

# Pushdown Systems

Anders Schlichtkrull, Morten Konggaard Schou, Jiří Srba and Dmitriy Traytel

## Abstract

We formalize pushdown systems and the correctness of the pushdown reachability algorithms  $\text{post}^*$  (forward search),  $\text{pre}^*$  (backward search) and  $\text{dual}^*$  (bi-directional search). For  $\text{pre}^*$  we refine the algorithm to an executable version for which one can generate code using Isabelle’s code generator. For  $\text{pre}^*$  and  $\text{post}^*$  we follow Stefan Schwoon’s PhD thesis [Sch02a]. The  $\text{dual}^*$  algorithm is from a paper by Jensen et. al presented at ATVA2021 [JSS<sup>+</sup>21]. The formalization is described in our FMCAD2022 paper [SSST22] in which we also document how we have used it to do differential testing against a C++ implementation of pushdown reachability called PDAAAL. Lammich et al. [Lam09, LMW09] formalized the  $\text{pre}^*$  algorithm for dynamic pushdown networks (DPN) which is a generalization of pushdown systems. Our work is independent from that because the  $\text{post}^*$  of DPNs is not regular and additionally the DPN formalization does not support epsilon transitions which we use for  $\text{post}^*$  and  $\text{dual}^*$ .

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Automata</b>	<b>3</b>
2.1	P-Automaton locale . . . . .	3
2.2	Intersection P-Automaton locale . . . . .	4
<b>3</b>	<b>Automata with epsilon</b>	<b>7</b>
3.1	P-Automaton with epsilon locale . . . . .	7
3.2	Intersection P-Automaton with epsilon locale . . . . .	7
<b>4</b>	<b>PDS</b>	<b>14</b>
<b>5</b>	<b>PDS with P automata</b>	<b>14</b>
5.1	Saturations . . . . .	17
5.2	Saturation rules . . . . .	18
5.3	Pre* lemmas . . . . .	22
5.4	Post* lemmas . . . . .	28
5.5	Intersection Automata . . . . .	47
5.6	Intersection epsilon-Automata . . . . .	48
5.7	Dual search . . . . .	54

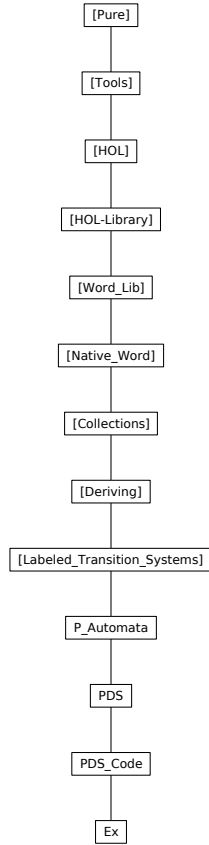


Figure 1: Theory dependency graph

## 1 Introduction

Pushdown reachability was studied by Büchi in 1964 [Büc64] and has been used for, among other things, interprocedural control-flow analysis of recursive programs [EK99, CNDE05], model checking [ES01, Sch02b, SSE05, BEM97] and communication network analysis [JKM<sup>+</sup>18, JKS<sup>+</sup>20, vDJJ<sup>+</sup>21]. In this formalization we formalize the  $\text{pre}^*$  and  $\text{post}^*$  algorithms [Sch02a] and the  $\text{dual}^*$  algorithm [JSS<sup>+</sup>21]. For  $\text{pre}^*$  we have also an executable version. In our FMCAD2022 paper [SSST22] we describe the formalization and use it to do differential testing against a C++ implementation of pushdown reachability called PDAAAL [JSS<sup>+</sup>21]. The differential testing revealed a number of bugs in PDAAAL that we were then able to fix.

theory *P\_Automata* imports *Labeled\_Transition\_Systems.LTS* begin

## 2 Automata

### 2.1 P-Automaton locale

locale *P\_Automaton* = *LTS transition\_relation*  
 for *transition\_relation* :: "('state::finite, 'label) transition set" +  
 fixes *Init* :: "'ctr\_loc::enum  $\Rightarrow$  'state"  
 and *finals* :: "'state set"  
 begin

definition *initials* :: "'state set" where  
 "*initials*  $\equiv$  *Init* 'UNIV"

lemma *initials\_list*:  
 "*initials* = set (map *Init Enum.enum*)"  
 using *enum\_UNIV unfolding initials\_def* by force

definition *accepts\_out* :: "'ctr\_loc  $\Rightarrow$  'label list  $\Rightarrow$  bool" where  
 "*accepts\_out*  $\equiv$   $\lambda p w. (\exists q \in \text{finals}. (Init\ p, w, q) \in \text{trans\_star})$ "

definition *lang\_out* :: "('ctr\_loc \* 'label list) set" where  
 "*lang\_out* = {(*p,w*). *accepts\_out* *p w*}"

definition *nonempty* where  
 "*nonempty*  $\longleftrightarrow$  *lang\_out*  $\neq$  {}"

lemma *nonempty\_alt*:  
 "*nonempty*  $\longleftrightarrow$  ( $\exists p. \exists q \in \text{finals}. \exists w. (Init\ p, w, q) \in \text{trans\_star}$ )"  
 unfolding *lang\_out\_def nonempty\_def accepts\_out\_def* by auto

typedef 'a *mark\_state* = "{(Q :: 'a set, I). I  $\subseteq$  Q}"  
 by auto

setup-lifting *type\_definition\_mark\_state*

lift-definition *get\_visited* :: "'a *mark\_state*  $\Rightarrow$  'a set" is *fst* .

lift-definition *get\_next* :: "'a *mark\_state*  $\Rightarrow$  'a set" is *snd* .

lift-definition *make\_mark\_state* :: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a *mark\_state*" is " $\lambda Q J. (Q \cup J, J)$ " by auto

lemma *get\_next\_get\_visited*: "*get\_next* *ms*  $\subseteq$  *get\_visited* *ms*"

by transfer auto

lemma *get\_next\_set\_next[simp]*: "*get\_next* (*make\_mark\_state* *Q J*) = *J*"

by transfer auto

lemma *get\_visited\_set\_next[simp]*: "*get\_visited* (*make\_mark\_state* *Q J*) = *Q*  $\cup$  *J*"

by transfer auto

function *mark* where

"*mark* *ms*  $\longleftrightarrow$

(let *Q* = *get\_visited* *ms*; *I* = *get\_next* *ms* in

if *I*  $\cap$  *finals*  $\neq$  {} then True

else let *J* = ( $\bigcup (q,w,q') \in \text{transition\_relation}. \text{if } q \in I \wedge q' \notin Q \text{ then } \{q'\} \text{ else } \{\}$ ) in

if *J* = {} then False else *mark* (*make\_mark\_state* *Q J*)"

by auto

termination by (relation "*measure* ( $\lambda ms. \text{card } (UNIV :: 'a \text{ set}) - \text{card } (get\_visited\ ms :: 'a \text{ set}))$ ")

(*fastforce intro!*: *diff\_less\_mono2 psubset\_card\_mono split: if\_splits*)+

declare *mark.simps[simp del]*

lemma *trapped\_transitions*: " $(p, w, q) \in \text{trans\_star} \implies$   
 $\forall p \in Q. (\forall \gamma q. (p, \gamma, q) \in \text{transition\_relation} \longrightarrow q \in Q) \implies$   
 $p \in Q \implies q \in Q$ "  
 by (*induct* *p w q* rule: *trans\_star.induct*) auto

lemma *mark\_complete*: " $(p, w, q) \in \text{trans\_star} \implies (get\_visited\ ms - get\_next\ ms) \cap \text{finals} = \{\} \implies$

```

 $\forall p \in \text{get\_visited } ms - \text{get\_next } ms. \forall q \gamma. (p, \gamma, q) \in \text{transition\_relation} \longrightarrow q \in \text{get\_visited } ms \implies$ 
 $p \in \text{get\_visited } ms \implies q \in \text{finals} \implies \text{mark } ms$ 
proof (induct p w q arbitrary: ms rule: trans_star.induct)
  case (trans_star_refl p)
  then show ?case by (subst mark.simps) (auto simp: Let_def)
next
  case step: (trans_star_step p  $\gamma$  q' w q)
  define J where “ $J \equiv \bigcup (q, w, q') \in \text{transition\_relation}. \text{if } q \in \text{get\_next } ms \wedge q' \notin \text{get\_visited } ms \text{ then } \{q'\} \text{ else } \{\}$ ”
  show ?case
  proof (cases “ $J = \{\}$ ”)
    case True
    then have “ $q' \in \text{get\_visited } ms$ ”
    by (smt (z3) DiffI Diff_disjoint Int_iff J_def SUP_bot_conv(2) case_prod_conv insertI1
      step.hyps(1) step.prem(2) step.prem(3))
    with True show ?thesis
    using step(1,2,4,5,7)
    by (subst mark.simps)
      (auto 10 0 intro!: step(3) elim!: set_mp[of _ “get_next ms”] simp: split_beta J_def
        dest: trapped_transitions[of q' w q “get_visited ms”])
    next
    case False
    then have [simp]: “ $\text{get\_visited } ms \cup J - J = \text{get\_visited } ms$ ”
    by (auto simp: J_def split: if_splits)
    then have “ $p \in \text{get\_visited } ms \implies (p, \gamma, q) \in \text{transition\_relation} \implies q \notin \text{get\_visited } ms \implies q \in J$ ” for p  $\gamma$  q
    using step(5)
    by (cases “ $p \in \text{get\_next } ms$ ”)
      (auto simp only: J_def simp_thms if_True if_False intro!: UN_I[of “(p,  $\gamma$ , q)”])
    with False show ?thesis
    using step(1,4,5,6,7)
    by (subst mark.simps)
      (auto 0 2 simp add: Let_def J_def[symmetric] disj_commute
        intro!: step(3)[of “make_mark_state (get_visited ms) J”])
  qed
qed

```

**lemma** mark\_sound: “ $\text{mark } ms \implies (\exists p \in \text{get\_next } ms. \exists q \in \text{finals}. \exists w. (p, w, q) \in \text{trans\_star})$ ”

**by** (induct ms rule: mark.induct)  
(subst (asm) (2) mark.simps, fastforce dest: trans\_star\_step simp: Let\_def split: if\_splits)

**lemma** nonempty\_code[code]: “ $\text{nonempty} = \text{mark } (\text{make\_mark\_state } \{\} (\text{set } (\text{map } \text{Init } \text{Enum.enum})))$ ”

**using** mark\_complete[of \_ \_ \_ “make\_mark\_state { } initials”]  
mark\_sound[of “make\_mark\_state { } initials”] nonempty\_alt  
**unfolding** initials\_def initials\_list[symmetric] **by** auto

**end**

## 2.2 Intersection P-Automaton locale

```

locale Intersection_P_Automaton =
  A1: P_Automaton ts1 Init finals1 +
  A2: P_Automaton ts2 Init finals2
  for ts1 :: “('state :: finite, 'label) transition set”
  and Init :: “'ctr_loc :: enum  $\Rightarrow$  'state”
  and finals1 :: “'state set”
  and ts2 :: “('state, 'label) transition set”
  and finals2 :: “'state set”
begin
sublocale pa: P_Automaton “inters ts1 ts2” “( $\lambda p. (\text{Init } p, \text{Init } p)$ )” “inters_finals finals1 finals2”

```

**definition** *accepts\_aut\_inters* **where**  
 “*accepts\_aut\_inters*  $p\ w = pa.accepts\_aut\ p\ w$ ”

**definition** *lang\_aut\_inters* :: “(*ctr\_loc* \* *label list*) *set*” **where**  
 “*lang\_aut\_inters* = { $(p,w).$  *accepts\_aut\_inters*  $p\ w$ }”

**lemma** *trans\_star\_inter*:

**assumes** “ $(p1, w, p2) \in A1.trans\_star$ ”  
**assumes** “ $(q1, w, q2) \in A2.trans\_star$ ”  
**shows** “ $((p1,q1), w :: 'label\ list, (p2,q2)) \in pa.trans\_star$ ”  
**using** *assms*

**proof** (*induction*  $w$  *arbitrary*:  $p1\ q1$ )

**case** (*Cons*  $\alpha\ w1'$ )  
**obtain**  $p'$  **where**  $p'_p$ : “ $(p1, \alpha, p') \in ts1 \wedge (p', w1', p2) \in A1.trans\_star$ ”  
**using** *Cons* **by** (*metis* *LTS.trans\_star\_cons*)  
**obtain**  $q'$  **where**  $q'_p$ : “ $(q1, \alpha, q') \in ts2 \wedge (q', w1', q2) \in A2.trans\_star$ ”  
**using** *Cons* **by** (*metis* *LTS.trans\_star\_cons*)  
**have** *ind*: “ $((p', q'), w1', p2, q2) \in pa.trans\_star$ ”  
**proof** –  
**have** “*Suc* (*length*  $w1'$ ) = *length* ( $\alpha\#\ w1'$ )”  
**by** *auto*  
**moreover**  
**have** “ $(p', w1', p2) \in A1.trans\_star$ ”  
**using**  $p'_p$  **by** *simp*  
**moreover**  
**have** “ $(q', w1', q2) \in A2.trans\_star$ ”  
**using**  $q'_p$  **by** *simp*  
**ultimately**  
**show** “ $((p', q'), w1', p2, q2) \in pa.trans\_star$ ”  
**using** *Cons*(1) **by** *auto*

**qed**

**moreover**  
**have** “ $((p1, q1), \alpha, (p', q')) \in (inters\ ts1\ ts2)$ ”  
**by** (*simp* *add*: *inters\_def*  $p'_p\ q'_p$ )

**ultimately**  
**have** “ $((p1, q1), \alpha\#\ w1', p2, q2) \in pa.trans\_star$ ”  
**by** (*meson* *LTS.trans\_star.trans\_star\_step*)

**moreover**  
**have** “*length* ( $(\alpha\#\ w1')$ ) > 0”  
**by** *auto*

**moreover**  
**have** “*hd* ( $(\alpha\#\ w1')$ ) =  $\alpha$ ”  
**by** *auto*

**ultimately**  
**show** *?case*  
**by** *force*

**next**

**case** *Nil*  
**then show** *?case*  
**by** (*metis* *LTS.trans\_star.trans\_star\_refl* *LTS.trans\_star\_empty*)

**qed**

**lemma** *inters\_trans\_star1*:

**assumes** “ $(p1q2, w :: 'label\ list, p2q2) \in pa.trans\_star$ ”  
**shows** “ $(fst\ p1q2, w, fst\ p2q2) \in A1.trans\_star$ ”  
**using** *assms*

**proof** (*induction* *rule*: *LTS.trans\_star.induct*[*OF* *assms*(1)])

**case** (1  $p$ )  
**then show** *?case*  
**by** (*simp* *add*: *LTS.trans\_star.trans\_star\_refl*)

**next**

**case** (2  $p\ \gamma\ q'\ w\ q$ )  
**then have** *ind*: “ $(fst\ q', w, fst\ q) \in A1.trans\_star$ ”

**by auto**  
**from**  $2(1)$  **have** “ $(p, \gamma, q') \in \{(p1, q1), \alpha, p2, q2 \mid p1\ q1\ \alpha\ p2\ q2. (p1, \alpha, p2) \in ts1 \wedge (q1, \alpha, q2) \in ts2\}$ ”  
**unfolding inters\_def** **by auto**  
**then have** “ $\exists p1\ q1. p = (p1, q1) \wedge (\exists p2\ q2. q' = (p2, q2) \wedge (p1, \gamma, p2) \in ts1 \wedge (q1, \gamma, q2) \in ts2)$ ”  
**by simp**  
**then obtain**  $p1\ q1$  **where** “ $p = (p1, q1) \wedge (\exists p2\ q2. q' = (p2, q2) \wedge (p1, \gamma, p2) \in ts1 \wedge (q1, \gamma, q2) \in ts2)$ ”  
**by auto**  
**then show** *?case*  
**using** *LTS.trans\_star.trans\_star\_step ind* **by fastforce**  
**qed**

**lemma inters\_trans\_star:**  
**assumes** “ $(p1q2, w :: 'label\ list, p2q2) \in pa.trans\_star$ ”  
**shows** “ $(snd\ p1q2, w, snd\ p2q2) \in A2.trans\_star$ ”  
**using** *assms*  
**proof** (*induction rule: LTS.trans\_star.induct[OF assms(1)]*)  
**case**  $(1\ p)$   
**then show** *?case*  
**by** (*simp add: LTS.trans\_star.trans\_star\_refl*)  
**next**

**case**  $(2\ p\ \gamma\ q'\ w\ q)$   
**then have** *ind:* “ $(snd\ q', w, snd\ q) \in A2.trans\_star$ ”  
**by auto**  
**from**  $2(1)$  **have** “ $(p, \gamma, q') \in \{(p1, q1), \alpha, p2, q2 \mid p1\ q1\ \alpha\ p2\ q2. (p1, \alpha, p2) \in ts1 \wedge (q1, \alpha, q2) \in ts2\}$ ”  
**unfolding inters\_def** **by auto**  
**then have** “ $\exists p1\ q1. p = (p1, q1) \wedge (\exists p2\ q2. q' = (p2, q2) \wedge (p1, \gamma, p2) \in ts1 \wedge (q1, \gamma, q2) \in ts2)$ ”  
**by simp**  
**then obtain**  $p1\ q1$  **where** “ $p = (p1, q1) \wedge (\exists p2\ q2. q' = (p2, q2) \wedge (p1, \gamma, p2) \in ts1 \wedge (q1, \gamma, q2) \in ts2)$ ”  
**by auto**  
**then show** *?case*  
**using** *LTS.trans\_star.trans\_star\_step ind* **by fastforce**  
**qed**

**lemma inters\_trans\_star\_iff:**  
“ $((p1, q2), w :: 'label\ list, (p2, q2)) \in pa.trans\_star \iff (p1, w, p2) \in A1.trans\_star \wedge (q2, w, q2) \in A2.trans\_star$ ”  
**by** (*metis fst\_conv inters\_trans\_star inters\_trans\_star1 snd\_conv trans\_star\_inter*)

**lemma inters\_accept\_iff:** “ $accepts\_aut\ inters\ p\ w \iff A1.accepts\_aut\ p\ w \wedge A2.accepts\_aut\ p\ w$ ”  
**proof**

**assume** “ $accepts\_aut\ inters\ p\ w$ ”  
**then show** “ $A1.accepts\_aut\ p\ w \wedge A2.accepts\_aut\ p\ w$ ”  
**unfolding accepts\_aut\_inters\_def** *A1.accepts\_aut\_def* *A2.accepts\_aut\_def* *pa.accepts\_aut\_def*  
**unfolding inters\_finals\_def**  
**using** *inters\_trans\_star\_iff[of \_ \_ w \_]*  
**using** *SigmaE fst\_conv inters\_trans\_star inters\_trans\_star1 snd\_conv*  
**by** (*metis (no\_types, lifting)*)

**next**  
**assume** *a:* “ $A1.accepts\_aut\ p\ w \wedge A2.accepts\_aut\ p\ w$ ”  
**then have** “ $(\exists q \in finals1. (Init\ p, w, q) \in A1.trans\_star) \wedge (\exists q \in finals2. (Init\ p, w, q) \in A2.trans\_star)$ ”  
**unfolding** *A1.accepts\_aut\_def* *A2.accepts\_aut\_def* **by auto**  
**then show** “ $accepts\_aut\ inters\ p\ w$ ”  
**unfolding accepts\_aut\_inters\_def** *pa.accepts\_aut\_def* *inters\_finals\_def*  
**by** (*auto simp: P\_Automaton.accepts\_aut\_def intro: trans\_star\_inter*)  
**qed**

**lemma lang\_aut\_alt:**  
“ $pa.lang\_aut = \{(p, w). (p, w) \in lang\_aut\ inters\}$ ”  
**unfolding** *pa.lang\_aut\_def* *lang\_aut\_inters\_def* *accepts\_aut\_inters\_def* *pa.accepts\_aut\_def*  
**by auto**

**lemma** *inters\_lang*: “*lang\_aut\_inters* = *A1.lang\_aut*  $\cap$  *A2.lang\_aut*”  
**unfolding** *lang\_aut\_inters\_def* *A1.lang\_aut\_def* *A2.lang\_aut\_def* **using** *inters\_accept\_iff* **by** *auto*  
**end**

### 3 Automata with epsilon

#### 3.1 P-Automaton with epsilon locale

**locale** *P\_Automaton\_ε* = *LTS\_ε transition\_relation* **for** *transition\_relation* :: “(*'state::finite*, *'label option*) *transition set*” +  
**fixes** *finals* :: “*'state set*” **and** *Init* :: “*'ctr\_loc* :: *enum*  $\Rightarrow$  *'state*”  
**begin**

**definition** *accepts\_aut\_ε* :: “*'ctr\_loc*  $\Rightarrow$  *'label list*  $\Rightarrow$  *bool*” **where**  
“*accepts\_aut\_ε*  $\equiv$   $\lambda p w. (\exists q \in \text{finals}. (\text{Init } p, w, q) \in \text{trans\_star\_}\epsilon)$ ”

**definition** *lang\_aut\_ε* :: “(*'ctr\_loc* \* *'label list*) *set*” **where**  
“*lang\_aut\_ε* =  $\{(p,w). \text{accepts\_aut\_}\epsilon p w\}$ ”

**definition** *nonempty\_ε* **where**  
“*nonempty\_ε*  $\longleftrightarrow \text{lang\_aut\_}\epsilon \neq \{\}$ ”

**end**

#### 3.2 Intersection P-Automaton with epsilon locale

**locale** *Intersection\_P\_Automaton\_ε* =  
*A1: P\_Automaton\_ε ts1 finals1 Init* +  
*A2: P\_Automaton\_ε ts2 finals2 Init*  
**for** *ts1* :: “(*'state* :: *finite*, *'label option*) *transition set*”  
**and** *finals1* :: “*'state set*”  
**and** *Init* :: “*'ctr\_loc* :: *enum*  $\Rightarrow$  *'state*”  
**and** *ts2* :: “(*'state*, *'label option*) *transition set*”  
**and** *finals2* :: “*'state set*”  
**begin**

**abbreviation**  $\epsilon$  :: “*'label option*” **where**  
“ $\epsilon$  == *None*”

**sublocale** *pa: P\_Automaton\_ε* “*inters\_ε ts1 ts2*” “*inters\_finals finals1 finals2*” “( $\lambda p. (\text{Init } p, \text{Init } p)$ )”  
.

**definition** *accepts\_aut\_inters\_ε* **where**  
“*accepts\_aut\_inters\_ε*  $p w = \text{pa.accepts\_aut\_}\epsilon p w$ ”

**definition** *lang\_aut\_inters\_ε* :: “(*'ctr\_loc* \* *'label list*) *set*” **where**  
“*lang\_aut\_inters\_ε* =  $\{(p,w). \text{accepts\_aut\_inters\_}\epsilon p w\}$ ”

**lemma** *trans\_star\_trans\_star\_ε\_inter*:

**assumes** “*LTS\_ε.ε\_exp* *w1 w*”

**assumes** “*LTS\_ε.ε\_exp* *w2 w*”

**assumes** “(*p1*, *w1*, *p2*)  $\in$  *A1.trans\_star*”

**assumes** “(*q1*, *w2*, *q2*)  $\in$  *A2.trans\_star*”

**shows** “( $(p1,q1), w :: \text{'label list}, (p2,q2)$ )  $\in$  *pa.trans\_star\_ε*”

**using** *assms*

**proof** (*induction* “*length w1* + *length w2*” *arbitrary: w1 w2 w p1 q1 rule: less\_induct*)

**case** *less*

**then show** *?case*

**proof** (*cases* “ $\exists \alpha w1' w2' \beta. w1 = \text{Some } \alpha \# w1' \wedge w2 = \text{Some } \beta \# w2'$ ”)

**case** *True*

**from** *True* **obtain**  $\alpha \beta w1' w2'$  **where** *True''*:

```

    "w1=Some α#w1'"
    "w2=Some β#w2'"
  by auto
have "α = β"
  by (metis True''(1) True''(2) LTS_ε.ε_exp_Some_hd less.prem(1) less.prem(2))
then have True':
  "w1=Some α#w1'"
  "w2=Some α#w2'"
  using True'' by auto
define w' where "w' = tl w"
obtain p' where p'_p: "(p1, Some α, p') ∈ ts1 ∧ (p', w1', p2) ∈ A1.trans_star"
  using less True'(1) by (metis LTS_ε.trans_star_cons_ε)
obtain q' where q'_p: "(q1, Some α, q') ∈ ts2 ∧ (q', w2', q2) ∈ A2.trans_star"
  using less True'(2) by (metis LTS_ε.trans_star_cons_ε)
have ind: "((p', q'), w', p2, q2) ∈ pa.trans_star_ε"
proof -
  have "length w1' + length w2' < length w1 + length w2"
    using True'(1) True'(2) by simp
  moreover
  have "LTS_ε.ε_exp w1' w'"
    by (metis (no_types) LTS_ε.ε_exp_def less(2) True'(1) list.map(2) list.sel(3)
      option.simps(3) removeAll.simps(2) w'_def)
  moreover
  have "LTS_ε.ε_exp w2' w'"
    by (metis (no_types) LTS_ε.ε_exp_def less(3) True'(2) list.map(2) list.sel(3)
      option.simps(3) removeAll.simps(2) w'_def)
  moreover
  have "(p', w1', p2) ∈ A1.trans_star"
    using p'_p by simp
  moreover
  have "(q', w2', q2) ∈ A2.trans_star"
    using q'_p by simp
  ultimately
  show "((p', q'), w', p2, q2) ∈ pa.trans_star_ε"
    using less(1)[of w1' w2' w' p' q'] by auto
qed
moreover
have "((p1, q1), Some α, (p', q')) ∈ (inters_ε ts1 ts2)"
  by (simp add: inters_ε_def p'_p q'_p)
ultimately
have "((p1, q1), α#w', p2, q2) ∈ pa.trans_star_ε"
  by (meson LTS_ε.trans_star_ε.trans_star_ε_step_γ)
moreover
have "length w > 0"
  using less(3) True' LTS_ε.ε_exp_Some_length by metis
moreover
have "hd w = α"
  using less(3) True' LTS_ε.ε_exp_Some_hd by metis
ultimately
show ?thesis
  using w'_def by force
next
case False
note False_outer_outer_outer_outer = False
show ?thesis
proof (cases "w1 = [] ∧ w2 = []")
  case True
  then have same: "p1 = p2 ∧ q1 = q2"
    by (metis LTS.trans_star_empty less.prem(3) less.prem(4))
  have "w = []"
    using True less(2) LTS_ε.exp_empty_empty by auto
  then show ?thesis
    using less True

```



```

    by (simp add: LTS_ε.trans_star_ε.trans_star_ε_refl same)
next
case False
note False_outer_outer_outer = False
show ?thesis
proof (cases "∃ w1'. w1 = ε # w1'")
case True
then obtain w1' where True':
  "w1 = ε # w1'"
  by auto
obtain p' where p'_p: "(p1, ε, p') ∈ ts1 ∧ (p', w1', p2) ∈ A1.trans_star"
using less True'(1) by (metis LTS_ε.trans_star_cons_ε)
have q'_p: "(q1, w2, q2) ∈ A2.trans_star"
using less by metis
have ind: "(p', q1), w, p2, q2) ∈ pa.trans_star_ε"
proof -
  have "length w1' + length w2 < length w1 + length w2"
    using True'(1) by simp
  moreover
  have "LTS_ε.ε_exp w1' w"
    by (metis (no_types) LTS_ε.ε_exp_def less(2) True'(1) removeAll.simps(2))
  moreover
  have "LTS_ε.ε_exp w2 w"
    by (metis (no_types) less(3))
  moreover
  have "(p', w1', p2) ∈ A1.trans_star"
    using p'_p by simp
  moreover
  have "(q1, w2, q2) ∈ A2.trans_star"
    using q'_p by simp
  ultimately
  show "(p', q1), w, p2, q2) ∈ pa.trans_star_ε"
    using less(1)[of w1' w2 w p' q1] by auto
qed
moreover
have "((p1, q1), ε, (p', q1)) ∈ (inters_ε ts1 ts2)"
  by (simp add: inters_ε_def p'_p q'_p)
ultimately
have "((p1, q1), w, p2, q2) ∈ pa.trans_star_ε"
  using LTS_ε.trans_star_ε.simps by fastforce
then
show ?thesis
  by force
next
case False
note False_outer_outer_outer = False
then show ?thesis
proof (cases "∃ w2'. w2 = ε # w2'")
case True
then obtain w2' where True':
  "w2 = ε # w2'"
  by auto
have p'_p: "(p1, w1, p2) ∈ A1.trans_star"
  using less by (metis)
obtain q' where q'_p: "(q1, ε, q') ∈ ts2 ∧ (q', w2', q2) ∈ A2.trans_star"
  using less True'(1) by (metis LTS_ε.trans_star_cons_ε)
have ind: "(p1, q'), w, p2, q2) ∈ pa.trans_star_ε"
proof -
  have "length w1 + length w2' < length w1 + length w2"
    using True'(1) True'(1) by simp
  moreover
  have "LTS_ε.ε_exp w1 w"
    by (metis (no_types) less(2))

