{U}}$ (*Abs* (Unf_G (*theG* χ))) *w* {*q*. *dom* $\pi_{\mathcal{A}}$ $\models_P$ *Rep q*} *i*) $\cup$ {*Abs* (*theG* χ)} $\cup$ {*q*. *dom* $\pi_{\mathcal{A}}$ $\models_P$ *Rep q*}

using *almost-all-commutative*′[*OF* ‹*finite* (*dom* $\pi_{\mathcal{A}}$)›] *X unfold-*$\mathcal{S}$*-eq*[*OF* ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)›] *finite-*Σ *bounded-w* **by** *simp*

then obtain *i* **where** *i-def*: $\bigwedge j\ \chi.\ j \geq i \Longrightarrow \chi \in dom\ \pi_{\mathcal{A}} \Longrightarrow$ *mojmir-def.*$\mathcal{S}$ Σ $\uparrow af_G$ (*Abs* (*theG* χ)) *w* {*q. dom* $\pi_{\mathcal{A}} \models_P$ *Rep q*} (*Suc j*)
= (λq. *step-abs q* (*w j*)) ‘ (*mojmir-def.*$\mathcal{S}$ Σ $\uparrow af_{G\mathfrak{U}}$ (*Abs* (*Unf*$_G$ (*theG* χ))) *w* {*q. dom* $\pi_{\mathcal{A}} \models_P$ *Rep q*} *j*) $\cup$ {*Abs* (*theG* χ)} $\cup$ {*q. dom* $\pi_{\mathcal{A}} \models_P$ *Rep q*}
unfolding *MOST-nat-le* **by** *blast*

obtain *j* **where** *limit* $r_{\mathcal{A}}$ = *range* (*suffix j* $r_{\mathcal{A}}$)
and *limit* $r_{\mathfrak{U}}$ = *range* (*suffix j* $r_{\mathfrak{U}}$)
using ‹*finite* (*range* $r_{\mathcal{A}}$)› ‹*finite* (*range* $r_{\mathfrak{U}}$)›
by (*rule common-range-limit*)
hence *limit* $r_{\mathcal{A}}$ = *range* (*suffix* (*j* + *i* + *1*) $r_{\mathcal{A}}$)
and *limit* $r_{\mathfrak{U}}$ = *range* (*suffix* (*j* + *i*) $r_{\mathfrak{U}}$)
by (*meson le-add1 limit-range-suffix-incr*)+
moreover
have $\bigwedge j.\ j \geq i \Longrightarrow r_{\mathcal{A}}$ (*Suc j*) $\in$ *M-fin* $\pi_{\mathcal{A}} \longleftrightarrow r_{\mathfrak{U}}$ *j* $\in$ $M_{\mathfrak{U}}$*-fin* $\pi_{\mathfrak{U}}$
proof −
fix *j*
assume $j \geq i$

obtain $\varphi_{\mathcal{A}}$ $m_{\mathcal{A}}$ *x* **where** $r_{\mathcal{A}}$*-def*′: $r_{\mathcal{A}}$ (*Suc j*) = (($\varphi_{\mathcal{A}}$, $m_{\mathcal{A}}$), *w* (*Suc j*), *x*)
unfolding $r_{\mathcal{A}}$*-def run*$_t$*.simps* **using** *prod.exhaust* **by** *fastforce*

obtain $\varphi_{\mathfrak{U}}$ $m_{\mathfrak{U}}$ *y* **where** $r_{\mathfrak{U}}$*-def*′: $r_{\mathfrak{U}}$ *j* = (($\varphi_{\mathfrak{U}}$, $m_{\mathfrak{U}}$), *w j*, *y*)
unfolding $r_{\mathfrak{U}}$*-def run*$_t$*.simps* **using** *prod.exhaust* **by** *fastforce*

have $m_{\mathcal{A}}$*-def*: $\bigwedge\chi\ q.\ \chi \in \mathbf{G}\ \varphi \Longrightarrow$ *the* ($m_{\mathcal{A}}$ χ) *q* = *semi-mojmir-def.state-rank* Σ $\uparrow af_G$ (*Abs* (*theG* χ)) *w q* (*Suc j*)
using $\mathcal{A}$*.run-properties*(*2*)[*of - φ Suc j*] $r_{\mathcal{A}}$*-def*′[*unfolded* $r_{\mathcal{A}}$*-def*] *prod.sel* **by** *simp*

have $m_{\mathfrak{U}}$*-def*: $\bigwedge\chi\ q.\ \chi \in \mathbf{G}\ \varphi \Longrightarrow$ *the* ($m_{\mathfrak{U}}$ χ) *q* = *semi-mojmir-def.state-rank* Σ $\uparrow af_{G\mathfrak{U}}$ (*Abs* (*Unf*$_G$ (*theG* χ))) *w q j*
using *run-properties*(*2*)[*of - φ j*] $r_{\mathfrak{U}}$*-def*′[*unfolded* $r_{\mathfrak{U}}$*-def*] *prod.sel* **by** *simp*

{
have *upt-Suc-0*: [*0..<Suc j*] = [*0..<j*] @ [*j*]
by *simp*
have *Rep* (*fst* (*fst* ($r_{\mathcal{A}}$ (*Suc j*)))) $\equiv_P$ *step* (*Rep* (*fst* (*fst* ($r_{\mathfrak{U}}$ *j*)))) (*w*

j)

unfolding $r_{\mathcal{A}}$-*def* $r_{\mathfrak{U}}$-*def* run_t.*simps fst-conv* $\mathcal{A}$.*run-properties*(*1*)[*of* φ *Suc j*] *run-properties*(*1*) *comp-apply*

unfolding *subsequence-def upt-Suc-0 map-append map-def list.map af-abs-equiv Unf-abs.abs-eq* **using** *Rep-step* **by** *auto*

hence *A*: $\bigwedge S.\ S \models_P Rep\ \varphi_{\mathcal{A}} \longleftrightarrow S \models_P step\ (Rep\ \varphi_{\mathfrak{U}})\ (w\ j)$

unfolding $r_{\mathcal{A}}$-*def'* $r_{\mathfrak{U}}$-*def'* *prod.sel ltl-prop-equiv-def* **..**

{

fix *S* **assume** $\bigwedge\chi.\ \chi \in dom\ \pi_{\mathcal{A}} \Longrightarrow S \models_P \chi$

hence $dom\ \pi_{\mathcal{A}} \subseteq S$

using ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)› *assms* **by** (*metis ltl-prop-entailment.simps*(*8*) *subsetI*)

{

fix χ **assume** $\chi \in dom\ \pi_{\mathcal{A}}$

interpret $\mathfrak{M}$: *ltl-FG-to-rabin* Σ *theG* χ *dom* $\pi_{\mathcal{A}}$

by (*unfold-locales*, *insert* ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)› *bounded-w finite-*Σ)

interpret $\mathfrak{U}$: *ltl-FG-to-rabin-opt* Σ *theG* χ *dom* $\pi_{\mathcal{A}}$

by (*unfold-locales*, *insert* ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)› *bounded-w finite-*Σ)

have $\bigwedge q\ \nu.\ dom\ \pi_{\mathcal{A}} \models_P Rep\ q \Longrightarrow dom\ \pi_{\mathcal{A}} \models_P Rep\ (step\text{-}abs\ q\ \nu)$

using ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)› **by** (*metis ltl-prop-equiv-def Rep-step step-*$\mathcal{G}$)

then have *subsetStep*: $\bigwedge\nu.\ (\lambda q.\ step\text{-}abs\ q\ \nu)$ ‘ $\{q.\ dom\ \pi_{\mathcal{A}} \models_P Rep\ q\} \subseteq \{q.\ dom\ \pi_{\mathcal{A}} \models_P Rep\ q\}$

by *auto*

let $?P = \lambda q.\ S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (Rep\ q)$

have $\bigwedge q\ \nu.\ (dom\ \pi_{\mathcal{A}}) \models_P Rep\ q \Longrightarrow ?P\ q$

using ‹*Only-G* (*dom* $\pi_{\mathcal{A}}$)› $eval_G$-*prop-entailment* ‹($dom\ \pi_{\mathcal{A}}) \subseteq S$› **by** *blast*

hence $\bigwedge q.\ q \in \{q.\ (dom\ \pi_{\mathcal{A}}) \models_P Rep\ q\} \Longrightarrow ?P\ q$

by *simp*

moreover

have *Y*: $\mathfrak{M}.\mathcal{S}\ (Suc\ j) = (\lambda q.\ step\text{-}abs\ q\ (w\ j))$ ‘ $(\mathfrak{U}.\mathcal{S}\ j) \cup \{Abs\ (theG\ \chi)\} \cup \{q.\ dom\ \pi_{\mathcal{A}} \models_P Rep\ q\}$

using *i-def*[*OF* ‹$j \geq i$› ‹$\chi \in dom\ \pi_{\mathcal{A}}$›] **by** *simp*

have *1*: $\mathfrak{M}$.*smallest-accepting-rank* = ($\pi_{\mathcal{A}}\ \chi$)

and *2*: $\mathfrak{U}$.*smallest-accepting-rank* = ($\pi_{\mathfrak{U}}\ \chi$)

and *3*: $\chi \in \mathbf{G}\ \varphi$

using ‹$\chi \in$ *dom* $\pi_{\mathcal{A}}$› *assms*[*unfolded ltl-prop-entails-abs.rep-eq*]
by *auto*
ultimately
have $(\forall q \in \mathfrak{M}.\mathcal{S}\ (Suc\ j).\ ?P\ q) = (\forall q \in (\lambda q.\ step\text{-}abs\ q\ (w\ j))$ ‘ $(\mathfrak{U}.\mathcal{S}\ j) \cup \{Abs\ (theG\ \chi)\}.\ ?P\ q)$
unfolding Y **by** *blast*
hence *4*: $(\forall q.\ (\exists j \geq the\ (\pi_{\mathcal{A}}\ \chi).\ the\ (m_{\mathcal{A}}\ \chi)\ q = Some\ j) \longrightarrow ?P\ q) = ((\forall q.\ (\exists j \geq the\ (\pi_{\mathfrak{U}}\ \chi).\ the\ (m_{\mathfrak{U}}\ \chi)\ q = Some\ j) \longrightarrow ?P\ (step\text{-}abs\ q\ (w\ j))) \wedge ?P\ (Abs\ (theG\ \chi)))$
using ‹$\bigwedge q.\ q \in \{q.\ dom\ \pi_{\mathcal{A}} \models_P Rep\ q\} \Longrightarrow ?P\ q$› *subsetStep*
unfolding $m_{\mathcal{A}}$-*def*[*OF 3*, *symmetric*] $m_{\mathfrak{U}}$-*def*[*OF 3*, *symmetric*] $\mathfrak{M}.\mathcal{S}$.*simps* $\mathfrak{U}.\mathcal{S}$.*simps 1 2 Set.image-Un option.sel* **by** *blast*
have $S \models_P \chi \wedge (\forall q.\ (\exists j \geq the\ (\pi_{\mathcal{A}}\ \chi).\ the\ (m_{\mathcal{A}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (Rep\ q)) \longleftrightarrow$
$S \models_P \chi \wedge S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (theG\ \chi) \wedge (\forall q.\ (\exists j \geq the\ (\pi_{\mathfrak{U}}\ \chi).\ the\ (m_{\mathfrak{U}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (step\ (Rep\ q)\ (w\ j)))$
unfolding *4* **using** $eval_G$-*respectfulness*(*2*)[*OF Rep-Abs-equiv*, *unfolded ltl-prop-equiv-def*]
using $eval_G$-*respectfulness*(*2*)[*OF Rep-step*, *unfolded ltl-prop-equiv-def*]
by *blast*
}
hence $((\forall \chi \in dom\ \pi_{\mathcal{A}}.\ S \models_P \chi \wedge (\forall q.\ (\exists j \geq the\ (\pi_{\mathcal{A}}\ \chi).\ the\ (m_{\mathcal{A}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (Rep\ q))) \longrightarrow S \models_P Rep\ \varphi_{\mathcal{A}})$
$\longleftrightarrow ((\forall \chi \in dom\ \pi_{\mathfrak{U}}.\ S \models_P \chi \wedge S \models_P eval_G\ (dom\ \pi_{\mathfrak{U}})\ (theG\ \chi) \wedge (\forall q.\ (\exists j \geq the\ (\pi_{\mathfrak{U}}\ \chi).\ the\ (m_{\mathfrak{U}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathfrak{U}})\ (step\ (Rep\ q)\ (w\ j)))) \longrightarrow S \models_P step\ (Rep\ \varphi_{\mathfrak{U}})\ (w\ j))$
by (*simp add*: ‹$\bigwedge \chi.\ \chi \in dom\ \pi_{\mathcal{A}} \Longrightarrow (S \models_P \chi \wedge (\forall q.\ (\exists j{\geq}the\ (\pi_{\mathcal{A}}\ \chi).\ the\ (m_{\mathcal{A}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (Rep\ q))) = (S \models_P \chi \wedge S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (theG\ \chi) \wedge (\forall q.\ (\exists j{\geq}the\ (\pi_{\mathfrak{U}}\ \chi).\ the\ (m_{\mathfrak{U}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (step\ (Rep\ q)\ (w\ j))))$› *A assms*(*2*))
}
hence $(\forall S.\ (\forall \chi \in dom\ \pi_{\mathcal{A}}.\ S \models_P \chi \wedge (\forall q.\ (\exists j \geq the\ (\pi_{\mathcal{A}}\ \chi).\ the\ (m_{\mathcal{A}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathcal{A}})\ (Rep\ q))) \longrightarrow S \models_P Rep\ \varphi_{\mathcal{A}}) \longleftrightarrow$
$(\forall S.\ (\forall \chi \in dom\ \pi_{\mathfrak{U}}.\ S \models_P \chi \wedge S \models_P eval_G\ (dom\ \pi_{\mathfrak{U}})\ (theG\ \chi) \wedge (\forall q.\ (\exists j \geq the\ (\pi_{\mathfrak{U}}\ \chi).\ the\ (m_{\mathfrak{U}}\ \chi)\ q = Some\ j) \longrightarrow S \models_P eval_G\ (dom\ \pi_{\mathfrak{U}})\ (step\ (Rep\ q)\ (w\ j)))) \longrightarrow S \models_P step\ (Rep\ \varphi_{\mathfrak{U}})\ (w\ j))$
unfolding *assms* **by** *auto*
}
hence $((\varphi_{\mathcal{A}}, m_{\mathcal{A}}), w\ (Suc\ j), x) \in M\text{-}fin\ \pi_{\mathcal{A}} \longleftrightarrow ((\varphi_{\mathfrak{U}}, m_{\mathfrak{U}}), w\ j, y) \in M_{\mathfrak{U}}\text{-}fin\ \pi_{\mathfrak{U}}$
unfolding *M-fin.simps* $M_{\mathfrak{U}}$-*fin.simps ltl-prop-entails-abs.abs-eq*[*symmetric*]

$eval_G$*-abs.abs-eq*[*symmetric*] ltl_P*-abs-rep step-abs.abs-eq*[*symmetric*] **by** *blast*
thus *?thesis j*
unfolding $r_{\mathcal{A}}$*-def'* $r_{\mathfrak{U}}$*-def'* **.**
qed
hence $\bigwedge n.\ r_{\mathcal{A}}\ (j + i + 1 + n) \in M\text{-}fin\ \pi_{\mathcal{A}} \longleftrightarrow r_{\mathfrak{U}}\ (j + i + n) \in M_{\mathfrak{U}}\text{-}fin\ \pi_{\mathfrak{U}}$
by *simp*
hence $range\ (suffix\ (j + i + 1)\ r_{\mathcal{A}}) \cap M\text{-}fin\ \pi_{\mathcal{A}} = \{\} \longleftrightarrow range\ (suffix\ (j + i)\ r_{\mathfrak{U}}) \cap M_{\mathfrak{U}}\text{-}fin\ \pi_{\mathfrak{U}} = \{\}$
unfolding *suffix-def* **by** *blast*
ultimately
show $limit\ r_{\mathcal{A}} \cap M\text{-}fin\ \pi_{\mathcal{A}} = \{\} \longleftrightarrow limit\ r_{\mathfrak{U}} \cap M_{\mathfrak{U}}\text{-}fin\ \pi_{\mathfrak{U}} = \{\}$
by *force*
qed
moreover
have $limit\ r_{\mathcal{A}} \cap UNIV \neq \{\}$ **and** $limit\ r_{\mathfrak{U}} \cap UNIV \neq \{\}$
using *limit-nonempty* ‹*finite* (*range* $r_{\mathfrak{U}}$)› ‹*finite* (*range* $r_{\mathcal{A}}$)› **by** *auto*
ultimately
show *?thesis*
unfolding *accepting-pair*$_R$*-def fst-conv snd-conv* $r_{\mathcal{A}}$*-def*[*symmetric*] $r_{\mathfrak{U}}$*-def*[*symmetric*] *Let-def* **by** *blast*
qed

theorem *ltl-to-generalized-rabin-correct*:
$w \models \varphi \longleftrightarrow accept_{GR}\ (\mathcal{A}_{\mathfrak{U}}\ \Sigma\ \varphi)\ w$
(**is** - $\longleftrightarrow$ *?rhs*)
proof (*unfold ltl-to-generalized-rabin-af-correct*[*OF finite-*Σ *bounded-w*], *standard*)
let *?lhs* = $accept_{GR}$ (*ltl-to-generalized-rabin-af* $\Sigma\ \varphi$) *w*

interpret $\mathcal{A}$: *ltl-to-rabin-af* Σ *w*
using *ltl-to-generalized-rabin-af-wellformed bounded-w finite-*Σ **by** *auto*

{
assume *?lhs*
then obtain π **where** *I*: $dom\ \pi \subseteq \mathbf{G}\ \varphi$
and *II*: $\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow the\ (\pi\ \chi) < \mathcal{A}.max\text{-}rank\text{-}of\ \Sigma\ \chi$
and *III*: *accepting-pair*$_R$ (*ltl-to-rabin-af*.$\delta_{\mathcal{A}}$ Σ) (*ltl-to-rabin-af*.$\iota_{\mathcal{A}}$ φ) (*M-fin* π, *UNIV*) *w*
and *IV*: $\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow$ *accepting-pair*$_R$ ($\mathcal{A}.\delta_{\mathcal{A}}$ Σ) (*ltl-to-rabin-af*.$\iota_{\mathcal{A}}$ φ) ($\mathcal{A}$.*Acc* $\Sigma\ \pi\ \chi$) *w*
by (*unfold ltl-to-generalized-rabin-af.simps*; *blast intro*: $\mathcal{A}.accept_{GR}$*-I*)

— Normalise π to the smallest accepting ranks

then obtain $\pi_{\mathcal{A}}$ **where** A: $dom\ \pi = dom\ \pi_{\mathcal{A}}$
and B: $\bigwedge\chi.\ \chi \in dom\ \pi_{\mathcal{A}} \Longrightarrow \pi_{\mathcal{A}}\ \chi = mojmir\text{-}def.smallest\text{-}accepting\text{-}rank$ $\Sigma\ \uparrow af_G\ (Abs\ (theG\ \chi))\ w\ \{q.\ dom\ \pi_{\mathcal{A}}\ \uparrow\models_P\ q\}$
and C: $accepting\text{-}pair_R\ (\mathcal{A}.\delta_{\mathcal{A}}\ \Sigma)\ (\mathcal{A}.\iota_{\mathcal{A}}\ \varphi)\ (M\text{-}fin\ \pi_{\mathcal{A}},\ UNIV)\ w$
and D: $\bigwedge\chi.\ \chi \in dom\ \pi_{\mathcal{A}} \Longrightarrow accepting\text{-}pair_R\ (\mathcal{A}.\delta_{\mathcal{A}}\ \Sigma)\ (\mathcal{A}.\iota_{\mathcal{A}}\ \varphi)$ $(\mathcal{A}.Acc\ \Sigma\ \pi_{\mathcal{A}}\ \chi)\ w$
using $\mathcal{A}.normalize\text{-}\pi$ **by** *blast*

— Properties about the domain of π
note $\mathcal{G}$*-properties*[*OF* ‹$dom\ \pi \subseteq \mathbf{G}\ \varphi$›]
hence $\mathfrak{M}$*-Accept*: $\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow mojmir\text{-}def.accept\ af\text{-}G\text{-}letter\text{-}abs$ $(Abs\ (theG\ \chi))\ w\ \{q.\ dom\ \pi\ \uparrow\models_P\ q\}$
using *I II IV* $\mathcal{A}$*.Acc-to-mojmir-accept* **unfolding** *ltl-to-rabin-base-def.max-rank-of-def* **by** (*metis ltl.sel*(*8*))
hence $\mathfrak{U}$*-Accept*: $\bigwedge\chi.\ \chi \in dom\ \pi \Longrightarrow mojmir\text{-}def.accept\ af\text{-}G\text{-}letter\text{-}abs\text{-}opt$ $(Abs\ (Unf_G\ (theG\ \chi)))\ w\ \{q.\ dom\ \pi\ \uparrow\models_P\ q\}$
using *unfold-accept-eq*[*OF* ‹*Only-G* (*dom* π)› *finite-*Σ *bounded-w*] **unfolding** *ltl-prop-entails-abs.rep-eq* **by** *blast*

