# 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 post* (forward search), pre* (backward search) and dual* (bi-directional search). For pre* we refine the algorithm to an executable version for which one can generate code using Isabelle's code generator. For pre* and post* we follow Stefan Schwoon's PhD thesis [Sch02a]. The dual* algorithm is from a paper by Jensen et. al presented at ATVA2021 [JSS+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 pre* algorithm for dynamic pushdown networks (DPN) which is a generalization of pushdown systems. Our work is independent from that because the post* of DPNs is not regular and additionally the DPN formalization does not support epsilon transitions which we use for post* and dual*.

# Contents

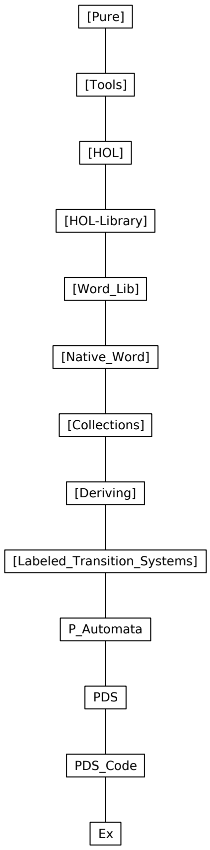


Figure 1: Theory dependency graph

# 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+18, JKS+20, vDJJ+21]. In this formalization we formalize the pre* and post* algorithms [Sch02a] and the dual* algorithm [JSS+21]. For 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+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 ⇒ 'state"
    **and** *finals* :: "'state set"
**begin**

**definition** *initials* :: "'state set" **where**
  "initials ≡ Init ' UNIV"

**lemma** *initials_list*:
  "initials = set (map Init Enum.enum)"
  **using** *enum_UNIV* **unfolding** *initials_def* **by** *force*

**definition** *accepts_aut* :: "'ctr_loc ⇒ 'label list ⇒ bool" **where**
  "accepts_aut ≡ λp w. (∃ q ∈ finals. (Init p, w, q) ∈ trans_star)"

**definition** *lang_aut* :: "('ctr_loc * 'label list) set" **where**
  "lang_aut = {(p,w). accepts_aut p w}"

**definition** *nonempty* **where**
  "nonempty ⟷ lang_aut ≠ {}"

**lemma** *nonempty_alt*:
  "nonempty ⟷ (∃ p. ∃ q ∈ finals. ∃ w. (Init p, w, q) ∈ trans_star)"
  **unfolding** *lang_aut_def nonempty_def accepts_aut_def* **by** *auto*

**typedef** *'a mark_state* = "{(Q :: 'a set, I). I ⊆ Q}"
  **by** *auto*
**setup-lifting** *type_definition_mark_state*
**lift-definition** *get_visited* :: "'a mark_state ⇒ 'a set" **is** *fst* **.**
**lift-definition** *get_next* :: "'a mark_state ⇒ 'a set" **is** *snd* **.**
**lift-definition** *make_mark_state* :: "'a set ⇒ 'a set ⇒ 'a mark_state" **is** "λQ J. (Q ∪ J, J)" **by** *auto*
**lemma** *get_next_get_visited*: "get_next ms ⊆ 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 ∪ J"
  **by** *transfer auto*

**function** *mark* **where**
  "mark ms ⟷
    (let Q = get_visited ms; I = get_next ms in
    if I ∩ finals ≠ {} then True
    else let J = (⋃ (q,w,q')∈transition_relation. if q ∈ I ∧ q' ∉ Q then {q'} else {}) in
      if J = {} then False else mark (make_mark_state Q J))"
  **by** *auto*
**termination by** (*relation* "measure (λms. card (UNIV :: 'a set) − card (get_visited ms :: 'a set))")
    (*fastforce intro*!: *diff_less_mono2 psubset_card_mono split*: *if_splits*)+

**declare** *mark.simps*[*simp del*]

**lemma** *trapped_transitions*: "(p, w, q) ∈ trans_star ⟹
  ∀ p ∈ Q. (∀ γ q. (p, γ, q) ∈ transition_relation ⟶ q ∈ Q) ⟹
  p ∈ Q ⟹ q ∈ Q"
  **by** (*induct p w q rule*: *trans_star.induct*) *auto*