```

```

moreover
have “ $LTS_{\varepsilon.\varepsilon\_exp} w2' w$ ”
  by (metis (no_types)  $LTS_{\varepsilon.\varepsilon\_exp\_def}$  less(3)  $True'(1)$   $removeAll.simps(2)$ )
moreover
have “ $(p1, w1, p2) \in A1.trans\_star$ ”
  using  $p'_p$  by simp
moreover
have “ $(q', w2', q2) \in A2.trans\_star$ ”
  using  $q'_p$  by simp
ultimately
show “ $((p1, q'), w, p2, q2) \in pa.trans\_star_{\varepsilon}$ ”
  using less(1)[of  $w1 w2' w p1 q'$ ] by auto
qed
moreover
have “ $((p1, q1), \varepsilon, (p1, q')) \in inters_{\varepsilon} ts1 ts2$ ”
  by (simp add:  $inters_{\varepsilon\_def}$   $p'_p q'_p$ )
ultimately
have “ $((p1, q1), w, p2, q2) \in pa.trans\_star_{\varepsilon}$ ”
  using  $LTS_{\varepsilon.trans\_star_{\varepsilon}.simps}$  by fastforce
then
show ?thesis
  by force
next
case False
then have “ $(w1 = [] \wedge (\exists \alpha w2'. w2 = Some \alpha \# w2')) \vee ((\exists \alpha w1'. w1 = Some \alpha \# w1') \wedge w2 = [])$ ”
  using  $False\_outer\_outer$   $False\_outer\_outer\_outer$   $False\_outer\_outer\_outer\_outer$ 
  by (metis  $neq\_Nil\_conv$   $option.exhaust\_sel$ )
then show ?thesis
  by (metis  $LTS_{\varepsilon.\varepsilon\_exp\_def}$   $LTS_{\varepsilon.\varepsilon\_exp\_Some\_length}$  less.prem(1) less.prem(2)
    less_numer(1)  $list.simps(8)$   $list.size(3)$   $removeAll.simps(1)$ )
qed
qed
qed
qed
qed

```

**lemma**  $trans\_star_{\varepsilon\_inter}$ :

**assumes** “ $(p1, w :: 'label\ list, p2) \in A1.trans\_star_{\varepsilon}$ ”

**assumes** “ $(q1, w, q2) \in A2.trans\_star_{\varepsilon}$ ”

**shows** “ $((p1, q1), w, (p2, q2)) \in pa.trans\_star_{\varepsilon}$ ”

**proof** –

**have** “ $\exists w1'. LTS_{\varepsilon.\varepsilon\_exp} w1' w \wedge (p1, w1', p2) \in A1.trans\_star$ ”

**using**  $assms$  **by** (simp add:  $LTS_{\varepsilon.trans\_star_{\varepsilon}\_exp\_trans\_star}$ )

**then** obtain  $w1'$  **where** “ $LTS_{\varepsilon.\varepsilon\_exp} w1' w \wedge (p1, w1', p2) \in A1.trans\_star$ ”

**by** auto

**moreover**

**have** “ $\exists w2'. LTS_{\varepsilon.\varepsilon\_exp} w2' w \wedge (q1, w2', q2) \in A2.trans\_star$ ”

**using**  $assms$  **by** (simp add:  $LTS_{\varepsilon.trans\_star_{\varepsilon}\_exp\_trans\_star}$ )

**then** obtain  $w2'$  **where** “ $LTS_{\varepsilon.\varepsilon\_exp} w2' w \wedge (q1, w2', q2) \in A2.trans\_star$ ”

**by** auto

**ultimately**

**show** ?thesis

**using**  $trans\_star\_trans\_star_{\varepsilon\_inter}$  **by** metis

**qed**

**lemma**  $inters\_trans\_star_{\varepsilon 1}$ :

**assumes** “ $(p1q2, w :: 'label\ list, p2q2) \in pa.trans\_star_{\varepsilon}$ ”

**shows** “ $(fst\ p1q2, w, fst\ p2q2) \in A1.trans\_star_{\varepsilon}$ ”

**using**  $assms$

**proof** (induction rule:  $LTS_{\varepsilon.trans\_star_{\varepsilon}.induct[OF\ assms(1)]}$ )

**case** (1  $p$ )

**then** show ?case

**by** (simp add:  $LTS_{\varepsilon.trans\_star_{\varepsilon}.trans\_star_{\varepsilon}\_refl}$ )

```

next
case (2 p γ q' w q)
then have ind: "(fst q', w, fst q) ∈ A1.trans_star_ε"
  by auto
from 2(1) have "(p, Some γ, q') ∈
  {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
  {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
  {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts1}"
  unfolding inters_ε_def by auto
moreover
{
  assume "(p, Some γ, q') ∈ {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
  then have "∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, Some γ, p2) ∈ ts1 ∧ (q1, Some γ, q2) ∈ ts2)"
  by simp
  then obtain p1 q1 where "p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, Some γ, p2) ∈ ts1 ∧ (q1, Some γ, q2) ∈ ts2)"
  by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_γ ind by fastforce
}
moreover
{
  assume "(p, Some γ, q') ∈ {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1}"
  then have ?case
    by auto
}
moreover
{
  assume "(p, Some γ, q') ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts1}"
  then have ?case
    by auto
}
ultimately
show ?case
  by auto
next
case (3 p q' w q)
then have ind: "(fst q', w, fst q) ∈ A1.trans_star_ε"
  by auto
from 3(1) have "(p, ε, q') ∈
  {((p1, q1), α, (p2, q2)) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
  {((p1, q1), ε, (p2, q1)) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
  {((p1, q1), ε, (p1, q2)) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
  unfolding inters_ε_def by auto
moreover
{
  assume "(p, ε, q') ∈ {((p1, q1), α, (p2, q2)) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
  then have "∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)"
  by simp
  then obtain p1 q1 where "p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)"
  by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
moreover
{
  assume "(p, ε, q') ∈ {((p1, q1), ε, (p2, q1)) | p1 p2 q1. (p1, ε, p2) ∈ ts1}"
  then have "∃ p1 p2 q1. p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1"
  by auto
  then obtain p1 p2 q1 where "p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1"
  by auto
  then have ?case
}

```

```

    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
  }
  moreover
  {
    assume “(p, ε, q′) ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}”
    then have “∃ p1 q1 q2. p = (p1, q1) ∧ q′ = (p1, q2) ∧ (q1, ε, q2) ∈ ts2”
      by auto
    then obtain p1 q1 q2 where “p = (p1, q1) ∧ q′ = (p1, q2) ∧ (q1, ε, q2) ∈ ts2”
      by auto
    then have ?case
      using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
  }
  ultimately
  show ?case
    by auto
qed

lemma inters_trans_star_ε:
  assumes “(p1q2, w :: 'label list, p2q2) ∈ pa.trans_star_ε”
  shows “(snd p1q2, w, snd p2q2) ∈ A2.trans_star_ε”
  using assms
proof (induction rule: LTS_ε.trans_star_ε.induct[OF assms(1)])
  case (1 p)
  then show ?case
    by (simp add: LTS_ε.trans_star_ε.trans_star_ε_refl)
next
  case (2 p γ q′ w q)
  then have ind: “(snd q′, w, snd q) ∈ A2.trans_star_ε”
    by auto
  from 2(1) have “(p, Some γ, q′) ∈
    {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
    {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
    {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}”
  unfolding inters_ε_def by auto
  moreover
  {
    assume “(p, Some γ, q′) ∈ {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}”
    then have “∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q′ = (p2, q2) ∧ (p1, Some γ, p2) ∈ ts1 ∧ (q1, Some γ, q2) ∈
    ts2)”
      by simp
    then obtain p1 q1 where “p = (p1, q1) ∧ (∃ p2 q2. q′ = (p2, q2) ∧ (p1, Some γ, p2) ∈ ts1 ∧ (q1, Some γ,
    q2) ∈ ts2)”
      by auto
    then have ?case
      using LTS_ε.trans_star_ε.trans_star_ε_step_γ ind by fastforce
  }
  moreover
  {
    assume “(p, Some γ, q′) ∈ {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1}”
    then have ?case
      by auto
  }
  moreover
  {
    assume “(p, Some γ, q′) ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}”
    then have ?case
      by auto
  }
  ultimately
  show ?case
    by auto
next
  case (3 p q′ w q)

```

```

then have ind: “(snd q', w, snd q) ∈ A2.trans_star_ε”
  by auto
from 3(1) have “(p, ε, q') ∈
  {((p1, q1), α, (p2, q2)) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
  {((p1, q1), ε, (p2, q1)) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
  {((p1, q1), ε, (p1, q2)) | p1 q1 q2. (q1, ε, q2) ∈ ts2}”
  unfolding inters_ε_def by auto
moreover
{
  assume “(p, ε, q') ∈ {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}”
  then have “∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)”
    by simp
  then obtain p1 q1 where “p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)”
    by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
moreover
{
  assume “(p, ε, q') ∈ {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1}”
  then have “∃ p1 p2 q1. p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1”
    by auto
  then obtain p1 p2 q1 where “p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1”
    by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
moreover
{
  assume “(p, ε, q') ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}”
  then have “∃ p1 q1 q2. p = (p1, q1) ∧ q' = (p1, q2) ∧ (q1, ε, q2) ∈ ts2”
    by auto
  then obtain p1 q1 q2 where “p = (p1, q1) ∧ q' = (p1, q2) ∧ (q1, ε, q2) ∈ ts2”
    by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
ultimately
show ?case
  by auto
qed

```

**lemma** *inters\_trans\_star\_ε\_iff*:

```

“((p1,q2), w :: 'label list, (p2,q2)) ∈ pa.trans_star_ε ↔
(p1, w, p2) ∈ A1.trans_star_ε ∧ (q2, w, q2) ∈ A2.trans_star_ε”
by (metis fst_conv inters_trans_star_ε inters_trans_star_ε1 snd_conv trans_star_ε_inter)

```

**lemma** *inters\_ε\_accept\_ε\_iff*:

```

“accepts_aut_inters_ε p w ↔ A1.accepts_aut_ε p w ∧ A2.accepts_aut_ε p w”

```

**proof**

```

assume “accepts_aut_inters_ε p w”
then show “A1.accepts_aut_ε p w ∧ A2.accepts_aut_ε p w”
  unfolding accepts_aut_inters_ε_def A1.accepts_aut_ε_def A2.accepts_aut_ε_def pa.accepts_aut_ε_def
  unfolding inters_finals_def
  using inters_trans_star_ε_iff[of _ _ w _]
  using SigmaE fst_conv inters_trans_star_ε inters_trans_star_ε1 snd_conv
  by (metis (no_types, lifting))

```

**next**

```

assume a: “A1.accepts_aut_ε p w ∧ A2.accepts_aut_ε p w”
then have “(∃ q∈finals1. (Init p, w, q) ∈ A1.trans_star_ε) ∧
  (∃ q∈finals2. (Init p, w, q) ∈ LTS_ε.trans_star_ε ts2)”
  unfolding A1.accepts_aut_ε_def A2.accepts_aut_ε_def by auto
then show “accepts_aut_inters_ε p w”

```

**unfolding** *accepts\_aut\_inters\_ε\_def* pa.*accepts\_aut\_ε\_def inters\_finals\_def*  
**by** (*auto simp: P\_Automaton\_ε.accepts\_aut\_ε\_def intro: trans\_star\_ε\_inter*)  
**qed**

**lemma** *inters\_ε\_lang\_ε*: “*lang\_aut\_inters\_ε = A1.lang\_aut\_ε ∩ A2.lang\_aut\_ε*”  
**unfolding** *lang\_aut\_inters\_ε\_def P\_Automaton\_ε.lang\_aut\_ε\_def* **using** *inters\_ε\_accept\_ε\_iff* **by** *auto*

**end**

**end**

**theory** *PDS* **imports** “*P\_Automata*” “*HOL-Library.While\_Combinator*” **begin**

## 4 PDS

**datatype** *'label operation = pop | swap 'label | push 'label 'label*  
**type-synonym** (*'ctr\_loc, 'label*) *rule =* “(*'ctr\_loc × 'label*) × (*'ctr\_loc × 'label operation*)”  
**type-synonym** (*'ctr\_loc, 'label*) *conf =* “*'ctr\_loc × 'label list*”

We define push down systems.

**locale** *PDS =*  
**fixes**  $\Delta ::$  “(*'ctr\_loc, 'label::finite*) *rule set*”

**begin**

**primrec** *lbl ::* “*'label operation ⇒ 'label list*” **where**  
*“lbl pop = []”*  
*| “lbl (swap  $\gamma$ ) = [ $\gamma$ ]”*  
*| “lbl (push  $\gamma$   $\gamma'$ ) = [ $\gamma, \gamma'$ ]”*

**definition** *is\_rule ::* “(*'ctr\_loc × 'label ⇒ 'ctr\_loc × 'label operation ⇒ bool*)” (**infix** “ $\hookrightarrow$ ” 80) **where**  
*“ $p\gamma \hookrightarrow p'w \equiv (p\gamma, p'w) \in \Delta$ ”*

**inductive-set** *transition\_rel ::* “(*'ctr\_loc, 'label*) *conf × unit × ('ctr\_loc, 'label*) *conf*) *set*” **where**  
*“( $p, \gamma$ )  $\hookrightarrow$  ( $p', w$ )  $\implies$   
( $p, \gamma\#w'$ ), ( $\cdot$ ), ( $p', (lbl\ w)\@w'$ )  $\in$  *transition\_rel*”*

**interpretation** *LTS transition\_rel .*

**notation** *step\_relp* (**infix** “ $\Rightarrow$ ” 80)  
**notation** *step\_starp* (**infix** “ $\Rightarrow^*$ ” 80)

**lemma** *step\_relp\_def2*:  
*“( $p, \gamma w'$ )  $\Rightarrow$  ( $p', ww'$ )  $\iff$  ( $\exists \gamma w' w. \gamma w' = \gamma\#w' \wedge ww' = (lbl\ w)\@w' \wedge (p, \gamma) \hookrightarrow (p', w)$ )”*  
**by** (*metis (no\_types, lifting) PDS.transition\_rel.intros step\_relp\_def transition\_rel.cases*)

**end**

## 5 PDS with P automata

**type-synonym** (*'ctr\_loc, 'label*) *sat\_rule =* “(*'ctr\_loc, 'label*) *transition set ⇒ ('ctr\_loc, 'label*) *transition set ⇒ bool*”

**datatype** (*'ctr\_loc, 'noninit, 'label*) *state =*  
*is\_Init: Init (the\_Ctr\_Loc: 'ctr\_loc)*  
*| is\_Noninit: Noninit (the\_St: 'noninit)*  
*| is\_Isolated: Isolated (the\_Ctr\_Loc: 'ctr\_loc) (the\_Label: 'label)*

**lemma** *finitely\_many\_states*:  
**assumes** “*finite (UNIV :: 'ctr\_loc set)*”  
**assumes** “*finite (UNIV :: 'noninit set)*”  
**assumes** “*finite (UNIV :: 'label set)*”

```

shows "finite (UNIV :: ('ctr_loc, 'noninit, 'label) state set)"
proof -
define Isolated' :: "('ctr_loc * 'label) ⇒ ('ctr_loc, 'noninit, 'label) state" where
  "Isolated' == λ(c :: 'ctr_loc, l:: 'label). Isolated c l"
define Init' :: "'ctr_loc ⇒ ('ctr_loc, 'noninit, 'label) state" where
  "Init' = Init"
define Noninit' :: "'noninit ⇒ ('ctr_loc, 'noninit, 'label) state" where
  "Noninit' = Noninit"

have split: "UNIV = (Init' ' UNIV) ∪ (Noninit' ' UNIV) ∪ (Isolated' ' (UNIV :: (('ctr_loc * 'label) set)))"
  unfolding Init'_def Noninit'_def
proof (rule; rule; rule; rule)
fix x :: "('ctr_loc, 'noninit, 'label) state"
assume "x ∈ UNIV"
moreover
assume "x ∉ range Isolated'"
moreover
assume "x ∉ range Noninit'"
ultimately
show "x ∈ range Init'"
  by (metis Isolated'_def prod.simps(2) range_eqI state.exhaust)
qed

have "finite (Init' ' (UNIV:: 'ctr_loc set))"
  using assms by auto
moreover
have "finite (Noninit' ' (UNIV:: 'noninit set))"
  using assms by auto
moreover
have "finite (UNIV :: (('ctr_loc * 'label) set))"
  using assms by (simp add: finite_Prod_UNIV)
then have "finite (Isolated' ' (UNIV :: (('ctr_loc * 'label) set)))"
  by auto
ultimately
show "finite (UNIV :: ('ctr_loc, 'noninit, 'label) state set)"
  unfolding split by auto
qed

instantiation state :: (finite, finite, finite) finite begin

instance by standard (simp add: finitely_many_states)

end

locale PDS_with_P_automata = PDS Δ
  for Δ :: "('ctr_loc::enum, 'label::finite) rule set"
  +
  fixes final_inits :: "('ctr_loc::enum) set"
  fixes final_noninits :: "('noninit::finite) set"
begin

definition finals :: "('ctr_loc, 'noninit::finite, 'label) state set" where
  "finals = Init ' final_inits ∪ Noninit ' final_noninits"

lemma F_not_Ext: "¬(∃ f ∈ finals. is_Isolated f)"
  using finals_def by fastforce

definition inits :: "('ctr_loc, 'noninit, 'label) state set" where
  "inits = {q. is_Init q}"

```

**lemma** *inits\_code*[code]: “*inits = set (map Init Enum.enum)*”  
**by** (*auto simp: inits\_def is\_Init\_def simp flip: UNIV\_enum*)

**definition** *noninits* :: “(*'ctr\_loc, 'noninit, 'label*) state set” **where**  
“*noninits = {q. is\_Noninit q}*”

**definition** *isols* :: “(*'ctr\_loc, 'noninit, 'label*) state set” **where**  
“*isols = {q. is\_Isolated q}*”

**sublocale** *LTS transition\_rel* .  
**notation** *step\_relp* (**infix** “ $\Rightarrow$ ” 80)  
**notation** *step\_starp* (**infix** “ $\Rightarrow^*$ ” 80)

**definition** *accepts* :: “(*'ctr\_loc, 'noninit, 'label*) state, *'label*) transition set  $\Rightarrow$  (*'ctr\_loc, 'label*) conf  $\Rightarrow$  bool” **where**  
“*accepts ts  $\equiv$   $\lambda(p,w). (\exists q \in \text{finals}. (Init\ p,w,q) \in LTS.trans\_star\ ts)$* ”

**lemma** *accepts\_accepts\_aut*: “*accepts ts (p, w)  $\longleftrightarrow$  P\_Automaton.accepts\_aut ts Init finals p w*”  
**unfolding** *accepts\_def P\_Automaton.accepts\_aut\_def inits\_def* **by** *auto*

**definition** *accepts\_ε* :: “(*'ctr\_loc, 'noninit, 'label*) state, *'label option*) transition set  $\Rightarrow$  (*'ctr\_loc, 'label*) conf  $\Rightarrow$  bool” **where**  
“*accepts\_ε ts  $\equiv$   $\lambda(p,w). (\exists q \in \text{finals}. (Init\ p,w,q) \in LTS_ε.trans\_star_ε\ ts)$* ”

**abbreviation** *ε* :: “*'label option*” **where**  
“*ε == None*”

**lemma** *accepts\_mono*[*mono*]: “*mono accepts*”

**proof** (*rule, rule*)  
**fix** *c* :: “(*'ctr\_loc, 'label*) conf”  
**fix** *ts ts'* :: “(*'ctr\_loc, 'noninit, 'label*) state, *'label*) transition set”  
**assume** *accepts\_ts*: “*accepts ts c*”  
**assume** *tstst'*: “*ts  $\subseteq$  ts'*”  
**obtain** *p l* **where** *pl\_p*: “*c = (p,l)*”  
**by** (*cases c*)  
**obtain** *q* **where** *q\_p*: “*q  $\in$  finals  $\wedge$  (Init p, l, q)  $\in$  LTS.trans\_star ts*”  
**using** *accepts\_ts unfolding pl\_p accepts\_def* **by** *auto*  
**then have** “*(Init p, l, q)  $\in$  LTS.trans\_star ts'*”  
**using** *tstst' LTS\_trans\_star\_mono monoD* **by** *blast*  
**then have** “*accepts ts' (p,l)*”  
**unfolding** *accepts\_def using q\_p* **by** *auto*  
**then show** “*accepts ts' c*”  
**unfolding** *pl\_p* .

**qed**

**lemma** *accepts\_cons*: “*(Init p, γ, Init p')  $\in$  ts  $\implies$  accepts ts (p', w)  $\implies$  accepts ts (p, γ # w)*”  
**using** *LTS.trans\_star.trans\_star\_step accepts\_def* **by** *fastforce*

**definition** *lang* :: “(*'ctr\_loc, 'noninit, 'label*) state, *'label*) transition set  $\Rightarrow$  (*'ctr\_loc, 'label*) conf set” **where**  
“*lang ts = {c. accepts ts c}*”

**lemma** *lang\_lang\_aut*: “*lang ts = ( $\lambda(s,w). (s, w)$ ) ‘(P\_Automaton.lang\_aut ts Init finals)*”  
**unfolding** *lang\_def P\_Automaton.lang\_aut\_def*  
**by** (*auto simp: inits\_def accepts\_def P\_Automaton.accepts\_aut\_def image\_iff intro!: exI[of \_ “Init \_”]*)

**lemma** *lang\_aut\_lang*: “*P\_Automaton.lang\_aut ts Init finals = lang ts*”  
**unfolding** *lang\_lang\_aut*  
**by** (*auto 0 3 simp: P\_Automaton.lang\_aut\_def P\_Automaton.accepts\_aut\_def inits\_def image\_iff*)

**definition** *lang\_ε* :: “(*'ctr\_loc, 'noninit, 'label*) state, *'label option*) transition set  $\Rightarrow$  (*'ctr\_loc, 'label*) conf set”  
**where**  
“*lang\_ε ts = {c. accepts\_ε ts c}*”



## 5.1 Saturations

**definition** *saturated* :: “(*c*, *l*) *sat\_rule*  $\Rightarrow$  (*c*, *l*) transition set  $\Rightarrow$  bool” where  
“saturated rule *ts*  $\longleftrightarrow$  ( $\nexists$  *ts'*. rule *ts ts'*)”

**definition** *saturation* :: “(*c*, *l*) *sat\_rule*  $\Rightarrow$  (*c*, *l*) transition set  $\Rightarrow$  (*c*, *l*) transition set  $\Rightarrow$  bool” where  
“saturation rule *ts ts'*  $\longleftrightarrow$  rule\*\* *ts ts'*  $\wedge$  saturated rule *ts'*”

**lemma** *no\_infinite*:

**assumes** “ $\bigwedge$  *ts ts'* :: (*c* :: finite, *l* :: finite) transition set. rule *ts ts'*  $\Longrightarrow$  card *ts'* = Suc (card *ts*)”

**assumes** “ $\forall$  *i* :: nat. rule (*tts i*) (*tts (Suc i)*)”

**shows** “False”

**proof** –

**define** *f* **where** “*f i* = card (*tts i*)” **for** *i*

**have** *f\_Suc*: “ $\forall$  *i*. *f i* < *f (Suc i)*”

**using** *assms f\_def lessI* **by** *metis*

**have** “ $\forall$  *i*.  $\exists$  *j*. *f j* > *i*”

**proof**

**fix** *i*

**show** “ $\exists$  *j*. *i* < *f j*”

**proof**(*induction i*)

**case** 0

**then show** ?*case*

**by** (*metis f\_Suc neq0\_conv*)

**next**

**case** (*Suc i*)

**then show** ?*case*

**by** (*metis Suc\_lessI f\_Suc*)

**qed**

**qed**

**then have** “ $\exists$  *j*. *f j* > card (*UNIV* :: (*c*, *l*) transition set)”

**by** *auto*

**then show** False

**by** (*metis card\_seteq f\_def finite\_UNIV le\_eq\_less\_or\_eq nat\_neq\_iff subset\_UNIV*)

**qed**

**lemma** *saturation\_termination*:

**assumes** “ $\bigwedge$  *ts ts'* :: (*c* :: finite, *l* :: finite) transition set. rule *ts ts'*  $\Longrightarrow$  card *ts'* = Suc (card *ts*)”

**shows** “ $\neg(\exists$  *tts*. ( $\forall$  *i* :: nat. rule (*tts i*) (*tts (Suc i)*)))”

**using** *assms no\_infinite* **by** *blast*

**lemma** *saturation\_exi*:

**assumes** “ $\bigwedge$  *ts ts'* :: (*c* :: finite, *l* :: finite) transition set. rule *ts ts'*  $\Longrightarrow$  card *ts'* = Suc (card *ts*)”

**shows** “ $\exists$  *ts'*. saturation rule *ts ts'*”

**proof** (*rule ccontr*)

**assume** *a*: “ $\nexists$  *ts'*. saturation rule *ts ts'*”

**define** *g* **where** “*g ts* = (SOME *ts'*. rule *ts ts'*)” **for** *ts*

**define** *tts* **where** “*tts i* = (*g*  $\widetilde{\sim}$  *i*) *ts*” **for** *i*

**have** “ $\forall$  *i* :: nat. rule\*\* *ts (tts i)*  $\wedge$  rule (*tts i*) (*tts (Suc i)*)”

**proof**

**fix** *i*

**show** “rule\*\* *ts (tts i)*  $\wedge$  rule (*tts i*) (*tts (Suc i)*)”

**proof** (*induction i*)

**case** 0

**have** “rule *ts (g ts)*”

**by** (*metis g\_def a rtranclp.rtrancl\_refl saturation\_def saturated\_def someI*)

**then show** ?*case*

**using** *tts\_def a saturation\_def* **by** *auto*

**next**

**case** (*Suc i*)

**then have** *sat\_Suc*: “rule\*\* *ts (tts (Suc i))*”

**by** *fastforce*

**then have** “rule (*g ((g*  $\widetilde{\sim}$  *i*) *ts*)) (*g ((g*  $\widetilde{\sim}$  *i*) *ts*))”

**by** (*metis Suc.IH tts\_def g\_def a\_r\_into\_rtranclp\_rtranclp\_trans saturation\_def saturated\_def*)

```

      someI)
    then have "rule (tts (Suc i)) (tts (Suc (Suc i)))"
      unfolding tts_def by simp
    then show ?case
      using sat_Suc by auto
  qed
qed
then have "∀ i. rule (tts i) (tts (Suc i))"
  by auto
then show False
  using no_infinite_assms by auto
qed

```

## 5.2 Saturation rules

**inductive** *pre\_star\_rule* :: "(('ctr\_loc, 'noninit, 'label) state, 'label) transition set ⇒ (('ctr\_loc, 'noninit, 'label) state, 'label) transition set ⇒ bool" **where**  
*add\_trans*: " $(p, \gamma) \hookrightarrow (p', w) \implies (Init\ p',\ lbl\ w,\ q) \in LTS.trans\_star\ ts \implies (Init\ p,\ \gamma,\ q) \notin ts \implies pre\_star\_rule\ ts\ (ts \cup \{(Init\ p,\ \gamma,\ q)\})$ "

**definition** *pre\_star1* :: "(('ctr\_loc, 'noninit, 'label) state, 'label) transition set ⇒ (('ctr\_loc, 'noninit, 'label) state, 'label) transition set" **where**  
*pre\_star1* *ts* =  
 $(\bigcup ((p, \gamma), (p', w)) \in \Delta. \bigcup q \in LTS.reach\ ts\ (Init\ p')\ (lbl\ w). \{(Init\ p,\ \gamma,\ q)\})$

**lemma** *pre\_star1\_mono*: "mono *pre\_star1*"  
**unfolding** *pre\_star1\_def*  
**by** (auto simp: mono\_def LTS.trans\_star\_code[symmetric] elim!: beX[rotated] LTS\_trans\_star\_mono[THEN monoD, THEN subsetD])

**lemma** *pre\_star\_rule\_pre\_star1*:  
**assumes** " $X \subseteq pre\_star1\ ts$ "  
**shows** " $pre\_star\_rule^{**}\ ts\ (ts \cup X)$ "

**proof** –  
**have** "finite *X*"  
**by** *simp*  
**from** *this* **assms** **show** ?thesis  
**proof** (induct *X* arbitrary: *ts* rule: finite\_induct)  
**case** (insert *x F*)  
**then obtain** *p* *γ* *p'* *w* *q* **where** \*: " $(p, \gamma) \hookrightarrow (p', w)$ "  
" $(Init\ p',\ lbl\ w,\ q) \in LTS.trans\_star\ ts$ " **and** *x*:  
" $x = (Init\ p,\ \gamma,\ q)$ "  
**by** (auto simp: *pre\_star1\_def* is\_rule\_def LTS.trans\_star\_code)  
**with** insert **show** ?case  
**proof** (cases " $(Init\ p,\ \gamma,\ q) \in ts$ ")  
**case** False  
**with** insert(1,2,4) *x* **show** ?thesis  
**by** (intro converse\_rtranclp\_into\_rtranclp[of *pre\_star\_rule*, OF *add\_trans*[OF \* False]])  
(auto intro!: insert(3)[of "insert *x ts*", simplified *x Un\_insert\_left*]  
intro: *pre\_star1\_mono*[THEN monoD, THEN set\_mp, of *ts*])  
**qed** (simp add: insert\_absorb)  
**qed** *simp*  
**qed**