— Define π for the other automaton
define $\pi_{\mathfrak{U}}$
where $\pi_{\mathfrak{U}}\ \chi = (if\ \chi \in dom\ \pi\ then\ mojmir\text{-}def.smallest\text{-}accepting\text{-}rank$ $\Sigma\ af\text{-}G\text{-}letter\text{-}abs\text{-}opt\ (Abs\ (Unf_G\ (theG\ \chi)))\ w\ \{q.\ dom\ \pi\ \uparrow\models_P\ q\}\ else$ $None)$
for χ

have *1*: $dom\ \pi_{\mathfrak{U}} = dom\ \pi$
using $\mathfrak{U}$*-Accept* **by** (*auto simp add*: $\pi_{\mathfrak{U}}$*-def dom-def mojmir-def.smallest-accepting-rank-def*)

hence $dom\ \pi_{\mathfrak{U}} = dom\ \pi_{\mathcal{A}}$ **and** $dom\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$ **and** $dom\ \pi_{\mathfrak{U}} \subseteq \mathbf{G}\ \varphi$
using A ‹$dom\ \pi \subseteq \mathbf{G}\ \varphi$› **by** *blast+*
have *2*: $\bigwedge\chi.\ \chi \in dom\ \pi_{\mathfrak{U}} \Longrightarrow \pi_{\mathfrak{U}}\ \chi = mojmir\text{-}def.smallest\text{-}accepting\text{-}rank$ $\Sigma\ af\text{-}G\text{-}letter\text{-}abs\text{-}opt\ (Abs\ (Unf_G\ (theG\ \chi)))\ w\ \{q.\ dom\ \pi_{\mathfrak{U}}\ \uparrow\models_P\ q\}$
using *1* **unfolding** ‹$dom\ \pi_{\mathfrak{U}} = dom\ \pi$› $\pi_{\mathfrak{U}}$*-def* **by** *simp*
hence *3*: $\bigwedge\chi.\ \chi \in dom\ \pi_{\mathfrak{U}} \Longrightarrow the\ (\pi_{\mathfrak{U}}\ \chi) < semi\text{-}mojmir\text{-}def.max\text{-}rank$ $\Sigma\ af\text{-}G\text{-}letter\text{-}abs\text{-}opt\ (Abs\ (Unf_G\ (theG\ \chi)))$
using *wellformed-mojmir-opt*[*OF* $\mathcal{G}$*-elements*[*OF* ‹$dom\ \pi_{\mathfrak{U}} \subseteq \mathbf{G}\ \varphi$›] *finite-*$\Sigma$ *bounded-w*, *THEN mojmir.smallest-accepting-rank-properties*(*6*)]
unfolding *ltl-prop-entails-abs.rep-eq* **by** *fastforce*

— Use correctness of the translation of individual accepting pairs
have *Acc*: $\bigwedge\chi.\ \chi \in dom\ \pi_{\mathfrak{U}} \Longrightarrow accepting\text{-}pair_R\ (\delta_{\mathfrak{U}}\ \Sigma)\ (\iota_{\mathfrak{U}}\ \varphi)\ (Acc_{\mathfrak{U}}\ \Sigma$ $\pi_{\mathfrak{U}}\ \chi)\ w$
using *mojmir-accept-to-Acc*[*OF* - ‹$dom\ \pi_{\mathfrak{U}} \subseteq \mathbf{G}\ \varphi$›] $\mathcal{G}$*-elements*[*OF*

‹*dom* $\pi_{\mathfrak{U}} \subseteq$ **G** φ›]
using *1 2*[*of G -*] *3*[*of G -*] $\mathfrak{U}$-*Accept*[*of G -*] *ltl.sel*(*8*) **unfolding** *comp-apply* **by** *metis*
have *M*: *accepting-pair*$_R$ ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) ($M_{\mathfrak{U}}$-*fin* $\pi_{\mathfrak{U}}$, *UNIV*) *w*
using *unfold-optimisation-correct-M*[*OF* ‹*dom* $\pi_{\mathcal{A}} \subseteq$ **G** φ› ‹*dom* $\pi_{\mathfrak{U}}$ = *dom* $\pi_{\mathcal{A}}$› *B 2*] *C*
using ‹*dom* $\pi_{\mathfrak{U}}$ = *dom* $\pi_{\mathcal{A}}$› **by** *blast+*

show *?rhs*
using *Acc 3* ‹*dom* $\pi_{\mathfrak{U}} \subseteq$ **G** φ› *combine-rabin-pairs-UNIV*[*OF M combine-rabin-pairs*]
by (*simp only*: *accept*$_{GR}$-*def fst-conv snd-conv ltl-to-generalized-rabin.simps rabin-pairs.simps max-rank-of-def comp-apply*) *blast*
}

{
assume *?rhs*
then obtain π **where** *I*: *dom* $\pi \subseteq$ **G** φ
and *II*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *the* (π χ) < *max-rank-of* Σ χ
and *III*: *accepting-pair*$_R$ ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) ($M_{\mathfrak{U}}$-*fin* π, *UNIV*) *w*
and *IV*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *accepting-pair*$_R$ ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) (*Acc*$_{\mathfrak{U}}$ Σ π χ) *w*
by (*blast intro*: *accept*$_{GR}$-*I*)

— Normalize π to the smallest accepting ranks
then obtain $\pi_{\mathfrak{U}}$ **where** *A*: *dom* π = *dom* $\pi_{\mathfrak{U}}$
and *B*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi_{\mathfrak{U}} \Longrightarrow \pi_{\mathfrak{U}}$ χ = *mojmir-def.smallest-accepting-rank* Σ $\uparrow$*af*$_{G\mathfrak{U}}$ (*Abs* (*Unf*$_G$ (*theG* χ))) *w* {*q*. *dom* $\pi_{\mathfrak{U}}$ $\uparrow\models_P$ *q*}
and *C*: *accepting-pair*$_R$ ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) ($M_{\mathfrak{U}}$-*fin* $\pi_{\mathfrak{U}}$, *UNIV*) *w*
and *D*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi_{\mathfrak{U}} \Longrightarrow$ *accepting-pair*$_R$ ($\delta_{\mathfrak{U}}$ Σ) ($\iota_{\mathfrak{U}}$ φ) (*Acc*$_{\mathfrak{U}}$ Σ $\pi_{\mathfrak{U}}$ χ) *w*
using *normalize-*π **unfolding** *comp-apply* **by** *blast*

— Properties about the domain of π
note $\mathcal{G}$-*properties*[*OF* ‹*dom* $\pi \subseteq$ **G** φ›]
hence $\mathfrak{U}$-*Accept*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *mojmir-def.accept af-G-letter-abs-opt* (*Abs* (*Unf*$_G$ (*theG* χ))) *w* {*q*. *dom* π $\uparrow\models_P$ *q*}
using *I II IV Acc-to-mojmir-accept* **unfolding** *max-rank-of-def comp-apply* **by** (*metis ltl.sel*(*8*))
hence $\mathfrak{M}$-*Accept*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *mojmir-def.accept af-G-letter-abs* (*Abs* (*theG* χ)) *w* {*q*. *dom* π $\uparrow\models_P$ *q*}
using *unfold-accept-eq*[*OF* ‹*Only-G* (*dom* π)› *finite-*Σ *bounded-w*]
unfolding *ltl-prop-entails-abs.rep-eq* **by** *blast*

— Define π for the other automaton
define $\pi_{\mathcal{A}}$
where $\pi_{\mathcal{A}}\ \chi = (\textit{if}\ \chi \in \textit{dom}\ \pi\ \textit{then mojmir-def.smallest-accepting-rank}$
$\Sigma\ \uparrow\!af_G\ (\textit{Abs}\ (\textit{theG}\ \chi))\ w\ \{q.\ \textit{dom}\ \pi\ \uparrow\models_P\ q\}\ \textit{else None})$
for χ

have *1*: $\textit{dom}\ \pi_{\mathcal{A}} = \textit{dom}\ \pi$
using $\mathfrak{M}$*-Accept* **by** (*auto simp add*: $\pi_{\mathcal{A}}$*-def dom-def mojmir-def.smallest-accepting-rank-def*)

hence $\textit{dom}\ \pi_{\mathfrak{U}} = \textit{dom}\ \pi_{\mathcal{A}}$ **and** $\textit{dom}\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$ **and** $\textit{dom}\ \pi_{\mathfrak{U}} \subseteq \mathbf{G}\ \varphi$
using *A* ‹$\textit{dom}\ \pi \subseteq \mathbf{G}\ \varphi$› **by** *blast+*
hence *ltl-FG-to-rabin* Σ ($\textit{dom}\ \pi_{\mathcal{A}}$) *w*
by (*unfold-locales*; *insert* $\mathcal{G}$*-elements*[*OF* ‹$\textit{dom}\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$›] *finite-*$\Sigma$
bounded-w)
have *2*: $\bigwedge\chi.\ \chi \in \textit{dom}\ \pi_{\mathcal{A}} \Longrightarrow \pi_{\mathcal{A}}\ \chi = \textit{mojmir-def.smallest-accepting-rank}$
$\Sigma\ \uparrow\!af_G\ (\textit{Abs}\ (\textit{theG}\ \chi))\ w\ \{q.\ \textit{dom}\ \pi_{\mathcal{A}}\ \uparrow\models_P\ q\}$
using *1* **unfolding** ‹$\textit{dom}\ \pi_{\mathcal{A}} = \textit{dom}\ \pi$› $\pi_{\mathcal{A}}$*-def* **by** *simp*
hence *3*: $\bigwedge\chi.\ \chi \in \textit{dom}\ \pi_{\mathcal{A}} \Longrightarrow \textit{the}\ (\pi_{\mathcal{A}}\ \chi) < \textit{semi-mojmir-def.max-rank}$
$\Sigma\ \uparrow\!af_G\ (\textit{Abs}\ (\textit{theG}\ \chi))$
using *ltl-FG-to-rabin.smallest-accepting-rank-properties*(*6*)[*OF* ‹*ltl-FG-to-rabin*
Σ ($\textit{dom}\ \pi_{\mathcal{A}}$) *w*›]
unfolding *ltl-prop-entails-abs.rep-eq* **by** *fastforce*

— Use correctness of the translation of individual accepting pairs
have *Acc*: $\bigwedge\chi.\ \chi \in \textit{dom}\ \pi_{\mathcal{A}} \Longrightarrow \textit{accepting-pair}_R\ (\mathcal{A}.\delta_{\mathcal{A}}\ \Sigma)\ (\mathcal{A}.\iota_{\mathcal{A}}\ \varphi)$
$(\mathcal{A}.\textit{Acc}\ \Sigma\ \pi_{\mathcal{A}}\ \chi)\ w$
using $\mathcal{A}$*.mojmir-accept-to-Acc*[*OF* - ‹$\textit{dom}\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$›] $\mathcal{G}$*-elements*[*OF*
‹$\textit{dom}\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$›]
using *1 2*[*of G* -] *3*[*of G* -] $\mathfrak{M}$*-Accept*[*of G* -] *ltl.sel*(*8*) **by** *metis*
have *M*: $\textit{accepting-pair}_R\ (\mathcal{A}.\delta_{\mathcal{A}}\ \Sigma)\ (\mathcal{A}.\iota_{\mathcal{A}}\ \varphi)\ (\textit{M-fin}\ \pi_{\mathcal{A}},\ \textit{UNIV})\ w$
using *unfold-optimisation-correct-M*[*OF* ‹$\textit{dom}\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$› ‹$\textit{dom}\ \pi_{\mathfrak{U}} =$
$\textit{dom}\ \pi_{\mathcal{A}}$› *2 B*] *C*
using ‹$\textit{dom}\ \pi_{\mathfrak{U}} = \textit{dom}\ \pi_{\mathcal{A}}$› **by** *blast+*

show *?lhs*
using *Acc 3* ‹$\textit{dom}\ \pi_{\mathcal{A}} \subseteq \mathbf{G}\ \varphi$› *combine-rabin-pairs-UNIV*[*OF M*
combine-rabin-pairs]
by (*simp only*: $\textit{accept}_{GR}$*-def fst-conv snd-conv* $\mathcal{A}$*.ltl-to-generalized-rabin.simps*
$\mathcal{A}$*.rabin-pairs.simps*
ltl-to-generalized-rabin-af.simps $\mathcal{A}$*.max-rank-of-def*
comp-apply) *blast*
}
qed

end

fun *ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$
where
ltl-to-generalized-rabin-af$_{\mathfrak{U}}$ Σ φ = *ltl-to-rabin-base-def.ltl-to-generalized-rabin* $\uparrow$*af*$_{\mathfrak{U}}$ $\uparrow$*af*$_{G\mathfrak{U}}$ (*Abs* $\circ$ *Unf*) (*Abs* $\circ$ *Unf*$_G$) *M*$_{\mathfrak{U}}$*-fin* Σ φ

lemma *ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$*-wellformed*:
finite Σ $\Longrightarrow$ *range w* $\subseteq$ Σ $\Longrightarrow$ *ltl-to-rabin-af-unf* Σ *w*
apply (*unfold-locales*)
apply (*auto simp add*: *af-G-letter-opt-sat-core-lifted ltl-prop-entails-abs.rep-eq intro*: *finite-reach-af-opt finite-reach-af-G-opt*)
apply (*meson le-trans ltl-semi-mojmir*[*unfolded semi-mojmir-def*])+
done

theorem *ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$*-correct*:
assumes *finite* Σ
assumes *range w* $\subseteq$ Σ
shows *w* $\models$ φ = *accept*$_{GR}$ (*ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$ Σ φ) *w*
using *ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$*-wellformed*[*OF assms*, *THEN ltl-to-rabin-af-unf.ltl-to-generalized-*
by *simp*

thm *ltl-FG-to-generalized-rabin-correct ltl-to-generalized-rabin-af-correct ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$*-*

end

15 LTL Translation Layer

theory *LTL-Compat*
imports *Main LTL.LTL ../LTL-FGXU*
begin

— The following infrastructure translates the generic **datatype** *'a ltln* = *true*$_n$ | *false*$_n$ | *Prop-ltln 'a* | *Nprop-ltln 'a* | *And-ltln* (*'a ltln*) (*'a ltln*) | *Or-ltln* (*'a ltln*) (*'a ltln*) | *Next-ltln* (*'a ltln*) | *Until-ltln* (*'a ltln*) (*'a ltln*) | *Release-ltln* (*'a ltln*) (*'a ltln*) | *WeakUntil-ltln* (*'a ltln*) (*'a ltln*) | *StrongRelease-ltln* (*'a ltln*) (*'a ltln*) datatype to special structure used in this project

abbreviation *LTLRelease* :: *'a ltl* ⇒ *'a ltl* ⇒ *'a ltl* (‹- *R* -› [*87*,*87*] *86*)
where
φ *R* ψ ≡ (*G* ψ) *or* (ψ *U* (φ *and* ψ))

abbreviation *LTLWeakUntil* :: *'a ltl* ⇒ *'a ltl* ⇒ *'a ltl* (‹- *W* -› [*87*,*87*] *86*)

where
  $\varphi\ W\ \psi \equiv (\varphi\ U\ \psi)\ or\ (G\ \varphi)$

abbreviation *LTLStrongRelease* :: *'a ltl* ⇒ *'a ltl* ⇒ *'a ltl* (‹- *M* -› [*87*,*87*] *86*)
where
  $\varphi\ M\ \psi \equiv \psi\ U\ (\varphi\ and\ \psi)$

fun *ltln-to-ltl* :: *'a ltln* ⇒ *'a ltl*
where
  ltln-to-ltl $true_n$ = *true*
| *ltln-to-ltl* $false_n$ = *false*
| *ltln-to-ltl* $prop_n(q) = p(q)$
| *ltln-to-ltl* $nprop_n(q) = np(q)$
| *ltln-to-ltl* $(\varphi\ and_n\ \psi)$ = *ltln-to-ltl* φ *and ltln-to-ltl* ψ
| *ltln-to-ltl* $(\varphi\ or_n\ \psi)$ = *ltln-to-ltl* φ *or ltln-to-ltl* ψ
| *ltln-to-ltl* $(\varphi\ U_n\ \psi)$ = (*if* $\varphi = true_n$ *then F* (*ltln-to-ltl* ψ) *else* (*ltln-to-ltl* φ) *U* (*ltln-to-ltl* ψ))
| *ltln-to-ltl* $(\varphi\ R_n\ \psi)$ = (*if* $\varphi = false_n$ *then G* (*ltln-to-ltl* ψ) *else* (*ltln-to-ltl* φ) *R* (*ltln-to-ltl* ψ))
| *ltln-to-ltl* $(\varphi\ W_n\ \psi)$ = (*if* $\psi = false_n$ *then G* (*ltln-to-ltl* φ) *else* (*ltln-to-ltl* φ) *W* (*ltln-to-ltl* ψ))
| *ltln-to-ltl* $(\varphi\ M_n\ \psi)$ = (*if* $\psi = true_n$ *then F* (*ltln-to-ltl* φ) *else* (*ltln-to-ltl* φ) *M* (*ltln-to-ltl* ψ))
| *ltln-to-ltl* $(X_n\ \varphi)$ = *X* (*ltln-to-ltl* φ)

lemma *ltln-to-ltl-semantics*:
  $w \models$ *ltln-to-ltl* $\varphi \longleftrightarrow w \models_n \varphi$
  by (*induction* φ *arbitrary*: *w*)
    (*simp-all del*: *semantics-ltln.simps*(*9*−*11*), *unfold ltln-Release-alterdef ltln-weak-strong*(*1*) *ltl-StrongRelease-Until-con*, *insert nat-less-le*, *auto*)

lemma *ltln-to-ltl-atoms*:
  vars (*ltln-to-ltl* φ) = *atoms-ltln* φ
  by (*induction* φ) *auto*

fun *atoms-list* :: *'a ltln* ⇒*'a list*
where
  atoms-list $(\varphi\ and_n\ \psi)$ = *List.union* (*atoms-list* φ) (*atoms-list* ψ)
| *atoms-list* $(\varphi\ or_n\ \psi)$ = *List.union* (*atoms-list* φ) (*atoms-list* ψ)
| *atoms-list* $(\varphi\ U_n\ \psi)$ = *List.union* (*atoms-list* φ) (*atoms-list* ψ)
| *atoms-list* $(\varphi\ R_n\ \psi)$ = *List.union* (*atoms-list* φ) (*atoms-list* ψ)
| *atoms-list* $(\varphi\ W_n\ \psi)$ = *List.union* (*atoms-list* φ) (*atoms-list* ψ)
| *atoms-list* $(\varphi\ M_n\ \psi)$ = *List.union* (*atoms-list* φ) (*atoms-list* ψ)

| *atoms-list* (X_n φ) = *atoms-list* φ
| *atoms-list* ($prop_n(a)$) = [a]
| *atoms-list* ($nprop_n(a)$) = [a]
| *atoms-list* - = []

lemma *atoms-list-correct*:
  set (*atoms-list* φ) = *atoms-ltln* φ
  by (*induction* φ) *auto*

lemma *atoms-list-distinct*:
  distinct (*atoms-list* φ)
  by (*induction* φ) *auto*

end

16 LTL Code Equations

theory *LTL-Impl*
  imports *Main*
    ../LTL-FGXU
    Boolean-Expression-Checkers.Boolean-Expression-Checkers
    Boolean-Expression-Checkers.Boolean-Expression-Checkers-AList-Mapping
begin