**lemma** *mark_complete*: "(p, w, q) ∈ trans_star ⟹ (get_visited ms − get_next ms) ∩ finals = {} ⟹

```
  ∀p ∈ get_visited ms − get_next ms. ∀q γ. (p, γ, q) ∈ transition_relation ⟶ q ∈ get_visited ms ⟹
  p ∈ get_visited ms ⟹ q ∈ finals ⟹ 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 γ q' w q)
  define J where "J ≡ ⋃(q, w, q')∈transition_relation. if q ∈ get_next ms ∧ q' ∉ get_visited ms then {q'} else
{}"
  show ?case
  proof (cases "J = {}")
    case True
    then have "q' ∈ 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.prems(2) step.prems(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]: "get_visited ms ∪ J − J = get_visited ms"
      by (auto simp: J_def split: if_splits)
   then have "p ∈ get_visited ms ⟹ (p, γ, q) ∈ transition_relation ⟹ q ∉ get_visited ms ⟹ q ∈ J" for p γ q
      using step(5)
      by (cases "p ∈ get_next ms")
        (auto simp only: J_def simp_thms if_True if_False intro!: UN_I[of "(p, γ, 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: "mark ms ⟹ (∃p ∈ get_next ms. ∃q ∈ finals. ∃w. (p, w, q) ∈ 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]: "nonempty = mark (make_mark_state {} (set (map Init 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 ⇒ 'state"
    and finals1 :: "'state set"
    and ts2 :: "('state, 'label) transition set"
    and finals2 :: "'state set"
begin

sublocale pa: P_Automaton "inters ts1 ts2" "(λp. (Init p, 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) ∈ A1.trans_star"
  assumes "(q1, w, q2) ∈ A2.trans_star"
  shows "((p1,q1), w :: 'label list, (p2,q2)) ∈ pa.trans_star"
  using assms
proof (induction w arbitrary: p1 q1)
  case (Cons α w1')
  obtain p' where p'_p: "(p1, α, p') ∈ ts1 ∧ (p', w1', p2) ∈ A1.trans_star"
    using Cons by (metis LTS.trans_star_cons)
  obtain q' where q'_p: "(q1, α, q') ∈ ts2 ∧(q', w1', q2) ∈ A2.trans_star"
    using Cons by (metis LTS.trans_star_cons)
  have ind: "((p', q'), w1', p2, q2) ∈ pa.trans_star"
  proof -
    have "Suc (length w1') = length (α#w1')"
      by auto
    moreover
    have "(p', w1', p2) ∈ A1.trans_star"
      using p'_p by simp
    moreover
    have "(q', w1', q2) ∈ A2.trans_star"
      using q'_p by simp
    ultimately
    show "((p', q'), w1', p2, q2) ∈ pa.trans_star"
      using Cons(1) by auto
  qed
  moreover
  have "((p1, q1), α, (p', q')) ∈ (inters ts1 ts2)"
    by (simp add: inters_def p'_p q'_p)
  ultimately
  have "((p1, q1), α#w1', p2, q2) ∈ pa.trans_star"
    by (meson LTS.trans_star.trans_star_step)
  moreover
  have "length ((α#w1')) > 0"
    by auto
  moreover
  have "hd ((α#w1')) = α"
    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) ∈ pa.trans_star"
  shows "(fst p1q2, w, fst p2q2) ∈ 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 γ q' w q)
  then have ind: "(fst q', w, fst q) ∈ A1.trans_star"
```

```
    by auto
  from 2(1) have "(p, γ, q') ∈
                    {((p1, q1), α, p2, q2) |p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
    unfolding inters_def by auto
  then have "∃p1 q1. p = (p1, q1) ∧ (∃p2 q2. q' = (p2, q2) ∧ (p1, γ, p2) ∈ ts1 ∧ (q1, γ, q2) ∈ ts2)"
    by simp