**lemma** *pre\_star\_rule\_pre\_star1s*: " $pre\_star\_rule^{**}\ ts\ (((\lambda s. s \cup pre\_star1\ s) \rightsquigarrow k)\ ts)$ "  
**by** (induct *k*) (auto elim!: rtranclp\_trans intro: *pre\_star\_rule\_pre\_star1*)

**definition** "*pre\_star\_loop* = while\_option ( $\lambda s. s \cup pre\_star1\ s \neq s$ ) ( $\lambda s. s \cup pre\_star1\ s$ )"

**definition** "*pre\_star\_exec* = the o *pre\_star\_loop*"

**definition** "*pre\_star\_exec\_check* *A* = (if inits  $\subseteq LTS.srcs\ A$  then *pre\_star\_loop* *A* else None)"

**definition** "*accept\_pre\_star\_exec\_check* *A* *c* = (if inits  $\subseteq LTS.srcs\ A$  then Some (accepts (*pre\_star\_exec* *A*) *c*) else None)"

**lemma** *while\_option\_finite\_subset\_Some*: **fixes**  $C :: 'a \text{ set}$   
**assumes** “*mono f*” **and** “ $\forall X. X \subseteq C \implies f X \subseteq C$ ” **and** “*finite C*” **and**  $X: “X \subseteq C” “X \subseteq f X”$   
**shows** “ $\exists P. \text{while\_option } (\lambda A. f A \neq A) f X = \text{Some } P$ ”  
**proof**(*rule measure\_while\_option\_Some*[**where**  
 $f = “\%A::'a \text{ set}. \text{card } C - \text{card } A”$  **and**  $P = “\%A. A \subseteq C \wedge A \subseteq f A”$  **and**  $s = X$ ])  
**fix**  $A$  **assume**  $A: “A \subseteq C \wedge A \subseteq f A” “f A \neq A”$   
**show** “ $(f A \subseteq C \wedge f A \subseteq f (f A)) \wedge \text{card } C - \text{card } (f A) < \text{card } C - \text{card } A$ ”  
(is “ $?L \wedge ?R$ ”)  
**proof**  
**show**  $?L$  **by**(*metis A(1) assms(2) monoD[OF <mono f>]*)  
**show**  $?R$   
**by** (*metis A assms(2,3) card\_seteq diff\_less\_mono2 equalityI linorder\_le\_less\_linear rev\_finite\_subset*)  
**qed**  
**qed** (*simp add: X*)

**lemma** *pre\_star\_exec\_terminates*: “ $\exists t. \text{pre\_star\_loop } s = \text{Some } t$ ”  
**unfolding** *pre\_star\_loop\_def*  
**by** (*rule while\_option\_finite\_subset\_Some*[**where**  $C = \text{UNIV}$ ])  
(*auto simp: mono\_def dest: pre\_star1\_mono[THEN monoD]*)

**lemma** *pre\_star\_exec\_code*[*code*]:  
“*pre\_star\_exec s = (let s' = pre\_star1 s in if s'  $\subseteq$  s then s else pre\_star\_exec (s  $\cup$  s'))*”  
**unfolding** *pre\_star\_exec\_def pre\_star\_loop\_def o\_apply*  
**by** (*subst while\_option\_unfold*)(*auto simp: Let\_def*)

**lemma** *saturation\_pre\_star\_exec*: “*saturation pre\_star\_rule ts (pre\_star\_exec ts)*”  
**proof** –  
**from** *pre\_star\_exec\_terminates* **obtain**  $t$  **where**  $t: “\text{pre\_star\_loop } ts = \text{Some } t”$   
**by** *blast*  
**obtain**  $k$  **where**  $k: “t = ((\lambda s. s \cup \text{pre\_star1 } s) \rightsquigarrow k) ts”$  **and**  $le: “\text{pre\_star1 } t \subseteq t”$   
**using** *while\_option\_stop2*[*OF t[unfolding pre\_star\_loop\_def]*] **by** *auto*  
**have** “ $(\bigcup \{us. \text{pre\_star\_rule } t us\}) - t \subseteq \text{pre\_star1 } t$ ”  
**by** (*auto simp: pre\_star1\_def LTS.trans\_star\_code[symmetric] prod.splits is\_rule\_def pre\_star\_rule.simps*)  
**from** *subset\_trans*[*OF this le*] **show** *?thesis*  
**unfolding** *saturation\_def saturated\_def pre\_star\_exec\_def o\_apply k t*  
**by** (*auto 9 0 simp: pre\_star\_rule\_pre\_star1s subset\_eq pre\_star\_rule.simps*)  
**qed**

**inductive** *post\_star\_rules* :: “((*ctr\_loc*, *'noninit*, *'label*) *state*, *'label option*) *transition set*  $\implies$  ((*ctr\_loc*, *'noninit*, *'label*) *state*, *'label option*) *transition set*  $\implies$  *bool*” **where**  
*add\_trans\_pop*:  
“ $(p, \gamma) \leftrightarrow (p', \text{pop}) \implies$   
 $(\text{Init } p, [\gamma], q) \in \text{LTS}_{\varepsilon}.\text{trans\_star}_{\varepsilon} ts \implies$   
 $(\text{Init } p', \varepsilon, q) \notin ts \implies$   
 $\text{post\_star\_rules } ts (ts \cup \{(\text{Init } p', \varepsilon, q)\})”$   
| *add\_trans\_swap*:  
“ $(p, \gamma) \leftrightarrow (p', \text{swap } \gamma') \implies$   
 $(\text{Init } p, [\gamma], q) \in \text{LTS}_{\varepsilon}.\text{trans\_star}_{\varepsilon} ts \implies$   
 $(\text{Init } p', \text{Some } \gamma', q) \notin ts \implies$   
 $\text{post\_star\_rules } ts (ts \cup \{(\text{Init } p', \text{Some } \gamma', q)\})”$   
| *add\_trans\_push\_1*:  
“ $(p, \gamma) \leftrightarrow (p', \text{push } \gamma' \gamma'') \implies$   
 $(\text{Init } p, [\gamma], q) \in \text{LTS}_{\varepsilon}.\text{trans\_star}_{\varepsilon} ts \implies$   
 $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin ts \implies$   
 $\text{post\_star\_rules } ts (ts \cup \{(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma')\})”$   
| *add\_trans\_push\_2*:  
“ $(p, \gamma) \leftrightarrow (p', \text{push } \gamma' \gamma'') \implies$   
 $(\text{Init } p, [\gamma], q) \in \text{LTS}_{\varepsilon}.\text{trans\_star}_{\varepsilon} ts \implies$   
 $(\text{Isolated } p' \gamma', \text{Some } \gamma'', q) \notin ts \implies$   
 $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \in ts \implies$   
 $\text{post\_star\_rules } ts (ts \cup \{(\text{Isolated } p' \gamma', \text{Some } \gamma'', q)\})”$

**lemma** *pre\_star\_rule\_mono*:

“ $pre\_star\_rule\ ts\ ts' \implies ts \subset ts'$ ”

**unfolding** *pre\_star\_rule.simps* **by** *auto*

**lemma** *post\_star\_rules\_mono*:

“ $post\_star\_rules\ ts\ ts' \implies ts \subset ts'$ ”

**proof**(*induction* *rule*: *post\_star\_rules.induct*)

**case** (*add\_trans\_pop* *p*  $\gamma$  *p'* *q* *ts*)

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

**next**

**case** (*add\_trans\_swap* *p*  $\gamma$  *p'*  $\gamma'$  *q* *ts*)

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

**next**

**case** (*add\_trans\_push\_1* *p*  $\gamma$  *p'*  $\gamma'$   $\gamma''$  *q* *ts*)

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

**next**

**case** (*add\_trans\_push\_2* *p*  $\gamma$  *p'*  $\gamma'$   $\gamma''$  *q* *ts*)

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

**qed**

**lemma** *pre\_star\_rule\_card\_Suc*: “ $pre\_star\_rule\ ts\ ts' \implies card\ ts' = Suc\ (card\ ts)$ ”

**unfolding** *pre\_star\_rule.simps* **by** *auto*

**lemma** *post\_star\_rules\_card\_Suc*: “ $post\_star\_rules\ ts\ ts' \implies card\ ts' = Suc\ (card\ ts)$ ”

**proof**(*induction* *rule*: *post\_star\_rules.induct*)

**case** (*add\_trans\_pop* *p*  $\gamma$  *p'* *q* *ts*)

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

**next**

**case** (*add\_trans\_swap* *p*  $\gamma$  *p'*  $\gamma'$  *q* *ts*)

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

**next**

**case** (*add\_trans\_push\_1* *p*  $\gamma$  *p'*  $\gamma'$   $\gamma''$  *q* *ts*)

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

**next**

**case** (*add\_trans\_push\_2* *p*  $\gamma$  *p'*  $\gamma'$   $\gamma''$  *q* *ts*)

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

**qed**

**lemma** *pre\_star\_saturation\_termination*:

“ $\neg(\exists\ tss. (\forall\ i :: nat. pre\_star\_rule\ (tts\ i)\ (tts\ (Suc\ i))))$ ”

**using** *no\_infinite\_pre\_star\_rule\_card\_Suc* **by** *blast*

**lemma** *post\_star\_saturation\_termination*:

“ $\neg(\exists\ tss. (\forall\ i :: nat. post\_star\_rules\ (tts\ i)\ (tts\ (Suc\ i))))$ ”

**using** *no\_infinite\_post\_star\_rules\_card\_Suc* **by** *blast*

**lemma** *pre\_star\_saturation\_exi*:

**shows** “ $\exists\ ts'. saturation\ pre\_star\_rule\ ts\ ts'$ ”

**using** *pre\_star\_rule\_card\_Suc\_saturation\_exi* **by** *blast*

**lemma** *post\_star\_saturation\_exi*:

**shows** “ $\exists\ ts'. saturation\ post\_star\_rules\ ts\ ts'$ ”

**using** *post\_star\_rules\_card\_Suc\_saturation\_exi* **by** *blast*

**lemma** *pre\_star\_rule\_incr*: “ $pre\_star\_rule\ A\ B \implies A \subseteq B$ ”

**proof**(*induction* *rule*: *pre\_star\_rule.inducts*)

**case** (*add\_trans* *p*  $\gamma$  *p'* *w* *q* *rel*)

**then show** *?case*

**by** *auto*

**qed**

**lemma** *post\_star\_rules\_incr*: “ $\text{post\_star\_rules } A \ B \implies A \subseteq B$ ”

**proof**(*induction rule*: *post\_star\_rules.inducts*)

**case** (*add\_trans\_pop*  $p \ \gamma \ p' \ q \ ts$ )

**then show** *?case*

**by** *auto*

**next**

**case** (*add\_trans\_swap*  $p \ \gamma \ p' \ \gamma' \ q \ ts$ )

**then show** *?case*

**by** *auto*

**next**

**case** (*add\_trans\_push\_1*  $p \ \gamma \ p' \ \gamma' \ \gamma'' \ q \ ts$ )

**then show** *?case*

**by** *auto*

**next**

**case** (*add\_trans\_push\_2*  $p \ \gamma \ p' \ \gamma' \ \gamma'' \ q \ ts$ )

**then show** *?case*

**by** *auto*

**qed**

**lemma** *saturation\_rtranclp\_pre\_star\_rule\_incr*: “ $\text{pre\_star\_rule}^{**} \ A \ B \implies A \subseteq B$ ”

**proof** (*induction rule*: *rtranclp\_induct*)

**case** *base*

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

**next**

**case** (*step*  $y \ z$ )

**then show** *?case*

**using** *pre\_star\_rule\_incr* **by** *auto*

**qed**

**lemma** *saturation\_rtranclp\_post\_star\_rule\_incr*: “ $\text{post\_star\_rules}^{**} \ A \ B \implies A \subseteq B$ ”

**proof** (*induction rule*: *rtranclp\_induct*)

**case** *base*

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

**next**

**case** (*step*  $y \ z$ )

**then show** *?case*

**using** *post\_star\_rules\_incr* **by** *auto*

**qed**

**lemma** *pre\_star'\_incr\_trans\_star*:

“ $\text{pre\_star\_rule}^{**} \ A \ A' \implies LTS.\text{trans\_star} \ A \subseteq LTS.\text{trans\_star} \ A'$ ”

**using** *mono\_def LTS\_trans\_star\_mono saturation\_rtranclp\_pre\_star\_rule\_incr* **by** *metis*

**lemma** *post\_star'\_incr\_trans\_star*:

“ $\text{post\_star\_rules}^{**} \ A \ A' \implies LTS.\text{trans\_star} \ A \subseteq LTS.\text{trans\_star} \ A'$ ”

**using** *mono\_def LTS\_trans\_star\_mono saturation\_rtranclp\_post\_star\_rule\_incr* **by** *metis*

**lemma** *post\_star'\_incr\_trans\_star\_ε*:

“ $\text{post\_star\_rules}^{**} \ A \ A' \implies LTS_{\epsilon}.\text{trans\_star}_{\epsilon} \ A \subseteq LTS_{\epsilon}.\text{trans\_star}_{\epsilon} \ A'$ ”

**using** *mono\_def LTS\_ε\_trans\_star\_ε\_mono saturation\_rtranclp\_post\_star\_rule\_incr* **by** *metis*

**lemma** *pre\_star\_lim'\_incr\_trans\_star*:

“ $\text{saturation pre\_star\_rule} \ A \ A' \implies LTS.\text{trans\_star} \ A \subseteq LTS.\text{trans\_star} \ A'$ ”

**by** (*simp add*: *pre\_star'\_incr\_trans\_star saturation\_def*)

**lemma** *post\_star\_lim'\_incr\_trans\_star*:

“ $\text{saturation post\_star\_rules} \ A \ A' \implies LTS.\text{trans\_star} \ A \subseteq LTS.\text{trans\_star} \ A'$ ”

**by** (*simp add*: *post\_star'\_incr\_trans\_star saturation\_def*)

**lemma** *post\_star\_lim'\_incr\_trans\_star\_ε*:

“ $\text{saturation post\_star\_rules} \ A \ A' \implies LTS_{\epsilon}.\text{trans\_star}_{\epsilon} \ A \subseteq LTS_{\epsilon}.\text{trans\_star}_{\epsilon} \ A'$ ”

**by** (*simp add*: *post\_star'\_incr\_trans\_star\_ε saturation\_def*)

### 5.3 Pre\* lemmas

**lemma** *inits\_srcs\_iff\_Ctr\_Loc\_srcs*:

“ $inits \subseteq LTS.srcs\ A \longleftrightarrow (\nexists q\ \gamma\ q'. (q, \gamma, Init\ q') \in A)$ ”

**proof**

**assume** “ $inits \subseteq LTS.srcs\ A$ ”

**then show** “ $\nexists q\ \gamma\ q'. (q, \gamma, Init\ q') \in A$ ”

**by** (*simp add: Collect\_mono\_iff LTS.srcs\_def inits\_def*)

**next**

**assume** “ $\nexists q\ \gamma\ q'. (q, \gamma, Init\ q') \in A$ ”

**show** “ $inits \subseteq LTS.srcs\ A$ ”

**by** (*metis LTS.srcs\_def2 inits\_def <math>\nexists q\ \gamma\ q'. (q, \gamma, Init\ q') \in A</math> mem\_Collect\_eq state.collapse(1) subsetI*)

**qed**

**lemma** *lemma\_3\_1*:

**assumes** “ $p'w \Rightarrow^* pv$ ”

**assumes** “ $pv \in lang\ A$ ”

**assumes** “*saturation pre\_star\_rule*  $A\ A'$ ”

**shows** “*accepts*  $A'\ p'w$ ”

**using** *assms*

**proof** (*induct rule: converse\_rtranclp\_induct*)

**case** *base*

**define**  $p$  **where** “ $p = fst\ pv$ ”

**define**  $v$  **where** “ $v = snd\ pv$ ”

**from** *base* **have** “ $\exists q \in finals. (Init\ p, v, q) \in LTS.trans\_star\ A'$ ”

**unfolding** *lang\_def p\_def v\_def* **using** *pre\_star\_lim'\_incr\_trans\_star accepts\_def* **by** *fastforce*

**then show** *?case*

**unfolding** *accepts\_def p\_def v\_def* **by** *auto*

**next**

**case** (*step*  $p'w\ p''u$ )

**define**  $p'$  **where** “ $p' = fst\ p'w$ ”

**define**  $w$  **where** “ $w = snd\ p'w$ ”

**define**  $p''$  **where** “ $p'' = fst\ p''u$ ”

**define**  $u$  **where** “ $u = snd\ p''u$ ”

**have**  $p'w\_def$ : “ $p'w = (p', w)$ ”

**using**  $p'\_def\ w\_def$  **by** *auto*

**have**  $p''u\_def$ : “ $p''u = (p'', u)$ ”

**using**  $p''\_def\ u\_def$  **by** *auto*

**then have** “*accepts*  $A'\ (p'', u)$ ”

**using** *step* **by** *auto*

**then obtain**  $q$  **where**  $q\_p$ : “ $q \in finals \wedge (Init\ p'', u, q) \in LTS.trans\_star\ A'$ ”

**unfolding** *accepts\_def* **by** *auto*

**have** “ $\exists \gamma\ w1\ u1. w = \gamma \# w1 \wedge u = lbl\ u1 @ w1 \wedge (p', \gamma) \hookrightarrow (p'', u1)$ ”

**using**  $p''u\_def\ p'w\_def\ step.hyps(1)\ step\_relp\_def2$  **by** *auto*

**then obtain**  $\gamma\ w1\ u1$  **where**  $\gamma\_w1\_u1\_p$ : “ $w = \gamma \# w1 \wedge u = lbl\ u1 @ w1 \wedge (p', \gamma) \hookrightarrow (p'', u1)$ ”

**by** *blast*

**then have** “ $\exists q1. (Init\ p'', lbl\ u1, q1) \in LTS.trans\_star\ A' \wedge (q1, w1, q) \in LTS.trans\_star\ A'$ ”

**using**  $q\_p\ LTS.trans\_star\_split$  **by** *auto*

**then obtain**  $q1$  **where**  $q1\_p$ : “ $(Init\ p'', lbl\ u1, q1) \in LTS.trans\_star\ A' \wedge (q1, w1, q) \in LTS.trans\_star\ A'$ ”

**by** *auto*

**then have**  $in\_A'$ : “ $(Init\ p', \gamma, q1) \in A'$ ”

**using**  $\gamma\_w1\_u1\_p\ add\_trans[of\ p'\ \gamma\ p''\ u1\ q1\ A']\ saturated\_def\ saturation\_def\ step.premis$  **by** *metis*

**then have** “ $(Init\ p', \gamma \# w1, q) \in LTS.trans\_star\ A'$ ”

**using**  $LTS.trans\_star.trans\_star\_step\ q1\_p$  **by** *meson*

**then have**  $t\_in\_A'$ : “ $(Init\ p', w, q) \in LTS.trans\_star\ A'$ ”

**using**  $\gamma\_w1\_u1\_p$  **by** *blast*

**from**  $q\_p\ t\_in\_A'$  **have** “ $q \in finals \wedge (Init\ p', w, q) \in LTS.trans\_star\ A'$ ”

```

  by auto
  then show ?case
    unfolding accepts_def p'w_def by auto
qed

```

```

lemma word_into_init_empty_states:
  fixes A :: “(('ctr_loc, 'noninit, 'label) state, 'label) transition set”
  assumes “(p, w, ss, Init q) ∈ LTS.trans_star_states A”
  assumes “inits ⊆ LTS.srcs A”
  shows “w = [] ∧ p = Init q ∧ ss=[p]”
proof -
  define q1 :: “('ctr_loc, 'noninit, 'label) state” where
    “q1 = Init q”
  have q1_path: “(p, w, ss, q1) ∈ LTS.trans_star_states A”
    by (simp add: assms(1) q1_def)
  moreover
  have “q1 ∈ inits”
    by (simp add: inits_def q1_def)
  ultimately
  have “w = [] ∧ p = q1 ∧ ss=[p]”
  proof(induction rule: LTS.trans_star_states.induct[OF q1_path])
    case (1 p)
    then show ?case by auto
  next
    case (2 p γ q' w ss q)
    have “∄ q γ q'. (q, γ, Init q') ∈ A”
      using assms(2) unfolding inits_def LTS.srcs_def by (simp add: Collect_mono_iff)
    then show ?case
      using 2 assms(2) by (metis inits_def is_Init_def mem_Collect_eq)
  qed
  then show ?thesis
    using q1_def by fastforce
qed

```

```

lemma word_into_init_empty:
  fixes A :: “(('ctr_loc, 'noninit, 'label) state, 'label) transition set”
  assumes “(p, w, Init q) ∈ LTS.trans_star A”
  assumes “inits ⊆ LTS.srcs A”
  shows “w = [] ∧ p = Init q”
  using assms word_into_init_empty_states LTS.trans_star_trans_star_states by metis

```

```

lemma step_relp_append_aux:
  assumes “pu ⇒* p1y”
  shows “(fst pu, snd pu @ v) ⇒* (fst p1y, snd p1y @ v)”
  using assms
proof (induction rule: rtranclp_induct)
  case base
  then show ?case by auto
next
  case (step p'w p1y)
  define p where “p = fst pu”
  define u where “u = snd pu”
  define p' where “p' = fst p'w”
  define w where “w = snd p'w”
  define p1 where “p1 = fst p1y”
  define y where “y = snd p1y”
  have step_1: “(p,u) ⇒* (p',w)”
    by (simp add: p'_def p_def step.hyps(1) u_def w_def)
  have step_2: “(p',w) ⇒ (p1,y)”
    by (simp add: p'_def p1_def step.hyps(2) w_def y_def)
  have step_3: “(p, u @ v) ⇒* (p', w @ v)”
    by (simp add: p'_def p_def step.IH u_def w_def)

```

```

note step' = step_1 step_2 step_3

from step'(2) have “ $\exists \gamma w' wa. w = \gamma \# w' \wedge y = \text{lbl } wa @ w' \wedge (p', \gamma) \leftrightarrow (p1, wa)$ ”
  using step_relp_def2[of p' w p1 y] by auto
then obtain  $\gamma w' wa$  where  $\gamma_w'_wa_p$ : “ $w = \gamma \# w' \wedge y = \text{lbl } wa @ w' \wedge (p', \gamma) \leftrightarrow (p1, wa)$ ”
  by metis
then have “ $(p, u @ v) \Rightarrow^* (p1, y @ v)$ ”
  by (metis (no_types, lifting) PDS.step_relp_def2 append.assoc append_Cons rtranclp.simps step_3)
then show ?case
  by (simp add: p1_def p_def u_def y_def)
qed

```

```

lemma step_relp_append:
  assumes “ $(p, u) \Rightarrow^* (p1, y)$ ”
  shows “ $(p, u @ v) \Rightarrow^* (p1, y @ v)$ ”
  using assms step_relp_append_aux by auto

```

```

lemma step_relp_append_empty:
  assumes “ $(p, u) \Rightarrow^* (p1, [])$ ”
  shows “ $(p, u @ v) \Rightarrow^* (p1, v)$ ”
  using step_relp_append[OF assms] by auto

```

```

lemma lemma_3_2_a':
  assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
  assumes “ $\text{pre\_star\_rule}^{**} A A'$ ”
  assumes “ $(\text{Init } p, w, q) \in \text{LTS.trans\_star } A'$ ”
  shows “ $\exists p' w'. (\text{Init } p', w', q) \in \text{LTS.trans\_star } A \wedge (p, w) \Rightarrow^* (p', w')$ ”
  using assms(2) assms(3)

```

```

proof (induction arbitrary: p q w rule: rtranclp_induct)
  case base
  then have “ $(\text{Init } p, w, q) \in \text{LTS.trans\_star } A \wedge (p, w) \Rightarrow^* (p, w)$ ”
    by auto
  then show ?case
    by auto
next

```

```

case (step Aminus1 Ai)

from step(2) obtain p1  $\gamma$  p2 w2 q' where p1_ $\gamma$ _p2_w2_q'_p:
  “ $Ai = \text{Aminus1} \cup \{(\text{Init } p1, \gamma, q')\}$ ”
  “ $(p1, \gamma) \leftrightarrow (p2, w2)$ ”
  “ $(\text{Init } p2, \text{lbl } w2, q') \in \text{LTS.trans\_star } \text{Aminus1}$ ”
  “ $(\text{Init } p1, \gamma, q') \notin \text{Aminus1}$ ”
  by (meson pre_star_rule.cases)

```

```

define t :: “ $(('ctr\_loc, 'noninit, 'label) \text{state}, 'label) \text{transition}$ ”
  where “ $t = (\text{Init } p1, \gamma, q')$ ”

```

```

obtain ss where ss_p: “ $(\text{Init } p, w, ss, q) \in \text{LTS.trans\_star\_states } Ai$ ”
  using step(4) LTS.trans_star_trans_star_states by metis

```

```

define j where “ $j = \text{count } (\text{transitions\_of } (\text{Init } p, w, ss, q)) t$ ”

```

```

from j_def ss_p show ?case
proof (induction j arbitrary: p q w ss)
  case 0
  then have “ $(\text{Init } p, w, q) \in \text{LTS.trans\_star } \text{Aminus1}$ ”
    using count_zero_remove_trans_star_states_trans_star p1_ $\gamma$ _p2_w2_q'_p(1) t_def by metis
  then show ?case
    using step.IH by metis
next
  case (Suc j')
  have “ $\exists u v u\_ss v\_ss.$ ”

```



$ss = u\_ss @ v\_ss \wedge w = u @ [\gamma] @ v \wedge$   
 $(Init\ p, u, u\_ss, Init\ p1) \in LTS.trans\_star\_states\ Aminus1 \wedge$   
 $(Init\ p1, [\gamma], q') \in LTS.trans\_star\ Ai \wedge$   
 $(q', v, v\_ss, q) \in LTS.trans\_star\_states\ Ai \wedge$   
 $(Init\ p, w, ss, q) = ((Init\ p, u, u\_ss, Init\ p1), \gamma) @ @^\gamma (q', v, v\_ss, q)$ ”  
**using** *split\_at\_first\_t* [of “Init p” w ss q Ai j’ “Init p1”  $\gamma$  q’ Aminus1]  
**using** *Suc(2,3) t\_def p1\_γ\_p2\_w2\_q'\_p(1,4) t\_def* **by** *auto*  
**then obtain**  $u\ v\ u\_ss\ v\_ss$  **where**  $u\ v\ u\_ss\ v\_ss\ p$ :  
“ $ss = u\_ss @ v\_ss \wedge w = u @ [\gamma] @ v$ ”  
“ $(Init\ p, u, u\_ss, Init\ p1) \in LTS.trans\_star\_states\ Aminus1$ ”  
“ $(Init\ p1, [\gamma], q') \in LTS.trans\_star\ Ai$ ”  
“ $(q', v, v\_ss, q) \in LTS.trans\_star\_states\ Ai$ ”  
“ $(Init\ p, w, ss, q) = ((Init\ p, u, u\_ss, Init\ p1), \gamma) @ @^\gamma (q', v, v\_ss, q)$ ”  
**by** *blast*  
**from** *this(2)* **have** “ $\exists p''\ w''.$   $(Init\ p'', w'', Init\ p1) \in LTS.trans\_star\ A \wedge (p, u) \Rightarrow^* (p'', w'')$ ”  
**using** *Suc(1)* [of  $p\ u$  “Init p1”] *step.IH step.prem(1)*  
**by** (*meson LTS.trans\\_star\\_states\\_trans\\_star LTS.trans\\_star\\_trans\\_star\\_states*)  
**from** *this this(1)* **have** *VIII*: “ $(p, u) \Rightarrow^* (p1, [])$ ”  
**using** *word\_into\_init\_empty\_assms(1)* **by** *blast*

**note** *IX* =  $p1\_γ\_p2\_w2\_q'_p(2)$   
**note** *III* =  $p1\_γ\_p2\_w2\_q'_p(3)$   
**from** *III* **have** *III\_2*: “ $\exists w2\_ss.$   $(Init\ p2, lbl\ w2, w2\_ss, q') \in LTS.trans\_star\_states\ Aminus1$ ”  
**using** *LTS.trans\\_star\\_trans\\_star\\_states* [of “Init p2” “lbl w2” q’ Aminus1] **by** *auto*  
**then obtain**  $w2\_ss$  **where** *III\_2*: “ $(Init\ p2, lbl\ w2, w2\_ss, q') \in LTS.trans\_star\_states\ Aminus1$ ”  
**by** *blast*

**from** *III* **have** *V*:  
“ $(Init\ p2, lbl\ w2, w2\_ss, q') \in LTS.trans\_star\_states\ Aminus1$ ”  
“ $(q', v, v\_ss, q) \in LTS.trans\_star\_states\ Ai$ ”  
**using** *III\_2*  $\langle (q', v, v\_ss, q) \in LTS.trans\_star\_states\ Ai \rangle$  **by** *auto*