16.1 Subformulae

fun *G-list* :: *'a ltl* ⇒ *'a ltl list*
where
  G-list (φ *and* ψ) = *List.union* (*G-list* φ) (*G-list* ψ)
| *G-list* (φ *or* ψ) = *List.union* (*G-list* φ) (*G-list* ψ)
| *G-list* (*F* φ) = *G-list* φ
| *G-list* (*G* φ) = *List.insert* (*G* φ) (*G-list* φ)
| *G-list* (*X* φ) = *G-list* φ
| *G-list* (φ *U* ψ) = *List.union* (*G-list* φ) (*G-list* ψ)
| *G-list* φ = []

lemma *G-eq-G-list*:
  G φ = *set* (*G-list* φ)
  by (*induction* φ) *auto*

lemma *G-list-distinct*:
  distinct (*G-list* φ)
  by (*induction* φ) *auto*

16.2 Propositional Equivalence

fun *ifex-of-ltl* :: $'a$ *ltl* $\Rightarrow$ $'a$ *ltl ifex*
where
ifex-of-ltl true = *Trueif*
| *ifex-of-ltl false* = *Falseif*
| *ifex-of-ltl* (φ *and* ψ) = *normif Mapping.empty* (*ifex-of-ltl* φ) (*ifex-of-ltl* ψ) *Falseif*
| *ifex-of-ltl* (φ *or* ψ) = *normif Mapping.empty* (*ifex-of-ltl* φ) *Trueif* (*ifex-of-ltl* ψ)
| *ifex-of-ltl* φ = *IF* φ *Trueif Falseif*

lemma *val-ifex*:
val-ifex (*ifex-of-ltl b*) *s* = ($\models_P$) $\{x.\ s\ x\}$ *b*
by (*induction b*) (*simp add*: *agree-Nil val-normif*)+

lemma *reduced-ifex*:
reduced (*ifex-of-ltl b*) {}
by (*induction b*) (*simp*; *metis keys-empty reduced-normif*)+

lemma *ifex-of-ltl-reduced-bdt-checker*:
reduced-bdt-checkers ifex-of-ltl ($\lambda y\ s.\ \{x.\ s\ x\} \models_P y$)
by (*unfold reduced-bdt-checkers-def*; *insert val-ifex reduced-ifex*; *blast*)

lemma [*code*]:
($\varphi \equiv_P \psi$) = *equiv-test ifex-of-ltl* φ ψ
by (*simp add*: *ltl-prop-equiv-def reduced-bdt-checkers.equiv-test*[*OF ifex-of-ltl-reduced-bdt-checker*]; *fastforce*)

lemma [*code*]:
($\varphi \longrightarrow_P \psi$) = *impl-test ifex-of-ltl* φ ψ
by (*simp add*: *ltl-prop-implies-def reduced-bdt-checkers.impl-test*[*OF ifex-of-ltl-reduced-bdt-checker*]; *force*)

— Check Code Export
export-code ($\equiv_P$) ($\longrightarrow_P$) **checking**

16.3 Remove Constants

fun *remove-constants*$_P$ :: $'a$ *ltl* $\Rightarrow$ $'a$ *ltl*
where
remove-constants$_P$ (φ *and* ψ) = (
case (*remove-constants*$_P$ φ) *of*
false $\Rightarrow$ *false*

| $true \Rightarrow$ *remove-constants*$_P$ ψ
| $\varphi' \Rightarrow$ (*case remove-constants*$_P$ ψ *of*
false $\Rightarrow$ *false*
| *true* $\Rightarrow \varphi'$
| $\psi' \Rightarrow \varphi'$ *and* ψ'))
| *remove-constants*$_P$ (φ *or* ψ) = (
case (*remove-constants*$_P$ φ) *of*
true $\Rightarrow$ *true*
| *false* $\Rightarrow$ *remove-constants*$_P$ ψ
| $\varphi' \Rightarrow$ (*case remove-constants*$_P$ ψ *of*
true $\Rightarrow$ *true*
| *false* $\Rightarrow \varphi'$
| $\psi' \Rightarrow \varphi'$ *or* ψ'))
| *remove-constants*$_P$ $\varphi = \varphi$

lemma *remove-constants-correct*:
$S \models_P \varphi \longleftrightarrow S \models_P$ *remove-constants*$_P$ φ
by (*induction* φ *arbitrary*: S) (*auto split*: *ltl.split*)

16.4 And/Or Constructors

fun *in-and*
where
in-and x (y *and* z) = (*in-and* x y $\vee$ *in-and* x z)
| *in-and* x y = ($x = y$)

fun *in-or*
where
in-or x (y *or* z) = (*in-or* x y $\vee$ *in-or* x z)
| *in-or* x y = ($x = y$)

lemma *in-entailment*:
in-and x $y \Longrightarrow S \models_P y \Longrightarrow S \models_P x$
in-or x $y \Longrightarrow S \models_P x \Longrightarrow S \models_P y$
by (*induction* y) *auto*

definition *mk-and*
where
mk-and f x y = (*case f x of false* $\Rightarrow$ *false* | *true* $\Rightarrow$ *f y*
| $x' \Rightarrow$ (*case f y of false* $\Rightarrow$ *false* | *true* $\Rightarrow x'$
| $y' \Rightarrow$ *if in-and* x' y' *then* y' *else if in-and* y' x' *then* x' *else* x' *and* y'))

definition *mk-and*$'$
where

mk-and′ x y ≡ case y of false ⇒ false | true ⇒ x | - ⇒ x and y

definition *mk-or*
where
mk-or f x y = (case f x of true ⇒ true | false ⇒ f y
| x′ ⇒ (case f y of true ⇒ true | false ⇒ x′
| y′ ⇒ if in-or x′ y′ then y′ else if in-or y′ x′ then x′ else x′ or y′))

definition *mk-or′*
where
mk-or′ x y ≡ case y of true ⇒ true | false ⇒ x | - ⇒ x or y

lemma *mk-and-correct*:
$S \models_P$ *mk-and f x y* $\longleftrightarrow$ $S \models_P$ *f x and f y*
proof −
have *X*: $\bigwedge$*x′ y′*. $S \models_P$ (*if in-and x′ y′ then y′ else if in-and y′ x′ then x′ else x′ and y′*)
$\longleftrightarrow$ $S \models_P$ *x′ and y′*
using *in-entailment* **by** *auto*
show *?thesis*
unfolding *mk-and-def ltl.split X* **by** (*cases f x*; *cases f y*; *simp*)
qed

lemma *mk-and′-correct*:
$S \models_P$ *mk-and′ x y* $\longleftrightarrow$ $S \models_P$ *x and y*
unfolding *mk-and′-def* **by** (*cases y*; *simp*)

lemma *mk-or-correct*:
$S \models_P$ *mk-or f x y* $\longleftrightarrow$ $S \models_P$ *f x or f y*
proof −
have *X*: $\bigwedge$*x′ y′*. $S \models_P$ (*if in-or x′ y′ then y′ else if in-or y′ x′ then x′ else x′ or y′*)
$\longleftrightarrow$ $S \models_P$ *x′ or y′*
using *in-entailment* **by** *auto*
show *?thesis*
unfolding *mk-or-def ltl.split X* **by** (*cases f x*; *cases f y*; *simp*)
qed

lemma *mk-or′-correct*:
$S \models_P$ *mk-or′ x y* $\longleftrightarrow$ $S \models_P$ *x or y*
unfolding *mk-or′-def* **by** (*cases y*; *simp*)

end

17 af - Unfolding Functions - Optimized Code Equations

theory *af-Impl*
 imports *Main ../af LTL-Impl*
begin

Provide optimized code definitions for $\uparrow af$ and other functions, which use heuristics to reduce the formula size

17.1 Helper Function

fun *remove-and-or*
where
 remove-and-or $(z$ *or* $y) = ($*case* z *of*
 $(((z'$ *and* $x')$ *or* $y')$ *and* $x) \Rightarrow$ *if* $x = x' \wedge y = y'$ *then* $((z'$ *and* $x')$ *or* $y')$ *else remove-and-or* z *or remove-and-or* y
 | - $\Rightarrow$ *remove-and-or* z *or remove-and-or* $y)$
| *remove-and-or* $(x$ *and* $y)$ = *remove-and-or* x *and remove-and-or* y
| *remove-and-or* $x = x$

lemma *remove-and-or-correct*:
 $S \models_P$ *remove-and-or* $x \longleftrightarrow S \models_P x$
proof (*induction* x)
 case (*LTLOr* x y)
 thus *?case*
 proof (*induction* x)
 case (*LTLAnd* x' y')
 thus *?case*
 proof (*induction* x')
 case (*LTLOr* x'' y'')
 thus *?case*
 by (*induction* x'') *auto*
 qed *auto*
 qed *auto*
qed *auto*

17.2 Optimized Equations

fun *af-letter-simp*
where
 af-letter-simp true ν = *true*
| *af-letter-simp false* ν = *false*
| *af-letter-simp* $p(a)$ ν = (*if* $a \in \nu$ *then true else false*)

| *af-letter-simp* $(np(a))$ ν = (*if* $a \notin \nu$ *then true else false*)
| *af-letter-simp* (φ *and* ψ) ν = (*case* φ *of*
    true $\Rightarrow$ *af-letter-simp* ψ ν
  | *false* $\Rightarrow$ *false*
  | $p(a)$ $\Rightarrow$ *if* $a \in \nu$ *then af-letter-simp* ψ ν *else false*
  | $np(a)$ $\Rightarrow$ *if* $a \notin \nu$ *then af-letter-simp* ψ ν *else false*
  | $G\ \varphi'$ $\Rightarrow$
    (*let*
      φ'' = *af-letter-simp* φ' ν;
      ψ'' = *af-letter-simp* ψ ν
     in
      (*if* $\varphi'' = \psi''$ *then mk-and'* ($G\ \varphi'$) φ'' *else mk-and id* (*mk-and'* ($G\ \varphi'$) φ'') ψ''))
  | - $\Rightarrow$ *mk-and id* (*af-letter-simp* φ ν) (*af-letter-simp* ψ ν))
| *af-letter-simp* (φ *or* ψ) ν = (*case* φ *of*
    true $\Rightarrow$ *true*
  | *false* $\Rightarrow$ *af-letter-simp* ψ ν
  | $p(a)$ $\Rightarrow$ *if* $a \in \nu$ *then true else af-letter-simp* ψ ν
  | $np(a)$ $\Rightarrow$ *if* $a \notin \nu$ *then true else af-letter-simp* ψ ν
  | $F\ \varphi'$ $\Rightarrow$
    (*let*
      φ'' = *af-letter-simp* φ' ν;
      ψ'' = *af-letter-simp* ψ ν
     in
      (*if* $\varphi'' = \psi''$ *then mk-or'* ($F\ \varphi'$) φ'' *else mk-or id* (*mk-or'* ($F\ \varphi'$) φ'') ψ''))
  | - $\Rightarrow$ *mk-or id* (*af-letter-simp* φ ν) (*af-letter-simp* ψ ν))
| *af-letter-simp* ($X\ \varphi$) ν = φ
| *af-letter-simp* ($G\ \varphi$) ν = *mk-and'* ($G\ \varphi$) (*af-letter-simp* φ ν)
| *af-letter-simp* ($F\ \varphi$) ν = *mk-or'* ($F\ \varphi$) (*af-letter-simp* φ ν)
| *af-letter-simp* ($\varphi\ U\ \psi$) ν = *mk-or'* (*mk-and'* ($\varphi\ U\ \psi$) (*af-letter-simp* φ ν)) (*af-letter-simp* ψ ν)

lemma *af-letter-simp-correct*:
  $S \models_P$ *af-letter* φ ν $\longleftrightarrow$ $S \models_P$ *af-letter-simp* φ ν
proof (*induction* φ)
  case (*LTLAnd* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-and-correct mk-and'-correct*)
next
  case (*LTLOr* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or'-correct*)
qed (*simp-all add*: *mk-and-correct mk-and'-correct mk-or-correct mk-or'-correct*)

fun *af-G-letter-simp*
where
  af-G-letter-simp true ν = *true*
| *af-G-letter-simp false* ν = *false*
| *af-G-letter-simp* $p(a)$ ν = (*if* $a \in \nu$ *then true else false*)
| *af-G-letter-simp* $(np(a))$ ν = (*if* $a \notin \nu$ *then true else false*)
| *af-G-letter-simp* (φ *and* ψ) ν = (*case* φ *of*
      true $\Rightarrow$ *af-G-letter-simp* ψ ν
    | *false* $\Rightarrow$ *false*
    | $p(a)$ $\Rightarrow$ *if* $a \in \nu$ *then af-G-letter-simp* ψ ν *else false*
    | $np(a)$ $\Rightarrow$ *if* $a \notin \nu$ *then af-G-letter-simp* ψ ν *else false*
    | - $\Rightarrow$ *mk-and id* (*af-G-letter-simp* φ ν) (*af-G-letter-simp* ψ ν))
| *af-G-letter-simp* (φ *or* ψ) ν = (*case* φ *of*
      true $\Rightarrow$ *true*
    | *false* $\Rightarrow$ *af-G-letter-simp* ψ ν
    | $p(a)$ $\Rightarrow$ *if* $a \in \nu$ *then true else af-G-letter-simp* ψ ν
    | $np(a)$ $\Rightarrow$ *if* $a \notin \nu$ *then true else af-G-letter-simp* ψ ν
    | F φ' $\Rightarrow$
      (*let*
        φ'' = *af-G-letter-simp* φ' ν;
        ψ'' = *af-G-letter-simp* ψ ν
       in
        (*if* $\varphi'' = \psi''$ *then mk-or'* (F φ') φ'' *else mk-or id* (*mk-or'* (F φ') φ'')
ψ''))
    | - $\Rightarrow$ *mk-or id* (*af-G-letter-simp* φ ν) (*af-G-letter-simp* ψ ν))
| *af-G-letter-simp* (X φ) ν = φ
| *af-G-letter-simp* (G φ) ν = G φ
| *af-G-letter-simp* (F φ) ν = *mk-or'* (F φ) (*af-G-letter-simp* φ ν)
| *af-G-letter-simp* (φ U ψ) ν = *mk-or'* (*mk-and'* (φ U ψ) (*af-G-letter-simp*
φ ν)) (*af-G-letter-simp* ψ ν)

lemma *af-G-letter-simp-correct*:
  $S \models_P$ *af-G-letter* φ ν $\longleftrightarrow$ $S \models_P$ *af-G-letter-simp* φ ν
proof (*induction* φ)
  case (*LTLAnd* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *mk-and-correct*)
next
  case (*LTLOr* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or'-correct*)
qed (*simp-all add*: *mk-and-correct mk-and'-correct mk-or-correct mk-or'-correct*)

fun *step-simp*
where
$\quad$ *step-simp* $p(a)\ \nu = (\textit{if } a \in \nu \textit{ then true else false})$
$|$ *step-simp* $(np(a))\ \nu\ = (\textit{if } a \notin \nu \textit{ then true else false})$
$|$ *step-simp* $(\varphi \textit{ and } \psi)\ \nu = (\textit{mk-and id } (\textit{step-simp } \varphi\ \nu)\ (\textit{step-simp } \psi\ \nu))$
$|$ *step-simp* $(\varphi \textit{ or } \psi)\ \nu = (\textit{mk-or id } (\textit{step-simp } \varphi\ \nu)\ (\textit{step-simp } \psi\ \nu))$
$|$ *step-simp* $(X\ \varphi)\ \nu = \textit{remove-constants}_P\ \varphi$
$|$ *step-simp* $\varphi\ \nu = \varphi$

lemma *step-simp-correct*:
$\quad S \models_P \textit{step } \varphi\ \nu \longleftrightarrow S \models_P \textit{step-simp } \varphi\ \nu$
proof (*induction* φ)
$\quad$**case** (*LTLAnd* $\varphi\ \psi$)
$\quad\quad$**thus** *?case*
$\quad\quad\quad$**by** (*cases* φ) (*auto simp add*: *Let-def mk-and-correct mk-and′-correct*)
next
$\quad$**case** (*LTLOr* $\varphi\ \psi$)
$\quad\quad$**thus** *?case*
$\quad\quad\quad$**by** (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or′-correct*)
qed (*simp-all add*: *mk-and-correct mk-and′-correct mk-or-correct mk-or′-correct remove-constants-correct*[*symmetric*])

fun *Unf-simp*
where
$\quad$ *Unf-simp* $(\varphi \textit{ and } \psi) = (\textit{case } \varphi \textit{ of}$
$\quad\quad$ $\textit{true} \Rightarrow \textit{Unf-simp } \psi$
$\quad$ $|\ \textit{false} \Rightarrow \textit{false}$
$\quad$ $|\ G\ \varphi' \Rightarrow$
$\quad\quad$ $(\textit{let}$
$\quad\quad\quad$ $\varphi'' = \textit{Unf-simp } \varphi';\ \psi'' = \textit{Unf-simp } \psi$
$\quad\quad$ $\textit{in}$
$\quad\quad$ $(\textit{if } \varphi'' = \psi'' \textit{ then mk-and}'\ (G\ \varphi')\ \varphi'' \textit{ else mk-and id } (\textit{mk-and}'\ (G\ \varphi')\ \varphi'')\ \psi''))$
$\quad$ $|\ \text{-} \Rightarrow \textit{mk-and id } (\textit{Unf-simp } \varphi)\ (\textit{Unf-simp } \psi))$
$|$ *Unf-simp* $(\varphi \textit{ or } \psi) = (\textit{case } \varphi \textit{ of}$
$\quad\quad$ $\textit{true} \Rightarrow \textit{true}$
$\quad$ $|\ \textit{false} \Rightarrow \textit{Unf-simp } \psi$
$\quad$ $|\ F\ \varphi' \Rightarrow$
$\quad\quad$ $(\textit{let}$
$\quad\quad\quad$ $\varphi'' = \textit{Unf-simp } \varphi';\ \psi'' = \textit{Unf-simp } \psi$
$\quad\quad$ $\textit{in}$
$\quad\quad$ $(\textit{if } \varphi'' = \psi'' \textit{ then mk-or}'\ (F\ \varphi')\ \varphi'' \textit{ else mk-or id } (\textit{mk-or}'\ (F\ \varphi')\ \varphi'')\ \psi''))$
$\quad$ $|\ \text{-} \Rightarrow \textit{mk-or id } (\textit{Unf-simp } \varphi)\ (\textit{Unf-simp } \psi))$

| *Unf-simp* (G φ) = *mk-and*$'$ (G φ) (*Unf-simp* φ)
| *Unf-simp* (F φ) = *mk-or*$'$ (F φ) (*Unf-simp* φ)
| *Unf-simp* (φ U ψ) = *mk-or*$'$ (*mk-and*$'$ (φ U ψ) (*Unf-simp* φ)) (*Unf-simp* ψ)
| *Unf-simp* φ = φ

lemma *Unf-simp-correct*:
$S \models_P$ *Unf* $\varphi \longleftrightarrow S \models_P$ *Unf-simp* φ
proof (*induction* φ)
case (*LTLAnd* φ ψ)
thus *?case*
by (*cases* φ) (*auto simp add*: *Let-def mk-and-correct mk-and*$'$*-correct*)
next
case (*LTLOr* φ ψ)
thus *?case*
by (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or*$'$*-correct*)
qed (*simp-all add*: *mk-and-correct mk-and*$'$*-correct mk-or-correct mk-or*$'$*-correct*)

fun Unf_G*-simp*
where
Unf_G*-simp* (φ *and* ψ) = *mk-and id* (Unf_G*-simp* φ) (Unf_G*-simp* ψ)
| Unf_G*-simp* (φ *or* ψ) = (*case* φ *of*
true $\Rightarrow$ *true*
| *false* $\Rightarrow$ Unf_G*-simp* ψ
| F φ' $\Rightarrow$
(*let*
φ'' = Unf_G*-simp* φ'; ψ'' = Unf_G*-simp* ψ
in
(*if* $\varphi'' = \psi''$ *then mk-or*$'$ (F φ') φ'' *else mk-or id* (*mk-or*$'$ (F φ') φ'') ψ''))
| - $\Rightarrow$ *mk-or id* (Unf_G*-simp* φ) (Unf_G*-simp* ψ))
| Unf_G*-simp* (F φ) = *mk-or*$'$ (F φ) (Unf_G*-simp* φ)
| Unf_G*-simp* (φ U ψ) = *mk-or*$'$ (*mk-and*$'$ (φ U ψ) (Unf_G*-simp* φ)) (Unf_G*-simp* ψ)
| Unf_G*-simp* φ = φ

lemma Unf_G*-simp-correct*:
$S \models_P Unf_G$ $\varphi \longleftrightarrow S \models_P$ Unf_G*-simp* φ
proof (*induction* φ)
case (*LTLAnd* φ ψ)
thus *?case*
by (*cases* φ) (*auto simp add*: *Let-def mk-and-correct mk-and*$'$*-correct*)
next
case (*LTLOr* φ ψ)

thus *?case*
by (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or'-correct*)
qed (*simp-all add*: *mk-and-correct mk-and'-correct mk-or-correct mk-or'-correct*)