  then obtain p1 q1 where "p = (p1, q1) ∧ (∃p2 q2. q' = (p2, q2) ∧ (p1, γ, p2) ∈ ts1 ∧ (q1, γ, q2) ∈ 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) ∈ pa.trans_star"
  shows "(snd p1q2, w, snd p2q2) ∈ 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 γ q' w q)
  then have ind: "(snd q', w, snd q) ∈ A2.trans_star"
    by auto
  from 2(1) have "(p, γ, q') ∈
                    {((p1, q1), α, p2, q2) |p1 q1 α p2 q2. (p1, α, p2) ∈ ts1 ∧ (q1, α, q2) ∈ ts2}"
    unfolding inters_def by auto
  then have "∃p1 q1. p = (p1, q1) ∧ (∃p2 q2. q' = (p2, q2) ∧ (p1, γ, p2) ∈ ts1 ∧ (q1, γ, q2) ∈ ts2)"
    by simp
  then obtain p1 q1 where "p = (p1, q1) ∧ (∃p2 q2. q' = (p2, q2) ∧ (p1, γ, p2) ∈ ts1 ∧ (q1, γ, q2) ∈ 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)) ∈ pa.trans_star ⟷ (p1, w, p2) ∈ A1.trans_star ∧ (q2, w, q2) ∈ 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 ⟷ A1.accepts_aut p w ∧ A2.accepts_aut p w"
proof
  assume "accepts_aut_inters p w"
  then show "A1.accepts_aut p w ∧ 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 ∧ A2.accepts_aut p w"
  then have "(∃q∈finals1. (Init p, w, q) ∈ A1.trans_star) ∧
             (∃q∈finals2. (Init p, w, q) ∈ 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) ∈ 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 ∩ 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 ⇒ 'state"*
**begin**

**definition** *accepts_aut_ε* :: *"'ctr_loc ⇒ 'label list ⇒ bool"* **where**
*"accepts_aut_ε ≡ λp w. (∃ q ∈ finals. (Init p, w, q) ∈ trans_star_ε)"*

**definition** *lang_aut_ε* :: *"('ctr_loc * 'label list) set"* **where**
*"lang_aut_ε = {(p,w). accepts_aut_ε p w}"*

**definition** *nonempty_ε* **where**
*"nonempty_ε ⟷ lang_aut_ε ≠ {}"*

**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 ⇒ 'state"*
**and** *ts2* :: *"('state, 'label option) transition set"*
**and** *finals2* :: *"'state set"*
**begin**

**abbreviation** *ε* :: *"'label option"* **where**
*"ε == None"*

**sublocale** *pa*: *P_Automaton_ε "inters_ε ts1 ts2" "inters_finals finals1 finals2" "(λp. (Init p, Init p))"*
.

**definition** *accepts_aut_inters_ε* **where**
*"accepts_aut_inters_ε p w = pa.accepts_aut_ε p w"*

**definition** *lang_aut_inters_ε* :: *"('ctr_loc * 'label list) set"* **where**
*"lang_aut_inters_ε = {(p,w). accepts_aut_inters_ε 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) ∈ A1.trans_star"*
**assumes** *"(q1, w2, q2) ∈ A2.trans_star"*
**shows** *"((p1,q1), w :: 'label list, (p2,q2)) ∈ 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 "∃α w1' w2' β. w1=Some α#w1' ∧ w2=Some β#w2'"*)
**case** *True*
**from** *True* **obtain** *α β 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.prems(1) less.prems(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.prems(3) less.prems(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 = 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_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) ∈ A1.trans_star"
            using p'_p by simp
          moreover
          have "(q', w2', q2) ∈ A2.trans_star"
            using q'_p by simp
          ultimately
          show "((p1, q'), w, p2, q2) ∈ pa.trans_star_ε"
            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) ∈ pa.trans_star_ε"