**define**  $w2v$  **where** “ $w2v = lbl\ w2 @ v$ ”  
**define**  $w2v\_ss$  **where** “ $w2v\_ss = w2\_ss @ tl\ v\_ss$ ”

**from** *V(1)* **have** “ $(Init\ p2, lbl\ w2, w2\_ss, q') \in LTS.trans\_star\_states\ Ai$ ”  
**using** *trans\\_star\\_states\\_mono p1\_γ\_p2\_w2\_q'\_p(1)* **using** *Un\_iff subsetI* **by** (*metis (no\_types)*)  
**then have** *V\_merged*: “ $(Init\ p2, w2v, w2v\_ss, q) \in LTS.trans\_star\_states\ Ai$ ”  
**using** *V(2) unfolding w2v\_def w2v\_ss\_def* **by** (*meson LTS.trans\\_star\\_states\\_append*)

**have**  $j'_count$ : “ $j' = count\ (transitions\_of'\ (Init\ p2, w2v, w2v\_ss, q))\ t$ ”  
**proof** –  
**define** *countts* **where**  
“ $countts == \lambda x. count\ (transitions\_of'\ x)\ t$ ”

**have** “ $countts\ (Init\ p, w, ss, q) = Suc\ j'$ ”  
**using** *Suc.prem(1) countts\_def* **by** *force*  
**moreover**  
**have** “ $countts\ (Init\ p, u, u\_ss, Init\ p1) = 0$ ”  
**using** *LTS.avoid\_count\_zero countts\_def p1\_γ\_p2\_w2\_q'\_p(4) t\_def u\_v\_u\_ss\_v\_ss\_p(2)*  
**by** *fastforce*  
**moreover**  
**from**  $u\ v\ u\_ss\ v\_ss\ p(5)$  **have** “ $countts\ (Init\ p, w, ss, q) = countts\ (Init\ p, u, u\_ss, Init\ p1) + 1 + countts$   
 $(q', v, v\_ss, q)$ ”  
**using** *count\_combine\_trans\_star\_states countts\_def t\_def u\_v\_u\_ss\_v\_ss\_p(2)*  
 $u\ v\ u\_ss\ v\_ss\ p(4)$  **by** *fastforce*

**ultimately**  
**have** “ $Suc\ j' = 0 + 1 + countts\ (q', v, v\_ss, q)$ ”  
**by** *auto*  
**then have** “ $j' = countts\ (q', v, v\_ss, q)$ ”  
**by** *auto*  
**moreover**  
**have** “ $countts\ (Init\ p2, lbl\ w2, w2\_ss, q') = 0$ ”

```

using III_2 LTS.avoid_count_zero countts_def p1_γ_p2_w2_q'_p(4) t_def by fastforce
moreover
have “(Init p2, w2v, w2v_ss, q) = (Init p2, lbl w2, w2_ss, q') @@' (q', v, v_ss, q)”
  using w2v_def w2v_ss_def by auto
then have “countts (Init p2, w2v, w2v_ss, q) = countts (Init p2, lbl w2, w2_ss, q') + countts (q', v, v_ss, q)”
  using ‹(Init p2, lbl w2, w2_ss, q') ∈ LTS.trans_star_states Ai›
  count_append_trans_star_states countts_def t_def u_v_u_ss_v_ss_p(4) by fastforce
ultimately
show ?thesis
  by (simp add: countts_def)
qed

have “∃ p' w'. (Init p', w', q) ∈ LTS.trans_star A ∧ (p2, w2v) ⇒* (p', w)”
  using Suc(1) using j'_count V_merged by auto
then obtain p' w' where p'_w'_p: “(Init p', w', q) ∈ LTS.trans_star A” “(p2, w2v) ⇒* (p', w)”
  by blast

note X = p'_w'_p(2)

have “(p, w) = (p, u@[γ]@v)”
  using ‹ss = u_ss @ v_ss ∧ w = u @ [γ] @ v› by blast

have “(p, u@[γ]@v) ⇒* (p1, γ#v)”
  using VIII_step_relp_append_empty by auto

from X have “(p1, γ#v) ⇒ (p2, w2v)”
  by (metis IX LTS.step_relp_def transition_rel.intros w2v_def)

from X have
  “(p2, w2v) ⇒* (p', w)”
  by simp

have “(p, w) ⇒* (p', w)”
  using X ‹(p, u @ [γ] @ v) ⇒* (p1, γ # v)› ‹(p, w) = (p, u @ [γ] @ v)› ‹(p1, γ # v) ⇒ (p2, w2v)› by auto

then have “(Init p', w', q) ∈ LTS.trans_star A ∧ (p, w) ⇒* (p', w)”
  using p'_w'_p(1) by auto
then show ?case
  by metis
qed
qed

```

```

lemma lemma_3_2_a:
  assumes “inits ⊆ LTS.srccs A”
  assumes “saturation pre_star_rule A A'”
  assumes “(Init p, w, q) ∈ LTS.trans_star A'”
  shows “∃ p' w'. (Init p', w', q) ∈ LTS.trans_star A ∧ (p, w) ⇒* (p', w)”
  using assms lemma_3_2_a' saturation_def by metis

```

— Corresponds to one direction of Schwoon’s theorem 3.2

```

theorem pre_star_rule_subset_pre_star_lang:
  assumes “inits ⊆ LTS.srccs A”
  assumes “pre_star_rule** A A'”
  shows “{c. accepts A' c} ⊆ pre_star (lang A)”
proof
  fix c :: “'ctr_loc × 'label list”
  assume c_a: “c ∈ {w. accepts A' w}”
  define p where “p = fst c”
  define w where “w = snd c”
  from p_def w_def c_a have “accepts A' (p, w)”
    by auto
  then have “∃ q ∈ finals. (Init p, w, q) ∈ LTS.trans_star A'”
    unfolding accepts_def by auto

```

**then obtain**  $q$  **where**  $q\_p$ : “ $q \in \text{finals}$ ” “ $(\text{Init } p, w, q) \in \text{LTS.trans\_star } A'$ ”  
**by** *auto*  
**then have** “ $\exists p' w'. (p, w) \Rightarrow^* (p', w') \wedge (\text{Init } p', w', q) \in \text{LTS.trans\_star } A'$ ”  
**using** *lemma\_3\_2\_a' assms(1) assms(2)* **by** *metis*  
**then obtain**  $p' w'$  **where**  $p'_w'_p$ : “ $(p, w) \Rightarrow^* (p', w')$ ” “ $(\text{Init } p', w', q) \in \text{LTS.trans\_star } A'$ ”  
**by** *auto*  
**then have** “ $(p', w') \in \text{lang } A'$ ”  
**unfolding** *lang\_def* **unfolding** *accepts\_def* **using**  $q\_p(1)$  **by** *auto*  
**then have** “ $(p, w) \in \text{pre\_star } (\text{lang } A')$ ”  
**unfolding** *pre\\_star\_def* **using**  $p'_w'_p(1)$  **by** *auto*  
**then show** “ $c \in \text{pre\_star } (\text{lang } A')$ ”  
**unfolding** *p\_def w\_def* **by** *auto*  
**qed**

— Corresponds to Schwoon’s theorem 3.2

**theorem** *pre\_star\_rule\_accepts\_correct*:

**assumes** “ $\text{inits} \subseteq \text{LTS.srscs } A'$ ”  
**assumes** “ $\text{saturation pre\_star\_rule } A A'$ ”  
**shows** “ $\{c. \text{accepts } A' c\} = \text{pre\_star } (\text{lang } A')$ ”

**proof** (*rule; rule*)

**fix**  $c$  :: “ $'ctr\_loc \times 'label \text{ list}$ ”  
**define**  $p$  **where** “ $p = \text{fst } c$ ”  
**define**  $w$  **where** “ $w = \text{snd } c$ ”  
**assume** “ $c \in \text{pre\_star } (\text{lang } A')$ ”  
**then have** “ $(p, w) \in \text{pre\_star } (\text{lang } A')$ ”  
**unfolding** *p\_def w\_def* **by** *auto*  
**then have** “ $\exists p' w'. (p', w') \in \text{lang } A' \wedge (p, w) \Rightarrow^* (p', w')$ ”  
**unfolding** *pre\\_star\_def* **by** *force*  
**then obtain**  $p' w'$  **where** “ $(p', w') \in \text{lang } A' \wedge (p, w) \Rightarrow^* (p', w')$ ”  
**by** *auto*  
**then have** “ $\exists q \in \text{finals}. (\text{Init } p, w, q) \in \text{LTS.trans\_star } A'$ ”  
**using** *lemma\_3\_1 assms(2)* **unfolding** *accepts\_def* **by** *force*  
**then have** “ $\text{accepts } A' (p, w)$ ”  
**unfolding** *accepts\_def* **by** *auto*  
**then show** “ $c \in \{c. \text{accepts } A' c\}$ ”  
**using** *p\_def w\_def* **by** *auto*

**next**

**fix**  $c$  :: “ $'ctr\_loc \times 'label \text{ list}$ ”  
**assume** “ $c \in \{w. \text{accepts } A' w\}$ ”  
**then show** “ $c \in \text{pre\_star } (\text{lang } A')$ ”  
**using** *pre\_star\_rule\_subset\_pre\_star\_lang assms* **unfolding** *saturation\_def* **by** *auto*

**qed**

— Corresponds to Schwoon’s theorem 3.2

**theorem** *pre\_star\_rule\_correct*:

**assumes** “ $\text{inits} \subseteq \text{LTS.srscs } A'$ ”  
**assumes** “ $\text{saturation pre\_star\_rule } A A'$ ”  
**shows** “ $\text{lang } A' = \text{pre\_star } (\text{lang } A')$ ”  
**using** *assms(1) assms(2) lang\_def pre\_star\_rule\_accepts\_correct* **by** *auto*

**theorem** *pre\_star\_exec\_accepts\_correct*:

**assumes** “ $\text{inits} \subseteq \text{LTS.srscs } A'$ ”  
**shows** “ $\{c. \text{accepts } (\text{pre\_star\_exec } A) c\} = \text{pre\_star } (\text{lang } A')$ ”  
**using** *pre\_star\_rule\_accepts\_correct[of A “pre\_star\_exec A”] saturation\_pre\_star\_exec[of A] assms* **by** *auto*

**theorem** *pre\_star\_exec\_lang\_correct*:

**assumes** “ $\text{inits} \subseteq \text{LTS.srscs } A'$ ”  
**shows** “ $\text{lang } (\text{pre\_star\_exec } A) = \text{pre\_star } (\text{lang } A')$ ”  
**using** *pre\_star\_rule\_correct[of A “pre\_star\_exec A”] saturation\_pre\_star\_exec[of A] assms* **by** *auto*

**theorem** *pre\_star\_exec\_check\_accepts\_correct*:

**assumes** “ $\text{pre\_star\_exec\_check } A \neq \text{None}$ ”

shows “ $\{c. \text{ accepts } (\text{the } (\text{pre\_star\_exec\_check } A)) \ c\} = \text{pre\_star } (\text{lang } A)$ ”  
 using `pre_star_exec_accepts_correct` `assms` **unfolding** `pre_star_exec_check_def` `pre_star_exec_def`  
 by (`auto split: if_splits`)

**theorem** `pre_star_exec_check_correct`:  
 assumes “`pre_star_exec_check A ≠ None`”  
 shows “`lang (the (pre_star_exec_check A)) = pre_star (lang A)`”  
 using `pre_star_exec_check_accepts_correct` `assms` **unfolding** `lang_def` **by** `auto`

**theorem** `accept_pre_star_exec_correct_True`:  
 assumes “`inits ⊆ LTS.sracs A`”  
 assumes “`accepts (pre_star_exec A) c`”  
 shows “`c ∈ pre_star (lang A)`”  
 using `pre_star_exec_accepts_correct` `assms(1)` `assms(2)` **by** `blast`

**theorem** `accept_pre_star_exec_correct_False`:  
 assumes “`inits ⊆ LTS.sracs A`”  
 assumes “`¬accepts (pre_star_exec A) c`”  
 shows “`c ∉ pre_star (lang A)`”  
 using `pre_star_exec_accepts_correct` `assms(1)` `assms(2)` **by** `blast`

**theorem** `accept_pre_star_exec_correct_Some_True`:  
 assumes “`accept_pre_star_exec_check A c = Some True`”  
 shows “`c ∈ pre_star (lang A)`”

**proof** –  
 have “`inits ⊆ LTS.sracs A`”  
 using `assms` **unfolding** `accept_pre_star_exec_check_def`  
 by (`auto split: if_splits`)  
 moreover  
 have “`accepts (pre_star_exec A) c`”  
 using `assms`  
 using `accept_pre_star_exec_check_def` `calculation` **by** `auto`  
 ultimately  
 show “`c ∈ pre_star (lang A)`”  
 using `accept_pre_star_exec_correct_True` **by** `auto`

**qed**

**theorem** `accept_pre_star_exec_correct_Some_False`:  
 assumes “`accept_pre_star_exec_check A c = Some False`”  
 shows “`c ∉ pre_star (lang A)`”

**proof** –  
 have “`inits ⊆ LTS.sracs A`”  
 using `assms` **unfolding** `accept_pre_star_exec_check_def`  
 by (`auto split: if_splits`)  
 moreover  
 have “`¬accepts (pre_star_exec A) c`”  
 using `assms`  
 using `accept_pre_star_exec_check_def` `calculation` **by** `auto`  
 ultimately  
 show “`c ∉ pre_star (lang A)`”  
 using `accept_pre_star_exec_correct_False` **by** `auto`

**qed**

**theorem** `accept_pre_star_exec_correct_None`:  
 assumes “`accept_pre_star_exec_check A c = None`”  
 shows “`¬inits ⊆ LTS.sracs A`”  
 using `assms` **unfolding** `accept_pre_star_exec_check_def` **by** `auto`

## 5.4 Post\* lemmas

**lemma** `lemma_3_3'`:  
 assumes “`pv ⇒* p'w`”  
 and “`(fst pv, snd pv) ∈ lang_ε A`”  
 and “`saturation post_star_rules A A'`”

**shows** “ $\text{accepts}_\varepsilon A' (\text{fst } p'w, \text{snd } p'w)$ ”  
**using** *assms*  
**proof** (*induct arbitrary: pv rule: rtranclp\_induct*)  
**case** *base*  
**show** *?case*  
**using** *assms post\_star\_lim'\_incr\_trans\_star\_\varepsilon*  
**by** (*auto simp: lang\_\varepsilon\_def accepts\_\varepsilon\_def*)  
**next**  
**case** (*step p''u p'w*)  
**define** *p'* **where** “ $p' = \text{fst } p'w$ ”  
**define** *w* **where** “ $w = \text{snd } p'w$ ”  
**define** *p''* **where** “ $p'' = \text{fst } p''u$ ”  
**define** *u* **where** “ $u = \text{snd } p''u$ ”  
**have** *p'w\_def*: “ $p'w = (p', w)$ ”  
**using** *p'\_def w\_def* **by** *auto*  
**have** *p''u\_def*: “ $p''u = (p'', u)$ ”  
**using** *p''\_def u\_def* **by** *auto*  
  
**then** **have** “ $\text{accepts}_\varepsilon A' (p'', u)$ ”  
**using** *assms(2) p''\_def step.hyps(3) step.prem(2) u\_def* **by** *metis*  
**then** **have** “ $\exists q. q \in \text{finals} \wedge (\text{Init } p'', u, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**by** (*auto simp: accepts\_\varepsilon\_def*)  
**then** **obtain** *q* **where** *q\_p*: “ $q \in \text{finals} \wedge (\text{Init } p'', u, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**by** *metis*  
**then** **have** “ $\exists u_\varepsilon. q \in \text{finals} \wedge \text{LTS}_\varepsilon.\varepsilon\_exp u_\varepsilon u \wedge (\text{Init } p'', u_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**using** *LTS\_\varepsilon.trans\_star\_\varepsilon\_iff\_\varepsilon\_exp\_trans\_star*[of “*Init p''*” *u q A'*] **by** *auto*  
**then** **obtain** *u\_\varepsilon* **where** *II*: “ $q \in \text{finals}$ ” “ $\text{LTS}_\varepsilon.\varepsilon\_exp u_\varepsilon u$ ” “ $(\text{Init } p'', u_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**by** *blast*  
**have** “ $\exists \gamma u1 w1. u = \gamma \# u1 \wedge w = \text{lbl } w1 @ u1 \wedge (p'', \gamma) \hookrightarrow (p', w1)$ ”  
**using** *p''\_def p'w\_def step.hyps(2) step\_relp\_def2* **by** *auto*  
**then** **obtain**  $\gamma u1 w1$  **where** *III*: “ $u = \gamma \# u1$ ” “ $w = \text{lbl } w1 @ u1$ ” “ $(p'', \gamma) \hookrightarrow (p', w1)$ ”  
**by** *blast*  
  
**have** *p'\_inits*: “ $\text{Init } p' \in \text{inits}$ ”  
**unfolding** *inits\_def* **by** *auto*  
**have** *p''\_inits*: “ $\text{Init } p'' \in \text{inits}$ ”  
**unfolding** *inits\_def* **by** *auto*  
  
**have** “ $\exists \gamma_\varepsilon u1_\varepsilon. \text{LTS}_\varepsilon.\varepsilon\_exp \gamma_\varepsilon [\gamma] \wedge \text{LTS}_\varepsilon.\varepsilon\_exp u1_\varepsilon u1 \wedge (\text{Init } p'', \gamma_\varepsilon @ u1_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**proof** –  
**have** “ $\exists \gamma_\varepsilon u1_\varepsilon. \text{LTS}_\varepsilon.\varepsilon\_exp \gamma_\varepsilon [\gamma] \wedge \text{LTS}_\varepsilon.\varepsilon\_exp u1_\varepsilon u1 \wedge u_\varepsilon = \gamma_\varepsilon @ u1_\varepsilon$ ”  
**using** *LTS\_\varepsilon.\varepsilon\_exp\_split'*[of *u\_\varepsilon \gamma u1*] *II(2) III(1)* **by** *auto*  
**then** **obtain**  $\gamma_\varepsilon u1_\varepsilon$  **where** “ $\text{LTS}_\varepsilon.\varepsilon\_exp \gamma_\varepsilon [\gamma] \wedge \text{LTS}_\varepsilon.\varepsilon\_exp u1_\varepsilon u1 \wedge u_\varepsilon = \gamma_\varepsilon @ u1_\varepsilon$ ”  
**by** *auto*  
**then** **have** “ $(\text{Init } p'', \gamma_\varepsilon @ u1_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**using** *II(3)* **by** *auto*  
**then** **show** *?thesis*  
**using**  $\langle \text{LTS}_\varepsilon.\varepsilon\_exp \gamma_\varepsilon [\gamma] \wedge \text{LTS}_\varepsilon.\varepsilon\_exp u1_\varepsilon u1 \wedge u_\varepsilon = \gamma_\varepsilon @ u1_\varepsilon \rangle$  **by** *blast*  
**qed**  
**then** **obtain**  $\gamma_\varepsilon u1_\varepsilon$  **where**  
*iii*: “ $\text{LTS}_\varepsilon.\varepsilon\_exp \gamma_\varepsilon [\gamma]$ ” **and**  
*iv*: “ $\text{LTS}_\varepsilon.\varepsilon\_exp u1_\varepsilon u1$ ” “ $(\text{Init } p'', \gamma_\varepsilon @ u1_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**by** *blast*  
**then** **have** *VI*: “ $\exists q1. (\text{Init } p'', \gamma_\varepsilon, q1) \in \text{LTS}.\text{trans\_star } A' \wedge (q1, u1_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**by** (*simp add: LTS.trans\_star\_split*)  
**then** **obtain** *q1* **where** *VI*: “ $(\text{Init } p'', \gamma_\varepsilon, q1) \in \text{LTS}.\text{trans\_star } A'$ ” “ $(q1, u1_\varepsilon, q) \in \text{LTS}.\text{trans\_star } A'$ ”  
**by** *blast*  
  
**then** **have** *VI\_2*: “ $(\text{Init } p'', [\gamma], q1) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ” “ $(q1, u1, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**by** (*meson LTS\_\varepsilon.trans\_star\_\varepsilon\_iff\_\varepsilon\_exp\_trans\_star iii VI(2) iv(1)*)  
  
**show** *?case*

**proof** (*cases w1*)  
**case** *pop*  
**then have**  $r$ : “ $(p'', \gamma) \leftrightarrow (p', \text{pop})$ ”  
**using** *III(3)* **by** *blast*  
**then have** “ $(\text{Init } p', \varepsilon, q1) \in A'$ ”  
**using** *VI\_2(1)* *add\_trans\_pop* *assms saturated\_def saturation\_def p'\_inits* **by** *metis*  
**then have** “ $(\text{Init } p', w, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**using** *III(2)* *VI\_2(2)* *pop LTS\_\varepsilon.trans\\_star\_\varepsilon.trans\\_star\_\varepsilon\_step\_\varepsilon* **by** *fastforce*  
**then have** “*accepts\_\varepsilon*  $A' (p', w)$ ”  
**unfolding** *accepts\_\varepsilon\_def* **using** *II(1)* **by** *blast*  
**then show** *?thesis*  
**using** *p'\_def w\_def* **by** *force*

**next**  
**case** (*swap*  $\gamma'$ )  
**then have**  $r$ : “ $(p'', \gamma) \leftrightarrow (p', \text{swap } \gamma')$ ”  
**using** *III(3)* **by** *blast*  
**then have** “ $(\text{Init } p', \text{Some } \gamma', q1) \in A'$ ”  
**by** (*metis* *VI\_2(1)* *add\_trans\_swap* *assms(3)* *saturated\_def saturation\_def*)  
**have** “ $(\text{Init } p', w, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**using** *III(2)* *LTS\_\varepsilon.trans\\_star\_\varepsilon.trans\\_star\_\varepsilon\_step\_\gamma* *VI\_2(2)* *append\_Cons append\_self\_conv2*  
*lbl.simps(3)* *swap*  $\langle (\text{Init } p', \text{Some } \gamma', q1) \in A' \rangle$  **by** *fastforce*  
**then have** “*accepts\_\varepsilon*  $A' (p', w)$ ”  
**unfolding** *accepts\_\varepsilon\_def*  
**using** *II(1)* **by** *blast*  
**then show** *?thesis*  
**using** *p'\_def w\_def* **by** *force*

**next**  
**case** (*push*  $\gamma' \gamma''$ )  
**then have**  $r$ : “ $(p'', \gamma) \leftrightarrow (p', \text{push } \gamma' \gamma'')$ ”  
**using** *III(3)* **by** *blast*  
**from** *this* *VI\_2* *iii* *post\_star\_rules.intros(3)*[*OF this, of q1 A', OF VI\_2(1)*]  
**have** “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \in A'$ ”  
**using** *assms(3)* **by** (*meson* *saturated\_def saturation\_def*)  
**from** *this*  $r$  *VI\_2* *iii* *post\_star\_rules.intros(4)*[*OF r, of q1 A', OF VI\_2(1)*]  
**have** “ $(\text{Isolated } p' \gamma', \text{Some } \gamma'', q1) \in A'$ ”  
**using** *assms(3)* **using** *saturated\_def saturation\_def* **by** *metis*  
**have** “ $(\text{Init } p', [\gamma], \text{Isolated } p' \gamma') \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A' \wedge$   
 $(\text{Isolated } p' \gamma', [\gamma''], q1) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A' \wedge$   
 $(q1, u1, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**by** (*metis* *LTS\_\varepsilon.trans\\_star\_\varepsilon.simps* *VI\_2(2)*)  $\langle (\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \in A' \rangle$   
 $\langle (\text{Isolated } p' \gamma', \text{Some } \gamma'', q1) \in A' \rangle$   
**have** “ $(\text{Init } p', w, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A'$ ”  
**using** *III(2)* *VI\_2(2)*  $\langle (\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \in A' \rangle$   
 $\langle (\text{Isolated } p' \gamma', \text{Some } \gamma'', q1) \in A' \rangle$  *push* *LTS\_\varepsilon.append\_edge\_edge\_trans\_star\_\varepsilon* **by** *auto*  
**then have** “*accepts\_\varepsilon*  $A' (p', w)$ ”  
**unfolding** *accepts\_\varepsilon\_def*  
**using** *II(1)* **by** *blast*  
**then show** *?thesis*  
**using** *p'\_def w\_def* **by** *force*

**qed**  
**qed**

**lemma** *lemma\_3\_3*:  
**assumes** “ $(p, v) \Rightarrow^* (p', w)$ ”  
**and** “ $(p, v) \in \text{lang}_\varepsilon A$ ”  
**and** “*saturation post\_star\_rules*  $A A'$ ”  
**shows** “*accepts\_\varepsilon*  $A' (p', w)$ ”  
**using** *assms lemma\_3\_3'* **by** *force*

**lemma** *init\_only\_hd*:  
**assumes** “ $(ss, w) \in \text{LTS.path\_with\_word } A$ ”  
**assumes** “*inits*  $\subseteq \text{LTS.srcs } A$ ”

**assumes** “count (transitions\_of (ss, w)) t > 0”  
**assumes** “t = (Init p1,  $\gamma$ , q1)”  
**shows** “hd (transition\_list (ss, w)) = t  $\wedge$  count (transitions\_of (ss, w)) t = 1”  
**using** *assms LTS.source\_only\_hd* **by** (metis *LTS.sracs\_def2 inits\_sracs\_iff\_Ctr\_Loc\_sracs*)

**lemma** *no\_edge\_to\_Ctr\_Loc\_avoid\_Ctr\_Loc*:  
**assumes** “(p, w, qq)  $\in$  *LTS.trans\_star Aminus1*”  
**assumes** “w  $\neq$  []”  
**assumes** “inits  $\subseteq$  *LTS.sracs Aminus1*”  
**shows** “qq  $\notin$  inits”  
**using** *assms LTS.no\_end\_in\_source* **by** (metis *subset\_iff*)

**lemma** *no\_edge\_to\_Ctr\_Loc\_avoid\_Ctr\_Loc\_ε*:  
**assumes** “(p, [ $\gamma$ ], qq)  $\in$  *LTS.ε.trans\_star\_ε Aminus1*”  
**assumes** “inits  $\subseteq$  *LTS.sracs Aminus1*”  
**shows** “qq  $\notin$  inits”  
**using** *assms LTS.ε.no\_edge\_to\_source\_ε* **by** (metis *subset\_iff*)

**lemma** *no\_edge\_to\_Ctr\_Loc\_post\_star\_rules'*:  
**assumes** “*post\_star\_rules*\* A Ai”  
**assumes** “ $\nexists$  q  $\gamma$  q'. (q,  $\gamma$ , Init q')  $\in$  A”  
**shows** “ $\nexists$  q  $\gamma$  q'. (q,  $\gamma$ , Init q')  $\in$  Ai”  
**using** *assms*  
**proof** (induction rule: *rtranclp\_induct*)  
  **case** base  
  **then show** ?case **by** auto  
**next**  
  **case** (step *Aminus1 Ai*)  
  **then have** *ind*: “ $\nexists$  q  $\gamma$  q'. (q,  $\gamma$ , Init q')  $\in$  *Aminus1*”  
  **by** auto  
  **from** *step(2)* **show** ?case  
  **proof** (cases rule: *post\_star\_rules.cases*)  
  **case** (add\_trans\_pop p  $\gamma$  p' q)  
  **have** “q  $\notin$  inits”  
  **using** *ind no\_edge\_to\_Ctr\_Loc\_avoid\_Ctr\_Loc\_ε inits\_sracs\_iff\_Ctr\_Loc\_sracs*  
  **by** (metis *local.add\_trans\_pop(3)*)  
  **then have** “ $\nexists$  qq. q = Init qq”  
  **by** (simp add: *inits\_def is\_Init\_def*)  
  **then show** ?thesis  
  **using** *ind local.add\_trans\_pop(1)* **by** auto  
  **next**  
  **case** (add\_trans\_swap p  $\gamma$  p'  $\gamma'$  q)  
  **have** “q  $\notin$  inits”  
  **using** *add\_trans\_swap ind no\_edge\_to\_Ctr\_Loc\_avoid\_Ctr\_Loc\_ε inits\_sracs\_iff\_Ctr\_Loc\_sracs*  
  **by** metis  
  **then have** “ $\nexists$  qq. q = Init qq”  
  **by** (simp add: *inits\_def is\_Init\_def*)  
  **then show** ?thesis  
  **using** *ind local.add\_trans\_swap(1)* **by** auto  
  **next**  
  **case** (add\_trans\_push\_1 p  $\gamma$  p'  $\gamma'$   $\gamma''$  q)  
  **have** “q  $\notin$  inits”  
  **using** *add\_trans\_push\_1 ind no\_edge\_to\_Ctr\_Loc\_avoid\_Ctr\_Loc\_ε inits\_sracs\_iff\_Ctr\_Loc\_sracs*  
  **by** metis  
  **then have** “ $\nexists$  qq. q = Init qq”  
  **by** (simp add: *inits\_def is\_Init\_def*)  
  **then show** ?thesis  
  **using** *ind local.add\_trans\_push\_1(1)* **by** auto  
  **next**  
  **case** (add\_trans\_push\_2 p  $\gamma$  p'  $\gamma'$   $\gamma''$  q)  
  **have** “q  $\notin$  inits”  
  **using** *add\_trans\_push\_2 ind no\_edge\_to\_Ctr\_Loc\_avoid\_Ctr\_Loc\_ε inits\_sracs\_iff\_Ctr\_Loc\_sracs*  
  **by** metis

```

then have “ $\nexists qq. q = \text{Init } qq$ ”
  by (simp add: inits_def is_Init_def)
then show ?thesis
  using ind local.add_trans_push_2(1) by auto
qed
qed