fun *af-letter-opt-simp*
where
af-letter-opt-simp true ν = *true*
| *af-letter-opt-simp false* ν = *false*
| *af-letter-opt-simp* $p(a)$ ν = (*if* $a \in \nu$ *then true else false*)
| *af-letter-opt-simp* ($np(a)$) ν = (*if* $a \notin \nu$ *then true else false*)
| *af-letter-opt-simp* (φ *and* ψ) ν = (*case* φ *of*
true $\Rightarrow$ *af-letter-opt-simp* ψ ν
| *false* $\Rightarrow$ *false*
| $p(a)$ $\Rightarrow$ *if* $a \in \nu$ *then af-letter-opt-simp* ψ ν *else false*
| $np(a)$ $\Rightarrow$ *if* $a \notin \nu$ *then af-letter-opt-simp* ψ ν *else false*
| G φ' $\Rightarrow$
(*let*
φ'' = *Unf-simp* φ';
ψ'' = *af-letter-opt-simp* ψ ν
in
(*if* $\varphi'' = \psi''$ *then mk-and'* (G φ') φ'' *else mk-and id* (*mk-and'* (G φ') φ'') ψ''))
| - $\Rightarrow$ *mk-and id* (*af-letter-opt-simp* φ ν) (*af-letter-opt-simp* ψ ν))
| *af-letter-opt-simp* (φ *or* ψ) ν = (*case* φ *of*
true $\Rightarrow$ *true*
| *false* $\Rightarrow$ *af-letter-opt-simp* ψ ν
| $p(a)$ $\Rightarrow$ *if* $a \in \nu$ *then true else af-letter-opt-simp* ψ ν
| $np(a)$ $\Rightarrow$ *if* $a \notin \nu$ *then true else af-letter-opt-simp* ψ ν
| F φ' $\Rightarrow$
(*let*
φ'' = *Unf-simp* φ';
ψ'' = *af-letter-opt-simp* ψ ν
in
(*if* $\varphi'' = \psi''$ *then mk-or'* (F φ') φ'' *else mk-or id* (*mk-or'* (F φ') φ'') ψ''))
| - $\Rightarrow$ *mk-or id* (*af-letter-opt-simp* φ ν) (*af-letter-opt-simp* ψ ν))
| *af-letter-opt-simp* (X φ) ν = *Unf-simp* φ
| *af-letter-opt-simp* (G φ) ν = *mk-and'* (G φ) (*Unf-simp* φ)
| *af-letter-opt-simp* (F φ) ν = *mk-or'* (F φ) (*Unf-simp* φ)
| *af-letter-opt-simp* (φ U ψ) ν = *mk-or'* (*mk-and'* (φ U ψ) (*Unf-simp* φ)) (*Unf-simp* ψ)

lemma *af-letter-opt-simp-correct*:
$S \models_P$ *af-letter-opt* φ ν $\longleftrightarrow$ $S \models_P$ *af-letter-opt-simp* φ ν

proof (*induction* φ)
  case (*LTLAnd* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-and-correct mk-and′-correct*)
next
  case (*LTLOr* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or′-correct*)
qed (*simp-all add*: *mk-and-correct mk-and′-correct mk-or-correct mk-or′-correct Unf-simp-correct*)

fun *af-G-letter-opt-simp*
where
  af-G-letter-opt-simp true ν = *true*
| *af-G-letter-opt-simp false* ν = *false*
| *af-G-letter-opt-simp* $p(a)$ ν = (*if* $a \in \nu$ *then true else false*)
| *af-G-letter-opt-simp* ($np(a)$) ν = (*if* $a \notin \nu$ *then true else false*)
| *af-G-letter-opt-simp* (φ *and* ψ) ν = (*case* φ *of*
      true ⇒ *af-G-letter-opt-simp* ψ ν
    | *false* ⇒ *false*
    | $p(a)$ ⇒ *if* $a \in \nu$ *then af-G-letter-opt-simp* ψ ν *else false*
    | $np(a)$ ⇒ *if* $a \notin \nu$ *then af-G-letter-opt-simp* ψ ν *else false*
    | - ⇒ *mk-and id* (*af-G-letter-opt-simp* φ ν) (*af-G-letter-opt-simp* ψ ν))
| *af-G-letter-opt-simp* (φ *or* ψ) ν = (*case* φ *of*
      true ⇒ *true*
    | *false* ⇒ *af-G-letter-opt-simp* ψ ν
    | $p(a)$ ⇒ *if* $a \in \nu$ *then true else af-G-letter-opt-simp* ψ ν
    | $np(a)$ ⇒ *if* $a \notin \nu$ *then true else af-G-letter-opt-simp* ψ ν
    | *F* φ' ⇒
      (*let*
        φ'' = Unf_G*-simp* φ';
        ψ'' = *af-G-letter-opt-simp* ψ ν
       in
        (*if* $\varphi'' = \psi''$ *then mk-or′* (*F* φ') φ'' *else mk-or id* (*mk-or′* (*F* φ') φ'') ψ''))
    | - ⇒ *mk-or id* (*af-G-letter-opt-simp* φ ν) (*af-G-letter-opt-simp* ψ ν))
| *af-G-letter-opt-simp* (*X* φ) ν = Unf_G*-simp* φ
| *af-G-letter-opt-simp* (*G* φ) ν = *G* φ
| *af-G-letter-opt-simp* (*F* φ) ν = *mk-or′* (*F* φ) (Unf_G*-simp* φ)
| *af-G-letter-opt-simp* (φ *U* ψ) ν = *mk-or′* (*mk-and′* (φ *U* ψ) (Unf_G*-simp* φ)) (Unf_G*-simp* ψ)

lemma *af-G-letter-opt-simp-correct*:
  $S \models_P$ *af-G-letter-opt* φ ν $\longleftrightarrow$ $S \models_P$ *af-G-letter-opt-simp* φ ν

proof (*induction* φ)
  case (*LTLAnd* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-and-correct mk-and′-correct*)
next
  case (*LTLOr* φ ψ)
    thus *?case*
      by (*cases* φ) (*auto simp add*: *Let-def mk-or-correct mk-or′-correct*)
qed (*simp-all add*: *mk-and-correct mk-and′-correct mk-or-correct mk-or′-correct* Unf_G*-simp-correct*)

17.3 Register Code Equations

lemma [*code*]:
  ↑*af* (*Abs* φ) ν = *Abs* (*remove-and-or* (*af-letter-simp* φ ν))
  unfolding *af-abs.f-abs.abs-eq af-letter-abs-def ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*
  using *af-letter-simp-correct remove-and-or-correct* **by** *blast*

lemma [*code*]:
  ↑af_G (*Abs* φ) ν = *Abs* (*remove-and-or* (*af-G-letter-simp* φ ν))
  unfolding *af-G-abs.f-abs.abs-eq af-G-letter-abs-def ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*
  using *af-G-letter-simp-correct remove-and-or-correct* **by** *blast*

lemma [*code*]:
  ↑*step* (*Abs* φ) ν = *Abs* (*step-simp* φ ν)
  unfolding *step-abs.abs-eq ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*
  using *step-simp-correct* **by** *blast*

lemma [*code*]:
  ↑*Unf* (*Abs* φ) = *Abs* (*remove-and-or* (*Unf-simp* φ))
  unfolding *Unf-abs.abs-eq ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*
  using *Unf-simp-correct remove-and-or-correct* **by** *blast*

lemma [*code*]:
  ↑Unf_G (*Abs* φ) = *Abs* (*remove-and-or* (Unf_G*-simp* φ))
  unfolding Unf_G*-abs.abs-eq ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*
  using Unf_G*-simp-correct remove-and-or-correct* **by** *blast*

lemma [*code*]:
  ↑$af_{\mathfrak{U}}$ (*Abs* φ) ν = *Abs* (*remove-and-or* (*af-letter-opt-simp* φ ν))
  unfolding *af-abs-opt.f-abs.abs-eq af-letter-abs-opt-def ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*

using *af-letter-opt-simp-correct remove-and-or-correct* **by** *blast*

lemma [*code*]:
$\uparrow af_{G\mathfrak{U}}$ (*Abs* φ) ν = *Abs* (*remove-and-or* (*af-G-letter-opt-simp* φ ν))
unfolding *af-G-abs-opt.f-abs.abs-eq af-G-letter-abs-opt-def ltl-prop-equiv-quotient.abs-eq-iff ltl-prop-equiv-def*
using *af-G-letter-opt-simp-correct remove-and-or-correct* **by** *blast*

end

18 Executable Translation from Mojmir to Rabin Automata

theory *Mojmir-Rabin-Impl*
imports *Main ../Mojmir-Rabin*
begin

— Ranking functions are stored as lists sorted ascending by the state rank

fun *init* :: *'a* ⇒ *'a list*
where
init q_0 = [q_0]

fun *nxt* :: *'b set* ⇒ (*'a*, *'b*) *DTS* ⇒ *'a* ⇒ (*'a list*, *'b*) *DTS*
where
nxt Σ δ q_0 = (λ*qs* ν. *remdups-fwd* ((*filter* (λ*q*. ¬*semi-mojmir-def.sink* Σ δ q_0 *q*) (*map* (λ*q*. δ *q* ν) *qs*)) @ [q_0]))

— Recompute the rank from the list

fun *rk* :: *'a list* ⇒ *'a* ⇒ *nat option*
where
rk qs q = (*let i* = *index qs q in if i* ≠ *length qs then Some i else None*)

— Instead of computing the whole sets for fail, merge, and succeed, we define filters (a.k.a. characteristic functions)

fun *fail-filt* :: *'b set* ⇒ (*'a*, *'b*) *DTS* ⇒ *'a* ⇒ (*'a* ⇒ *bool*) ⇒ (*'a list*, *'b*) *transition* ⇒ *bool*
where
fail-filt Σ δ q_0 *F* (*r*, ν, -) = (∃ *q* ∈ *set r*. *let q'* = δ *q* ν *in* (¬*F q'*) ∧ *semi-mojmir-def.sink* Σ δ q_0 *q'*)

fun *merge-filt* :: ('a, 'b) *DTS* ⇒ 'a ⇒ ('a ⇒ *bool*) ⇒ *nat* ⇒ ('a *list*, 'b) *transition* ⇒ *bool*
where
merge-filt δ q_0 *F i* (*r*, ν, -) = ($\exists$ *q* ∈ *set r*. *let* q' = δ *q* ν *in the* (*rk r q*) < *i* ∧ ¬*F* q' ∧ (($\exists$ q'' ∈ *set r*. q'' ≠ *q* ∧ δ q'' ν = q') ∨ q' = q_0))

fun *succeed-filt* :: ('a, 'b) *DTS* ⇒ 'a ⇒ ('a ⇒ *bool*) ⇒ *nat* ⇒ ('a *list*, 'b) *transition* ⇒ *bool*
where
succeed-filt δ q_0 *F i* (*r*, ν, -) = ($\exists$ *q* ∈ *set r*. *let* q' = δ *q* ν *in rk r q* = *Some i* ∧ (¬*F q* ∨ *q* = q_0) ∧ *F* q')

18.0.1 nxt Properties

lemma *nxt-run-distinct*:
distinct (*run* (*nxt* Σ Δ q_0) (*init* q_0) *w n*)
by (*cases n*; *simp del*: *remdups-fwd.simps*; *metis* (*no-types*) *remdups-fwd-distinct*)

lemma *nxt-run-reverse-step*:
fixes Σ δ q_0 *w*
defines *r* ≡ *run* (*nxt* Σ δ q_0) (*init* q_0) *w*
assumes *q* ∈ *set* (*r* (*Suc n*))
assumes *q* ≠ q_0
shows $\exists$ q' ∈ *set* (*r n*). δ q' (*w n*) = *q*
using *assms*(*2*−*3*) **unfolding** *r-def run.simps nxt.simps remdups-fwd-set*
by *auto*

lemma *nxt-run-sink-free*:
q ∈ *set* (*run* (*nxt* Σ δ q_0) (*init* q_0) *w n*) ⟹ ¬*semi-mojmir-def.sink* Σ δ q_0 *q*
by (*induction n*) (*simp-all add*: *semi-mojmir-def.sink-def del*: *remdups-fwd.simps*, *blast*)

18.0.2 rk Properties

lemma *rk-bounded*:
rk xs x = *Some i* ⟹ *i* < *length xs*
by (*simp add*: *Let-def*) (*metis index-conv-size-if-notin index-less-size-conv option.distinct*(*1*) *option.inject*)

lemma *rk-facts*:
x ∈ *set xs* ⟷ *rk xs x* ≠ *None*
x ∈ *set xs* ⟷ ($\exists$ *i*. *rk xs x* = *Some i*)
using *rk-bounded* **by** (*simp add*: *index-size-conv*)+

lemma *rk-split*:
  $y \notin set\ xs \Longrightarrow rk\ (xs\ @\ y\ \#\ zs)\ y = Some\ (length\ xs)$
  by (*induction xs*) *auto*

lemma *rk-split-card*:
  $y \notin set\ xs \Longrightarrow distinct\ xs \Longrightarrow rk\ (xs\ @\ y\ \#\ zs)\ y = Some\ (card\ (set\ xs))$
  using *rk-split* **by** (*metis length-remdups-card-conv remdups-id-iff-distinct*)

lemma *rk-split-card-takeWhile*:
  assumes $x \in set\ xs$
  assumes *distinct xs*
  shows $rk\ xs\ x = Some\ (card\ (set\ (takeWhile\ (\lambda y.\ y \neq x)\ xs)))$
proof –
  obtain *ys zs* **where** $xs = ys\ @\ x\ \#\ zs$ **and** $x \notin set\ ys$
    using *assms* **by** (*blast dest*: *split-list-first*)
  moreover
  hence *distinct ys* **and** $ys = takeWhile\ (\lambda y.\ y \neq x)\ xs$
    using *takeWhile-foo assms* **by** (*simp*, *fast*)
  ultimately
  show *?thesis*
    using *rk-split-card* **by** *metis*
qed

lemma *take-rk*:
  assumes *distinct xs*
  shows $set\ (take\ i\ xs) = \{q.\ \exists j < i.\ rk\ xs\ q = Some\ j\}$
  (**is** *?rhs* = *?lhs*)
  using *assms*
proof (*induction i arbitrary*: *xs*)
  case (*Suc i*)
    thus *?case*
    proof (*induction xs*)
      case (*Cons x xs*)
        have $set\ (take\ (Suc\ i)\ (x\ \#\ xs)) = \{x\} \cup set\ (take\ i\ xs)$
          by *simp*
        also
        have $\ldots = \{x\} \cup \{q.\ \exists j < i.\ rk\ xs\ q = Some\ j\}$
          using *Cons* **by** *simp*
        finally
        show *?case*
          by *force*
    qed *simp*
qed *simp*

lemma *drop-rk*:
assumes *distinct xs*
shows *set* (*drop i xs*) = {*q*. ∃ *j* ≥ *i*. *rk xs q* = *Some j*}
proof −
have *set xs* = {*q*. ∃ *j*. *rk xs q* = *Some j*} (**is** - = *?U*)
using *rk-facts*(*2*)[*of* - *xs*] **by** *blast*
moreover
have *?U* = {*q*. ∃ *j* ≥ *i*. *rk xs q* = *Some j*} ∪ {*q*. ∃ *j* < *i*. *rk xs q* = *Some j*}
and {} = {*q*. ∃ *j* ≥ *i*. *rk xs q* = *Some j*} ∩ {*q*. ∃ *j* < *i*. *rk xs q* = *Some j*}
by *auto*
moreover
have *set xs* = *set* (*drop i xs*) ∪ *set* (*take i xs*)
and {} = *set* (*drop i xs*) ∩ *set* (*take i xs*)
by (*metis assms append-take-drop-id inf-sup-aci*(*1*,*5*) *distinct-append set-append*)+
ultimately
show *?thesis*
using *take-rk*[*OF assms*] **by** *blast*
qed

18.0.3 Relation to (Semi) Mojmir Automata

lemma (**in** *semi-mojmir*) *nxt-run-configuration*:
defines $r \equiv run\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)\ w$
shows *q* ∈ *set* (*r n*) ⟷ ¬*sink q* ∧ *configuration q n* ≠ {}
proof (*induction n arbitrary*: *q*)
case (*Suc n*)
thus *?case*
proof (*cases* $q \neq q_0$)
case *True*
{
assume *q* ∈ *set* (*r* (*Suc n*))
hence ¬ *sink q*
using *r-def nxt-run-sink-free* **by** *metis*
moreover
obtain *q′* **where** *q′* ∈ *set* (*r n*) **and** δ *q′* (*w n*) = *q*
using ‹*q* ∈ *set* (*r* (*Suc n*))› *nxt-run-reverse-step*[*OF* - ‹$q \neq q_0$›] **unfolding** *r-def* **by** *blast*
hence *configuration q* (*Suc n*) ≠ {} **and** ¬ *sink q*
unfolding *configuration-step-eq*[*OF True*] *Suc* **using** *True* ‹¬ *sink q*› **by** *auto*

}
moreover
{
assume $\neg$*sink q* **and** *configuration q* (*Suc n*) $\neq$ {}
then obtain q' **where** *configuration* q' $n \neq$ {} **and** δ q' (w n) = q
unfolding *configuration-step-eq*[*OF True*] **by** *blast*
moreover
hence $\neg$*sink* q'
using ‹$\neg$*sink q*› *sink-rev-step assms* **by** *blast*
ultimately
have $q' \in$ *set* (*r n*)
unfolding *Suc* **by** *blast*
hence $q \in$ *set* (*r* (*Suc n*))
using ‹δ q' (w n) = q› ‹$\neg$*sink q*›
unfolding *r-def run.simps set-filter comp-def remdups-fwd-set set-map set-append image-def*
unfolding *r-def*[*symmetric*] **by** *auto*
}
ultimately
show *?thesis*
by *blast*
qed (*insert assms*, *auto simp add*: *r-def sink-def*)
qed (*insert assms*, *auto simp add*: *r-def sink-def*)

lemma (**in** *semi-mojmir*) *nxt-run-sorted*:
defines $r \equiv$ *run* (*nxt* Σ δ q_0) (*init* q_0) *w*
shows *sorted* (*map* (λq. *the* (*oldest-token q n*)) (*r n*))
proof (*induction n*)
case (*Suc n*)
let *?f-n* = λq. *the* (*oldest-token q n*)
let *?f-Suc-n* = λq. *the* (*oldest-token q* (*Suc n*))
let *?step* = *filter* (λq. $\neg$*sink q*) ((*map* (λq. δ *q* (*w n*)) (*r n*)) @ [q_0])

have $\bigwedge$*q p qs ps*. *remdups-fwd ?step* = *qs* @ *q* # *p* # *ps* $\Longrightarrow$ *?f-Suc-n q* $\leq$ *?f-Suc-n p*
proof –
fix *q qs p ps*
assume *remdups-fwd ?step* = *qs* @ *q* # *p* # *ps*
then obtain *zs* zs' zs'' **where** *step-def*: *?step* = *zs* @ *q* # zs' @ *p* # zs''
and *remdups-fwd zs* = *qs*
and *remdups-fwd-acc* (*set qs* $\cup$ {*q*}) zs' = []
and *remdups-fwd-acc* (*set qs* $\cup$ {*q*, *p*}) zs'' = *ps*
and $q \notin$ *set zs*

and $p \notin$ *set zs* $\cup \{q\}$
unfolding *remdups-fwd.simps remdups-fwd-split-exact-iff remdups-fwd-split-exact-iff*[**where** *?ys* = [], *simplified*] *insert-commute*
by *auto*
hence $p \notin$ *set zs* $\cup$ *set zs'* $\cup \{q\}$
and $q \neq p$ **unfolding** *remdups-fwd-acc-empty*[*symmetric*] **by** *auto*
hence $p \notin$ *set zs* $\cup$ *set zs'* $\cup$ *set* [q]
by *simp*
hence $\{q, p\} \subseteq$ *set ?step*
using *step-def* **by** *simp*
hence $\neg$ *sink q* **and** $\neg$ *sink p*
unfolding *set-map set-filter* **by** *blast+*

show *?f-Suc-n q* $\leq$ *?f-Suc-n p*
proof (*cases zs''* = [])
case *True*
hence $p = q_0$ **and** *q-def*: *filter* ($\lambda q.$ $\neg$*sink q*) (*map* ($\lambda q.$ δ *q* (*w n*)) (*r n*)) = *zs* @ [*q*] @ *zs'*
using *step-def* **unfolding** *sink-def* **by** *simp+*
hence $q_0 \notin$ *set* (*filter* ($\lambda q.$ $\neg$*sink q*) (*map* ($\lambda q.$ δ *q* (*w n*)) (*r n*)))
using ‹$p \notin$ *set zs* $\cup$ *set zs'* $\cup \{q\}$› **unfolding** ‹$p = q_0$› *sink-def* **by** *simp*
hence $q_0 \notin$ ($\lambda q.$ δ *q* (*w n*)) ‘ {*q'*. *configuration q' n* $\neq$ {}}
using *nxt-run-configuration bounded-w* **unfolding** *set-map set-filter r-def sink-def init.simps* **by** *blast*
hence *configuration p* (*Suc n*) = {*Suc n*} **using** *assms*
unfolding ‹$p = q_0$› **using** *configuration-step-eq-*q_0 **by** *blast*
hence *?f-Suc-n p* = *Suc n*
using *assms* **by** *force*
moreover
have $q \in$ ($\lambda q.$ δ *q* (*w n*)) ‘ *set* (*r n*)
using ‹$p \notin$ *set zs* $\cup$ *set zs'* $\cup \{q\}$› *image-set* **unfolding** *filter-map-split-iff*[*of* ($\lambda q.$ $\neg$ *sink q*) $\lambda q.$ δ *q* (*w n*)]
by (*metis* (*no-types*, *lifting*) *Un-insert-right* ‹$p = q_0$› ‹$\{q, p\} \subseteq$ *set* [$q \leftarrow$*map* ($\lambda q.$ δ *q* (*w n*)) (*r n*) @ [q_0] . $\neg$ *sink q*]› *append-Nil2 insert-iff insert-subset list.simps*(*15*) *mem-Collect-eq set-append set-filter*)
hence $q \in$ ($\lambda q.$ δ *q* (*w n*)) ‘ {*q'*. *configuration q' n* $\neq$ {}}
using *nxt-run-configuration* **unfolding** *r-def* **by** *auto*
hence *configuration q* (*Suc n*) $\neq$ {}
using *configuration-step assms* **by** *blast*
hence *?f-Suc-n q* $\leq$ *Suc n*
using *assms oldest-token-bounded*[*of q Suc n*]
by (*simp del*: *configuration.simps*)
ultimately