          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.prems(1) less.prems(2)
              less_numeral_extra(3) list.simps(8) list.size(3) removeAll.simps(1))
      qed
    qed
  qed
 qed
qed

lemma trans_star_ε_inter:
  assumes "(p1, w :: 'label list, p2) ∈ A1.trans_star_ε"
  assumes "(q1, w, q2) ∈ A2.trans_star_ε"
  shows "((p1, q1), w, (p2, q2)) ∈ pa.trans_star_ε"
proof -
  have "∃w1'. LTS_ε.ε_exp w1' w ∧ (p1, w1', p2) ∈ A1.trans_star"
    using assms by (simp add: LTS_ε.trans_star_ε_ε_exp_trans_star)
  then obtain w1' where "LTS_ε.ε_exp w1' w ∧ (p1, w1', p2) ∈ A1.trans_star"
    by auto
  moreover
  have "∃w2'. LTS_ε.ε_exp w2' w ∧ (q1, w2', q2) ∈ A2.trans_star"
    using assms by (simp add: LTS_ε.trans_star_ε_ε_exp_trans_star)
  then obtain w2' where "LTS_ε.ε_exp w2' w ∧ (q1, w2', q2) ∈ A2.trans_star"
    by auto
  ultimately
  show ?thesis
    using trans_star_trans_star_ε_inter by metis
qed

lemma inters_trans_star_ε1:
  assumes "(p1q2, w :: 'label list, p2q2) ∈ pa.trans_star_ε"
  shows "(fst p1q2, w, fst p2q2) ∈ A1.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: "(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** Δ :: "(*'ctr_loc, 'label::finite*) *rule set*"

**begin**

**primrec** *lbl* :: "*'label operation ⇒ 'label list*" **where**
"*lbl pop* = []"
| "*lbl* (*swap γ*) = [*γ*]"
| "*lbl* (*push γ γ'*) = [*γ, γ'*]"

**definition** *is_rule* :: "*'ctr_loc × 'label ⇒ 'ctr_loc × 'label operation ⇒ bool*" (**infix** "↪" *80*) **where**
"*pγ ↪ p'w ≡ (pγ,p'w) ∈ Δ*"

**inductive-set** *transition_rel* :: "((*'ctr_loc, 'label*) *conf* × *unit* × (*'ctr_loc, 'label*) *conf*) *set*" **where**
"(*p, γ*) ↪ (*p', w*) ⟹
((*p, γ#w'*), (), (*p'*, (*lbl w*)@*w'*)) ∈ *transition_rel*"

**interpretation** *LTS transition_rel* **.**

**notation** *step_relp* (**infix** "⇒" *80*)
**notation** *step_starp* (**infix** "⇒*" *80*)

**lemma** *step_relp_def2*:
"(*p, γw'*) ⇒ (*p',ww'*) ⟷ (∃*γ w' w. γw' = γ#w' ∧ ww'* = (*lbl w*)@*w'* ∧ (*p, γ*) ↪ (*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 "⇒" 80)
notation step_starp (infix "⇒*" 80)

definition accepts :: "(('ctr_loc, 'noninit, 'label) state, 'label) transition set ⇒ ('ctr_loc, 'label) conf ⇒ bool" where
  "accepts ts ≡ λ(p,w). (∃q ∈ finals. (Init p,w,q) ∈ LTS.trans_star ts)"

lemma accepts_accepts_aut: "accepts ts (p, w) ⟷ 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 ⇒ ('ctr_loc, 'label) conf ⇒
bool" where
  "accepts_ε ts ≡ λ(p,w). (∃q ∈ finals. (Init p,w,q) ∈ 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 tsts': "ts ⊆ ts'"
  obtain p l where pl_p: "c = (p,l)"
    by (cases c)
  obtain q where q_p:  "q ∈ finals ∧ (Init p, l, q) ∈ LTS.trans_star ts"
    using accepts_ts unfolding pl_p accepts_def by auto
  then have "(Init p, l, q) ∈ LTS.trans_star ts'"
    using tsts' 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') ∈ ts ⟹ accepts ts (p', w) ⟹ 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 ⇒ ('ctr_loc, 'label) conf set" where
  "lang ts = {c. accepts ts c}"

lemma lang_lang_aut: "lang ts = (λ(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 ⇒ ('ctr_loc, 'label) conf set"
where
  "lang_ε ts = {c. accepts_ε ts c}"
```