```

```

lemma no_edge_to_Ctr_Loc_post_star_rules:
  assumes “ $\text{post\_star\_rules}^{**} A Ai$ ”
  assumes “ $\text{inits} \subseteq \text{LTS.srcs } A$ ”
  shows “ $\text{inits} \subseteq \text{LTS.srcs } Ai$ ”
  using assms no_edge_to_Ctr_Loc_post_star_rules' inits_srcs_iff_Ctr_Loc_srcs by metis

```

```

lemma source_and_sink_isolated:
  assumes “ $N \subseteq \text{LTS.srcs } A$ ”
  assumes “ $N \subseteq \text{LTS.sinks } A$ ”
  shows “ $\forall p \gamma q. (p, \gamma, q) \in A \longrightarrow p \notin N \wedge q \notin N$ ”
  by (metis LTS.srcs_def2 LTS.sinks_def2 assms(1) assms(2) in_mono)

```

```

lemma post_star_rules_Isolated_source_invariant':

```

```

  assumes “ $\text{post\_star\_rules}^{**} A A'$ ”
  assumes “ $\text{isols} \subseteq \text{LTS.isolated } A$ ”
  assumes “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin A'$ ”
  shows “ $\nexists p \gamma. (p, \gamma, \text{Isolated } p' \gamma') \in A'$ ”
  using assms

```

```

proof (induction rule: rtranclp_induct)

```

```

  case base
  then show ?case
    unfolding isols_def is_Isolated_def using LTS.isolated_no_edges by fastforce

```

```

next

```

```

  case (step Aminus1 Ai)
  from step(2) show ?case
  proof (cases rule: post_star_rules.cases)
    case (add_trans_pop p'''  $\gamma''$  p'' q)
    then have “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin Ai$ ”
      using step.prem(2) by blast
    then have nin: “ $\nexists p \gamma. (p, \gamma, \text{Isolated } p' \gamma') \in Aminus1$ ”
      using local.add_trans_pop(1) step.IH step.prem(1,2) by fastforce
    then have “ $\text{Isolated } p' \gamma' \neq q$ ”
      using add_trans_pop(4) LTS. $\epsilon$ .trans_star_not_to_source  $\epsilon$  LTS.srcs_def2
      by (metis local.add_trans_pop(3) state.distinct(3))
    then have “ $\nexists p \gamma. (p, \gamma, \text{Isolated } p' \gamma') = (\text{Init } p'', \epsilon, q)$ ”
      by auto
    then show ?thesis
      using nin add_trans_pop(1) by auto

```

```

next

```

```

  case (add_trans_swap p''''  $\gamma''$  p''  $\gamma'''$  q)
  then have “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin Ai$ ”
    using step.prem(2) by blast
  then have nin: “ $\nexists p \gamma. (p, \gamma, \text{Isolated } p' \gamma') \in Aminus1$ ”
    using local.add_trans_swap(1) step.IH step.prem(1,2) by fastforce
  then have “ $\text{Isolated } p' \gamma' \neq q$ ”
    using LTS.srcs_def2
    by (metis state.distinct(4) LTS. $\epsilon$ .trans_star_not_to_source  $\epsilon$  local.add_trans_swap(3))
  then have “ $\nexists p \gamma. (p, \gamma, \text{Isolated } p' \gamma') = (\text{Init } p'', \text{Some } \gamma''', q)$ ”
    by auto
  then show ?thesis
    using nin add_trans_swap(1) by auto

```

```

next

```

```

  case (add_trans_push_1 p''''  $\gamma''$  p''  $\gamma'''$   $\gamma''''$  q)
  then have “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin Ai$ ”
    using step.prem(2) by blast
  then show ?thesis

```



```

    using add_trans_push_1(1) Un_iff state.inject(2) prod.inject singleton_iff step.IH
      step.prem(1,2) by blast
next
case (add_trans_push_2 p''''  $\gamma''$  p''  $\gamma'''$   $\gamma''''$  q)
have “(Init p', Some  $\gamma'$ , Isolated p'  $\gamma'$ )  $\notin$  Ai”
  using step.prem(2) .
then have nin: “ $\nexists$  p  $\gamma$ . (p,  $\gamma$ , Isolated p'  $\gamma'$ )  $\in$  Aminus1”
  using local.add_trans_push_2(1) step.IH step.prem(1) by fastforce
then have “Isolated p'  $\gamma' \neq$  q”
  using LTS.srscs_def2 local.add_trans_push_2(3)
  by (metis state.disc(1,3) LTS_ε.trans_star_not_to_source_ε)
then have “ $\nexists$  p  $\gamma$ . (p,  $\gamma$ , Isolated p'  $\gamma'$ ) = (Init p'', ε, q)”
  by auto
then show ?thesis
  using nin add_trans_push_2(1) by auto
qed
qed

lemma post_star_rules_Isolated_source_invariant:
  assumes “post_star_rules** A A'”
  assumes “isols  $\subseteq$  LTS.isolated A”
  assumes “(Init p', Some  $\gamma'$ , Isolated p'  $\gamma'$ )  $\notin$  A'”
  shows “Isolated p'  $\gamma' \in$  LTS.srscs A'”
  by (meson LTS.srscs_def2 assms(1) assms(2) assms(3) post_star_rules_Isolated_source_invariant')

lemma post_star_rules_Isolated_sink_invariant':
  assumes “post_star_rules** A A'”
  assumes “isols  $\subseteq$  LTS.isolated A”
  assumes “(Init p', Some  $\gamma'$ , Isolated p'  $\gamma'$ )  $\notin$  A'”
  shows “ $\nexists$  p  $\gamma$ . (Isolated p'  $\gamma'$ ,  $\gamma$ , p)  $\in$  A'”
  using assms
proof (induction rule: rtranclp_induct)
  case base
  then show ?case
    unfolding isols_def is_Isolated_def
    using LTS.isolated_no_edges by fastforce
next
case (step Aminus1 Ai)
from step(2) show ?case
proof (cases rule: post_star_rules.cases)
  case (add_trans_pop p''''  $\gamma''$  p'' q)
  then have “(Init p', Some  $\gamma'$ , Isolated p'  $\gamma'$ )  $\notin$  Ai”
    using step.prem(2) by blast
  then have nin: “ $\nexists$  p  $\gamma$ . (Isolated p'  $\gamma'$ ,  $\gamma$ , p)  $\in$  Aminus1”
    using local.add_trans_pop(1) step.IH step.prem(1,2) by fastforce
  then have “Isolated p'  $\gamma' \neq$  q”
    using add_trans_pop(4)
    LTS_ε.trans_star_not_to_source_ε[of “Init p''''” “[ $\gamma'$ ”] q Aminus1 “Isolated p'  $\gamma'$ ”]
    post_star_rules_Isolated_source_invariant local.add_trans_pop(1) step.hyps(1) step.prem(1,2)
    UnI1 local.add_trans_pop(3) by (metis (full_types) state.distinct(3))
  then have “ $\nexists$  p  $\gamma$ . (p,  $\gamma$ , Isolated p'  $\gamma'$ ) = (Init p'', ε, q)”
    by auto
  then show ?thesis
    using nin add_trans_pop(1) by auto
next
case (add_trans_swap p''''  $\gamma''$  p''  $\gamma'''$  q)
then have “(Init p', Some  $\gamma'$ , Isolated p'  $\gamma'$ )  $\notin$  Ai”
  using step.prem(2) by blast
then have nin: “ $\nexists$  p  $\gamma$ . (Isolated p'  $\gamma'$ ,  $\gamma$ , p)  $\in$  Aminus1”
  using local.add_trans_swap(1) step.IH step.prem(1,2) by fastforce
then have “Isolated p'  $\gamma' \neq$  q”
  using LTS_ε.trans_star_not_to_source_ε[of “Init p''''” “[ $\gamma'$ ”] q Aminus1]
  local.add_trans_swap(3) post_star_rules_Isolated_source_invariant[of _ Aminus1 p'  $\gamma'$ ] UnCI

```

```

    local.add_trans_swap(1) step.hyps(1) step.prem(1,2) state.simps(7) by metis
  then have “ $\nexists p \gamma. (p, \gamma, \text{Isolated } p' \gamma') = (\text{Init } p'', \text{Some } \gamma''', q)$ ”
    by auto
  then show ?thesis
    using nin add_trans_swap(1) by auto
next
case (add_trans_push_1 p''''  $\gamma''$  p''  $\gamma'''$   $\gamma''''$  q)
then have “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin A_i$ ”
  using step.prem(2) by blast
then show ?thesis
  using add_trans_push_1(1) Un_iff state.inject prod.inject singleton_iff step.IH
  step.prem(1,2) by blast
next
case (add_trans_push_2 p''''  $\gamma''$  p''  $\gamma'''$   $\gamma''''$  q)
have “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin A_i$ ”
  using step.prem(2) by blast
then have nin: “ $\nexists p \gamma. (\text{Isolated } p' \gamma', \gamma, p) \in A_{\text{minus}1}$ ”
  using local.add_trans_push_2(1) step.IH step.prem(1,2) by fastforce
then have “ $\text{Isolated } p' \gamma' \neq q$ ”
  using state.disc(3)
  LTS_ε.trans_star_not_to_source_ε[of “Init p''''” “[ $\gamma'$ ”] q A_{\text{minus}1} “Isolated p'  $\gamma'$ ”]
  local.add_trans_push_2(3)
  using post_star_rules_Isolated_source_invariant[of _ A_{\text{minus}1} p'  $\gamma'$ ] UnCI
  local.add_trans_push_2(1) step.hyps(1) step.prem(1,2) state.disc(1) by metis
then have “ $\nexists p \gamma. (\text{Isolated } p' \gamma', \gamma, p) = (\text{Init } p'', \varepsilon, q)$ ”
  by auto
then show ?thesis
  using nin add_trans_push_2(1)
  using local.add_trans_push_2 step.prem(2) by auto
qed
qed

```

lemma post\_star\_rules\_Isolated\_sink\_invariant:

```

  assumes “post_star_rules* A A'”
  assumes “isols  $\subseteq$  LTS.isolated A”
  assumes “ $(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma') \notin A'$ ”
  shows “ $\text{Isolated } p' \gamma' \in \text{LTS.sinks } A'$ ”
  by (meson LTS.sinks_def2 assms(1,2,3) post_star_rules_Isolated_sink_invariant)

```

— Corresponds to Schwoon’s lemma 3.4

lemma rtranclp\_post\_star\_rules\_contains\_successors\_states:

```

  assumes “post_star_rules* A A'”
  assumes “inits  $\subseteq$  LTS.srcs A”
  assumes “isols  $\subseteq$  LTS.isolated A”
  assumes “ $(\text{Init } p, w, ss, q) \in \text{LTS.trans_star_states } A'$ ”
  shows “ $(\neg \text{is\_Isolated } q \longrightarrow (\exists p' w'. (\text{Init } p', w', q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A \wedge (p', w') \Rightarrow^* (p, \text{LTS}_\varepsilon.\text{remove}_\varepsilon w))) \wedge$ 
    ( $\text{is\_Isolated } q \longrightarrow (\text{the\_Ctr\_Loc } q, [\text{the\_Label } q]) \Rightarrow^* (p, \text{LTS}_\varepsilon.\text{remove}_\varepsilon w))$ ”

```

using assms

proof (induction arbitrary: p q w ss rule: rtranclp\_induct)

case base

{

assume ctr\_loc: “ $\text{is\_Init } q \vee \text{is\_Noninit } q$ ”

then have “ $(\text{Init } p, \text{LTS}_\varepsilon.\text{remove}_\varepsilon w, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A$ ”

using base LTS\_ε.trans\_star\_states\_trans\_star\_ε by metis

then have “ $\exists p' w'. (p', w', q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A$ ”

by auto

then have ?case

using ctr\_loc  $\langle (\text{Init } p, \text{LTS}_\varepsilon.\text{remove}_\varepsilon w, q) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon A \rangle$  by blast

}

moreover

{

```

assume ‐is_Isolated q‐
then have ?case
proof (cases w)
  case Nil
  then have False using base
    using LTS.trans_star_empty LTS.trans_star_states_trans_star ‐is_Isolated q‐
    by (metis state.disc(7))
  then show ?thesis
    by metis
next
case (Cons γ w_rest)
then have ‐(Init p, γ#w_rest, ss, q) ∈ LTS.trans_star_states A‐
  using base Cons by blast
then have ‐∃ s γ'. (s, γ', q) ∈ A‐
  using LTS.trans_star_states_transition_relation by metis
then have False
  using ‐is_Isolated q‐ isols_def base.premis(2) LTS.isolated_no_edges
  by (metis mem_Collect_eq subset_eq)
then show ?thesis
  by auto
qed
}
ultimately
show ?case
  by (meson state.exhaust_disc)
next
case (step Aminus1 Ai)
from step(2) have ‐∃ p1 γ p2 w2 q1. Ai = Aminus1 ∪ {(p1, γ, q1)} ∧ (p1, γ, q1) ∉ Aminus1‐
  by (cases rule: post_star_rules.cases) auto
then obtain p1 γ q1 where p1_γ_p2_w2_q'_p:
  ‐Ai = Aminus1 ∪ {(p1, γ, q1)}‐
  ‐(p1, γ, q1) ∉ Aminus1‐
  by auto

define t where ‐t = (p1, γ, q1)‐
define j where ‐j = count (transitions_of' (Init p, w, ss, q)) t‐

note ss_p = step(6)

from j_def ss_p show ?case
proof (induction j arbitrary: p q w ss)
  case 0
  then have ‐(Init p, w, ss, q) ∈ LTS.trans_star_states Aminus1‐
    using count_zero_remove_path_with_word_trans_star_states p1_γ_p2_w2_q'_p(1) t_def
    by metis
  then show ?case
    using step by auto
next
case (Suc j)
from step(2) show ?case
proof (cases rule: post_star_rules.cases)
  case (add_trans_pop p2 γ2 p1 q1)
  note III = add_trans_pop(3)
  note VI = add_trans_pop(2)
  have t_def: ‐t = (Init p1, ε, q1)‐
    using local.add_trans_pop(1) local.add_trans_pop p1_γ_p2_w2_q'_p(1) t_def by blast
  have init_Ai: ‐inits ⊆ LTS.srcs Ai‐
    using step(1,2) step(4)
    using no_edge_to_Ctr_Loc_post_star_rules
    using r_into_rtranclp by (metis)
  have t_hd_once: ‐hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1‐
  proof –
    have ‐(ss, w) ∈ LTS.path_with_word Ai‐

```

```

    using Suc(3) LTS.trans_star_states_path_with_word by metis
  moreover
  have "inits  $\subseteq$  LTS.srcls Ai"
    using init_Ai by auto
  moreover
  have "0 < count (transitions_of (ss, w)) t"
    by (metis Suc.prem(1) transitions_of'.simps zero_less_Suc)
  moreover
  have "t = (Init p1,  $\varepsilon$ , q1)"
    using t_def by auto
  moreover
  have "Init p1  $\in$  inits"
    by (simp add: inits_def)
  ultimately
  show "hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1"
    using init_only_hd[of ss w Ai t p1  $\varepsilon$  q1] by auto
qed

have "transition_list (ss, w)  $\neq$  []"
  by (metis LTS.trans_star_states_path_with_word LTS.path_with_word.simps Suc.prem(1)
    Suc.prem(2) count_empty_less_not_refl2 list.distinct(1) transition_list.simps(1)
    transitions_of'.simps transitions_of'.simps(2) zero_less_Suc)
then have ss_w_split: "([Init p1, q1], [ $\varepsilon$ ]) @' (tl ss, tl w) = (ss, w)"
  using t_hd_once t_def hd_transition_list_append_path_with_word by metis
then have ss_w_split': "(Init p1, [ $\varepsilon$ ], [Init p1, q1], q1) @@@' (q1, tl w, tl ss, q) = (Init p1, w, ss, q)"
  by auto
have VII: "p = p1"
proof -
  have "(Init p, w, ss, q)  $\in$  LTS.trans_star_states Ai"
    using Suc.prem(2) by blast
  moreover
  have "t = hd (transition_list' (Init p, w, ss, q))"
    using <hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1>
    by fastforce
  moreover
  have "transition_list' (Init p, w, ss, q)  $\neq$  []"
    by (simp add: <transition_list (ss, w)  $\neq$  []>)
  moreover
  have "t = (Init p1,  $\varepsilon$ , q1)"
    using t_def by auto
  ultimately
  show "p = p1"
    using LTS.hd_is_hd by fastforce
qed
have "j=0"
  using Suc(2) <hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1>
  by force
have "(Init p1, [ $\varepsilon$ ], [Init p1, q1], q1)  $\in$  LTS.trans_star_states Ai"
proof -
  have "(Init p1,  $\varepsilon$ , q1)  $\in$  Ai"
    using local.add_trans_pop(1) by auto
  moreover
  have "(Init p1,  $\varepsilon$ , q1)  $\notin$  Aiminus1"
    by (simp add: local.add_trans_pop)
  ultimately
  show "(Init p1, [ $\varepsilon$ ], [Init p1, q1], q1)  $\in$  LTS.trans_star_states Ai"
    by (meson LTS.trans_star_states.trans_star_states_refl
      LTS.trans_star_states.trans_star_states_step)
qed
have "(q1, tl w, tl ss, q)  $\in$  LTS.trans_star_states Aiminus1"
proof -
  from Suc(3) have "(ss, w)  $\in$  LTS.path_with_word Ai"
    by (meson LTS.trans_star_states_path_with_word)

```

```

then have  $tl\_ss\_w\_Ai$ : “ $(tl\ ss, tl\ w) \in LTS.path\_with\_word\ Ai$ ”
  by (metis  $LTS.path\_with\_word.simps$   $\langle transition\_list\ (ss, w) \neq [] \rangle list.sel(3)$ 
     $transition\_list.simps(2)$ )
from  $t\_hd\_once$  have  $zero\_p1\_e\_q1$ : “ $0 = count\ (transitions\_of\ (tl\ ss, tl\ w))\ (Init\ p1, \epsilon, q1)$ ”
  using  $count\_append\_path\_with\_word\_gamma$ [of “[ $hd\ ss$ ]” “[]” “ $tl\ ss$ ” “ $hd\ w$ ” “ $tl\ w$ ” “ $Init\ p1$ ” “ $\epsilon$ ” “ $q1$ ” simplified]
   $\langle (Init\ p1, [\epsilon], [Init\ p1, q1], q1) \in LTS.trans\_star\_states\ Ai \rangle$ 
   $\langle transition\_list\ (ss, w) \neq [] \rangle Suc.prem(2)$  VII
   $LTS.transition\_list\_Cons$ [of “ $Init\ p$ ” “ $w\ ss\ q\ Ai\ \epsilon\ q1$ ”] by (auto simp:  $t\_def$ )
have  $Ai\_Aminus1$ : “ $Ai = Aminus1 \cup \{(Init\ p1, \epsilon, q1)\}$ ”
  using  $local.add\_trans\_pop(1)$  by auto
from  $t\_hd\_once$   $tl\_ss\_w\_Ai$   $zero\_p1\_e\_q1$   $Ai\_Aminus1$ 
   $count\_zero\_remove\_path\_with\_word$ [OF  $tl\_ss\_w\_Ai$ , of “ $Init\ p1$ ” “ $\epsilon$ ” “ $q1$ ”  $Aminus1$ ]
have “ $(tl\ ss, tl\ w) \in LTS.path\_with\_word\ Aminus1$ ”
  by auto
moreover
have “ $hd\ (tl\ ss) = q1$ ”
  using  $Suc.prem(2)$  VII  $\langle transition\_list\ (ss, w) \neq [] \rangle t\_def$ 
   $LTS.transition\_list\_Cons\ t\_hd\_once$  by fastforce
moreover
have “ $last\ ss = q$ ”
  by (metis  $LTS.trans\_star\_states\_last\ Suc.prem(2)$ )
ultimately
show “ $(q1, tl\ w, tl\ ss, q) \in LTS.trans\_star\_states\ Aminus1$ ”
  by (metis (no\_types, lifting)  $LTS.trans\_star\_states\_path\_with\_word$ 
     $LTS.path\_with\_word\_trans\_star\_states\ LTS.path\_with\_word\_not\_empty\ Suc.prem(2)$ 
     $last\_ConsR\ list.collapse$ )
qed
have “ $w = \epsilon \# (tl\ w)$ ”
  by (metis  $Suc(3)$  VII  $\langle transition\_list\ (ss, w) \neq [] \rangle list.distinct(1)\ list.exhaust\_sel$ 
     $list.sel(1)\ t\_def\ LTS.transition\_list\_Cons\ t\_hd\_once$ )
then have  $w\_tl\_e$ : “ $LTS\_e.remove\_e\ w = LTS\_e.remove\_e\ (tl\ w)$ ”
  by (metis  $LTS\_e.remove\_e\_def\ removeAll.simps(2)$ )

have “ $\exists \gamma 2'. LTS\_e.e\_exp\ \gamma 2' [\gamma 2] \wedge (Init\ p2, \gamma 2', q1) \in LTS.trans\_star\ Aminus1$ ”
  using  $add\_trans\_pop$ 
  by (simp add:  $LTS\_e.trans\_star\_e\_e\_exp\_trans\_star$ )
then obtain  $\gamma 2'$  where “ $LTS\_e.e\_exp\ \gamma 2' [\gamma 2] \wedge (Init\ p2, \gamma 2', q1) \in LTS.trans\_star\ Aminus1$ ”
  by blast
then have “ $\exists ss2. (Init\ p2, \gamma 2', ss2, q1) \in LTS.trans\_star\_states\ Aminus1$ ”
  by (simp add:  $LTS.trans\_star\_trans\_star\_states$ )
then obtain  $ss2$  where  $III\_1$ : “ $(Init\ p2, \gamma 2', ss2, q1) \in LTS.trans\_star\_states\ Aminus1$ ”
  by blast
have  $III\_2$ : “ $(q1, tl\ w, tl\ ss, q) \in LTS.trans\_star\_states\ Aminus1$ ”
  using  $ss\_w\_split'$   $Suc(3)$   $Suc(2)$   $\langle j=0 \rangle$ 
  using  $\langle (q1, tl\ w, tl\ ss, q) \in LTS.trans\_star\_states\ Aminus1 \rangle$  by blast
have  $III$ : “ $(Init\ p2, \gamma 2' @ tl\ w, ss2 @ (tl\ (tl\ ss)), q) \in LTS.trans\_star\_states\ Aminus1$ ”
  using  $III\_1\ III\_2$  by (meson  $LTS.trans\_star\_states\_append$ )

from  $Suc(1)$ [of  $p2$  “ $\gamma 2' @ tl\ w$ ” “ $ss2 @ (tl\ (tl\ ss))$ ”  $q$ ]
have  $V$ : “ $\neg is\_Isolated\ q \longrightarrow$ 
   $(\exists p' w'. (Init\ p', w', q) \in LTS\_e.trans\_star\_e\ A \wedge (p', w') \Rightarrow^* (p2, LTS\_e.remove\_e\ (\gamma 2' @ tl\ w))) \wedge$ 
   $(is\_Isolated\ q \longrightarrow (the\_Ctr\_Loc\ q, [the\_Label\ q]) \Rightarrow^* (p2, LTS\_e.remove\_e\ (\gamma 2' @ tl\ w)))$ ”
  using  $III$ 
  using  $step.IH\ step.prem(1,2,3)$  by blast

have “ $\neg is\_Isolated\ q \vee is\_Isolated\ q$ ”
  using  $state.exhaust\_disc$  by blast
then show ?thesis
proof
  assume  $ctr\_q$ : “ $\neg is\_Isolated\ q$ ”
  then have “ $\exists p' w'. (Init\ p', w', q) \in LTS\_e.trans\_star\_e\ A \wedge (p', w') \Rightarrow^* (p2, LTS\_e.remove\_e\ (\gamma 2' @ tl$ 
 $w))$ ”
  using  $V$  by auto

```

**then obtain  $p' w'$  where**  
 VIII: “(Init  $p', w', q) \in LTS_{\varepsilon}.trans\_star_{\varepsilon} A$ ” **and steps:** “ $(p', w') \Rightarrow^* (p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w))$ ”  
 w))”  
**by blast**  
**then have** “ $(p', w') \Rightarrow^* (p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w)) \wedge$   
 $(p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w)) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} (tl w))$ ”  
**proof –**  
**have**  $\gamma2'_{\gamma2}$ : “ $LTS_{\varepsilon}.remove_{\varepsilon} \gamma2' = [\gamma2]$ ”  
**by** (metis  $LTS_{\varepsilon}.\varepsilon\_exp\_def LTS_{\varepsilon}.remove_{\varepsilon}\_def \langle LTS_{\varepsilon}.\varepsilon\_exp \gamma2' [\gamma2] \wedge (Init p2, \gamma2', q1) \in LTS.trans\_star Aminus1 \rangle$ )  
**have** “ $(p', w') \Rightarrow^* (p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w))$ ”  
**using steps by auto**  
**moreover**  
**have rule:** “ $(p2, \gamma2) \hookrightarrow (p, pop)$ ”  
**using VI VII by auto**  
**from steps have steps':** “ $(p', w') \Rightarrow^* (p2, \gamma2 \# (LTS_{\varepsilon}.remove_{\varepsilon} (tl w)))$ ”  
**using  $\gamma2'_{\gamma2}$**   
**by** (metis  $Cons\_eq\_append1 LTS_{\varepsilon}.remove_{\varepsilon}\_append\_dist self\_append\_conv2$ )  
**from rule steps' have** “ $(p2, \gamma2 \# (LTS_{\varepsilon}.remove_{\varepsilon} (tl w))) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} (tl w))$ ”  
**using VIII**  
**by** (metis  $PDS.transition\_rel.intros\_append\_self\_conv2 lbl.simps(1) r\_into\_rtranclp step\_relp\_def$ )  
**then have** “ $(p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w)) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} (tl w))$ ”  
**by** (simp add:  $LTS_{\varepsilon}.remove_{\varepsilon}\_append\_dist \gamma2'_{\gamma2}$ )  
**ultimately**  
**show** “ $(p', w') \Rightarrow^* (p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w)) \wedge$   
 $(p2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma2' @ tl w)) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} (tl w))$ ”  
**by auto**  
**qed**  
**then have** “ $(p', w') \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} (tl w)) \wedge (Init p', w', q) \in LTS_{\varepsilon}.trans\_star_{\varepsilon} A$ ”  
**using VIII by force**  
**then have** “ $\exists p' w'. (Init p', w', q) \in LTS_{\varepsilon}.trans\_star_{\varepsilon} A \wedge (p', w') \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} w)$ ”  
**using  $w\_tl_{\varepsilon}$  by auto**  
**then show ?thesis**  
**using  $ctr\_q \langle p = p1 \rangle$  by blast**  
**next**  
**assume** “ $is\_Isolated q$ ”  
**from V have** “ $(the\_Ctr\_Loc q, [the\_Label q]) \Rightarrow^* (p2, \gamma2 \# (LTS_{\varepsilon}.remove_{\varepsilon} w))$ ”  
**by** (metis  $LTS_{\varepsilon}.\varepsilon\_exp\_def LTS_{\varepsilon}.remove_{\varepsilon}\_append\_dist LTS_{\varepsilon}.remove_{\varepsilon}\_def \langle LTS_{\varepsilon}.\varepsilon\_exp \gamma2' [\gamma2] \wedge (Init p2, \gamma2', q1) \in LTS.trans\_star Aminus1 \rangle \langle is\_Isolated q \rangle append\_Cons append\_self\_conv2 w\_tl_{\varepsilon}$ )  
  