show *?f-Suc-n q* ≤ *?f-Suc-n p*
by *presburger*
next
case *False*
hence *X*: *filter* (λ*q*. ¬*sink q*) (*map* (λ*q*. δ *q* (*w n*)) (*r n*)) = *zs* @ [*q*] @ *zs'* @ [*p*] @ *butlast zs''*
using *step-def* **unfolding** *map-append filter-append sink-def* **apply** *simp*
by (*metis* (*no-types*, *lifting*) *butlast.simps*(*2*) *list.distinct*(*1*) *butlast-append append-is-Nil-conv butlast-snoc*)
obtain *qs' sq' sp' ps' ps''* **where** *r-def'*: *r n* = *qs'* @ *sq'* @ *ps'* @ *sp'* @ *ps''*
and *1*: *filter* (λ*q*. ¬*sink q*) (*map* (λ*q*. δ *q* (*w n*)) *qs'*) = *zs*
and *2*: *filter* (λ*q*. ¬*sink q*) (*map* (λ*q*. δ *q* (*w n*)) *sq'*) = [*q*]
and *3*: *filter* (λ*q*. ¬*sink q*) (*map* (λ*q*. δ *q* (*w n*)) *ps'*) = *zs'*
and *filter* (λ*q*. ¬*sink q*) (*map* (λ*q*. δ *q* (*w n*)) *sp'*) = [*p*]
and *filter* (λ*q*. ¬*sink q*) (*map* (λ*q*. δ *q* (*w n*)) *ps''*) = *butlast zs''*
using *X* **unfolding** *filter-map-split-iff* **by** (*blast*)
hence *21*: *Set.filter* (λ*q*. ¬*sink q*) ((λ*q*. δ *q* (*w n*)) ‘ *set sq'*) = {*q*}
and *41*: *Set.filter* (λ*q*. ¬*sink q*) ((λ*q*. δ *q* (*w n*)) ‘ *set sp'*) = {*p*}
by (*metis filter-set image-set list.set*(*1*) *list.simps*(*15*))+
from *21* **obtain** *q'* **where** *q'* ∈ *set sq'* **and** ¬ *sink q'* **and** *q* = δ *q'* (*w n*)
using *sink-rev-step*(*2*)[*OF* ‹¬ *sink q*›, *of* - *n*] **by** *fastforce*
from *41* **obtain** *p'* **where** *p'* ∈ *set sp'* **and** ¬ *sink p'* **and** *p* = δ *p'* (*w n*)
using *sink-rev-step*(*2*)[*OF* ‹¬ *sink p*›, *of* - *n*] **by** *fastforce*
from *Suc* **have** *?f-n q'* ≤ *?f-n p'*
unfolding *r-def' map-append sorted-append set-append set-map* **using** ‹*q'* ∈ *set sq'*› ‹*p'* ∈ *set sp'*› **by** *auto*
moreover
{
have *oldest-token q' n* ≠ *None*
using *nxt-run-configuration option.distinct*(*1*) *r-def r-def'* ‹*q'* ∈ *set sq'*› ‹*p'* ∈ *set sp'*› *set-append*
unfolding *init.simps oldest-token.simps* **by** (*metis UnCI*)
moreover
hence *oldest-token q* (*Suc n*) ≠ *None*
using ‹*q* = δ *q'* (*w n*)› **by** (*metis oldest-token.simps option.distinct*(*1*) *configuration-step-non-empty*)
ultimately
obtain *x y* **where** *x-def*: *oldest-token q' n* = *Some x*
and *y-def*: *oldest-token q* (*Suc n*) = *Some y*
by *blast*

moreover
hence $x \leq n$ **and** *token-run x n* $= q'$
using *oldest-token-bounded push-down-oldest-token-token-run assms* **by** *blast+*
moreover
hence *token-run x* (*Suc n*) $= q$
using ‹$q = \delta\ q'\ (w\ n)$› **by** (*rule token-run-step*)
ultimately
have $x \geq y$
using *oldest-token-monotonic-Suc assms* **by** *blast*
moreover
{
have $\bigwedge q''.\ q = \delta\ q''\ (w\ n) \Longrightarrow q'' \notin$ *set qs'*
using ‹$q \notin$ *set zs*› **unfolding** ‹*filter* ($\lambda q.\ \neg$*sink q*) (*map* ($\lambda q.\ \delta\ q\ (w\ n)$) *qs'*) = *zs*›[*symmetric*] *set-map set-filter* **apply** *simp* **using** ‹¬ *sink q*› **by** *blast*
moreover
{
obtain *us vs* **where** *1*: *map* ($\lambda q.\ \delta\ q\ (w\ n)$) *sq'* = *us* @ [*q*] @ *vs* **and** ∀ *u*∈*set us. sink u* **and** [] = [*q*←*vs* . ¬ *sink q*]
using ‹*filter* ($\lambda q.\ \neg$*sink q*) (*map* ($\lambda q.\ \delta\ q\ (w\ n)$) *sq'*) = [*q*]› **unfolding** *filter-eq-Cons-iff* **by** *auto*
moreover
hence $\bigwedge q''.\ q'' \in$ (*set us*) ∪ (*set vs*) $\Longrightarrow$ *sink* q''
by (*metis UnE filter-empty-conv*)
hence $q \notin$ (*set us*) ∪ (*set vs*)
using ‹¬ *sink q*› **by** *blast*
ultimately
have $\bigwedge q''.\ q'' \in$ (*set sq'* − {*q'*}) $\Longrightarrow \delta\ q''\ (w\ n) \neq q$
using *1* ‹$q = \delta\ q'\ (w\ n)$› ‹$q' \in$ *set sq'*› **by** (*fastforce dest*: *split-list elim*: *map-splitE*)
}
ultimately
have $\bigwedge q''.\ q = \delta\ q''\ (w\ n) \Longrightarrow$ *configuration* q'' *n* $\neq$ {} $\Longrightarrow q'' \in$ *set* (*ps'* @ *sp'* @ *ps''*) ∨ $q'' = q'$
using *nxt-run-configuration*[*of - n*] ‹¬ *sink q*› *sink-rev-step*
unfolding *r-def'*[*unfolded r-def*] *set-append*
by *blast*
moreover
have $\bigwedge q''.\ q'' \in$ *set* (*ps'* @ *sp'* @ *ps''*) $\Longrightarrow x \leq$ *?f-n* q''
using *x-def* **using** *Suc* **unfolding** *r-def' map-append sorted-append set-append set-map* **using** ‹$q' \in$ *set sq'*› ‹$p' \in$ *set sp'*›
apply (*simp del*: *oldest-token.simps*) **by** *fastforce*
moreover

have $\bigwedge q''.\ q'' = q' \Longrightarrow x \leq$ *?f-n* q''
using *x-def* **by** *simp*
moreover
have $\bigwedge q''\ x.\ x \in$ *configuration* $q''\ n \Longrightarrow$ *the* (*oldest-token* $q''\ n$) $\leq x$
using *assms* **by** *auto*
ultimately
have $\bigwedge z.\ z \in \bigcup\{$*configuration* $q'\ n\ |q'.\ q = \delta\ q'\ (w\ n)\} \Longrightarrow x \leq z$
by *fastforce*
}
hence $\bigwedge z.\ z \in$ *configuration* q (*Suc* n) $\Longrightarrow x \leq z$
unfolding *configuration-step-eq-unified* **using** ‹$x \leq n$›
by (*cases* $q = q_0$; *auto*)
hence $x \leq y$
using *y-def Min.boundedI configuration-finite* **using** *push-down-oldest-token-configuration* **by** *presburger*
ultimately
have *?f-n* q' = *?f-Suc-n* q
using *x-def y-def* **by** *fastforce*
}
moreover
{
have *oldest-token* $p'\ n \neq$ *None*
using *nxt-run-configuration oldest-token.simps option.distinct*(*1*) *r-def r-def'* ‹$q' \in$ *set sq'*› ‹$p' \in$ *set sp'*› *set-append*
unfolding *init.simps* **by** (*metis UnCI*)
moreover
hence *oldest-token* p (*Suc* n) $\neq$ *None*
using ‹$p = \delta\ p'\ (w\ n)$› **by** (*metis oldest-token.simps option.distinct*(*1*) *configuration-step-non-empty*)
ultimately
obtain $x\ y$ **where** *x-def*: *oldest-token* $p'\ n$ = *Some* x
and *y-def*: *oldest-token* p (*Suc* n) = *Some* y
by *blast*
moreover
hence $x \leq n$ **and** *token-run* $x\ n = p'$
using *oldest-token-bounded push-down-oldest-token-token-run assms* **by** *blast+*
moreover
hence *token-run* x (*Suc* n) = p
using ‹$p = \delta\ p'\ (w\ n)$› *assms token-run-step* **by** *simp*
ultimately
have $x \geq y$

using *oldest-token-monotonic-Suc assms* **by** *blast*
moreover
{
have $\bigwedge q''$. $p = \delta\ q''\ (w\ n) \Longrightarrow q'' \notin$ *set qs'* $\cup$ *set sq'* $\cup$ *set ps'*
using ‹$p \notin$ *set zs* $\cup$ *set zs'* $\cup$ *set* [*q*]› ‹¬ *sink p*› **unfolding** *1*[*symmetric*] *2*[*symmetric*] *3*[*symmetric*] *set-map set-filter* **by** *blast*
moreover
{
obtain *us vs* **where** *1*: *map* (λq. δ *q* (*w n*)) *sp'* = *us* @ [*p*] @ *vs* **and** ∀ *u*∈*set us*. *sink u* **and** [] = [*q*←*vs* . ¬ *sink q*]
using ‹*filter* (λq. ¬*sink q*) (*map* (λq. δ *q* (*w n*)) *sp'*) = [*p*]› **unfolding** *filter-eq-Cons-iff* **by** *auto*
moreover
hence $\bigwedge q''$. $q'' \in$ (*set us*) $\cup$ (*set vs*) $\Longrightarrow$ *sink* q''
by (*metis UnE filter-empty-conv*)
hence $p \notin$ (*set us*) $\cup$ (*set vs*)
using ‹¬ *sink p*› **by** *blast*
ultimately
have $\bigwedge q''$. $q'' \in$ (*set sp'* − {*p'*}) $\Longrightarrow \delta\ q''\ (w\ n) \neq p$
using *1* ‹$p = \delta\ p'\ (w\ n)$› ‹$p' \in$ *set sp'*› **by** (*fastforce dest*: *split-list elim*: *map-splitE*)
}
ultimately
have $\bigwedge q''$. $p = \delta\ q''\ (w\ n) \Longrightarrow$ *configuration* q'' *n* $\neq$ {} $\Longrightarrow q'' \in$ *set ps''* $\vee\ q'' = p'$
using *nxt-run-configuration*[*of - n*] ‹¬ *sink p*›[*THEN sink-rev-step*(*2*)] **unfolding** *r-def'*[*unfolded r-def*] *set-append*
by *blast*
moreover
have $\bigwedge q''$. $q'' \in$ *set ps''* $\Longrightarrow x \leq$ *?f-n* q''
using *x-def* **using** *Suc* **unfolding** *r-def'* *map-append sorted-append set-append set-map* **using** ‹$q' \in$ *set sq'*› ‹$p' \in$ *set sp'*›
apply (*simp del*: *oldest-token.simps*) **by** *fastforce*
moreover
have $\bigwedge q''$. $q'' = p' \Longrightarrow x \leq$ *?f-n* q''
using *x-def* **by** *simp*
moreover
have $\bigwedge q''\ x$. $x \in$ *configuration* q'' *n* $\Longrightarrow$ *the* (*oldest-token* q'' *n*) $\leq x$
using *assms* **by** *auto*
ultimately
have $\bigwedge z$. $z \in \bigcup$ {*configuration* p' *n* |p'. $p = \delta\ p'\ (w\ n)$} $\Longrightarrow x \leq z$
by *fastforce*

}
hence $\bigwedge z.\ z \in$ *configuration p* $(Suc\ n) \Longrightarrow x \leq z$
unfolding *configuration-step-eq-unified* **using** ‹$x \leq n$›
by (*cases* $p = q_0$; *auto*)
hence $x \leq y$
using *y-def Min.boundedI configuration-finite* **using** *push-down-oldest-token-configuration*
by *presburger*
ultimately
have *?f-n p'* = *?f-Suc-n p*
using *x-def y-def* **by** *fastforce*
}
ultimately
show *?thesis*
by *presburger*
qed
qed
hence $\bigwedge x\ y\ xs\ ys.$ *map ?f-Suc-n* (*remdups-fwd ?step*) = *xs* @ [*x*, *y*] @ *ys* $\Longrightarrow x \leq y$
by (*auto elim*: *map-splitE simp del*: *remdups-fwd.simps*)
hence *sorted* (*map ?f-Suc-n* (*remdups-fwd* (*?step*)))
using *sorted-pre* **by** *metis*
thus *?case*
by (*simp add*: *r-def sink-def*)
qed (*simp add*: *r-def*)

lemma (**in** *semi-mojmir*) *nxt-run-senior-states*:
defines $r \equiv$ *run* (*nxt* $\Sigma\ \delta\ q_0$) (*init* q_0) *w*
assumes $\neg$*sink q*
assumes *configuration q n* $\neq \{\}$
shows *senior-states q n* = *set* (*takeWhile* ($\lambda q'.\ q' \neq q$) (*r n*))
(**is** *?lhs* = *?rhs*)
proof (*rule set-eqI*, *rule*)
fix q' **assume** q'*-def*: $q' \in$ *senior-states q n*
then obtain *x y* **where** *oldest-token* q' *n* = *Some y* **and** *oldest-token q n* = *Some x* **and** $y < x$
using *senior-states.simps* **using** *assms* **by** *blast*
hence *the* (*oldest-token* q' *n*) < *the* (*oldest-token q n*)
by *fastforce*
moreover
hence $\neg$*sink* q' **and** *configuration* q' *n* $\neq \{\}$
using q'*-def option.distinct*(*1*) ‹*oldest-token* q' *n* = *Some y*›
oldest-token.simps **using** *assms* **by** (*force*, *metis*)
hence $q' \in$ *set* (*r n*) **and** $q \in$ *set* (*r n*)
using *nxt-run-configuration assms* **by** *blast+*

moreover
have *distinct* (*r n*)
unfolding *r-def* **using** *nxt-run-distinct* **by** *fast*
ultimately
obtain r' r'' r''' **where** *r-alt-def*: $r\ n = r' @ q' \# r'' @ q \# r'''$
using *sorted-list*[*OF - - nxt-run-sorted*] *assms* **unfolding** *r-def* **by** *presburger*
hence $q' \in set\ (r' @ q' \# r'')$
by *simp*
thus $q' \in set\ (takeWhile\ (\lambda q'.\ q' \neq q)\ (r\ n))$
using ‹*distinct* (*r n*)› *takeWhile-distinct*[*of* $r' @ q' \# r''$ q r''' q'] **unfolding** *r-alt-def* **by** *simp*
next
fix q' **assume** *q'-def*: $q' \in set\ (takeWhile\ (\lambda q'.\ q' \neq q)\ (r\ n))$
moreover
hence $q' \in set\ (r\ n)$
by (*blast dest*: *set-takeWhileD*)+
hence *5*: $\neg$ *sink* q'
using *nxt-run-configuration assms* **by** *simp*
have $q \in set\ (r\ n)$
using *nxt-run-configuration assms* **by** *blast*+
ultimately
obtain r' r'' r''' **where** *r-alt-def*: $r\ n = r' @ q' \# r'' @ q \# r'''$
using *takeWhile-split* **by** *metis*
have *distinct* (*r n*)
unfolding *r-def* **using** *nxt-run-distinct* **by** *fast*
have *1*: *the* (*oldest-token* q' *n*) $\leq$ *the* (*oldest-token q n*)
using *nxt-run-sorted*[*of n*, *unfolded r-def*[*symmetric*]] *assms*
unfolding *r-alt-def map-append list.map*
unfolding *sorted-append* **by** (*simp del*: *oldest-token.simps*)
have $q \neq q'$
using ‹*distinct* (*r n*)› *r-alt-def* **by** *auto*
moreover
have *2*: *oldest-token* q' *n* $\neq$ *None* **and** *3*: *oldest-token q n* $\neq$ *None*
using *assms* ‹$q' \in set\ (r\ n)$› *nxt-run-configuration* **by** *force*+
ultimately
have *4*: *the* (*oldest-token* q' *n*) $\neq$ *the* (*oldest-token q n*)
by (*metis oldest-token-equal option.collapse*)

show $q' \in$ *senior-states q n*
using *1 2 3 4 5 assms* **by** *fastforce*
qed

lemma (**in** *semi-mojmir*) *nxt-run-state-rank*:

state-rank q n = *rk* (*run* (*nxt* Σ δ q_0) (*init* q_0) w n) q
by (*cases* $\neg$*sink* q $\wedge$ *configuration* q $n \neq \{\}$, *unfold state-rank.simps*)
(*metis nxt-run-senior-states rk-split-card-takeWhile nxt-run-distinct nxt-run-configuration*, *metis nxt-run-configuration rk-facts*(*1*))

lemma (**in** *semi-mojmir*) *nxt-foldl-state-rank*:
state-rank q n = *rk* (*foldl* (*nxt* Σ δ q_0) (*init* q_0) (*map* w [*0*..<n])) q
unfolding *nxt-run-state-rank run-foldl* **..**

lemma (**in** *semi-mojmir*) *nxt-run-step-run*:
run step initial w = *rk* o (*run* (*nxt* Σ δ q_0) (*init* q_0) w)
using *nxt-run-state-rank state-rank-step-foldl*[*unfolded run-foldl*[*symmetric*]]
unfolding *comp-def* **by** *presburger*

definition (**in** *semi-mojmir-def*) Q_E
where
$Q_E \equiv$ *reach* Σ (*nxt* Σ δ q_0) (*init* q_0)

lemma (**in** *semi-mojmir*) *finite-Q*:
finite Q_E
proof −
{
fix i **fix** w :: *nat* $\Rightarrow$ *'a*
assume *range* $w \subseteq \Sigma$
then interpret $\mathfrak{H}$: *semi-mojmir* Σ δ q_0 w
using *finite-reach finite-*Σ **by** (*unfold-locales*, *blast*)
have *set* (*run* (*nxt* Σ δ q_0) (*init* q_0) w i) $\subseteq$ {$\mathfrak{H}$.*token-run* j i | j. $j \leq i$} (**is** *?S1* $\subseteq$ -)
using $\mathfrak{H}$.*nxt-run-configuration* **by** *auto*
also
have ... $\subseteq$ *reach* Σ δ q_0 (**is** - $\subseteq$ *?S2*)
unfolding *reach-def token-run.simps* **using** ‹*range* $w \subseteq \Sigma$› **by** *fastforce*
finally
have *?S1* $\subseteq$ *?S2* **.**
}
hence *set* ‘ $Q_E \subseteq$ *Pow* (*reach* Σ δ q_0)
unfolding Q_E*-def reach-def* **by** *blast*
hence *finite* (*set* ‘ Q_E)
using *finite-reach* **by** (*blast dest*: *finite-subset*)
moreover
have $\bigwedge xs.\ xs \in Q_E \Longrightarrow$ *distinct xs*
unfolding Q_E*-def reach-def* **using** *nxt-run-distinct* **by** *fastforce*
ultimately
show *finite* Q_E

using *set-list* **by** *auto*
qed

lemma (**in** *mojmir-to-rabin-def*) *filt-equiv*:
$(rk\ x,\ \nu,\ y) \in fail_R \longleftrightarrow fail\text{-}filt\ \Sigma\ \delta\ q_0\ (\lambda x.\ x \in F)\ (x,\ \nu,\ y')$
$(rk\ x,\ \nu,\ y) \in succeed_R\ i \longleftrightarrow succeed\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ (x,\ \nu,\ y')$
$(rk\ x,\ \nu,\ y) \in merge_R\ i \longleftrightarrow merge\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ (x,\ \nu,\ y')$
by (*simp add*: $fail_R$*-def* $succeed_R$*-def* $merge_R$*-def del*: *rk.simps*; *metis* (*no-types*, *lifting*) *option.sel rk-facts*(*2*))+

lemma *fail-filt-eq*:
$fail\text{-}filt\ \Sigma\ \delta\ q_0\ P\ (x,\ \nu,\ y) \longleftrightarrow (rk\ x,\ \nu,\ y') \in mojmir\text{-}to\text{-}rabin\text{-}def.fail_R\ \Sigma\ \delta\ q_0\ \{x.\ P\ x\}$
unfolding *mojmir-to-rabin-def.filt-equiv*(*1*)[**where** $y' = y$] **by** *simp*

lemma *merge-filt-eq*:
$merge\text{-}filt\ \delta\ q_0\ P\ i\ (x,\ \nu,\ y) \longleftrightarrow (rk\ x,\ \nu,\ y') \in mojmir\text{-}to\text{-}rabin\text{-}def.merge_R\ \delta\ q_0\ \{x.\ P\ x\}\ i$
unfolding *mojmir-to-rabin-def.filt-equiv*(*3*)[**where** $y' = y$] **by** *simp*

lemma *succeed-filt-eq*:
$succeed\text{-}filt\ \delta\ q_0\ P\ i\ (x,\ \nu,\ y) \longleftrightarrow (rk\ x,\ \nu,\ y') \in mojmir\text{-}to\text{-}rabin\text{-}def.succeed_R\ \delta\ q_0\ \{x.\ P\ x\}\ i$
unfolding *mojmir-to-rabin-def.filt-equiv*(*2*)[**where** $y' = y$] **by** *simp*

theorem (**in** *mojmir-to-rabin*) *rabin-accept-iff-rabin-list-accept-rank*:
$accepting\text{-}pair_R\ \delta_{\mathcal{R}}\ q_{\mathcal{R}}\ (Acc_{\mathcal{R}}\ i)\ w \longleftrightarrow accepting\text{-}pair_R\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)\ (\{t.\ fail\text{-}filt\ \Sigma\ \delta\ q_0\ (\lambda x.\ x \in F)\ t\} \cup \{t.\ merge\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ t\},\ \{t.\ succeed\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ t\})\ w$
(**is** $accepting\text{-}pair_R\ \delta_{\mathcal{R}}\ q_{\mathcal{R}}\ (?F,\ ?I)\ w \longleftrightarrow accepting\text{-}pair_R\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)\ (?F',\ ?I')\ w$)
proof −
have *finite* $(reach_t\ \Sigma\ \delta_{\mathcal{R}}\ q_{\mathcal{R}})$
using *wellformed-*$\mathcal{R}$ *finite-*Σ *finite-reach*$_t$ **by** *fast*
moreover
have *finite* $(reach_t\ \Sigma\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0))$
using *finite-Q finite-*Σ *finite-reach*$_t$ **by** (*auto simp add*: Q_E*-def*)
moreover
have $run_t\ step\ initial\ w = (\lambda(x,\ \nu,\ y).\ (rk\ x,\ \nu,\ rk\ y))\ o\ (run_t\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)\ w)$
using *nxt-run-step-run* **by** *auto*
moreover
note *bounded-w filt-equiv*
ultimately

show *?thesis*
by (*intro accepting-pair$_R$-abstract*) *auto*
qed

18.1 Compute Rabin Automata List Representation

fun *mojmir-to-rabin-exec*
where
mojmir-to-rabin-exec Σ δ q_0 F = (
let
$q_0{}'$ = *init* q_0;
δ' = δ_L Σ (*nxt* (*set* Σ) δ q_0) $q_0{}'$;
max-rank = *card* (*Set.filter* (*Not o semi-mojmir-def.sink* (*set* Σ) δ q_0) (Q_L Σ δ q_0));
fail = *Set.filter* (*fail-filt* (*set* Σ) δ q_0 F) δ';
merge = (λi. *Set.filter* (*merge-filt* δ q_0 F i) δ');
succeed = (λi. *Set.filter* (*succeed-filt* δ q_0 F i) δ')
in
(δ', $q_0{}'$, (λi. (*fail* $\cup$ (*merge* i), *succeed* i)) ‘ {*0..<max-rank*}))

18.2 Code Generation

declare *semi-mojmir-def.sink-def* [*code*]

— Drop computation of length by different code equation
fun *index-option* :: *nat* ⇒ *'a list* ⇒ *'a* ⇒ *nat option*
where
index-option n [] *y* = *None*
| *index-option n* (*x* # *xs*) *y* = (*if x* = *y then Some n else index-option* (*Suc n*) *xs y*)

declare *rk.simps* [*code del*]

lemma *rk-eq-index-option* [*code*]:
rk xs x = *index-option 0 xs x*
proof –
have *A*: $\bigwedge n$. *x* ∈ *set xs* ⟹ *index xs x* + *n* = *the* (*index-option n xs x*)
and *B*: $\bigwedge n$. *x* ∉ *set xs* ⟷ *index-option n xs x* = *None*
by (*induction xs*) (*auto*, *metis add-Suc-right*)
thus *?thesis*
proof (*cases x* ∈ *set xs*)
case *True*
moreover
hence *index xs x* = *the* (*index-option 0 xs x*)

using *A*[*OF True*, *of 0*] **by** *simp*
ultimately
show *?thesis*
unfolding *rk.simps* **by** (*metis* (*mono-tags*, *lifting*) *B True index-less-size-conv less-irrefl option.collapse*)
qed *simp*
qed

— Check Code Export
export-code *init nxt fail-filt succeed-filt merge-filt mojmir-to-rabin-exec* **checking**

lemma (**in** *mojmir*) *max-rank-card*:
assumes $\Sigma = set\ \Sigma'$
shows $max\text{-}rank = card\ (Set.filter\ (Not\ o\ semi\text{-}mojmir\text{-}def.sink\ (set\ \Sigma')\ \delta\ q_0)\ (Q_L\ \Sigma'\ \delta\ q_0))$
unfolding *max-rank-def* Q_L*-reach*[*OF finite-reach*[*unfolded* ‹$\Sigma = set\ \Sigma'$›]]

by (*simp add*: *Set.filter-def set-diff-eq assms*(*1*))

theorem (**in** *mojmir-to-rabin*) *exec-correct*:
assumes $\Sigma = set\ \Sigma'$
shows $accept \longleftrightarrow accept_R\text{-}LTS\ (mojmir\text{-}to\text{-}rabin\text{-}exec\ \Sigma'\ \delta\ q_0\ (\lambda x.\ x \in F))\ w$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
have *F1*: $finite\ (reach\ \Sigma\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0))$
using *finite-Q* **by** (*simp add*: Q_E*-def*)
hence *F2*: $finite\ (reach_t\ \Sigma\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0))$
using *finite-*Σ **by** (*rule finite-reach*$_t$)

let $?\delta' = \delta_L\ \Sigma'\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)$
have δ'*-Def*: $?\delta' = reach_t\ \Sigma\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)$
using δ_L*-reach*[*OF F2*[*unfolded assms*]] **unfolding** *assms* **by** *simp*

have *3*: $snd\ (snd\ ((mojmir\text{-}to\text{-}rabin\text{-}exec\ \Sigma'\ \delta\ q_0\ (\lambda x.\ x \in F))))$
$= \{((\{t.\ fail\text{-}filt\ \Sigma\ \delta\ q_0\ (\lambda x.\ x \in F)\ t\} \cup \{t.\ merge\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ t\}) \cap reach_t\ \Sigma\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0),$
$\{t.\ succeed\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ t\}\ \cap reach_t\ \Sigma\ (nxt\ \Sigma\ \delta\ q_0)\ (init\ q_0)) \mid i.\ i < max\text{-}rank\}$
unfolding *assms mojmir-to-rabin-exec.simps Let-def fst-conv snd-conv set-map* δ'*-Def*[*unfolded assms*] *max-rank-card*[*OF assms*, *symmetric*]
unfolding *assms*[*symmetric*] *Set.filter-def* **by** *auto*

have *?lhs* $\longleftrightarrow accept_R\ (\delta_{\mathcal{R}},\ q_{\mathcal{R}},\ \{(Acc_{\mathcal{R}}\ i) \mid i.\ i < max\text{-}rank\})\ w$

using *mojmir-accept-iff-rabin-accept* **by** *blast*

moreover

have ... $\longleftrightarrow$ $accept_R$ ($nxt\ \Sigma\ \delta\ q_0$, $init\ q_0$, $\{(\{t.\ fail\text{-}filt\ \Sigma\ \delta\ q_0\ (\lambda x.\ x \in F)\ t\} \cup \{t.\ merge\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ t\}, \{t.\ succeed\text{-}filt\ \delta\ q_0\ (\lambda x.\ x \in F)\ i\ t\}) \mid i.\ i < max\text{-}rank\}$) w
unfolding $accept_R$*-def fst-conv snd-conv* **using** *rabin-accept-iff-rabin-list-accept-rank* **by** *blast*

moreover

have ... $\longleftrightarrow$ *?rhs*
apply (*subst* $accept_R$*-restrict*[*OF bounded-w*])
unfolding *3*[*unfolded mojmir-to-rabin-exec.simps Let-def snd-conv*, *symmetric*] *assms*[*symmetric*] *mojmir-to-rabin-exec.simps Let-def* **unfolding** *assms* δ'*-Def*[*unfolded assms*]
unfolding $accept_R$*-LTS*[*OF bounded-w*[*unfolded assms*], *symmetric*, *unfolded assms*] **by** *simp*

ultimately

show *?thesis*
by *blast*
qed

end

19 Executable Translation from LTL to Rabin Automata

theory *LTL-Rabin-Impl*
imports *Main ../Auxiliary/Map2 ../LTL-Rabin ../LTL-Rabin-Unfold-Opt af-Impl Mojmir-Rabin-Impl*
begin

19.1 Template

19.1.1 Definition

locale *ltl-to-rabin-base-code-def* = *ltl-to-rabin-base-def* +
fixes
$M\text{-}fin_C$ :: $'a\ ltl \Rightarrow ('a\ ltl, nat)\ mapping \Rightarrow ('a\ ltl_P \times ('a\ ltl, 'a\ ltl_P\ list)\ mapping, 'a\ set)\ transition \Rightarrow bool$

begin

— Transition Function and Initial State

fun $deltaC$
where
$delta_C\ \Sigma = \delta \times \uparrow\Delta_\times\ (nxt\ \Sigma\ \delta_M\ o\ q_{0M}\ o\ theG)$

fun $initial_C$
where
$initial_C\ \varphi = (q_0\ \varphi,\ Mapping.tabulate\ (G\text{-}list\ \varphi)\ (init\ o\ q_{0M}\ o\ theG))$

— Acceptance Condition

definition $max\text{-}rank\text{-}of_C$
where
$max\text{-}rank\text{-}of_C\ \Sigma\ \psi = card\ (Set.filter\ (Not\ o\ semi\text{-}mojmir\text{-}def.sink\ (set\ \Sigma)$ $\delta_M\ (q_{0M}\ (theG\ \psi)))\ (Q_L\ \Sigma\ \delta_M\ (q_{0M}\ (theG\ \psi))))$

fun $Acc\text{-}fin_C$
where
$Acc\text{-}fin_C\ \Sigma\ \pi\ \chi\ ((\text{-},\ m'),\ \nu,\ \text{-}) = ($
let
$t = (the\ (Mapping.lookup\ m'\ \chi),\ \nu,\ [\,]);$ — Third element is unused. Hence it is safe to pass a dummy value.
$\mathcal{G} = Mapping.keys\ \pi$
in
$fail\text{-}filt\ \Sigma\ \delta_M\ (q_{0M}\ (theG\ \chi))\ (ltl\text{-}prop\text{-}entails\text{-}abs\ \mathcal{G})\ t$
$\vee\ merge\text{-}filt\ \delta_M\ (q_{0M}\ (theG\ \chi))\ (ltl\text{-}prop\text{-}entails\text{-}abs\ \mathcal{G})\ (the\ (Mapping.lookup$ $\pi\ \chi))\ t)$

fun $Acc\text{-}inf_C$
where
$Acc\text{-}inf_C\ \pi\ \chi\ ((\text{-},\ m'),\ \nu,\ \text{-}) = ($
let
$t = (the\ (Mapping.lookup\ m'\ \chi),\ \nu,\ [\,]);$ — Third element is unused. Hence it is safe to pass a dummy value.
$\mathcal{G} = Mapping.keys\ \pi$
in
$succeed\text{-}filt\ \delta_M\ (q_{0M}\ (theG\ \chi))\ (ltl\text{-}prop\text{-}entails\text{-}abs\ \mathcal{G})\ (the\ (Mapping.lookup$ $\pi\ \chi))\ t)$

definition $mappings_C$ :: $'a\ set\ list \Rightarrow 'a\ ltl \Rightarrow ('a\ ltl,\ nat)\ mapping\ set$
where

mappings$_C$ Σ φ $\equiv$ $\{\pi$. *Mapping.keys* $\pi \subseteq$ **G** $\varphi \wedge (\forall \chi \in$ (*Mapping.keys* π). *the* (*Mapping.lookup* π χ) $<$ *max-rank-of*$_C$ Σ χ)$\}$

definition *reachable-transitions*$_C$
where
reachable-transitions$_C$ Σ $\varphi \equiv \delta_L$ Σ (*delta*$_C$ (*set* Σ)) (*initial*$_C$ φ)

fun *ltl-to-generalized-rabin*$_C$
where
ltl-to-generalized-rabin$_C$ Σ φ = (
let
δ-*LTS* = *reachable-transitions*$_C$ Σ φ;
α-*fin-filter* = $\lambda\pi$ *t*. *M-fin*$_C$ φ π *t* $\vee$ ($\exists \chi \in$ *Mapping.keys* π. *Acc-fin*$_C$ (*set* Σ) π χ *t*);
to-pair = $\lambda\pi$. (*Set.filter* (α-*fin-filter* π) δ-*LTS*, ($\lambda\chi$. *Set.filter* (*Acc-inf*$_C$ π χ) δ-*LTS*) ` *Mapping.keys* π);
α = *to-pair* ` (*mappings*$_C$ Σ φ) — Multi-thread here!, prove *mappings* (*set* ...) equation
in
(δ-*LTS*, *initial*$_C$ φ, α))

lemma *mappings*$_C$-*code*:
mappings$_C$ Σ φ = (
let
Gs = *G-list* φ;
max-rank = *Mapping.lookup* (*Mapping.tabulate Gs* (*max-rank-of*$_C$ Σ))
in
set (*concat* (*map* (*mapping-generator-list* (λx. [*0* ..< *the* (*max-rank* *x*)])) (*subseqs Gs*))))
(**is** *?lhs* = *?rhs*)
proof −
{
fix *xs* :: *'a ltl list*
have *subset-G*: $\bigwedge x$. $x \in$ *set* (*subseqs* (*xs*)) $\Longrightarrow$ *set x* $\subseteq$ *set xs*
apply (*induction xs*)
apply (*simp*)
by (*insert subseqs-powset*; *blast*)
}
hence *subset-G*: $\bigwedge x$. $x \in$ *set* (*subseqs* (*G-list* φ)) $\Longrightarrow$ *set x* $\subseteq$ **G** φ
unfolding *G-eq-G-list* **by** *auto*

have *?lhs* = $\bigcup\{\{\pi$. *Mapping.keys* π = *xs* $\wedge$ ($\forall\chi\in$*Mapping.keys* π. *the* (*Mapping.lookup* π χ) $<$ *max-rank-of*$_C$ Σ χ)$\}$ | *xs*. *xs* $\in$ *set* ` (*set* (*subseqs* (*G-list* φ)))$\}$

unfolding *mappings*$_C$*-def G-eq-G-list subseqs-powset* **by** *auto*
also
have ... = $\bigcup${{π. *Mapping.keys* π = *set xs* $\wedge$ ($\forall$ χ $\in$ *set xs*. *the* (*Mapping.lookup* π χ) < *max-rank-of*$_C$ Σ χ)} |
xs. *xs* $\in$ *set* (*subseqs* (*G-list* φ))}
by *auto*
also
have ... = *?rhs*
using *subset-G*
by (*auto simp add*: *Let-def mapping-generator-code* [*symmetric*]
lookup-tabulate G-eq-G-list [*symmetric*] *mapping-generator-set-eq*
cong del: *SUP-cong-simp*; *blast*)
finally
show *?thesis*
by *simp*
qed

lemma *reach-delta-initial*:
assumes (x, y) $\in$ *reach* Σ (*delta*$_C$ Σ) (*initial*$_C$ φ)
assumes χ $\in$ **G** φ
shows *Mapping.lookup y* χ $\neq$ *None* (**is** *?t1*)
and *distinct* (*the* (*Mapping.lookup y* χ)) (**is** *?t2*)
proof –
from *assms*(*1*) **obtain** *w i* **where** *y-def*: *y* = *run* ($\uparrow\Delta_\times$ (*nxt* Σ δ_M *o* q_{0M} *o theG*)) (*Mapping.tabulate* (*G-list* φ) (*init o* q_{0M} *o theG*)) *w i*
unfolding *reach-def delta*$_C$*.simps initial*$_C$*.simps simple-product-run* **by** *fast*
from *assms*(*2*) *nxt-run-distinct* **show** *?t1*
unfolding *y-def* **using** *product-abs-run-Some*[*of Mapping.tabulate* (*G-list* φ) (*init o* q_{0M} *o theG*) χ] **unfolding** *G-eq-G-list*
unfolding *lookup-tabulate* **by** *fastforce*
with *nxt-run-distinct* **show** *?t2*
unfolding *y-def* **using** *lookup-tabulate*
by (*metis* (*no-types*) *G-eq-G-list assms*(*2*) *comp-eq-dest-lhs option.sel product-abs-run-Some*)
qed

end

19.1.2 Correctness

fun *abstract-state* :: $'x \times ('y, 'z\ list)\ mapping \Rightarrow 'x \times ('y \rightharpoonup 'z \rightharpoonup nat)$
where
abstract-state (*a*, *b*) = (*a*, (*map-option rk*) *o* (*Mapping.lookup b*))

fun *abstract-transition*
where
abstract-transition $(q,\ \nu,\ q') = (\mathit{abstract\text{-}state}\ q,\ \nu,\ \mathit{abstract\text{-}state}\ q')$

locale *ltl-to-rabin-base-code* = *ltl-to-rabin-base* + *ltl-to-rabin-base-code-def* +
assumes
$\mathit{M\text{-}fin}_C$-*correct*: $[\![t \in \mathit{reach}_t\ \Sigma\ (\mathit{delta}_C\ \Sigma)\ (\mathit{initial}_C\ \varphi);\ \mathit{dom}\ \pi \subseteq \mathbf{G}\ \varphi]\!] \Longrightarrow$
$\mathit{abstract\text{-}transition}\ t \in \mathit{M\text{-}fin}\ \pi = \mathit{M\text{-}fin}_C\ \varphi\ (\mathit{Mapping.Mapping}\ \pi)\ t$
begin

lemma $\mathit{finite\text{-}reach}_C$:
$\mathit{finite}\ (\mathit{reach}_t\ \Sigma\ (\mathit{delta}_C\ \Sigma)\ (\mathit{initial}_C\ \varphi))$
proof −
note *finite-reach'*
moreover
have $(\bigwedge x.\ x \in \mathbf{G}\ \varphi \Longrightarrow \mathit{finite}\ (\mathit{reach}\ \Sigma\ ((\mathit{nxt}\ \Sigma\ \delta_M\ o\ q_{0M}\ o\ \mathit{theG})\ x)\ ((\mathit{init}\ o\ q_{0M}\ o\ \mathit{theG})\ x)))$
using *semi-mojmir.finite-Q*[*OF semi-mojmir*] **unfolding** *G-eq-G-list semi-mojmir-def*.Q_E-*def* **by** *simp*
hence $\mathit{finite}\ (\mathit{reach}\ \Sigma\ (\uparrow\Delta_\times\ (\mathit{nxt}\ \Sigma\ \delta_M\ o\ q_{0M}\ o\ \mathit{theG}))\ (\mathit{Mapping.tabulate}\ (\mathit{G\text{-}list}\ \varphi)\ (\mathit{init}\ o\ q_{0M}\ o\ \mathit{theG})))$
by (*metis* (*no-types*) *finite-reach-product-abs*[*OF finite-keys-tabulate*] *G-eq-G-list keys-tabulate lookup-tabulate-Some*)
ultimately
have $\mathit{finite}\ (\mathit{reach}\ \Sigma\ (\mathit{delta}_C\ \Sigma)\ (\mathit{initial}_C\ \varphi))$
using *finite-reach-simple-product* **by** *fastforce*
thus *?thesis*
using *finite*-Σ **by** (*simp add*: $\mathit{finite\text{-}reach}_t$)
qed

lemma $\mathit{max\text{-}rank\text{-}of}_C$-*eq*:
assumes $\Sigma = \mathit{set}\ \Sigma'$
shows $\mathit{max\text{-}rank\text{-}of}_C\ \Sigma'\ \psi = \mathit{max\text{-}rank\text{-}of}\ \Sigma\ \psi$
proof −
interpret $\mathfrak{M}$: $\mathit{semi\text{-}mojmir}\ \mathit{set}\ \Sigma'\ \delta_M\ q_{0M}\ (\mathit{theG}\ \psi)\ w$
using *semi-mojmir assms* **by** *force*
show *?thesis*
unfolding *max-rank-of-def* $\mathit{max\text{-}rank\text{-}of}_C$-*def* Q_L-*reach*[*OF* $\mathfrak{M}$.*finite-reach*] *semi-mojmir-def*.*max-rank-def*
by (*simp add*: *Set.filter-def set-diff-eq assms*)
qed

lemma *reachable-transitions*$_C$*-eq*:
  assumes $\Sigma = set\ \Sigma'$
  shows *reachable-transitions*$_C$ Σ' φ = *reach*$_t$ Σ (*delta*$_C$ Σ) (*initial*$_C$ φ)
 by (*simp only*: *reachable-transitions*$_C$*-def* δ_L*-reach*[*OF finite-reach*$_C$[*unfolded assms*]] *assms*)

lemma *run-abstraction-correct*:
  run (*delta* Σ) (*initial* φ) *w* = *abstract-state o* (*run* (*delta*$_C$ Σ) (*initial*$_C$ φ) *w*)
proof –
  {
    fix *i*

    let $?\delta_2 = \Delta_\times$ ($\lambda\chi$. *semi-mojmir-def.step* Σ δ_M (q_{0M} (*theG* χ)))
    let $?q_2 = \iota_\times$ (**G** φ) ($\lambda\chi$. *semi-mojmir-def.initial* (q_{0M} (*theG* χ)))

    let $?\delta_2{}' = \uparrow\Delta_\times$ (*nxt* Σ δ_M *o* q_{0M} *o theG*)
    let $?q_2{}'$ = *Mapping.tabulate* (*G-list* φ) (*init o* q_{0M} *o theG*)