**then have** “ $(the\_Ctr\_Loc q, [the\_Label q]) \Rightarrow^* (p1, LTS_{\varepsilon}.remove_{\varepsilon} w)$ ”  
**using VI by** (metis  $append\_Nil lbl.simps(1) rtranclp.simps step\_relp\_def2$ )  
**then have** “ $(the\_Ctr\_Loc q, [the\_Label q]) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} w)$ ”  
**using VII by auto**  
**then show ?thesis**  
**using  $\langle is\_Isolated q \rangle$  by blast**  
**qed**  
**next**  
**case** (add\_trans\_swap  $p2 \gamma2 p1 \gamma' q1$ )  
**note III = add\_trans\_swap(3)**  
**note VI = add\_trans\_swap(2)**  
**have  $t\_def$ :** “ $t = (Init p1, Some \gamma', q1)$ ”  
**using local.add\_trans\_swap(1) local.add\_trans\_swap  $p1_{\gamma} p2 w2 q' p(1) t\_def$  by blast**  
**have init\_Ai:** “ $inits \subseteq LTS.srscs Ai$ ”  
**using step(1,2,4) no\_edge\_to\_Ctr\_Loc\_post\_star\_rules by** (meson  $r\_into\_rtranclp$ )  
**have  $t\_hd\_once$ :** “ $hd (transition\_list (ss, w)) = t \wedge count (transitions\_of (ss, w)) t = 1$ ”  
**proof –**  
**have** “ $(ss, w) \in LTS.path\_with\_word Ai$ ”  
**using Suc(3)  $LTS.trans\_star\_states\_path\_with\_word$  by metis**  
**moreover**  
**have** “ $inits \subseteq LTS.srscs Ai$ ”

```

    using init_Ai by auto
  moreover
  have “0 < count (transitions_of (ss, w)) t”
    by (metis Suc.prem(1) transitions_of'.sims zero_less_Suc)
  moreover
  have “t = (Init p1, Some  $\gamma'$ , q1)”
    using t_def
    by auto
  moreover
  have “Init p1  $\in$  inits”
    using inits_def by force
  ultimately
  show “hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1”
    using init_only_hd[of ss w Ai t p1 q1] by auto
qed

have “transition_list (ss, w)  $\neq$  []”
  by (metis LTS.trans_star_states_path_with_word LTS.path_with_word.sims Suc.prem(1,2)
    count_empty_less_not_refl2 list.distinct(1) transition_list.sims(1) transitions_of'.sims
    transitions_of.sims(2) zero_less_Suc)
then have ss_w_split: “([Init p1, q1], [Some  $\gamma'$ ]) @' (tl ss, tl w) = (ss, w)”
  using t_hd_once t_def hd_transition_list_append_path_with_word by metis
then have ss_w_split': “(Init p1, [Some  $\gamma'$ ], [Init p1, q1], q1) @@' (q1, tl w, tl ss, q) = (Init p1, w, ss, q)”
  by auto
have VII: “p = p1”
proof -
  have “(Init p, w, ss, q)  $\in$  LTS.trans_star_states Ai”
    using Suc.prem(2) by blast
  moreover
  have “t = hd (transition_list' (Init p, w, ss, q))”
    using  $\langle$ hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1 $\rangle$  by fastforce
  moreover
  have “transition_list' (Init p, w, ss, q)  $\neq$  []”
    by (simp add:  $\langle$ transition_list (ss, w)  $\neq$  [] $\rangle$ )
  moreover
  have “t = (Init p1, Some  $\gamma'$ , q1)”
    using t_def by auto
  ultimately
  show “p = p1”
    using LTS.hd_is_hd by fastforce
qed
have “j=0”
  using Suc(2)  $\langle$ hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1 $\rangle$  by force
have “(Init p1, [Some  $\gamma'$ ], [Init p1, q1], q1)  $\in$  LTS.trans_star_states Ai”
proof -
  have “(Init p1, Some  $\gamma'$ , q1)  $\in$  Ai”
    using local.add_trans_swap(1) by auto
  moreover
  have “(Init p1, Some  $\gamma'$ , q1)  $\notin$  Aiminus1”
    using local.add_trans_swap(4) by blast
  ultimately
  show “(Init p1, [Some  $\gamma'$ ], [Init p1, q1], q1)  $\in$  LTS.trans_star_states Ai”
    by (meson LTS.trans_star_states.trans_star_states_refl LTS.trans_star_states.trans_star_states_step)
qed
have “(q1, tl w, tl ss, q)  $\in$  LTS.trans_star_states Aiminus1”
proof -
  from Suc(3) have “(ss, w)  $\in$  LTS.path_with_word Ai”
    by (meson LTS.trans_star_states_path_with_word)
  then have tl_ss_w_Ai: “(tl ss, tl w)  $\in$  LTS.path_with_word Ai”
    by (metis LTS.path_with_word.sims  $\langle$ transition_list (ss, w)  $\neq$  [] $\rangle$  list.sel(3)
      transition_list.sims(2))
  from t_hd_once have zero_p1_ε_q1: “0 = count (transitions_of (tl ss, tl w)) (Init p1, Some  $\gamma'$ , q1)”
    using count_append_path_with_word_γ[of “[hd ss]” “[]” “tl ss” “hd w” “tl w” “Init p1” “Some  $\gamma'$ ” q1],

```

*simplified*]

```

  ⟨(Init p1, [Some γ'], [Init p1, q1], q1) ∈ LTS.trans_star_states Ai⟩ ⟨transition_list (ss, w) ≠ []⟩
  Suc.prem(2) VII LTS.transition_list_Cons[of "Init p" w ss q Ai "Some γ'" q1]
  by (auto simp: t_def)
  have Ai_Aiminus1: "Ai = Aiminus1 ∪ {(Init p1, Some γ', q1)}"
  using local.add_trans_swap(1) by auto
  from t_hd_once tl_ss_w_Ai zero_p1_ε_q1 Ai_Aiminus1
  count_zero_remove_path_with_word[OF tl_ss_w_Ai, of "Init p1" _ q1 Aiminus1]
  have "(tl ss, tl w) ∈ LTS.path_with_word Aiminus1"
  by auto
  moreover
  have "hd (tl ss) = q1"
  using Suc.prem(2) VII ⟨transition_list (ss, w) ≠ []⟩ t_def LTS.transition_list_Cons t_hd_once by
fastforce
  moreover
  have "last ss = q"
  by (metis LTS.trans_star_states_last Suc.prem(2))
  ultimately
  show "(q1, tl w, tl ss, q) ∈ LTS.trans_star_states Aiminus1"
  by (metis (no_types, lifting) LTS.trans_star_states_path_with_word
  LTS.path_with_word_trans_star_states LTS.path_with_word_not_empty Suc.prem(2)
  last_ConsR list.collapse)
qed
have "w = Some γ' # (tl w)"
  by (metis Suc(3) VII ⟨transition_list (ss, w) ≠ []⟩ list.distinct(1) list.exhaust_sel
  list.sel(1) t_def LTS.transition_list_Cons t_hd_once)
then have w_tl_ε: "LTS_ε.remove_ε w = LTS_ε.remove_ε (Some γ'#tl w)"
  using LTS_ε.remove_ε_def removeAll.simps(2)
  by presburger
have "∃ γ2'. LTS_ε.ε_exp γ2' [γ2] ∧ (Init p2, γ2', q1) ∈ LTS.trans_star Aiminus1"
  using add_trans_swap by (simp add: LTS_ε.trans_star_ε_ε_exp_trans_star)
then obtain γ2' where "LTS_ε.ε_exp γ2' [γ2] ∧ (Init p2, γ2', q1) ∈ LTS.trans_star Aiminus1"
  by blast
then have "∃ ss2. (Init p2, γ2', ss2, q1) ∈ LTS.trans_star_states Aiminus1"
  by (simp add: LTS.trans_star_trans_star_states)
then obtain ss2 where III_1: "(Init p2, γ2', ss2, q1) ∈ LTS.trans_star_states Aiminus1"
  by blast
have III_2: "(q1, tl w, tl ss, q) ∈ LTS.trans_star_states Aiminus1"
  using ss_w_split' Suc(3) Suc(2) ⟨j=0⟩
  using ⟨(q1, tl w, tl ss, q) ∈ LTS.trans_star_states Aiminus1⟩ by blast
have III: "(Init p2, γ2' @ tl w, ss2 @ (tl (tl ss)), q) ∈ LTS.trans_star_states Aiminus1"
  using III_1 III_2 by (meson LTS.trans_star_states_append)

from Suc(1)[of p2 "γ2' @ tl w" "ss2 @ (tl (tl ss))" q]
have V: "¬is_Isolated q →
(∃ p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl w))) ∧
(is_Isolated q → (the_Ctr_Loc q, [the_Label q]) ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl w)))"
  using III
  using step.IH step.prem(1,2,3) by blast

have "¬is_Isolated q ∨ is_Isolated q"
  using state.exhaust_disc by blast
then show ?thesis
proof
  assume ctr_q: "¬is_Isolated q"
  then have "∃ p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl
w))"
  using V by auto
  then obtain p' w' where
  VIII: "(Init p', w', q) ∈ LTS_ε.trans_star_ε A" and steps: "(p', w') ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl
w))"
  by blast
  then have "(p', w') ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl w)) ∧

```



$(p2, LTS\_ε.remove\_ε (\gamma2' @ tl w)) \Rightarrow^* (p, \gamma' \# LTS\_ε.remove\_ε (tl w))$

**proof** –

**have**  $\gamma2' \_ \gamma2$ : “ $LTS\_ε.remove\_ε \gamma2' = [\gamma2]$ ”

**by** (*metis*  $LTS\_ε.ε\_exp\_def$   $LTS\_ε.remove\_ε\_def$   $\langle LTS\_ε.ε\_exp \gamma2' [\gamma2] \wedge (Init\ p2, \gamma2', q1) \in LTS.trans\_star\ Aminus1 \rangle$ )

**have** “ $(p', w') \Rightarrow^* (p2, LTS\_ε.remove\_ε (\gamma2' @ tl w))$ ”

**using** *steps by auto*

**moreover**

**have** *rule*: “ $(p2, \gamma2) \hookrightarrow (p, swap\ \gamma')$ ”

**using** *VI VII by auto*

**from** *steps have steps'*: “ $(p', w') \Rightarrow^* (p2, \gamma2 \# (LTS\_ε.remove\_ε (tl w)))$ ”

**using**  $\gamma2' \_ \gamma2$

**by** (*metis*  $Cons\_eq\_append1$   $LTS\_ε.remove\_ε\_append\_dist$   $self\_append\_conv2$ )

**from** *rule steps' have* “ $(p2, \gamma2 \# (LTS\_ε.remove\_ε (tl w))) \Rightarrow^* (p, \gamma' \# LTS\_ε.remove\_ε (tl w))$ ”

**using** *VIII*

**using**  $PDS.transition\_rel.intros$   $append\_self\_conv2$   $lbl.simps(1)$   $r\_into\_rtranclp$   $step\_relp\_def$

**by** *fastforce*

**then** **have** “ $(p2, LTS\_ε.remove\_ε (\gamma2' @ tl w)) \Rightarrow^* (p, \gamma' \# LTS\_ε.remove\_ε (tl w))$ ”

**by** (*simp add*:  $LTS\_ε.remove\_ε\_append\_dist$   $\gamma2' \_ \gamma2$ )

**ultimately**

**show** “ $(p', w') \Rightarrow^* (p2, LTS\_ε.remove\_ε (\gamma2' @ tl w)) \wedge$   
 $(p2, LTS\_ε.remove\_ε (\gamma2' @ tl w)) \Rightarrow^* (p, \gamma' \# LTS\_ε.remove\_ε (tl w))$ ”

**by** *auto*

**qed**

**then** **have** “ $(p', w') \Rightarrow^* (p, \gamma' \# LTS\_ε.remove\_ε (tl w)) \wedge (Init\ p', w', q) \in LTS\_ε.trans\_star\_ε\ A$ ”

**using** *VIII by force*

**then** **have** “ $\exists p' w'. (Init\ p', w', q) \in LTS\_ε.trans\_star\_ε\ A \wedge (p', w') \Rightarrow^* (p, LTS\_ε.remove\_ε\ w)$ ”

**using**  $LTS\_ε.remove\_ε\_Cons\_tl$  **by** (*metis*  $\langle w = Some\ \gamma' \# tl\ w \rangle$ )

**then** **show** *?thesis*

**using**  $ctr\_q$   $\langle p = p1 \rangle$  **by** *blast*

**next**

**assume** “*is\\_Isolated*  $q$ ”

**from** *V this have* “ $(the\_Ctr\_Loc\ q, [the\_Label\ q]) \Rightarrow^* (p2, LTS\_ε.remove\_ε (\gamma2' @ tl w))$ ”

**by** *auto*

**then** **have** “ $(the\_Ctr\_Loc\ q, [the\_Label\ q]) \Rightarrow^* (p2, \gamma2 \# (LTS\_ε.remove\_ε (tl w)))$ ”

**by** (*metis*  $LTS\_ε.ε\_exp\_def$   $LTS\_ε.remove\_ε\_append\_dist$   $LTS\_ε.remove\_ε\_def$   $\langle LTS\_ε.ε\_exp \gamma2' [\gamma2] \wedge (Init\ p2, \gamma2', q1) \in LTS.trans\_star\ Aminus1 \rangle$   $append\_Cons$   $append\_self\_conv2$ )

**then** **have** “ $(the\_Ctr\_Loc\ q, [the\_Label\ q]) \Rightarrow^* (p1, \gamma' \# LTS\_ε.remove\_ε (tl w))$ ”

**using** *VI*

**by** (*metis* (*no\\_types*)  $append\_Cons$   $append\_Nil$   $lbl.simps(2)$   $rtranclp.rtrancl\_into\_rtrancl$   $step\_relp\_def2$ )

**then** **have** “ $(the\_Ctr\_Loc\ q, [the\_Label\ q]) \Rightarrow^* (p, \gamma' \# LTS\_ε.remove\_ε (tl w))$ ”

**using** *VII by auto*

**then** **show** *?thesis*

**using**  $\langle is\_Isolated\ q \rangle$

**by** (*metis*  $LTS\_ε.remove\_ε\_Cons\_tl\ w\_tl\_ε$ )

**qed**

**next**

**case** (*add\\_trans\\_push\\_1*  $p2\ \gamma2\ p1\ \gamma1\ \gamma''\ q1'$ )

**then** **have** *t\\_def*: “ $t = (Init\ p1, Some\ \gamma1, Isolated\ p1\ \gamma1)$ ”

**using**  $local.add\_trans\_pop(1)$   $local.add\_trans\_pop$   $p1\_γ\_p2\_w2\_q'\_p(1)$  *t\\_def* **by** *blast*

**have** *init\\_Ai*: “ $inits \subseteq LTS.srcs\ Ai$ ”

**using**  $step(1,2)$   $step(4)$

**using**  $no\_edge\_to\_Ctr\_Loc\_post\_star\_rules$

**by** (*meson*  $r\_into\_rtranclp$ )

**have** *t\\_hd\\_once*: “ $hd\ (transition\_list\ (ss, w)) = t \wedge count\ (transitions\_of\ (ss, w))\ t = 1$ ”

**proof** –

**have** “ $(ss, w) \in LTS.path\_with\_word\ Ai$ ”

**using**  $Suc(3)$   $LTS.trans\_star\_states\_path\_with\_word$  **by** *metis*

**moreover**

**have** “ $inits \subseteq LTS.srcs\ Ai$ ”

**using** *init\\_Ai* **by** *auto*

```

moreover
have “ $0 < \text{count}(\text{transitions\_of}(ss, w)) t$ ”
  by (metis Suc.prem(1) transitions_of'.simps zero_less_Suc)
moreover
have “ $t = (\text{Init } p1, \text{Some } \gamma1, \text{Isolated } p1 \ \gamma1)$ ”
  using t_def by auto
moreover
have “ $\text{Init } p1 \in \text{inits}$ ”
  using inits_def by fastforce
ultimately
show “ $\text{hd}(\text{transition\_list}(ss, w)) = t \wedge \text{count}(\text{transitions\_of}(ss, w)) t = 1$ ”
  using init_only_hd[of ss w Ai t] by auto
qed
have “ $\text{transition\_list}(ss, w) \neq []$ ”
  by (metis LTS.trans_star_states_path_with_word LTS.path_with_word.simps Suc.prem(1,2)
    count_empty_less_not_refl2 list.distinct(1) transition_list.simps(1)
    transitions_of'.simps transitions_of.simps(2) zero_less_Suc)

have VII: “ $p = p1$ ”
proof –
  have “ $(\text{Init } p, w, ss, q) \in \text{LTS.trans\_star\_states } Ai$ ”
    using Suc.prem(2) by blast
  moreover
  have “ $t = \text{hd}(\text{transition\_list}'(\text{Init } p, w, ss, q))$ ”
    using <hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1 > by fastforce
  moreover
  have “ $\text{transition\_list}'(\text{Init } p, w, ss, q) \neq []$ ”
    by (simp add: <transition_list (ss, w)  $\neq$  [] >)
  moreover
  have “ $t = (\text{Init } p1, \text{Some } \gamma1, \text{Isolated } p1 \ \gamma1)$ ”
    using t_def by auto
  ultimately
  show “ $p = p1$ ”
    using LTS.hd_is_hd by fastforce
qed
from add_trans_push_1(4) have “ $\nexists p \ \gamma. (\text{Isolated } p1 \ \gamma1, \gamma, p) \in \text{Aiminus1}$ ”
  using post_star_rules_Isolated_sink_invariant[of A Aiminus1 p1  $\gamma1$ ] step.hyps(1)
    step.prem(1,2,3) unfolding LTS.sinks_def by blast
then have “ $\nexists p \ \gamma. (\text{Isolated } p1 \ \gamma1, \gamma, p) \in Ai$ ”
  using local.add_trans_push_1(1) by blast
then have ss_w_short: “ $ss = [\text{Init } p1, \text{Isolated } p1 \ \gamma1] \wedge w = [\text{Some } \gamma1]$ ”
  using Suc.prem(2) VII <hd (transition_list (ss, w)) = t  $\wedge$  count (transitions_of (ss, w)) t = 1 > t_def
    LTS.nothing_after_sink[of “Init p1” “Isolated p1  $\gamma1$ ” “tl (tl ss)” “Some  $\gamma1$ ” “tl w” Ai] <transition_list (ss,
w)  $\neq$  [] >
    LTS.trans_star_states_path_with_word[of “Init p” w ss q Ai]
    LTS.transition_list_Cons[of “Init p” w ss q Ai]
  by (auto simp: LTS.sinks_def2)
then have q_ext: “ $q = \text{Isolated } p1 \ \gamma1$ ”
  using LTS.trans_star_states_last Suc.prem(2) by fastforce
have “ $(p1, [\gamma1]) \Rightarrow^* (p, \text{LTS}_\varepsilon.\text{remove}_\varepsilon w)$ ”
  using ss_w_short unfolding LTS_ε.remove_ε_def
  using VII by force
have “ $(\text{the\_Ctr\_Loc } q, [\text{the\_Label } q]) \Rightarrow^* (p, \text{LTS}_\varepsilon.\text{remove}_\varepsilon w)$ ”
  by (simp add: <(p1, [ $\gamma1$ ])  $\Rightarrow^*$  (p, LTS_ε.remove_ε w) > q_ext)
then show ?thesis
  using q_ext by auto
next
case (add_trans_push_2 p2  $\gamma2$  p1  $\gamma1$   $\gamma''$  q')
note IX = add_trans_push_2(3)
note XIII = add_trans_push_2(2)
have t_def: “ $t = (\text{Isolated } p1 \ \gamma1, \text{Some } \gamma'', q')$ ”
  using local.add_trans_push_2(1,4) p1_γ_p2_w2_q'_p(1) t_def by blast
have init_Ai: “ $\text{inits} \subseteq \text{LTS.srccs } Ai$ ”

```

```

using step(1,2) step(4)
using no_edge_to_Ctr_Loc_post_star_rules
by (meson r_into_rtranclp)

from Suc(2,3) split_at_first_t[of "Init p" w ss q Ai j "Isolated p1  $\gamma 1$ " "Some  $\gamma''$ " q' Aminus1] t_def
have " $\exists u v u\_ss v\_ss.$ 
   $ss = u\_ss @ v\_ss \wedge$ 
   $w = u @ [Some \gamma''] @ v \wedge$ 
   $(Init p, u, u\_ss, Isolated p1 \gamma 1) \in LTS.trans\_star\_states Aminus1 \wedge$ 
   $(Isolated p1 \gamma 1, [Some \gamma''], q') \in LTS.trans\_star Ai \wedge (q', v, v\_ss, q) \in LTS.trans\_star\_states Ai$ "
  using local.add_trans_push_2(1,4) by blast
then obtain u v u_ss v_ss where
  ss_split: " $ss = u\_ss @ v\_ss$ " and
  w_split: " $w = u @ [Some \gamma''] @ v$ " and
  X_1: " $(Init p, u, u\_ss, Isolated p1 \gamma 1) \in LTS.trans\_star\_states Aminus1$ " and
  out_trans: " $(Isolated p1 \gamma 1, [Some \gamma''], q') \in LTS.trans\_star Ai$ " and
  path: " $(q', v, v\_ss, q) \in LTS.trans\_star\_states Ai$ "
  by auto
from step(3)[of p u u_ss "Isolated p1  $\gamma 1$ "] X_1 have
  " $(\neg is\_Isolated (Isolated p1 \gamma 1) \longrightarrow$ 
   $(\exists p' w'. (Init p', w', Isolated p1 \gamma 1) \in LTS.\epsilon.trans\_star\_epsilon A \wedge (p', w') \Rightarrow^* (p, LTS.\epsilon.remove\_epsilon u))) \wedge$ 
   $(is\_Isolated (Isolated p1 \gamma 1) \longrightarrow$ 
   $(the\_Ctr\_Loc (Isolated p1 \gamma 1), [the\_Label (Isolated p1 \gamma 1)]) \Rightarrow^* (p, LTS.\epsilon.remove\_epsilon u))$ "
  using step.prem(1,2,3) by auto
then have " $(the\_Ctr\_Loc (Isolated p1 \gamma 1), [the\_Label (Isolated p1 \gamma 1)]) \Rightarrow^* (p, LTS.\epsilon.remove\_epsilon u)$ "
  by auto
then have p1_ $\gamma 1$ _p_u: " $(p1, [\gamma 1]) \Rightarrow^* (p, LTS.\epsilon.remove\_epsilon u)$ "
  by auto
from IX have " $\exists \gamma 2\epsilon \gamma 2ss. LTS.\epsilon.\epsilon\_exp \gamma 2\epsilon [\gamma 2] \wedge (Init p2, \gamma 2\epsilon, \gamma 2ss, q') \in LTS.trans\_star\_states$ 
  Aminus1"
  by (meson LTS.trans_star_trans_star_states LTS.\epsilon.trans_star_\epsilon_\epsilon_exp_trans_star)
then obtain  $\gamma 2\epsilon \gamma 2ss$  where XI_1: " $LTS.\epsilon.\epsilon\_exp \gamma 2\epsilon [\gamma 2] \wedge (Init p2, \gamma 2\epsilon, \gamma 2ss, q') \in LTS.trans\_star\_states$ 
  Aminus1"
  by blast
have " $(q', v, v\_ss, q) \in LTS.trans\_star\_states Ai$ "
  using path .
have ind:
  " $(\neg is\_Isolated q \longrightarrow (\exists p' w'. (Init p', w', q) \in LTS.\epsilon.trans\_star\_epsilon A \wedge (p', w') \Rightarrow^* (p2, LTS.\epsilon.remove\_epsilon$ 
   $(\gamma 2\epsilon @ v)))) \wedge$ 
   $(is\_Isolated q \longrightarrow (the\_Ctr\_Loc q, [the\_Label q]) \Rightarrow^* (p2, LTS.\epsilon.remove\_epsilon (\gamma 2\epsilon @ v)))$ "
  proof -
  have  $\gamma 2ss\_len$ : " $length \gamma 2ss = Suc (length \gamma 2\epsilon)$ "
    by (meson LTS.trans_star_states_length XI_1)

  have v_ss_empty: " $v\_ss \neq []$ "
    by (metis LTS.trans_star_states.simps path list.distinct(1))

  have  $\gamma 2ss\_last$ : " $last \gamma 2ss = hd v\_ss$ "
    by (metis LTS.trans_star_states_hd LTS.trans_star_states_last XI_1 path)

  have cv: " $j = count (transitions\_of ((v\_ss, v))) t$ "
  proof -
  have last_u_ss: " $Isolated p1 \gamma 1 = last u\_ss$ "
    by (meson LTS.trans_star_states_last X_1)
  have q'_hd_v_ss: " $q' = hd v\_ss$ "
    by (meson LTS.trans_star_states_hd path)

  have "count (transitions\_of' (((Init p, u, u\_ss, Isolated p1  $\gamma 1$ ), Some  $\gamma''$ ) @@ $\gamma$  (q', v, v\_ss, q)))
    (Isolated p1  $\gamma 1$ , Some  $\gamma''$ , q') =
    count (transitions\_of' (Init p, u, u\_ss, Isolated p1  $\gamma 1$ )) (Isolated p1  $\gamma 1$ , Some  $\gamma''$ , q') +
    (if Isolated p1  $\gamma 1 = last u\_ss \wedge q' = hd v\_ss \wedge Some \gamma'' = Some \gamma''$  then 1 else 0) +
    count (transitions\_of' (q', v, v\_ss, q)) (Isolated p1  $\gamma 1$ , Some  $\gamma''$ , q)"

```

**using** `count_append_trans_star_states_γ_length`[of `u_ss u v_ss` “Init p” “Isolated p1 γ1” “Some γ’’”  $q'$   $v$   $q$  “Isolated p1 γ1” “Some γ’’”  $q'$ ] `t_def ss_split w_split X_1`  
**by** (`meson LTS.trans_star_states_length v_ss_empty`)  
**then have** “`count (transitions_of (u_ss @ v_ss, u @ Some γ’’ # v)) (last u_ss, Some γ’’, hd v_ss) = Suc (count (transitions_of (u_ss, u)) (last u_ss, Some γ’’, hd v_ss) + count (transitions_of (v_ss, v)) (last u_ss, Some γ’’, hd v_ss))`”  
**using** `last_u_ss q' hd_v_ss` **by** `auto`  
**then have** “`j = count (transitions_of' ((q', v, v_ss, q))) t`”  
**using** `last_u_ss q' hd_v_ss X_1 ss_split w_split add_trans_push_2(4) Suc(2)`  
`LTS.avoid_count_zero`[of “Init p” `u u_ss` “Isolated p1 γ1” `Aiminus1` “Isolated p1 γ1” “Some γ’’”  $q'$ ]  
**by** (`auto simp: t_def`)  
**then show** “`j = count (transitions_of ((v_ss, v))) t`”  
**by** `force`  
**qed**  
**have** `p2_q'_states_Aiminus1`: “`(Init p2, γ2ε, γ2ss, q') ∈ LTS.trans_star_states Aiminus1`”  
**using** `XI_1` **by** `blast`  
**then have** `cγ2`: “`count (transitions_of (γ2ss, γ2ε)) t = 0`”  
**using** `LTS.avoid_count_zero local.add_trans_push_2(4) t_def` **by** `fastforce`  
**have** “`j = count (transitions_of ((γ2ss, γ2ε) @' (v_ss, v))) t`”  
**using** `LTS.count_append_path_with_word`[of `γ2ss γ2ε v_ss v` “Isolated p1 γ1” “Some γ’’”  $q'$ ] `t_def`  
`cγ2 cv γ2ss_len v_ss_empty γ2ss_last`  
**by** `force`  
**then have** `j_count`: “`j = count (transitions_of' (Init p2, γ2ε @ v, γ2ss @ tl v_ss, q)) t`”  
**by** `simp`  
  
**have** “`(γ2ss, γ2ε) ∈ LTS.path_with_word Aiminus1`”  
**by** (`meson LTS.trans_star_states_path_with_word p2_q'_states_Aiminus1`)  
**then have** `γ2ss_path`: “`(γ2ss, γ2ε) ∈ LTS.path_with_word Ai`”  
**using** `add_trans_push_2(1)`  
`path_with_word_mono`[of `γ2ss γ2ε Aiminus1 Ai`] **by** `auto`  
  
**have** `path'`: “`(v_ss, v) ∈ LTS.path_with_word Ai`”  
**by** (`meson LTS.trans_star_states_path_with_word path`)  
**have** “`(γ2ss, γ2ε) @' (v_ss, v) ∈ LTS.path_with_word Ai`”  
**using** `γ2ss_path path' LTS.append_path_with_word_path_with_word γ2ss_last`  
**by** `auto`  
**then have** “`(γ2ss @ tl v_ss, γ2ε @ v) ∈ LTS.path_with_word Ai`”  
**by** `auto`  
  
**have** “`(Init p2, γ2ε @ v, γ2ss @ tl v_ss, q) ∈ LTS.trans_star_states Ai`”  
**by** (`metis (no_types, lifting) LTS.path_with_word_trans_star_states LTS.trans_star_states_append LTS.trans_star_states_hd XI_1 path γ2ss_path γ2ss_last`)  
  
**from** `this Suc(1)`[of `p2` “`γ2ε @ v`” “`γ2ss @ tl v_ss`”  $q$ ]  
**show**  
“`(¬ is_Isolated q → (∃ p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p2, LTS_ε.remove_ε (γ2ε @ v)))) ∧`  
`(is_Isolated q → (the_Ctr_Loc q, [the_Label q]) ⇒* (p2, LTS_ε.remove_ε (γ2ε @ v)))`”  
**using** `j_count` **by** `fastforce`  
**qed**  
  