    {
      fix *q*
      assume $q \notin$ **G** φ
       hence $?q_2$ *q* = *None* **and** *Mapping.lookup* (*run* $?\delta_2{}'$ $?q_2{}'$ *w i*) *q* = *None*
          using *G-eq-G-list product-abs-run-None* **by** (*simp*, *metis domIff keys-dom-lookup keys-tabulate*)
      hence *run* $?\delta_2$ $?q_2$ *w i q* = (λm. (*map-option rk*) *o* (*Mapping.lookup m*)) (*run* $?\delta_2{}'$ $?q_2{}'$ *w i*) *q*
        using *product-run-None* **by** (*simp del*: *nxt.simps rk.simps*)
    }

    moreover

    {
      fix *q j*
      assume $q \in$ **G** φ
      hence *init*: $?q_2$ *q* = *Some* (*semi-mojmir-def.initial* (q_{0M} (*theG q*)))
        and *Mapping.lookup* (*run* $?\delta_2{}'$ $?q_2{}'$ *w i*) *q* = *Some* (*run* ((*nxt* Σ δ_M $\circ$ q_{0M} $\circ$ *theG*) *q*) ((*init* $\circ$ q_{0M} $\circ$ *theG*) *q*) *w i*)
        apply (*simp del*: *nxt.simps*)
      apply (*metis G-eq-G-list* ‹$q \in$ **G** φ› *lookup-tabulate product-abs-run-Some*)

        done

hence *run* $?\delta_2$ $?q_2$ *w i q* = ($\lambda m.$ (*map-option rk*) *o* (*Mapping.lookup* *m*)) (*run* $?\delta_2{}'$ $?q_2{}'$ *w i*) *q*
unfolding *product-run-Some*[*of* $\iota_\times$ (**G** φ) ($\lambda\chi$. *semi-mojmir-def.initial* (q_{0M} (*theG* χ))) *q*, *OF init*]
by (*simp del*: *product.simps nxt.simps rk.simps*; *unfold map-of-map semi-mojmir.nxt-run-step-run*[*OF semi-mojmir*]; *simp*)
}

ultimately

have *run* $?\delta_2$ $?q_2$ *w i* = ($\lambda m.$ (*map-option rk*) *o* (*Mapping.lookup m*)) (*run* $?\delta_2{}'$ $?q_2{}'$ *w i*)
by *blast*
}
hence $\bigwedge i.$ *run* (*delta* Σ) (*initial* φ) *w i* = *abstract-state* (*run* ($delta_C$ Σ) ($initial_C$ φ) *w i*)
using *finite-*Σ *bounded-w* **by** (*simp add*: *simple-product-run comp-def del*: *simple-product.simps*)
thus *?thesis*
by *auto*
qed

lemma
assumes $t \in reach_t$ Σ ($delta_C$ Σ) ($initial_C$ φ)
assumes $\chi \in$ **G** φ
shows *Acc-fin*$_C$*-correct*:
abstract-transition $t \in$ *Acc-fin* Σ π $\chi \longleftrightarrow$ *Acc-fin*$_C$ Σ (*Mapping.Mapping* π) χ *t* (**is** *?t1*)
and *Acc-inf*$_C$*-correct*:
abstract-transition $t \in$ *Acc-inf* π $\chi \longleftrightarrow$ *Acc-inf*$_C$ (*Mapping.Mapping* π) χ *t* (**is** *?t2*)
proof −
obtain $x\ y\ \nu\ z\ z'$ **where** *t-def* [*simp*]: $t = ((x, y), \nu, (z, z'))$
by (*metis prod.collapse*)
have $(x, y) \in$ *reach* Σ ($delta_C$ Σ) ($initial_C$ φ)
and $(z, z') \in$ *reach* Σ ($delta_C$ Σ) ($initial_C$ φ)
using *assms*(*1*) **unfolding** $reach_t$*-def reach-def* run_t*.simps t-def* **by** *blast+*
then obtain $m\ m'$ **where** [*simp*]: *Mapping.lookup y* χ = *Some m*
and *Mapping.lookup y* $\chi \neq$ *None*
and [*simp*]: *Mapping.lookup* z' χ = *Some* m'
using *assms*(*2*) **by** (*blast dest*: *reach-delta-initial*)+

have *FF* [*simp*]: *fail-filt* Σ δ_M (q_{0M} (*theG* χ)) (*ltl-prop-entails-abs* (*dom*

π)) (*the* (*Mapping.lookup* y χ), ν, [])
  = ((*the* (*map-option rk* (*Mapping.lookup* y χ)), ν, (λx. *Some 0*)) $\in$ *mojmir-to-rabin-def.fail*$_R$ Σ δ_M (q_{0M} (*theG* χ)) {q. *dom* π $\uparrow\models_P$ q})
  unfolding *option.map-sel*[*OF* ‹*Mapping.lookup y* $\chi \neq$ *None*›] *fail-filt-eq*[**where** y = [], *symmetric*] **by** *simp*

  have *MF* [*simp*]: $\bigwedge i$. *merge-filt* δ_M (q_{0M} (*theG* χ)) (*ltl-prop-entails-abs* (*dom* π)) *i* (*the* (*Mapping.lookup* y χ), ν, [])
  = ((*the* (*map-option rk* (*Mapping.lookup* y χ)), ν, (λx. *Some 0*)) $\in$ *mojmir-to-rabin-def.merge*$_R$ δ_M (q_{0M} (*theG* χ)) {q. *dom* π $\uparrow\models_P$ q} *i*)
  unfolding *option.map-sel*[*OF* ‹*Mapping.lookup y* $\chi \neq$ *None*›] *merge-filt-eq*[**where** y = [], *symmetric*] **by** *simp*

  have *SF* [*simp*]: $\bigwedge i$. *succeed-filt* δ_M (q_{0M} (*theG* χ)) (*ltl-prop-entails-abs* (*dom* π)) *i* (*the* (*Mapping.lookup* y χ), ν, [])
  = ((*the* (*map-option rk* (*Mapping.lookup* y χ)), ν, (λx. *Some 0*)) $\in$ *mojmir-to-rabin-def.succeed*$_R$ δ_M (q_{0M} (*theG* χ)) {q. *dom* π $\uparrow\models_P$ q} *i*)
  unfolding *option.map-sel*[*OF* ‹*Mapping.lookup y* $\chi \neq$ *None*›] *succeed-filt-eq*[**where** y = [], *symmetric*] **by** *simp*

  note *mojmir-to-rabin-def.fail*$_R$*-def* [*simp*]
  note *mojmir-to-rabin-def.merge*$_R$*-def* [*simp*]
  note *mojmir-to-rabin-def.succeed*$_R$*-def* [*simp*]

  show *?t1* **and** *?t2*
  by (*simp-all add*: *Let-def keys.abs-eq lookup.abs-eq del*: *rk.simps*)
  (*rule*; *metis option.distinct*(*1*) *option.sel option.collapse rk-facts*(*1*))+
qed

theorem *ltl-to-generalized-rabin*$_C$*-correct*:
  assumes Σ = *set* Σ'
  shows *accept*$_{GR}$ (*ltl-to-generalized-rabin* Σ φ) *w* $\longleftrightarrow$ *accept*$_{GR}$*-LTS* (*ltl-to-generalized-rabin*$_C$ Σ' φ) *w*
  (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
  let $?\delta$ = *delta* Σ
  let $?q_0$ = *initial* φ

  let $?\delta_C$ = *delta*$_C$ Σ
  let $?q_{0C}$ = *initial*$_C$ φ
  let *?reach*$_C$ = *reach*$_t$ Σ (*delta*$_C$ Σ) (*initial*$_C$ φ)

  note *reachable-transitions*$_C$*-simp*[*simp*] = *reachable-transitions*$_C$*-eq*[*OF*

assms]
note *max-rank-of*$_C$*-simp*[*simp*] = *max-rank-of*$_C$*-eq*[*OF assms*]

{
fix π :: *'a ltl* $\Rightarrow$ *nat option*
assume π*-wellformed*: *dom* $\pi \subseteq$ **G** φ

let *?F* = (*M-fin* $\pi \cup \bigcup\{$*Acc-fin* $\Sigma\ \pi\ \chi \mid \chi.\ \chi \in$ *dom* $\pi\}$, $\{$*Acc-inf* $\pi\ \chi \mid \chi.\ \chi \in$ *dom* $\pi\}$)
let *?fin* = $\{t.$ *M-fin*$_C$ φ (*Mapping.Mapping* π) $t\} \cup \{t.\ \exists\chi \in$ *dom* π. *Acc-fin*$_C$ Σ (*Mapping.Mapping* π) χ $t\}$
let *?inf* = $\{\{t.$ *Acc-inf*$_C$ (*Mapping.Mapping* π) χ $t\} \mid \chi.\ \chi \in$ *dom* $\pi\}$

have *finite-reach'*: *finite* (*reach*$_t$ Σ (*delta* Σ) (*initial* φ))
by (*meson finite-reach finite-*Σ *finite-reach*$_t$)

have *run-abstraction-correct'*:
run$_t$ (*delta* Σ) (*initial* φ) *w* = *abstract-transition o* (*run*$_t$ (*delta*$_C$ Σ) (*initial*$_C$ φ) *w*)
using *run-abstraction-correct comp-def* **by** *auto*

have *accepting-pair*$_{GR}$ *?*δ *?q*$_0$ *?F w* $\longleftrightarrow$ *accepting-pair*$_{GR}$ *?*δ_C *?q*$_{0C}$ (*?fin*, *?inf*) *w* (**is** *?l* $\longleftrightarrow$ -)
by (*rule accepting-pair*$_{GR}$*-abstract*[*OF finite-reach' finite-reach*$_C$ *bounded-w*];
insert ‹*dom* $\pi \subseteq$ **G** φ› *M-fin*$_C$*-correct Acc-fin*$_C$*-correct Acc-inf*$_C$*-correct run-abstraction-correct'*; *blast*)
also
have ... $\longleftrightarrow$ *accepting-pair*$_{GR}$*-LTS ?reach*$_C$ *?q*$_{0C}$ (*?fin* $\cap$ *?reach*$_C$, (λI. $I \cap$ *?reach*$_C$) ‘ *?inf*) *w* (**is** - $\longleftrightarrow$ *?r*)
using *bounded-w* **by** (*simp only*: *accepting-pair*$_{GR}$*-LTS*[*symmetric*] *accepting-pair*$_{GR}$*-restrict*[*symmetric*])
finally
have *?l* $\longleftrightarrow$ *?r* **.**
}

note *X* = *this*

{
assume *?lhs*
then obtain π **where** *1*: *dom* $\pi \subseteq$ **G** φ
and *2*: $\bigwedge\chi.\ \chi \in$ *dom* $\pi \Longrightarrow$ *the* ($\pi\ \chi$) < *max-rank-of* $\Sigma\ \chi$
and *3*: *accepting-pair*$_{GR}$ (*delta* Σ) (*initial* φ) (*M-fin* $\pi \cup \bigcup\{$*Acc-fin* Σ $\pi\ \chi \mid\chi.\ \chi \in$ *dom* $\pi\}$, $\{$*Acc-inf* $\pi\ \chi \mid\chi.\ \chi \in$ *dom* $\pi\}$) *w*
by *auto*

define π' **where** $\pi' = \textit{Mapping.Mapping}\ \pi$

have *dom* π = *Mapping.keys* π' **and** π = *Mapping.lookup* π'
by (*simp-all add*: *keys.abs-eq lookup.abs-eq* π'*-def*)

have *acc-pair-LTS*: *accepting-pair*$_{GR}$*-LTS ?reach*$_C$ *?q*$_{0C}$ (({*t*. *M-fin*$_C$ φ π' *t*} ∪ {*t*. ∃χ ∈ *Mapping.keys* π'. *Acc-fin*$_C$ Σ π' χ *t*}) ∩ *?reach*$_C$,
(λ*I*. *I* ∩ *?reach*$_C$) ‘ {{*t*. *Acc-inf*$_C$ π' χ *t*} | χ. χ ∈ *Mapping.keys* π'})
w
using *3* **unfolding** *X*[*OF 1*] **unfolding** ‹*dom* π = *Mapping.keys* π'› π'*-def*[*symmetric*] **by** *simp*

show *?rhs*
apply (*unfold ltl-to-generalized-rabin*$_C$*.simps Let-def*)
apply (*intro accept*$_{GR}$*-LTS-I*)
apply (*insert acc-pair-LTS*; *auto simp add*: *assms*[*symmetric*] *mappings*$_C$*-def*)
apply (*insert 1 2*; *unfold* ‹*dom* π = *Mapping.keys* π'›; *unfold* ‹π = *Mapping.lookup* π'›)
by (*auto simp add*: *assms*[*symmetric*] *Set.filter-def image-def mappings*$_C$*-def*)
}

moreover

{
assume *?rhs*
obtain *Fin Inf* **where** (*Fin*, *Inf*) ∈ *snd* (*snd* (*ltl-to-generalized-rabin*$_C$ Σ′ φ))
and *4*: *accepting-pair*$_{GR}$*-LTS ?reach*$_C$ (*initial*$_C$ φ) (*Fin*, *Inf*) *w*
using *accept*$_{GR}$*-LTS-E*[*OF* ‹*?rhs*›] **apply** (*simp add*: *Let-def assms del*: *accept*$_{GR}$*-LTS.simps*) **by** *auto*

then obtain π **where** *Y*: (*Fin*, *Inf*) = (*Set.filter* (λ*t*. *M-fin*$_C$ φ π *t* ∨ (∃χ ∈ *Mapping.keys* π. *Acc-fin*$_C$ Σ π χ *t*)) *?reach*$_C$,
(λχ. *Set.filter* (*Acc-inf*$_C$ π χ) *?reach*$_C$) ‘ (*Mapping.keys* π))
and *1*: *Mapping.keys* π ⊆ **G** φ **and** *2*: ⋀χ. χ ∈ *Mapping.keys* π ⟹ *the* (*Mapping.lookup* π χ) < *max-rank-of* Σ χ
unfolding *ltl-to-generalized-rabin*$_C$*.simps Let-def fst-conv snd-conv mappings*$_C$*-def assms reachable-transitions*$_C$*-simp max-rank-of*$_C$*-simp* **by** *auto*
define π' **where** $\pi' = \textit{Mapping.rep}\ \pi$
have *dom* π' = *Mapping.keys* π **and** *Mapping.Mapping* π' = π

by (*simp-all add*: π'*-def mapping.rep-inverse keys.rep-eq*)
have *1*: *dom* $\pi' \subseteq$ **G** φ **and** *2*: $\bigwedge\chi.\ \chi \in$ *dom* $\pi' \Longrightarrow$ *the* $(\pi'\ \chi) <$ *max-rank-of* $\Sigma\ \chi$
using *1 2* **unfolding** π'*-def Mapping.keys.rep-eq Mapping.mapping.rep-inverse* **by** (*simp add*: *lookup.rep-eq*)+
moreover
have $(\{a \in reach_t\ \Sigma\ (delta_C\ \Sigma)\ (initial_C\ \varphi).\ M\text{-}fin_C\ \varphi\ \pi\ a \vee (\exists\chi\in Mapping.keys\ \pi.\ Acc\text{-}fin_C\ \Sigma\ \pi\ \chi\ a)\}, \{y.\ \exists x\in Mapping.keys\ \pi.\ y = \{a \in reach_t\ \Sigma\ (delta_C\ \Sigma)\ (initial_C\ \varphi).\ Acc\text{-}inf_C\ \pi\ x\ a\}\})$
$= ((Collect\ (M\text{-}fin_C\ \varphi\ \pi) \cup \{t.\ \exists\chi\in Mapping.keys\ \pi.\ Acc\text{-}fin_C\ \Sigma\ \pi\ \chi\ t\}) \cap reach_t\ \Sigma\ (delta_C\ \Sigma)\ (initial_C\ \varphi), \{y.\ \exists x\in\{Collect\ (Acc\text{-}inf_C\ \pi\ \chi)\ |\chi.\ \chi \in Mapping.keys\ \pi\}.\ y = x \cap reach_t\ \Sigma\ (delta_C\ \Sigma)\ (initial_C\ \varphi)\})$
by *auto*
hence $accepting\text{-}pair_{GR}\ (delta\ \Sigma)\ (initial\ \varphi)\ (M\text{-}fin\ \pi' \cup \bigcup\{Acc\text{-}fin\ \Sigma\ \pi'\ \chi\ |\ \chi.\ \chi \in dom\ \pi'\}, \{Acc\text{-}inf\ \pi'\ \chi\ |\ \chi.\ \chi \in dom\ \pi'\})\ w$
unfolding *X*[*OF 1*] **using** *4* **unfolding** *Y Set.filter-def* **unfolding** ‹*dom* π' = *Mapping.keys* π› ‹*Mapping.Mapping* $\pi' = \pi$› *image-def* **by** *simp*

ultimately
show *?lhs*
unfolding *ltl-to-generalized-rabin.simps*
by (*intro* $Rabin.accept_{GR}$*-I*, *blast*; *auto*)
}
qed

end

19.2 Generalized Deterministic Rabin Automaton (af)

definition $M\text{-}fin_C\text{-}af\text{-}lhs$ :: *'a ltl* ⇒ (*'a ltl*, *nat*) *mapping* ⇒ (*'a ltl*, (*'a* ltl_P *list*)) *mapping* ⇒ *'a* ltl_P
where
$M\text{-}fin_C\text{-}af\text{-}lhs\ \varphi\ \pi\ m' \equiv$
let
$\mathcal{G}$ = *Mapping.keys* π;
$\mathcal{G}_L$ = *filter* ($\lambda x.\ x \in \mathcal{G}$) (*G-list* φ);
mk-conj = $\lambda\chi$. *foldl and-abs* (*Abs* χ) (*map* ($\uparrow eval_G\ \mathcal{G}$) (*drop* (*the* (*Mapping.lookup* $\pi\ \chi$)) (*the* (*Mapping.lookup* $m'\ \chi$))))
in
$\uparrow$*And* (*map mk-conj* $\mathcal{G}_L$)

fun $M\text{-}fin_C\text{-}af$ :: *'a ltl* ⇒ (*'a ltl*, *nat*) *mapping* ⇒ (*'a* ltl_P × ((*'a ltl*, (*'a* ltl_P *list*)) *mapping*), *'a set*) *transition* ⇒ *bool*
where