**show** `?thesis`  
**proof** (`cases “is_Init q ∨ is_Noninit q”`)  
**case** `True`  
**have** “`(∃ p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p2, LTS_ε.remove_ε (γ2ε @ v)))`”  
**using** `True ind` **by** `fastforce`  
**then obtain** `p' w' p`: “`(Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p2, LTS_ε.remove_ε (γ2ε @ v))`”  
**by** `auto`  
**then have** “`(p', w') ⇒* (p2, LTS_ε.remove_ε (γ2ε @ v))`”  
**by** `auto`  
**have** `p2_γ2εv_p1_γ1_γ''_v`: “`(p2, LTS_ε.remove_ε (γ2ε @ v)) ⇒* (p1, γ1 # γ'' # LTS_ε.remove_ε v)`”

```

proof –
  have “ $\gamma_2 \#(LTS_{\varepsilon}.remove_{\varepsilon} v) = LTS_{\varepsilon}.remove_{\varepsilon} (\gamma_2\varepsilon @ v)$ ”
    using XI_1
    by (metis LTS_{\varepsilon}.\varepsilon_exp_def LTS_{\varepsilon}.remove_{\varepsilon}_append_dist LTS_{\varepsilon}.remove_{\varepsilon}_def append_Cons self_append_conv2)
  moreover
  from XIII have “ $(p_2, \gamma_2 \#(LTS_{\varepsilon}.remove_{\varepsilon} v)) \Rightarrow^* (p_1, \gamma_1 \# \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v)$ ”
    by (metis PDS.transition_rel.intros append_Cons append_Nil lbl.simps(3) r_into_rtranclp step_relp_def)
  ultimately
  show “ $(p_2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma_2\varepsilon @ v)) \Rightarrow^* (p_1, \gamma_1 \# \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v)$ ”
    by auto
  qed
have  $p_1 \_ \gamma_1 \gamma'' v \_ p \_ uv$ : “ $(p_1, \gamma_1 \# \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v) \Rightarrow^* (p, (LTS_{\varepsilon}.remove_{\varepsilon} u) @ (\gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v))$ ”
  by (metis p1_{\gamma_1}_p_u append_Cons append_Nil step_relp_append)
have “ $(p, (LTS_{\varepsilon}.remove_{\varepsilon} u) @ (\gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v)) = (p, LTS_{\varepsilon}.remove_{\varepsilon} w)$ ”
  by (metis (no_types, lifting) Cons_eq_append_conv LTS_{\varepsilon}.remove_{\varepsilon}_Cons_tl LTS_{\varepsilon}.remove_{\varepsilon}_append_dist w_split hd_Cons_tl list.inject list.sel(1) list.simps(3) self_append_conv2)
  then show ?thesis
    using True p1_{\gamma_1}\gamma''v_p_uv p2_{\gamma_2\varepsilon}v_p1_{\gamma_1}\gamma''_v p'_w'_p by fastforce
next
case False
  then have  $q \_ nlq \_ p2_{\gamma_2\varepsilon}v$ : “ $(the\_Ctr\_Loc q, [the\_Label q]) \Rightarrow^* (p_2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma_2\varepsilon @ v))$ ”
    using ind.state.exhaust_disc
    by blast
  have  $p2_{\gamma_2\varepsilon}v \_ p1_{\gamma_1}\gamma''v$ : “ $(p_2, LTS_{\varepsilon}.remove_{\varepsilon} (\gamma_2\varepsilon @ v)) \Rightarrow^* (p_1, \gamma_1 \# \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v)$ ”
  by (metis (mono_tags) LTS_{\varepsilon}.\varepsilon_exp_def LTS_{\varepsilon}.remove_{\varepsilon}_append_dist LTS_{\varepsilon}.remove_{\varepsilon}_def XIII XI_1 append_Cons append_Nil lbl.simps(3) r_into_rtranclp step_relp_def2)

  have  $p1_{\gamma_1}\gamma''_v \_ p \_ u\gamma''v$ : “ $(p_1, \gamma_1 \# \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} u @ \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v)$ ”
  by (metis p1_{\gamma_1}_p_u append_Cons append_Nil step_relp_append)

  have “ $(p, LTS_{\varepsilon}.remove_{\varepsilon} u @ \gamma'' \# LTS_{\varepsilon}.remove_{\varepsilon} v) = (p, LTS_{\varepsilon}.remove_{\varepsilon} w)$ ”
  by (metis LTS_{\varepsilon}.remove_{\varepsilon}_Cons_tl LTS_{\varepsilon}.remove_{\varepsilon}_append_dist append_Cons append_Nil w_split hd_Cons_tl list.distinct(1) list.inject)

  then show ?thesis
    using False p1_{\gamma_1}\gamma''_v_p_u\gamma''v p2_{\gamma_2\varepsilon}v_p1_{\gamma_1}\gamma''_v q_nlq_p2_{\gamma_2\varepsilon}v
    by (metis (no_types, lifting) ind_rtranclp_trans)
  qed
qed
qed
qed

```

— Corresponds to Schwoon’s lemma 3.4

**lemma** *rtranclp\_post\_star\_rules\_constains\_successors*:

```

assumes “post_star_rules*  $A A'$ ”
assumes “inits  $\subseteq LTS.srcs A$ ”
assumes “isols  $\subseteq LTS.isolated A$ ”
assumes “ $(Init p, w, q) \in LTS.trans\_star A'$ ”
shows “ $(\neg is\_Isolated q \longrightarrow (\exists p' w'. (Init p', w', q) \in LTS_{\varepsilon}.trans\_star_{\varepsilon} A \wedge (p', w') \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} w))) \wedge$ 
  ( $is\_Isolated q \longrightarrow (the\_Ctr\_Loc q, [the\_Label q]) \Rightarrow^* (p, LTS_{\varepsilon}.remove_{\varepsilon} w)$ )”
using rtranclp_post_star_rules_constains_successors_states assms
by (metis LTS.trans_star_trans_star_states)

```

— Corresponds to Schwoon’s lemma 3.4

**lemma** *post\_star\_rules\_saturation\_constains\_successors*:

```

assumes “saturation_post_star_rules  $A A'$ ”

```

```

assumes "inits  $\subseteq$  LTS.srcs A"
assumes "isols  $\subseteq$  LTS.isolated A"
assumes "(Init p, w, q)  $\in$  LTS.trans_star A'"
shows "(-is_Isolated q  $\longrightarrow$  ( $\exists$  p' w'. (Init p', w', q)  $\in$  LTS_ε.trans_star_ε A  $\wedge$  (p', w')  $\Rightarrow^*$  (p, LTS_ε.remove_ε w)))  $\wedge$ 
(is_Isolated q  $\longrightarrow$  (the_Ctr_Loc q, [the_Label q])  $\Rightarrow^*$  (p, LTS_ε.remove_ε w)))"
using rtranclp_post_star_rules_constains_successors assms saturation_def by metis

```

— Corresponds to one direction of Schwoon's theorem 3.3

**theorem** *post\_star\_rules\_subset\_post\_star\_lang*:

```

assumes "post_star_rules** A A'"
assumes "inits  $\subseteq$  LTS.srcs A"
assumes "isols  $\subseteq$  LTS.isolated A"
shows "{c. accepts_ε A' c}  $\subseteq$  post_star (lang_ε A)"
proof
fix c :: "('ctr_loc, 'label) conf"
define p where "p = fst c"
define w where "w = snd c"
assume "c  $\in$  {c. accepts_ε A' c}"
then have "accepts_ε A' (p,w)"
  unfolding p_def w_def by auto
then obtain q where q_p: "q  $\in$  finals" "(Init p, w, q)  $\in$  LTS_ε.trans_star_ε A'"
  unfolding accepts_ε_def by auto
then obtain w' where w'_def: "LTS_ε.ε_exp w' w  $\wedge$  (Init p, w', q)  $\in$  LTS.trans_star A'"
  by (meson LTS_ε.trans_star_ε_iff_ε_exp_trans_star)
then have path: "(Init p, w', q)  $\in$  LTS.trans_star A'"
  by auto
have "¬ is_Isolated q"
  using F_not_Ext q_p(1) by blast
then obtain p' w'a where "(Init p', w'a, q)  $\in$  LTS_ε.trans_star_ε A  $\wedge$  (p', w'a)  $\Rightarrow^*$  (p, LTS_ε.remove_ε w)"
  using rtranclp_post_star_rules_constains_successors[OF assms(1) assms(2) assms(3) path] by auto
then have "(Init p', w'a, q)  $\in$  LTS_ε.trans_star_ε A  $\wedge$  (p', w'a)  $\Rightarrow^*$  (p, w)"
  using w'_def
  by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_def
     $\langle$ LTS_ε.ε_exp w' w  $\wedge$  (Init p, w', q)  $\in$  LTS.trans_star A' $\rangle$ )
then have "(p,w)  $\in$  post_star (lang_ε A)"
  using  $\langle$ q  $\in$  finals $\rangle$  unfolding LTS.post_star_def accepts_ε_def lang_ε_def by fastforce
then show "c  $\in$  post_star (lang_ε A)"
  unfolding p_def w_def by auto
qed

```

— Corresponds to Schwoon's theorem 3.3

**theorem** *post\_star\_rules\_accepts\_ε\_correct*:

```

assumes "saturation post_star_rules A A'"
assumes "inits  $\subseteq$  LTS.srcs A"
assumes "isols  $\subseteq$  LTS.isolated A"
shows "{c. accepts_ε A' c} = post_star (lang_ε A)"
proof (rule; rule)
fix c :: "('ctr_loc, 'label) conf"
define p where "p = fst c"
define w where "w = snd c"
assume "c  $\in$  post_star (lang_ε A)"
then obtain p' w' where "(p', w')  $\Rightarrow^*$  (p, w)  $\wedge$  (p', w')  $\in$  lang_ε A"
  by (auto simp: post_star_def p_def w_def)
then have "accepts_ε A' (p, w)"
  using lemma_3_3[of p' w' p w A A'] assms(1) by auto
then have "accepts_ε A' c"
  unfolding p_def w_def by auto
then show "c  $\in$  {c. accepts_ε A' c}"
  by auto
next
fix c :: "('ctr_loc, 'label) conf"
assume "c  $\in$  {c. accepts_ε A' c}"

```

**then show** “ $c \in \text{post\_star}(\text{lang\_}\varepsilon A)$ ”  
**using** *assms post\_star\_rules\_subset\_post\_star\_lang unfolding saturation\_def by blast*  
**qed**

— Corresponds to Schwoon’s theorem 3.3

**theorem** *post\_star\_rules\_correct*:

**assumes** “*saturation post\_star\_rules A A’*”  
**assumes** “*inits  $\subseteq LTS.\text{srcs } A$* ”  
**assumes** “*isols  $\subseteq LTS.\text{isolated } A$* ”  
**shows** “*lang\_* $\varepsilon A' = \text{post\_star}(\text{lang\_}\varepsilon A)$ ”  
**using** *assms lang\_* $\varepsilon$ *\_def post\_star\_rules\_accepts\_* $\varepsilon$ *\_correct by presburger*

**end**

## 5.5 Intersection Automata

**definition** *accepts\_inters* :: “ $((\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state} * (\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state}, \text{label}) \text{ transition set}$   
 $\Rightarrow ((\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state} * (\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state}) \text{ set} \Rightarrow (\text{ctr\_loc}, \text{label}) \text{ conf} \Rightarrow \text{bool}$ ” **where**  
“*accepts\_inters ts finals*  $\equiv \lambda(p,w). (\exists qq \in \text{finals}. ((\text{Init } p, \text{Init } p), w, qq) \in LTS.\text{trans\_star } ts)$ ”

**lemma** *accepts\_inters\_accepts\_aut\_inters*:

**assumes** “*ts12 = inters ts1 ts2*”  
**assumes** “*finals12 = inters\_finals finals1 finals2*”  
**shows** “*accepts\_inters ts12 finals12 (p,w)  $\longleftrightarrow$*   
*Intersection\_P\_Automaton.accepts\_aut\_inters ts1 Init finals1 ts2*  
*finals2 p w*”  
**by** (*simp add: Intersection\_P\_Automaton.accepts\_aut\_inters\_def PDS\_with\_P\_automata.inits\_def*  
*P\_Automaton.accepts\_aut\_def accepts\_inters\_def assms*)

**definition** *lang\_inters* :: “ $((\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state} * (\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state}, \text{label}) \text{ transition set}$   
 $\Rightarrow ((\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state} * (\text{ctr\_loc}, \text{noninit}, \text{label}) \text{ state}) \text{ set} \Rightarrow (\text{ctr\_loc}, \text{label}) \text{ conf set}$ ” **where**  
“*lang\_inters ts finals* = {*c. accepts\_inters ts finals c*}”

**lemma** *lang\_inters\_lang\_aut\_inters*:

**assumes** “*ts12 = inters ts1 ts2*”  
**assumes** “*finals12 = inters\_finals finals1 finals2*”  
**shows** “ $(\lambda(p,w). (p, w)) \text{ ‘lang\_inters ts12 finals12 =}$   
*Intersection\_P\_Automaton.lang\_aut\_inters ts1 Init finals1 ts2 finals2*”  
**using** *assms*  
**by** (*auto simp: Intersection\_P\_Automaton.lang\_aut\_inters\_def*  
*Intersection\_P\_Automaton.inters\_accept\_iff*  
*accepts\_inters\_accepts\_aut\_inters lang\_inters\_def is\_Init\_def*  
*PDS\_with\_P\_automata.inits\_def P\_Automaton.accepts\_aut\_def image\_iff*)

**lemma** *inters\_accept\_iff*:

**assumes** “*ts12 = inters ts1 ts2*”  
**assumes** “*finals12 = inters\_finals (PDS\_with\_P\_automata.finals final\_initss1 final\_noninits1)*  
*(PDS\_with\_P\_automata.finals final\_initss2 final\_noninits2)*”  
**shows**  
“*accepts\_inters ts12 finals12 (p,w)  $\longleftrightarrow$*   
*PDS\_with\_P\_automata.accepts final\_initss1 final\_noninits1 ts1 (p,w)  $\wedge$*   
*PDS\_with\_P\_automata.accepts final\_initss2 final\_noninits2 ts2 (p,w)*”  
**using** *accepts\_inters\_accepts\_aut\_inters Intersection\_P\_Automaton.inters\_accept\_iff assms*  
**by** (*simp add: Intersection\_P\_Automaton.inters\_accept\_iff accepts\_inters\_accepts\_aut\_inters*  
*PDS\_with\_P\_automata.accepts\_accepts\_aut*)

**lemma** *inters\_lang*:

**assumes** “*ts12 = inters ts1 ts2*”  
**assumes** “*finals12 =*  
*inters\_finals (PDS\_with\_P\_automata.finals final\_initss1 final\_noninits1)*  
*(PDS\_with\_P\_automata.finals final\_initss2 final\_noninits2)*”  
**shows** “*lang\_inters ts12 finals12 =*  
*PDS\_with\_P\_automata.lang final\_initss1 final\_noninits1 ts1  $\cap$*   
*PDS\_with\_P\_automata.lang final\_initss2 final\_noninits2 ts2*”

using *assms* by (auto simp add: PDS\_with\_P\_automata.lang\_def inters\_accept\_iff lang\_inters\_def)

## 5.6 Intersection epsilon-Automata

context *PDS\_with\_P\_automata* begin

interpretation *LTS* transition\_rel .

notation *step\_relp* (infix “ $\Rightarrow$ ” 80)

notation *step\_starp* (infix “ $\Rightarrow^*$ ” 80)

**definition** *accepts\_ε\_inters* :: “((‘ctr\_loc, ‘noninit, ‘label) state \* (‘ctr\_loc, ‘noninit, ‘label) state, ‘label option) transition set  $\Rightarrow$  (‘ctr\_loc, ‘label) conf  $\Rightarrow$  bool” **where**

“*accepts\_ε\_inters* *ts*  $\equiv$   $\lambda(p,w). (\exists q1 \in \text{finals}. \exists q2 \in \text{finals}. ((\text{Init } p, \text{Init } p), w, (q1, q2)) \in \text{LTS}_\varepsilon.\text{trans\_star}_\varepsilon \text{ts})$ ”

**definition** *lang\_ε\_inters* :: “((‘ctr\_loc, ‘noninit, ‘label) state \* (‘ctr\_loc, ‘noninit, ‘label) state, ‘label option) transition set  $\Rightarrow$  (‘ctr\_loc, ‘label) conf set” **where**

“*lang\_ε\_inters* *ts* = {*c*. *accepts\_ε\_inters* *ts* *c*}”

**lemma** *trans\_star\_trans\_star\_ε\_inter*:

**assumes** “*LTS\_ε.ε\_exp* *w1* *w*”

**assumes** “*LTS\_ε.ε\_exp* *w2* *w*”

**assumes** “(*p1*, *w1*, *p2*)  $\in$  *LTS.trans\_star* *ts1*”

**assumes** “(*q1*, *w2*, *q2*)  $\in$  *LTS.trans\_star* *ts2*”

**shows** “((*p1*, *q1*), *w* :: ‘label list, (*p2*, *q2*))  $\in$  *LTS\_ε.trans\_star\_ε* (*inters\_ε* *ts1* *ts2*)”

**using** *assms*

**proof** (induction “length *w1* + length *w2*” arbitrary: *w1* *w2* *w* *p1* *q1* rule: *less\_induct*)

**case** *less*

**then show** ?*case*

**proof** (cases “ $\exists \alpha w1' w2' \beta. w1 = \text{Some } \alpha \# w1' \wedge w2 = \text{Some } \beta \# w2'$ ”)

**case** *True*

**from** *True* **obtain**  $\alpha \beta w1' w2'$  **where** *True'*:

“*w1* = *Some*  $\alpha \# w1'$ ”

“*w2* = *Some*  $\beta \# w2'$ ”

**by** *auto*

**have** “ $\alpha = \beta$ ”

**by** (*metis* *True''*(1) *True''*(2) *LTS\_ε.ε\_exp\_Some\_hd* *less.prem*(1) *less.prem*(2))

**then have** *True'*:

“*w1* = *Some*  $\alpha \# w1'$ ”

“*w2* = *Some*  $\alpha \# w2'$ ”

**using** *True''* **by** *auto*

**define** *w'* **where** “*w'* = *tl* *w*”

**obtain** *p'* **where** *p'\_p*: “(*p1*, *Some*  $\alpha$ , *p'*)  $\in$  *ts1*  $\wedge$  (*p'*, *w1'*, *p2*)  $\in$  *LTS.trans\_star* *ts1*”

**using** *less* *True'*(1) **by** (*metis* *LTS\_ε.trans\_star\_cons\_ε*)

**obtain** *q'* **where** *q'\_p*: “(*q1*, *Some*  $\alpha$ , *q'*)  $\in$  *ts2*  $\wedge$  (*q'*, *w2'*, *q2*)  $\in$  *LTS.trans\_star* *ts2*”

**using** *less* *True'*(2) **by** (*metis* *LTS\_ε.trans\_star\_cons\_ε*)

**have** *ind*: “((*p'*, *q'*), *w'*, *p2*, *q2*)  $\in$  *LTS\_ε.trans\_star\_ε* (*inters\_ε* *ts1* *ts2*)”

**proof** –

**have** “length *w1'* + length *w2'* < length *w1* + length *w2*”

**using** *True'*(1) *True'*(2) **by** *simp*

**moreover**

**have** “*LTS\_ε.ε\_exp* *w1'* *w'*”

**by** (*metis* (*no\_types*) *LTS\_ε.ε\_exp\_def* *less*(2) *True'*(1) *list.map*(2) *list.sel*(3)

*option.simps*(3) *removeAll.simps*(2) *w'\_def*)

**moreover**

**have** “*LTS\_ε.ε\_exp* *w2'* *w'*”

**by** (*metis* (*no\_types*) *LTS\_ε.ε\_exp\_def* *less*(3) *True'*(2) *list.map*(2) *list.sel*(3)

*option.simps*(3) *removeAll.simps*(2) *w'\_def*)

**moreover**

**have** “(*p'*, *w1'*, *p2*)  $\in$  *LTS.trans\_star* *ts1*”

**using** *p'\_p* **by** *simp*

**moreover**

**have** “(*q'*, *w2'*, *q2*)  $\in$  *LTS.trans\_star* *ts2*”

**using** *q'\_p* **by** *simp*



```

ultimately
show “((p', q'), w', p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
  using less(1)[of w1' w2' w' p' q'] by auto
qed
moreover
have “((p1, q1), Some α, (p', q')) ∈ (inters_ε ts1 ts2)”
  by (simp add: inters_ε_def p'_p q'_p)
ultimately
have “((p1, q1), α#w', p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
  by (meson LTS_ε.trans_star_ε.trans_star_ε_step_γ)
moreover
have “length w > 0”
  using less(3) True' LTS_ε.ε_exp_Some_length by metis
moreover
have “hd w = α”
  using less(3) True' LTS_ε.ε_exp_Some_hd by metis
ultimately
show ?thesis
  using w'_def by force
next
case False
note False_outer_outer_outer_outer = False
show ?thesis
proof (cases “w1 = [] ∧ w2 = []”)
  case True
  then have same: “p1 = p2 ∧ q1 = q2”
    by (metis LTS.trans_star_empty less.prem(3) less.prem(4))
  have “w = []”
    using True less(2) LTS_ε.exp_empty_empty by auto
  then show ?thesis
    using less True
    by (simp add: LTS_ε.trans_star_ε.trans_star_ε_refl same)
next
case False
note False_outer_outer_outer = False
show ?thesis
proof (cases “∃ w1'. w1=ε#w1'”)
  case True
  then obtain w1' where True':
    “w1=ε#w1'”
    by auto
  obtain p' where p'_p: “(p1, ε, p') ∈ ts1 ∧ (p', w1', p2) ∈ LTS.trans_star ts1”
    using less True'(1) by (metis LTS_ε.trans_star_cons_ε)
  have q'_p: “(q1, w2, q2) ∈ LTS.trans_star ts2”
    using less by (metis)
  have ind: “((p', q1), w, p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
  proof -
    have “length w1' + length w2 < length w1 + length w2”
      using True'(1) by simp
    moreover
    have “LTS_ε.ε_exp w1' w”
      by (metis (no_types) LTS_ε.ε_exp_def less(2) True'(1) removeAll.simps(2))
    moreover
    have “LTS_ε.ε_exp w2 w”
      by (metis (no_types) less(3))
    moreover
    have “(p', w1', p2) ∈ LTS.trans_star ts1”
      using p'_p by simp
    moreover
    have “(q1, w2, q2) ∈ LTS.trans_star ts2”
      using q'_p by simp
    ultimately
    show “((p', q1), w, p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”

```

```

    using less(1)[of w1' w2 w p' q1] by auto
qed
moreover
have “((p1, q1), ε, (p', q1)) ∈ (inters_ε ts1 ts2)”
  by (simp add: inters_ε_def p'_p q'_p)
ultimately
have “((p1, q1), w, p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
  using LTS_ε.trans_star_ε.simps by fastforce
then
show ?thesis
  by force
next
case False
note False_outer_outer = False
then show ?thesis
proof (cases “∃ w2'. w2 = ε # w2'”)
case True
then obtain w2' where True':
  “w2=ε#w2'”
  by auto
have p'_p: “(p1, w1, p2) ∈ LTS.trans_star ts1”
  using less by (metis)
obtain q' where q'_p: “(q1, ε, q') ∈ ts2 ∧ (q', w2', q2) ∈ LTS.trans_star ts2”
  using less True'(1) by (metis LTS_ε.trans_star_cons_ε)
have ind: “((p1, q'), w, p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
proof -
  have “length w1 + length w2' < length w1 + length w2”
    using True'(1) True'(1) by simp
  moreover
  have “LTS_ε.ε_exp w1 w”
    by (metis (no_types) less(2))
  moreover
  have “LTS_ε.ε_exp w2' w”
    by (metis (no_types) LTS_ε.ε_exp_def less(3) True'(1) removeAll.simps(2))
  moreover
  have “(p1, w1, p2) ∈ LTS.trans_star ts1”
    using p'_p by simp
  moreover
  have “(q', w2', q2) ∈ LTS.trans_star ts2”
    using q'_p by simp
  ultimately
  show “((p1, q'), w, p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
    using less(1)[of w1 w2' w p1 q'] by auto
qed
moreover
have “((p1, q1), ε, (p1, q')) ∈ (inters_ε ts1 ts2)”
  by (simp add: inters_ε_def p'_p q'_p)
ultimately
have “((p1, q1), w, p2, q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)”
  using LTS_ε.trans_star_ε.simps by fastforce
then
show ?thesis
  by force
next
case False
then have “(w1 = [] ∧ (∃ α w2'. w2 = Some α # w2')) ∨ ((∃ α w1'. w1 = Some α # w1') ∧ w2 = [])”
  using False_outer_outer False_outer_outer_outer False_outer_outer_outer_outer
  by (metis neq_Nil_conv option.exhaust_sel)
then show ?thesis
  by (metis LTS_ε.ε_exp_def LTS_ε.ε_exp_Some_length less.premis(1,2) less_numerical_extra(3)
    list.simps(8) list.size(3) removeAll.simps(1))
qed
qed

```

qed  
 qed  
 qed

lemma *trans\_star\_ε\_inter*:

assumes “ $(p1, w :: \text{'label list}, p2) \in LTS_{\epsilon}.trans\_star_{\epsilon} ts1$ ”  
 assumes “ $(q1, w, q2) \in LTS_{\epsilon}.trans\_star_{\epsilon} ts2$ ”  
 shows “ $((p1, q1), w, (p2, q2)) \in LTS_{\epsilon}.trans\_star_{\epsilon} (inters_{\epsilon} ts1 ts2)$ ”

proof –

have “ $\exists w1'. LTS_{\epsilon}.exp w1' w \wedge (p1, w1', p2) \in LTS.trans\_star ts1$ ”  
 using *assms* by (*simp add: LTS\_{\epsilon}.trans\_star\_{\epsilon}\_{\epsilon}\_{exp}\_{trans\_star}*)  
 then obtain *w1'* where “ $LTS_{\epsilon}.exp w1' w \wedge (p1, w1', p2) \in LTS.trans\_star ts1$ ”  
 by *auto*

moreover

have “ $\exists w2'. LTS_{\epsilon}.exp w2' w \wedge (q1, w2', q2) \in LTS.trans\_star ts2$ ”  
 using *assms* by (*simp add: LTS\_{\epsilon}.trans\_star\_{\epsilon}\_{\epsilon}\_{exp}\_{trans\_star}*)  
 then obtain *w2'* where “ $LTS_{\epsilon}.exp w2' w \wedge (q1, w2', q2) \in LTS.trans\_star ts2$ ”  
 by *auto*

ultimately

show *?thesis*

using *trans\_star\_trans\_star\_{\epsilon}\_{inter}* by *metis*

qed

lemma *inters\_trans\_star\_{\epsilon}1*:

assumes “ $(p1q2, w :: \text{'label list}, p2q2) \in LTS_{\epsilon}.trans\_star_{\epsilon} (inters_{\epsilon} ts1 ts2)$ ”  
 shows “ $(fst p1q2, w, fst p2q2) \in LTS_{\epsilon}.trans\_star_{\epsilon} ts1$ ”

using *assms*

proof (*induction rule: LTS\_{\epsilon}.trans\_star\_{\epsilon}.induct[OF assms(1)]*)

case (1 *p*)

then show *?case*

by (*simp add: LTS\_{\epsilon}.trans\_star\_{\epsilon}.trans\_star\_{\epsilon}\_{refl}*)

next

case (2 *p*  $\gamma$  *q'* *w* *q*)

then have *ind*: “ $(fst q', w, fst q) \in LTS_{\epsilon}.trans\_star_{\epsilon} ts1$ ”

by *auto*

from 2(1) have “ $(p, \text{Some } \gamma, q') \in$

“ $\{((p1, q1), \alpha, p2, q2) \mid p1 q1 \alpha p2 q2. (p1, \alpha, p2) \in ts1 \wedge (q1, \alpha, q2) \in ts2\} \cup$   
 $\{((p1, q1), \epsilon, p2, q1) \mid p1 p2 q1. (p1, \epsilon, p2) \in ts1\} \cup$   
 $\{((p1, q1), \epsilon, p1, q2) \mid p1 q1 q2. (q1, \epsilon, q2) \in ts1\}$ ”

unfolding *inters\_{\epsilon}\_{def}* by *auto*

moreover

{

assume “ $(p, \text{Some } \gamma, q') \in \{((p1, q1), \alpha, p2, q2) \mid p1 q1 \alpha p2 q2. (p1, \alpha, p2) \in ts1 \wedge (q1, \alpha, q2) \in ts2\}$ ”

then have “ $\exists p1 q1. p = (p1, q1) \wedge (\exists p2 q2. q' = (p2, q2) \wedge (p1, \text{Some } \gamma, p2) \in ts1 \wedge (q1, \text{Some } \gamma, q2) \in ts2)$ ”

by *simp*

then obtain *p1 q1* where “ $p = (p1, q1) \wedge (\exists p2 q2. q' = (p2, q2) \wedge (p1, \text{Some } \gamma, p2) \in ts1 \wedge (q1, \text{Some } \gamma, q2) \in ts2)$ ”

by *auto*

then have *?case*

using *LTS\_{\epsilon}.trans\_star\_{\epsilon}.trans\_star\_{\epsilon}\_{step}\_{\gamma}\_{ind}* by *fastforce*

}

moreover

{

assume “ $(p, \text{Some } \gamma, q') \in \{((p1, q1), \epsilon, p2, q1) \mid p1 p2 q1. (p1, \epsilon, p2) \in ts1\}$ ”

then have *?case*

by *auto*

}

moreover

{

assume “ $(p, \text{Some } \gamma, q') \in \{((p1, q1), \epsilon, p1, q2) \mid p1 q1 q2. (q1, \epsilon, q2) \in ts1\}$ ”

then have *?case*

by *auto*

}

```

}
ultimately
show ?case
  by auto
next
case (3 p q' w q)
then have ind: "(fst q', w, fst q) ∈ LTS_ε.trans_star_ε ts1"
  by auto
from 3(1) have "(p, ε, q') ∈
  {((p1, q1), α, (p2, q2)) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
  {((p1, q1), ε, (p2, q1)) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
  {((p1, q1), ε, (p1, q2)) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
  unfolding inters_ε_def by auto
moreover
{
  assume "(p, ε, q') ∈ {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
  then have "∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)"
    by simp
  then obtain p1 q1 where "p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)"
    by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
moreover
{
  assume "(p, ε, q') ∈ {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1}"
  then have "∃ p1 p2 q1. p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1"
    by auto
  then obtain p1 p2 q1 where "p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1"
    by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
moreover
{
  assume "(p, ε, q') ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
  then have "∃ p1 q1 q2. p = (p1, q1) ∧ q' = (p1, q2) ∧ (q1, ε, q2) ∈ ts2"
    by auto
  then obtain p1 q1 q2 where "p = (p1, q1) ∧ q' = (p1, q2) ∧ (q1, ε, q2) ∈ ts2"
    by auto
  then have ?case
    using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
}
}
ultimately
show ?case
  by auto
qed

```

```

lemma inters_trans_star_ε:
  assumes "(p1q2, w :: 'label list, p2q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)"
  shows "(snd p1q2, w, snd p2q2) ∈ LTS_ε.trans_star_ε ts2"
  using assms
proof (induction rule: LTS_ε.trans_star_ε.induct[OF assms(1)])
  case (1 p)
  then show ?case
    by (simp add: LTS_ε.trans_star_ε.trans_star_ε_refl)
next
  case (2 p γ q' w q)
  then have ind: "(snd q', w, snd q) ∈ LTS_ε.trans_star_ε ts2"
    by auto
  from 2(1) have "(p, Some γ, q') ∈
    {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
    {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
    {((p1, q1), ε, (p1, q2)) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"

```

```

      {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
  unfolding inters_ε_def by auto
  moreover
  {
    assume "(p, Some γ, q') ∈ {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
    then have "∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, Some γ, p2) ∈ ts1 ∧ (q1, Some γ, q2) ∈
  ts2)"
      by simp
    then obtain p1 q1 where "p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, Some γ, p2) ∈ ts1 ∧ (q1, Some γ,
  q2) ∈ ts2)"
      by auto
    then have ?case
      using LTS_ε.trans_star_ε.trans_star_ε_step_γ ind by fastforce
  }
  moreover
  {
    assume "(p, Some γ, q') ∈ {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1}"
    then have ?case
      by auto
  }
  moreover
  {
    assume "(p, Some γ, q') ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
    then have ?case
      by auto
  }
  ultimately
  show ?case
    by auto
next
  case (∃ p q' w q)
  then have ind: "(snd q', w, snd q) ∈ LTS_ε.trans_star_ε ts2"
    by auto
  from ∃(1) have "(p, ε, q') ∈
    {((p1, q1), α, (p2, q2)) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2} ∪
    {((p1, q1), ε, (p2, q1)) | p1 p2 q1. (p1, ε, p2) ∈ ts1} ∪
    {((p1, q1), ε, (p1, q2)) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
    unfolding inters_ε_def by auto
  moreover
  {
    assume "(p, ε, q') ∈ {((p1, q1), α, p2, q2) | p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
    then have "∃ p1 q1. p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)"
      by simp
    then obtain p1 q1 where "p = (p1, q1) ∧ (∃ p2 q2. q' = (p2, q2) ∧ (p1, ε, p2) ∈ ts1 ∧ (q1, ε, q2) ∈ ts2)"
      by auto
    then have ?case
      using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
  }
  moreover
  {
    assume "(p, ε, q') ∈ {((p1, q1), ε, p2, q1) | p1 p2 q1. (p1, ε, p2) ∈ ts1}"
    then have "∃ p1 p2 q1. p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1"
      by auto
    then obtain p1 p2 q1 where "p = (p1, q1) ∧ q' = (p2, q1) ∧ (p1, ε, p2) ∈ ts1"
      by auto
    then have ?case
      using LTS_ε.trans_star_ε.trans_star_ε_step_ε ind by fastforce
  }
  moreover
  {
    assume "(p, ε, q') ∈ {((p1, q1), ε, p1, q2) | p1 q1 q2. (q1, ε, q2) ∈ ts2}"
    then have "∃ p1 q1 q2. p = (p1, q1) ∧ q' = (p1, q2) ∧ (q1, ε, q2) ∈ ts2"
      by auto
  }

```

```

then obtain  $p1\ q1\ q2$  where “ $p = (p1, q1) \wedge q' = (p1, q2) \wedge (q1, \varepsilon, q2) \in ts2$ ”
  by auto
then have ?case
  using  $LTS_{\varepsilon}.trans\_star_{\varepsilon}.trans\_star_{\varepsilon}.step_{\varepsilon}$  ind by fastforce
}
ultimately
show ?case
  by auto
qed

```

```

lemma  $inters\_trans\_star_{\varepsilon}$  iff:
  “ $((p1, q2), w :: 'label\ list, (p2, q2)) \in LTS_{\varepsilon}.trans\_star_{\varepsilon} (inters_{\varepsilon}\ ts1\ ts2) \longleftrightarrow$ 
   $(p1, w, p2) \in LTS_{\varepsilon}.trans\_star_{\varepsilon}\ ts1 \wedge (q2, w, q2) \in LTS_{\varepsilon}.trans\_star_{\varepsilon}\ ts2$ ”
  by (metis fst_conv inters_trans_star_{\varepsilon} inters_trans_star_{\varepsilon}1 snd_conv trans_star_{\varepsilon}_inter)

```

```

lemma  $inters_{\varepsilon}\_accept_{\varepsilon}$  iff:
  “ $accepts_{\varepsilon}\_inters (inters_{\varepsilon}\ ts1\ ts2)\ c \longleftrightarrow accepts_{\varepsilon}\ ts1\ c \wedge accepts_{\varepsilon}\ ts2\ c$ ”
proof
  assume “ $accepts_{\varepsilon}\_inters (inters_{\varepsilon}\ ts1\ ts2)\ c$ ”
  then show “ $accepts_{\varepsilon}\ ts1\ c \wedge accepts_{\varepsilon}\ ts2\ c$ ”
    using  $accepts_{\varepsilon}\_def\ accepts_{\varepsilon}\_inters\_def\ inters\_trans\_star_{\varepsilon}\ inters\_trans\_star_{\varepsilon}1$  by fastforce
next
  assume asm: “ $accepts_{\varepsilon}\ ts1\ c \wedge accepts_{\varepsilon}\ ts2\ c$ ”
  define  $p$  where “ $p = fst\ c$ ”
  define  $w$  where “ $w = snd\ c$ ”

```

```

  from asm have “ $accepts_{\varepsilon}\ ts1\ (p, w) \wedge accepts_{\varepsilon}\ ts2\ (p, w)$ ”
    using  $p\_def\ w\_def$  by auto
  then have “ $(\exists q \in finals. (Init\ p, w, q) \in LTS_{\varepsilon}.trans\_star_{\varepsilon}\ ts1) \wedge$ 
   $(\exists q \in finals. (Init\ p, w, q) \in LTS_{\varepsilon}.trans\_star_{\varepsilon}\ ts2)$ ”
    unfolding  $accepts_{\varepsilon}\_def$  by auto
  then show “ $accepts_{\varepsilon}\_inters (inters_{\varepsilon}\ ts1\ ts2)\ c$ ”
    using  $accepts_{\varepsilon}\_inters\_def\ p\_def\ trans\_star_{\varepsilon}_inter\ w\_def$  by fastforce
qed

```

```

lemma  $inters_{\varepsilon}\_lang_{\varepsilon}$ : “ $lang_{\varepsilon}\_inters (inters_{\varepsilon}\ ts1\ ts2) = lang_{\varepsilon}\ ts1 \cap lang_{\varepsilon}\ ts2$ ”
  unfolding  $lang_{\varepsilon}\_inters\_def\ lang_{\varepsilon}\_def$  using  $inters_{\varepsilon}\_accept_{\varepsilon}\_iff$  by auto

```

## 5.7 Dual search

```

lemma dual1:
  “ $post\_star (lang_{\varepsilon}\ A1) \cap pre\_star (lang\ A2) = \{c. \exists c1 \in lang_{\varepsilon}\ A1. \exists c2 \in lang\ A2. c1 \Rightarrow^* c \wedge c \Rightarrow^* c2\}$ ”
proof –
  have “ $post\_star (lang_{\varepsilon}\ A1) \cap pre\_star (lang\ A2) = \{c. c \in post\_star (lang_{\varepsilon}\ A1) \wedge c \in pre\_star (lang\ A2)\}$ ”
    by auto
  moreover
  have “ $\dots = \{c. (\exists c1 \in lang_{\varepsilon}\ A1. c1 \Rightarrow^* c) \wedge (\exists c2 \in lang\ A2. c \Rightarrow^* c2)\}$ ”
    unfolding  $post\_star\_def\ pre\_star\_def$  by auto
  moreover
  have “ $\dots = \{c. \exists c1 \in lang_{\varepsilon}\ A1. \exists c2 \in lang\ A2. c1 \Rightarrow^* c \wedge c \Rightarrow^* c2\}$ ”
    by auto
  ultimately
  show ?thesis by metis
qed

```

```

lemma dual2:
  “ $post\_star (lang_{\varepsilon}\ A1) \cap pre\_star (lang\ A2) \neq \{\} \longleftrightarrow (\exists c1 \in lang_{\varepsilon}\ A1. \exists c2 \in lang\ A2. c1 \Rightarrow^* c2)$ ”
proof (rule)
  assume “ $post\_star (lang_{\varepsilon}\ A1) \cap pre\_star (lang\ A2) \neq \{\}$ ”
  then show “ $\exists c1 \in lang_{\varepsilon}\ A1. \exists c2 \in lang\ A2. c1 \Rightarrow^* c2$ ”
    by (auto simp: pre_star_def post_star_def intro: rtranclp_trans)
next
  assume “ $\exists c1 \in lang_{\varepsilon}\ A1. \exists c2 \in lang\ A2. c1 \Rightarrow^* c2$ ”
  then show “ $post\_star (lang_{\varepsilon}\ A1) \cap pre\_star (lang\ A2) \neq \{\}$ ”

```

using dual1 by auto  
qed

lemma LTS\_ε\_of\_LTS\_Some: “(p, Some γ, q′) ∈ LTS\_ε\_of\_LTS A2′ ↔ (p, γ, q′) ∈ A2′”  
unfolding LTS\_ε\_of\_LTS\_def ε\_edge\_of\_edge\_def by (auto simp add: rev\_image\_eqI)

lemma LTS\_ε\_of\_LTS\_None: “(p, None, q′) ∉ LTS\_ε\_of\_LTS A2′”  
unfolding LTS\_ε\_of\_LTS\_def ε\_edge\_of\_edge\_def by (auto)

lemma trans\_star\_ε\_LTS\_ε\_of\_LTS\_trans\_star:  
assumes “(p,w,q) ∈ LTS\_ε.trans\_star\_ε (LTS\_ε\_of\_LTS A2′)”  
shows “(p,w,q) ∈ LTS.trans\_star A2′”  
using assms

proof (induction rule: LTS\_ε.trans\_star\_ε.induct[OF assms(1)])

case (1 p)  
then show ?case  
by (simp add: LTS.trans\_star.trans\_star\_refl)

next

case (2 p γ q′ w q)  
moreover  
have “(p, γ, q′) ∈ A2′”  
using 2(1) using LTS\_ε\_of\_LTS\_Some by metis  
moreover  
have “(q′, w, q) ∈ LTS.trans\_star A2′”  
using “2.IH” 2(2) by auto  
ultimately show ?case  
by (meson LTS.trans\_star.trans\_star\_step)

next

case (3 p q′ w q)  
then show ?case  
using LTS\_ε\_of\_LTS\_None by fastforce

qed

lemma trans\_star\_trans\_star\_ε\_LTS\_ε\_of\_LTS:  
assumes “(p,w,q) ∈ LTS.trans\_star A2′”  
shows “(p,w,q) ∈ LTS\_ε.trans\_star\_ε (LTS\_ε\_of\_LTS A2′)”  
using assms

proof (induction rule: LTS.trans\_star.induct[OF assms(1)])

case (1 p)  
then show ?case  
by (simp add: LTS\_ε.trans\_star\_ε.trans\_star\_ε\_refl)

next

case (2 p γ q′ w q)  
then show ?case  
by (meson LTS\_ε.trans\_star\_ε.trans\_star\_ε\_step\_γ LTS\_ε\_of\_LTS\_Some)

qed

lemma accepts\_ε\_LTS\_ε\_of\_LTS\_iff\_accepts: “accepts\_ε (LTS\_ε\_of\_LTS A2′) (p, w) ↔ accepts A2′ (p, w)”  
using accepts\_ε\_def accepts\_def trans\_star\_ε\_LTS\_ε\_of\_LTS\_trans\_star  
trans\_star\_trans\_star\_ε\_LTS\_ε\_of\_LTS by fastforce

lemma lang\_ε\_LTS\_ε\_of\_LTS\_is\_lang: “lang\_ε (LTS\_ε\_of\_LTS A2′) = lang A2′”  
unfolding lang\_ε\_def lang\_def using accepts\_ε\_LTS\_ε\_of\_LTS\_iff\_accepts by auto

theorem dual\_star\_correct\_early\_termination:

assumes “inits ⊆ LTS.srcs A1”  
assumes “inits ⊆ LTS.srcs A2”  
assumes “isols ⊆ LTS.isolated A1”  
assumes “isols ⊆ LTS.isolated A2”  
assumes “post\_star\_rules\*\* A1 A1′”  
assumes “pre\_star\_rule\*\* A2 A2′”  
assumes “lang\_ε\_inters (inters\_ε A1′ (LTS\_ε\_of\_LTS A2′)) ≠ {}”

shows “ $\exists c1 \in \text{lang}_\varepsilon A1. \exists c2 \in \text{lang } A2. c1 \Rightarrow^* c2$ ”

**proof** –

have “ $\{c. \text{accepts}_\varepsilon A1' c\} \subseteq \text{post\_star } (\text{lang}_\varepsilon A1)$ ”  
 using *assms* using *post\_star\_rules\_subset\_post\_star\_lang* **by auto**

then have *A1'\_correct*: “ $\text{lang}_\varepsilon A1' \subseteq \text{post\_star } (\text{lang}_\varepsilon A1)$ ”  
 unfolding *lang\_ε\_def* **by auto**

have “ $\{c. \text{accepts } A2' c\} \subseteq \text{pre\_star } (\text{lang } A2)$ ”  
 using *pre\_star\_rule\_subset\_pre\_star\_lang*[of *A2 A2'*] *assms* **by auto**

then have *A2'\_correct*: “ $\text{lang } A2' \subseteq \text{pre\_star } (\text{lang } A2)$ ”  
 unfolding *lang\_def* **by auto**

have “ $\text{lang}_\varepsilon \text{inters } (\text{inters}_\varepsilon A1' (\text{LTS}_\varepsilon \text{ of LTS } A2')) = \text{lang}_\varepsilon A1' \cap \text{lang}_\varepsilon (\text{LTS}_\varepsilon \text{ of LTS } A2')$ ”  
 using *inters\_ε\_lang\_ε*[of *A1' “(LTS\_ε of LTS A2’)”*] **by auto**

**moreover**

have “ $\dots = \text{lang}_\varepsilon A1' \cap \text{lang } A2'$ ”  
 using *lang\_ε\_LTS\_ε\_of\_LTS\_is\_lang* **by auto**

**moreover**

have “ $\dots \subseteq \text{post\_star } (\text{lang}_\varepsilon A1) \cap \text{pre\_star } (\text{lang } A2)$ ”  
 using *A1'\_correct A2'\_correct* **by auto**

**ultimately**

have *inters\_correct*: “ $\text{lang}_\varepsilon \text{inters } (\text{inters}_\varepsilon A1' (\text{LTS}_\varepsilon \text{ of LTS } A2')) \subseteq \text{post\_star } (\text{lang}_\varepsilon A1) \cap \text{pre\_star } (\text{lang } A2)$ ”  
**by metis**

**from** *assms* **have** “ $\text{post\_star } (\text{lang}_\varepsilon A1) \cap \text{pre\_star } (\text{lang } A2) \neq \{\}$ ”  
 using *inters\_correct* **by auto**

then show “ $\exists c1 \in \text{lang}_\varepsilon A1. \exists c2 \in \text{lang } A2. c1 \Rightarrow^* c2$ ”  
 using *dual2* **by auto**

**qed**

**theorem** *dual\_star\_correct\_saturation*:

**assumes** “ $\text{inits} \subseteq \text{LTS.sracs } A1$ ”  
**assumes** “ $\text{inits} \subseteq \text{LTS.sracs } A2$ ”  
**assumes** “ $\text{isols} \subseteq \text{LTS.isolated } A1$ ”  
**assumes** “ $\text{isols} \subseteq \text{LTS.isolated } A2$ ”  
**assumes** “*saturation post\_star\_rules* *A1 A1'*”  
**assumes** “*saturation pre\_star\_rule* *A2 A2'*”  
**shows** “ $\text{lang}_\varepsilon \text{inters } (\text{inters}_\varepsilon A1' (\text{LTS}_\varepsilon \text{ of LTS } A2')) \neq \{\} \iff (\exists c1 \in \text{lang}_\varepsilon A1. \exists c2 \in \text{lang } A2. c1 \Rightarrow^* c2)$ ”

**proof** –

have “ $\{c. \text{accepts}_\varepsilon A1' c\} = \text{post\_star } (\text{lang}_\varepsilon A1)$ ”  
 using *post\_star\_rules\_accepts\_ε\_correct*[of *A1 A1'*] *assms* **by auto**

then have *A1'\_correct*: “ $\text{lang}_\varepsilon A1' = \text{post\_star } (\text{lang}_\varepsilon A1)$ ”  
 unfolding *lang\_ε\_def* **by auto**

have “ $\{c. \text{accepts } A2' c\} = \text{pre\_star } (\text{lang } A2)$ ”  
 using *pre\_star\_rule\_accepts\_correct*[of *A2 A2'*] *assms* **by auto**

then have *A2'\_correct*: “ $\text{lang } A2' = \text{pre\_star } (\text{lang } A2)$ ”  
 unfolding *lang\_def* **by auto**

have “ $\text{lang}_\varepsilon \text{inters } (\text{inters}_\varepsilon A1' (\text{LTS}_\varepsilon \text{ of LTS } A2')) = \text{lang}_\varepsilon A1' \cap \text{lang}_\varepsilon (\text{LTS}_\varepsilon \text{ of LTS } A2')$ ”  
 using *inters\_ε\_lang\_ε*[of *A1' “(LTS\_ε of LTS A2’)”*] **by auto**

**moreover**

have “ $\dots = \text{lang}_\varepsilon A1' \cap \text{lang } A2'$ ”  
 using *lang\_ε\_LTS\_ε\_of\_LTS\_is\_lang* **by auto**

**moreover**

have “ $\dots = \text{post\_star } (\text{lang}_\varepsilon A1) \cap \text{pre\_star } (\text{lang } A2)$ ”  
 using *A1'\_correct A2'\_correct* **by auto**

**ultimately**

have *inters\_correct*: “ $\text{lang}_\varepsilon \text{inters } (\text{inters}_\varepsilon A1' (\text{LTS}_\varepsilon \text{ of LTS } A2')) = \text{post\_star } (\text{lang}_\varepsilon A1) \cap \text{pre\_star } (\text{lang } A2)$ ”



```

by metis

show ?thesis
proof
  assume "lang_ε_inters (inters_ε A1' (LTS_ε_of_LTS A2')) ≠ {}"
  then have "post_star (lang_ε A1) ∩ pre_star (lang A2) ≠ {}"
    using inters_correct by auto
  then show "∃ c1 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c2"
    using dual2 by auto
next
  assume "∃ c1 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c2"
  then have "post_star (lang_ε A1) ∩ pre_star (lang A2) ≠ {}"
    using dual2 by auto
  then show "lang_ε_inters (inters_ε A1' (LTS_ε_of_LTS A2')) ≠ {}"
    using inters_correct by auto
qed
qed

```

```

theorem dual_star_correct_early_termination_configs:
  assumes "inits ⊆ LTS.srcs A1"
  assumes "inits ⊆ LTS.srcs A2"
  assumes "isols ⊆ LTS.isolated A1"
  assumes "isols ⊆ LTS.isolated A2"
  assumes "lang_ε A1 = {c1}"
  assumes "lang A2 = {c2}"
  assumes "post_star_rules** A1 A1'"
  assumes "pre_star_rule** A2 A2'"
  assumes "lang_ε_inters (inters_ε A1' (LTS_ε_of_LTS A2')) ≠ {}"
  shows "c1 ⇒* c2"
  using dual_star_correct_early_termination assms by (metis singletonD)

```

```

theorem dual_star_correct_saturation_configs:
  assumes "inits ⊆ LTS.srcs A1"
  assumes "inits ⊆ LTS.srcs A2"
  assumes "isols ⊆ LTS.isolated A1"
  assumes "isols ⊆ LTS.isolated A2"
  assumes "lang_ε A1 = {c1}"
  assumes "lang A2 = {c2}"
  assumes "saturation_post_star_rules A1 A1'"
  assumes "saturation_pre_star_rule A2 A2'"
  shows "lang_ε_inters (inters_ε A1' (LTS_ε_of_LTS A2')) ≠ {} ↔ c1 ⇒* c2"
  using assms dual_star_correct_saturation by auto

```

end

end

theory PDS\_Code

imports PDS "Deriving.Derive"

begin

```

global-interpretation pds: PDS_with_P_automata Δ F_ctr_loc F_ctr_loc_st
  for Δ :: "('ctr_loc::{enum, linorder}, 'label::{finite, linorder}) rule set"
  and F_ctr_loc :: "('ctr_loc) set"
  and F_ctr_loc_st :: "('state::{finite}) set"
  defines pre_star = "PDS_with_P_automata.pre_star_exec Δ"
  and pre_star_check = "PDS_with_P_automata.pre_star_exec_check Δ"
  and inits = "PDS_with_P_automata.inits"
  and finals = "PDS_with_P_automata.finals F_ctr_loc F_ctr_loc_st"
  and accepts = "PDS_with_P_automata.accepts F_ctr_loc F_ctr_loc_st"
  and language = "PDS_with_P_automata.lang F_ctr_loc F_ctr_loc_st"
  and step_starp = "rtranclp (LTS.step_relp (PDS.transition_rel Δ))"
  and accepts_pre_star_check = "PDS_with_P_automata.accept_pre_star_exec_check Δ F_ctr_loc F_ctr_loc_st"
.

```

```

global-interpretation inter: Intersection_P_Automaton
  initial_automaton Init “finals initial_F_ctr_loc initial_F_ctr_loc_st”
  “pre_star Δ final_automaton” “finals final_F_ctr_loc final_F_ctr_loc_st”
  for  $\Delta$  :: “('ctr_loc::{enum, linorder}, 'label::{finite, linorder}) rule set”
  and initial_automaton :: “('ctr_loc, 'state::finite, 'label) state, 'label) transition set”
  and initial_F_ctr_loc :: “'ctr_loc set”
  and initial_F_ctr_loc_st :: “'state set”
  and final_automaton :: “('ctr_loc, 'state, 'label) state, 'label) transition set”
  and final_F_ctr_loc :: “'ctr_loc set”
  and final_F_ctr_loc_st :: “'state set”
  defines nonempty_inter = “P_Automaton.nonempty
    (inters initial_automaton (pre_star final_automaton))
    (( $\lambda p. (Init\ p, Init\ p)$ ))
    (inters_finals (finals initial_F_ctr_loc initial_F_ctr_loc_st)
      (finals final_F_ctr_loc final_F_ctr_loc_st))”
  .

definition “check Δ I IF IF_st F FF FF_st =
  (if pds.inits  $\subseteq$  LTS.srcs F then Some (nonempty_inter Δ I IF IF_st F FF FF_st) else None)”

lemma check_None: “check Δ I IF IF_st F FF FF_st = None  $\longleftrightarrow$   $\neg$  (inits  $\subseteq$  LTS.srcs F)”
  unfolding check_def by auto

lemma check_Some: “check Δ I IF IF_st F FF FF_st = Some b  $\longleftrightarrow$ 
  (inits  $\subseteq$  LTS.srcs F  $\wedge$  b = ( $\exists p\ w\ p'\ w'$ .
    (p, w)  $\in$  language IF IF_st I  $\wedge$ 
    (p', w')  $\in$  language FF FF_st F  $\wedge$ 
    step_starp Δ (p, w) (p', w')))”
  unfolding check_def nonempty_inter_def P_Automaton.nonempty_def
  inter.lang_aut_alt inter.inters_lang
  pds.lang_aut_lang
  by (auto 0 5 simp: pds.pre_star_exec_lang_correct pds.pre_star_def image_iff
  elim!: bezI[rotated])

declare P_Automaton.mark.simps[code]

export-code check checking SML

end

```

## References

- [BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR 1997*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
- [Büc64] J Richard Büchi. Regular canonical systems. *Archiv für mathematische Logik und Grundlagenforschung*, 6(3-4):91–111, 1964.
- [CNDE05] Christopher L. Conway, Kedar S. Namjoshi, Dennis Dams, and Stephen A. Edwards. Incremental algorithms for inter-procedural analysis of safety properties. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV 2005*, volume 3576 of *LNCS*, pages 449–461. Springer, 2005.
- [EK99] Javier Esparza and Jens Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In Wolfgang Thomas, editor, *FoSSaCS 1999*, volume 1578 of *LNCS*, pages 14–30. Springer, 1999.
- [ES01] Javier Esparza and Stefan Schwoon. A bdd-based model checker for recursive programs. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV 2001*, volume 2102 of *LNCS*, pages 324–336. Springer, 2001.

- [JKM<sup>+</sup>18] Jesper Stenbjerg Jensen, Troels Beck Krøgh, Jonas Sand Madsen, Stefan Schmid, Jirí Srba, and Marc Tom Thorgersen. P-Rex: fast verification of MPLS networks with multiple link failures. In Xenofontas A. Dimitropoulos, Alberto Dainotti, Laurent Vanbever, and Theophilus Benson, editors, *CoNEXT 2018*, pages 217–227. ACM, 2018.
- [JKS<sup>+</sup>20] Peter Gjøøl Jensen, Dan Kristiansen, Stefan Schmid, Morten Konggaard Schou, Bernhard Clemens Schrenk, and Jirí Srba. AalWiNes: a fast and quantitative what-if analysis tool for MPLS networks. In Dongsu Han and Anja Feldmann, editors, *CoNEXT 2020*, pages 474–481. ACM, 2020.
- [JSS<sup>+</sup>21] Peter Gjøøl Jensen, Stefan Schmid, Morten Konggaard Schou, Jirí Srba, Juan Vanerio, and Ingo van Duijn. Faster pushdown reachability analysis with applications in network verification. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 170–186. Springer, 2021.
- [Lam09] Peter Lammich. Formalization of dynamic pushdown networks in Isabelle/HOL, 2009. <https://www21.in.tum.de/~lammich/isabelle/dpn-document.pdf>.
- [LMW09] Peter Lammich, Markus Müller-Olm, and Alexander Wenner. Predecessor sets of dynamic pushdown networks with tree-regular constraints. In Ahmed Bouajjani and Oded Maler, editors, *CAV 2009*, volume 5643 of *LNCS*, pages 525–539. Springer, 2009.
- [Sch02a] Stefan Schwoon. *Model checking pushdown systems*. PhD thesis, Technical University Munich, Germany, 2002. <https://d-nb.info/96638976X/34>.
- [Sch02b] Stefan Schwoon. Moped. 2002. <http://www2.informatik.uni-stuttgart.de/fmi/szs/tools/moped/>.
- [SSE05] Dejavuth Suwimonteerabuth, Stefan Schwoon, and Javier Esparza. jMoped: A Java bytecode checker based on Moped. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS 2005*, volume 3440 of *LNCS*, pages 541–545. Springer, 2005.
- [SSST22] Anders Schlichtkrull, Morten Konggaard Schou, Jirí Srba, and Dmitriy Traytel. Differential testing of pushdown reachability with a formally verified oracle. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, pages 369–379. IEEE, 2022.
- [vDJJ<sup>+</sup>21] I. van Duijn, P.G. Jensen, J.S. Jensen, T.B. Krøgh, J.S. Madsen, S. Schmid, J. Srba, and M.T. Thorgersen. Automata-theoretic approach to verification of MPLS networks under link failures. *IEEE/ACM Transactions on Networking*, pages 1–16, 2021.