M-fin$_C$*-af* φ π ((φ', m'), -) = *Not* ((*M-fin*$_C$*-af-lhs* φ π m') $\uparrow\longrightarrow_P$ φ')

lemma *M-fin*$_C$*-af-correct*:
assumes $t \in$ *reach*$_t$ Σ (*ltl-to-rabin-base-code-def.delta*$_C$ $\uparrow$*af* $\uparrow$*af*$_G$ *Abs* Σ) (*ltl-to-rabin-base-code-def.initial*$_C$ *Abs Abs* φ)
assumes *dom* $\pi \subseteq$ **G** φ
shows *abstract-transition t* $\in$ *M-fin* π = *M-fin*$_C$*-af* φ (*Mapping.Mapping* π) *t*
proof –
let *?delta* = *ltl-to-rabin-base-code-def.delta*$_C$ $\uparrow$*af* $\uparrow$*af*$_G$ *Abs* Σ
let *?initial* = *ltl-to-rabin-base-code-def.initial*$_C$ *Abs Abs* φ

obtain *x y* ν *z* z' **where** *t-def* [*simp*]: *t* = ((*x*, *y*), ν, (*z*, z'))
by (*metis prod.collapse*)
have (*x*, *y*) $\in$ *reach* Σ *?delta ?initial*
using *assms*(*1*) **by** (*simp add*: *reach*$_t$*-def reach-def*; *blast*)
hence *N1*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *Mapping.lookup y* $\chi \neq$ *None*
and *D1*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *distinct* (*the* (*Mapping.lookup y* χ))
using *assms*(*2*) **by** (*blast dest*: *ltl-to-rabin-base-code-def.reach-delta-initial*)+

{
fix *S*
let *?m'* = $\lambda\chi$. *map-option rk* (*Mapping.lookup y* χ)

{
fix χ
assume $\chi \in$ *dom* π
hence *S* $\uparrow\models_P$ (*foldl and-abs* (*Abs* χ) (*map* ($\uparrow$*eval*$_G$ (*dom* π)) (*drop* (*the* (π χ)) (*the* (*Mapping.lookup y* χ)))))
$\longleftrightarrow$ *S* $\uparrow\models_P$ (*Abs* χ) $\wedge$ ($\forall$ *q*. ($\exists$ *j* $\geq$ *the* (π χ). *the* (*?m'* χ) *q* = *Some j*) $\longrightarrow$ *S* $\uparrow\models_P$ $\uparrow$*eval*$_G$ (*dom* π) *q*)
using *D1*[*THEN drop-rk*, *of - the* (π χ)] *N1*[*THEN option.map-sel*, *of - rk*]
by (*auto simp add*: *foldl-LTLAnd-prop-entailment-abs*)
}

hence *S* $\uparrow\models_P$ (*M-fin*$_C$*-af-lhs* φ (*Mapping.Mapping* π) *y*)
$\longleftrightarrow$ ($\forall\chi \in$ *dom* π. *S* $\uparrow\models_P$ (*Abs* χ) $\wedge$ ($\forall$ *q*. ($\exists$ *j* $\geq$ *the* (π χ). *the* (*?m'* χ) *q* = *Some j*) $\longrightarrow$ *S* $\uparrow\models_P$ $\uparrow$*eval*$_G$ (*dom* π) *q*))
unfolding *M-fin*$_C$*-af-lhs-def Let-def And-prop-entailment-abs set-map Ball-def keys.abs-eq lookup.abs-eq*
using *assms*(*2*) **by** (*simp add*: *image-def inter-set-filter*[*symmetric*] *G-eq-G-list*[*symmetric*]; *blast*)
}

thus *?thesis*
by (*simp add*: *ltl-prop-implies-def ltl-prop-implies-abs-def ltl-prop-entails-abs-def*)
qed

definition
ltl-to-generalized-rabin$_C$*-af* ≡ *ltl-to-rabin-base-code-def.ltl-to-generalized-rabin*$_C$
↑*af* ↑*af*$_G$ *Abs Abs M-fin*$_C$*-af*

theorem *ltl-to-generalized-rabin*$_C$*-af-correct*:
assumes *range w* ⊆ *set* Σ
shows *w* ⊨ φ ⟷ *accept*$_{GR}$*-LTS* (*ltl-to-generalized-rabin*$_C$*-af* Σ φ) *w*
(**is** *?lhs* ⟷ *?rhs*)
proof −
have *X*: *ltl-to-rabin-base-code* ↑*af* ↑*af*$_G$ *Abs Abs M-fin* (*set* Σ) *w M-fin*$_C$*-af*
using *ltl-to-generalized-rabin-af-wellformed*[*OF finite-set assms*] *M-fin*$_C$*-af-correct assms*
unfolding *ltl-to-rabin-af-def ltl-to-rabin-base-code-def ltl-to-rabin-base-code-axioms-def* **by** *blast*
have *?lhs* ⟷ *accept*$_{GR}$ (*ltl-to-generalized-rabin-af* (*set* Σ) φ) *w*
using *assms ltl-to-generalized-rabin-af-correct* **by** *auto*
also
have ... ⟷ *?rhs*
using *ltl-to-rabin-base-code.ltl-to-generalized-rabin*$_C$*-correct*[*OF X*]
unfolding *ltl-to-generalized-rabin*$_C$*-af-def* **by** *simp*
finally
show *?thesis* **.**
qed

19.3 Generalized Deterministic Rabin Automaton (eager af)

definition *M-fin*$_C$*-af*$_{\mathfrak{U}}$*-lhs* :: *'a ltl* ⇒ (*'a ltl*, *nat*) *mapping* ⇒ (*'a ltl*, (*'a ltl*$_P$ *list*)) *mapping* ⇒ *'a set* ⇒ *'a ltl*$_P$
where
M-fin$_C$*-af*$_{\mathfrak{U}}$*-lhs* φ π *m'* ν ≡
let
𝒢 = *Mapping.keys* π;
𝒢$_L$ = *filter* (λ*x*. *x* ∈ 𝒢) (*G-list* φ);
mk-conj = λχ. *foldl and-abs* (*and-abs* (*Abs* χ) (↑*eval*$_G$ 𝒢 (*Abs* (*theG* χ)))) (*map* (↑*eval*$_G$ 𝒢 *o* (λ*q*. ↑*step q* ν)) (*drop* (*the* (*Mapping.lookup* π χ)) (*the* (*Mapping.lookup m'* χ))))
in
↑*And* (*map mk-conj* 𝒢$_L$)

fun *M-fin*$_C$*-af*$_{\mathfrak{U}}$ :: *'a ltl* ⇒ (*'a ltl*, *nat*) *mapping* ⇒ (*'a ltl*$_P$ × ((*'a ltl*, (*'a*

ltl$_P$ *list*)) *mapping*), *'a set*) *transition* ⇒ *bool*
where
M-fin$_C$*-af*$_{\mathfrak{U}}$ φ π ((φ', m'), ν, -) = *Not* ((*M-fin*$_C$*-af*$_{\mathfrak{U}}$*-lhs* φ π m' ν) $\uparrow\!\longrightarrow_P$ ($\uparrow$*step* φ' ν))

lemma *M-fin*$_C$*-af*$_{\mathfrak{U}}$*-correct*:
assumes $t \in$ *reach*$_t$ Σ (*ltl-to-rabin-base-code-def.delta*$_C$ $\uparrow$*af*$_{\mathfrak{U}}$ $\uparrow$*af*$_{G\mathfrak{U}}$ (*Abs* $\circ$ *Unf*$_G$) Σ) (*ltl-to-rabin-base-code-def.initial*$_C$ (*Abs* $\circ$ *Unf*) (*Abs* $\circ$ *Unf*$_G$) φ)
assumes *dom* $\pi \subseteq$ **G** φ
shows *abstract-transition* $t \in M_{\mathfrak{U}}$*-fin* π = *M-fin*$_C$*-af*$_{\mathfrak{U}}$ φ (*Mapping.Mapping* π) t
proof −
let *?delta* = *ltl-to-rabin-base-code-def.delta*$_C$ $\uparrow$*af*$_{\mathfrak{U}}$ $\uparrow$*af*$_{G\mathfrak{U}}$ (*Abs* $\circ$ *Unf*$_G$) Σ
let *?initial* = *ltl-to-rabin-base-code-def.initial*$_C$ (*Abs* $\circ$ *Unf*) (*Abs* $\circ$ *Unf*$_G$) φ

obtain x y ν z z' **where** *t-def* [*simp*]: t = ((x, y), ν, (z, z'))
by (*metis prod.collapse*)
have (x, y) $\in$ *reach* Σ *?delta ?initial*
using *assms*(*1*) **by** (*simp add*: *reach*$_t$*-def reach-def*; *blast*)
hence *N1*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *Mapping.lookup* y $\chi \neq$ *None*
and *D1*: $\bigwedge\chi$. $\chi \in$ *dom* $\pi \Longrightarrow$ *distinct* (*the* (*Mapping.lookup* y χ))
using *assms*(*2*) **by** (*blast dest*: *ltl-to-rabin-base-code-def.reach-delta-initial*)+

{
fix S
let *?m'* = $\lambda\chi$. *map-option rk* (*Mapping.lookup* y χ)

{
fix χ
assume $\chi \in$ *dom* π
hence S $\uparrow\models_P$ (*foldl and-abs* (*and-abs* (*Abs* χ) ($\uparrow$*eval*$_G$ (*dom* π) (*Abs* (*theG* χ)))) (*map* ($\uparrow$*eval*$_G$ (*dom* π) *o* (λq. $\uparrow$*step* q ν)) (*drop* (*the* (π χ)) (*the* (*Mapping.lookup* y χ)))))
$\longleftrightarrow$ S $\uparrow\models_P$ *Abs* χ $\wedge$ S $\uparrow\models_P$ $\uparrow$*eval*$_G$ (*dom* π) (*Abs* (*theG* χ)) $\wedge$ ($\forall q$. ($\exists j \geq$ *the* (π χ). *the* (*?m'* χ) q = *Some* j) $\longrightarrow$ S $\uparrow\models_P$ $\uparrow$*eval*$_G$ (*dom* π) ($\uparrow$*step* q ν))
using *D1*[*THEN drop-rk*, *of - the* (π χ)] *N1*[*THEN option.map-sel*, *of - rk*]
by (*auto simp add*: *foldl-LTLAnd-prop-entailment-abs and-abs-conjunction simp del*: *rk.simps*)
}

hence $S \uparrow\models_P$ (*M-fin*$_C$-*af*$_{\mathfrak{U}}$-*lhs* φ (*Mapping.Mapping* π) *y* ν)
$\longleftrightarrow$ (($\forall\chi \in$ *dom* π. ($S \uparrow\models_P$ *Abs* $\chi \wedge S \uparrow\models_P \uparrow eval_G$ (*dom* π) (*Abs* (*theG* χ)) $\wedge$ ($\forall q$. ($\exists j \geq$ *the* (π χ). *the* (*?m'* χ) *q* = *Some j*) $\longrightarrow S \uparrow\models_P$ $\uparrow eval_G$ (*dom* π) ($\uparrow$*step q* ν)))))
unfolding *M-fin*$_C$-*af*$_{\mathfrak{U}}$-*lhs-def Let-def And-prop-entailment-abs set-map Ball-def keys.abs-eq lookup.abs-eq*
using *assms*(*2*) **by** (*simp add*: *image-def inter-set-filter*[*symmetric*] *G-eq-G-list*[*symmetric*]; *blast*)
}
thus *?thesis*
by (*simp add*: *ltl-prop-implies-def ltl-prop-implies-abs-def ltl-prop-entails-abs-def*)
qed

definition
ltl-to-generalized-rabin$_C$-*af*$_{\mathfrak{U}}$ $\equiv$ *ltl-to-rabin-base-code-def.ltl-to-generalized-rabin*$_C$ $\uparrow$*af*$_{\mathfrak{U}}$ $\uparrow$*af*$_{G\mathfrak{U}}$ (*Abs* $\circ$ *Unf*) (*Abs* $\circ$ *Unf*$_G$) *M-fin*$_C$-*af*$_{\mathfrak{U}}$

theorem *ltl-to-generalized-rabin*$_C$-*af*$_{\mathfrak{U}}$-*correct*:
assumes *range w* $\subseteq$ *set* Σ
shows $w \models \varphi \longleftrightarrow$ *accept*$_{GR}$-*LTS* (*ltl-to-generalized-rabin*$_C$-*af*$_{\mathfrak{U}}$ Σ φ) *w*
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof –
have *X*: *ltl-to-rabin-base-code* $\uparrow$*af*$_{\mathfrak{U}}$ $\uparrow$*af*$_{G\mathfrak{U}}$ (*Abs* $\circ$ *Unf*) (*Abs* $\circ$ *Unf*$_G$) *M*$_{\mathfrak{U}}$-*fin* (*set* Σ) *w M-fin*$_C$-*af*$_{\mathfrak{U}}$
using *ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$-*wellformed*[*OF finite-set assms*] *M-fin*$_C$-*af*$_{\mathfrak{U}}$-*correct assms*
unfolding *ltl-to-rabin-af-unf-def ltl-to-rabin-base-code-def ltl-to-rabin-base-code-axioms-def* **by** *blast*
have *?lhs* $\longleftrightarrow$ *accept*$_{GR}$ (*ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$ (*set* Σ) φ) *w*
using *assms ltl-to-generalized-rabin-af*$_{\mathfrak{U}}$-*correct* **by** *auto*
also
have ... $\longleftrightarrow$ *?rhs*
using *ltl-to-rabin-base-code.ltl-to-generalized-rabin*$_C$-*correct*[*OF X*]
unfolding *ltl-to-generalized-rabin*$_C$-*af*$_{\mathfrak{U}}$-*def* **by** *simp*
finally
show *?thesis* **.**
qed

end

20 Code Generation

theory *Export-Code*

imports *Main LTL-Compat LTL-Rabin-Impl*
HOL−Library.AList-Mapping
LTL.Rewriting
HOL−Library.Code-Target-Numeral
begin

20.1 External Interface

definition
ltlc-to-rabin eager mode (φ_c :: *String.literal ltlc*) ≡
(*let*
φ_n = *ltlc-to-ltln* φ_c;
Σ = *map set* (*subseqs* (*atoms-list* φ_n));
φ = *ltln-to-ltl* (*simplify mode* φ_n)
in
(*if eager then ltl-to-generalized-rabin*$_C$*-af*$_{\mathfrak{U}}$ Σ φ *else ltl-to-generalized-rabin*$_C$*-af*
Σ φ))

theorem *ltlc-to-rabin-exec-correct*:
assumes *range w* ⊆ *Pow* (*atoms-ltlc* φ_c)
shows $w \models_c \varphi_c \longleftrightarrow$ *accept*$_{GR}$*-LTS* (*ltlc-to-rabin eager mode* φ_c) *w*
(**is** *?lhs* = *?rhs*)
proof −
let *?*φ_n = *ltlc-to-ltln* φ_c
let *?*Σ = *map set* (*subseqs* (*atoms-list ?*φ_n))
let *?*φ = *ltln-to-ltl* (*simplify mode ?*φ_n)

have *set ?*Σ = *Pow* (*atoms-ltln ?*φ_n)
unfolding *atoms-list-correct*[*symmetric*] *subseqs-powset*[*symmetric*] *list.set-map*
..
hence *R*: *range w* ⊆ *set ?*Σ
using *assms ltlc-to-ltln-atoms*[*symmetric*] **by** *metis*

have $w \models_c \varphi_c \longleftrightarrow w \models$ *?*φ
by (*simp only*: *ltlc-to-ltln-semantics simplify-correct ltln-to-ltl-semantics*)
also
have ... ⟷ *?rhs*
using *ltl-to-generalized-rabin*$_C$*-af*$_{\mathfrak{U}}$*-correct*[*OF R*] *ltl-to-generalized-rabin*$_C$*-af-correct*[*OF*
R]
unfolding *ltlc-to-rabin-def Let-def* **by** *auto*
finally
show *?thesis*
by *simp*
qed

20.2 Normalize Equivalence Classes During DFS-Search

fun *norm-rep*
where
  norm-rep $(i, (q, \nu, p))\ (q', \nu', p') = ($
    let
      eq-q $= (q = q')$; *eq-p* $= (p = p')$;
      $q'' =$ *if eq-q then* q' *else if* $q = p'$ *then* p' *else* q;
      $p'' =$ *if eq-p then* p' *else if* $p = q'$ *then* q' *else* p
    in
      $(i \mid ($*eq-q* & *eq-p* & $\nu = \nu'), q'', \nu, p''))$

fun *norm-fold* :: $('a, 'b)$ *transition* $\Rightarrow$ $('a, 'b)$ *transition list* $\Rightarrow$ $(bool * 'a * 'b * 'a)$
where
  norm-fold (q, ν, p) *xs* = *foldl-break norm-rep fst* $(False, q, \nu,$ *if* $q = p$ *then* q *else* $p)$ *xs*

definition *norm-insert* :: $('a, 'b)$ *transition* $\Rightarrow$ $('a, 'b)$ *transition list* $\Rightarrow$ $(bool * ('a, 'b)$ *transition list*$)$
where
  norm-insert x xs $\equiv$ *let* $(i, x') =$ *norm-fold x xs in if i then* (i, xs) *else* $(i, x' \# xs)$

lemma *norm-fold*:
  norm-fold (q, ν, p) *xs* $= ((q, \nu, p) \in$ *set xs*$, q, \nu, p)$
proof (*induction xs rule*: *rev-induct*)
  case (*snoc x xs*)
    obtain q' ν' p' **where** *x-def*: $x = (q', \nu', p')$
      by (*blast intro*: *prod-cases3*)
    show *?case*
      using *snoc* **by** (*auto simp add*: *x-def foldl-break-append*)
qed *simp*

lemma *norm-insert*:
  norm-insert x xs $= (x \in$ *set xs*, *List.insert x xs*$)$
proof −
  obtain q ν p **where** *x-def*: $x = (q, \nu, p)$
    by (*blast intro*: *prod-cases3*)
  show *?thesis*
    unfolding *x-def norm-insert-def norm-fold* **by** *simp*
qed

declare *list-dfs-def* [*code del*]

declare *norm-insert-def* [*code-unfold*]

lemma *list-dfs-norm-insert* [*code*]:
list-dfs succ S [] = *S*
list-dfs succ S (*x* # *xs*) = (*let* (*memb*, *S'*) = *norm-insert x S in list-dfs succ S'* (*if memb then xs else succ x* @ *xs*))
unfolding *list-dfs-def Let-def norm-insert* **by** *simp+*

20.3 Register Code Equations

lemma [*code*]:
$\uparrow\Delta_\times$ *f* (*AList-Mapping.Mapping xs*) *c* = *AList-Mapping.Mapping* (*map-ran* (λ*a b. f a b c*) *xs*)
proof −
have $\bigwedge$*x.* ($\Delta_\times$ *f* (*map-of xs*) *c*) *x* = (*map-of* (*map* (λ(*k*, *v*). (*k*, *f k v c*)) *xs*)) *x*
by (*induction xs*) *auto*
thus *?thesis*
by (*transfer*; *simp add*: *map-ran-def*)
qed

lemmas *ltl-to-rabin-base-code-export* [*code*, *code-unfold*] =
ltl-to-rabin-base-code-def.ltl-to-generalized-rabin$_C$*.simps*
ltl-to-rabin-base-code-def.reachable-transitions$_C$*-def*
ltl-to-rabin-base-code-def.mappings$_C$*-code*
ltl-to-rabin-base-code-def.delta$_C$*.simps*
ltl-to-rabin-base-code-def.initial$_C$*.simps*
ltl-to-rabin-base-code-def.Acc-inf$_C$*.simps*
ltl-to-rabin-base-code-def.Acc-fin$_C$*.simps*
ltl-to-rabin-base-code-def.max-rank-of$_C$*-def*

lemmas *M-fin*$_C$*-lhs* [*code del*, *code-unfold*] =
M-fin$_C$*-af*$_{\mathfrak{U}}$*-lhs-def M-fin*$_C$*-af-lhs-def*

— Test code export
export-code *true*$_c$ *Iff-ltlc Nop true Abs AList-Mapping.Mapping set ltlc-to-rabin*
checking

— Export translator (and also constructors)
export-code *true*$_c$ *Iff-ltlc Nop true Abs AList-Mapping.Mapping set ltlc-to-rabin*

in *SML* **module-name** *LTL* **file** ‹*../Code/LTL-to-DRA-Translator.sml*›

end

References

[1] J. Esparza, J. Kretínský, and S. Sickert. From LTL to deterministic automata - A safraless compositional approach. *Formal Methods in System Design*, 49(3):219–271, 2016.