## 5.1 Saturations

```
definition saturated :: "('c, 'l) sat_rule ⇒ ('c, 'l) transition set ⇒ bool" where
  "saturated rule ts ⟷ (∄ts'. rule ts ts')"

definition saturation :: "('c, 'l) sat_rule ⇒ ('c, 'l) transition set ⇒ ('c, 'l) transition set ⇒ bool" where
  "saturation rule ts ts' ⟷ rule** ts ts' ∧ saturated rule ts'"

lemma no_infinite:
  assumes "⋀ts ts' :: ('c ::finite, 'l::finite) transition set. rule ts ts' ⟹ card ts' = Suc (card ts)"
  assumes "∀ i :: nat. rule (tts i) (tts (Suc i))"
  shows "False"
proof −
  define f where "f i = card (tts i)" for i
  have f_Suc: "∀ i. f i < f (Suc i)"
    using assms f_def lessI by metis
  have "∀ i. ∃ j. f j > i"
  proof
    fix i
    show "∃ 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 "∃ 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 "⋀ts ts' :: ('c ::finite, 'l::finite) transition set. rule ts ts' ⟹ card ts' = Suc (card ts)"
  shows "¬(∃ tts. (∀ i :: nat. rule (tts i) (tts (Suc i))))"
  using assms no_infinite by blast

lemma saturation_exi:
  assumes "⋀ts ts' :: ('c ::finite, 'l::finite) transition set. rule ts ts' ⟹ card ts' = Suc (card ts)"
  shows "∃ ts'. saturation rule ts ts'"
proof (rule ccontr)
  assume a: "∄ts'. saturation rule ts ts'"
  define g where "g ts = (SOME ts'. rule ts ts')" for ts
  define tts where "tts i = (g ^^ i) ts" for i
  have "∀ i :: nat. rule** ts (tts i) ∧ rule (tts i) (tts (Suc i))"
  proof
    fix i
    show "rule** ts (tts i) ∧ 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 ^^ i) ts)) (g (g ((g ^^ 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, γ) ↪ (p', w) ⟹ (Init p', lbl w, q) ∈ LTS.trans_star ts ⟹*
*(Init p, γ, q) ∉ ts ⟹ pre_star_rule ts (ts ∪ {(Init p, γ, q)})"*

**definition** *pre_star1* :: *"(('ctr_loc, 'noninit, 'label) state, 'label) transition set ⇒ (('ctr_loc, 'noninit, 'label) state, 'label) transition set"* **where**
*"pre_star1 ts =*
*(⋃((p, γ), (p', w)) ∈ Δ. ⋃ q ∈ LTS.reach ts (Init p') (lbl w). {(Init p, γ, q)})"*

**lemma** *pre_star1_mono*: *"mono pre_star1"*
**unfolding** *pre_star1_def*
**by** (*auto simp*: *mono_def LTS.trans_star_code*[*symmetric*] *elim*!: *bexI*[*rotated*]
*LTS_trans_star_mono*[*THEN monoD*, *THEN subsetD*])

**lemma** *pre_star_rule_pre_star1*:
**assumes** *"X ⊆ pre_star1 ts"*
**shows** *"pre_star_rule** ts (ts ∪ 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, γ) ↪ (p', w)"*
*"(Init p', lbl w, q) ∈ LTS.trans_star ts"* **and** *x*:
*"x = (Init p, γ, q)"*
**by** (*auto simp*: *pre_star1_def is_rule_def LTS.trans_star_code*)
**with** *insert* **show** *?case*
**proof** (*cases "(Init p, γ, q) ∈ 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 (((λs. s ∪ pre_star1 s) ^^ k) ts)"*
**by** (*induct k*) (*auto elim*!: *rtranclp_trans intro*: *pre_star_rule_pre_star1*)

**definition** *"pre_star_loop = while_option (λs. s ∪ pre_star1 s ≠ s) (λs. s ∪ pre_star1 s)"*
**definition** *"pre_star_exec = the o pre_star_loop"*
**definition** *"pre_star_exec_check A = (if inits ⊆ LTS.srcs A then pre_star_loop A else None)"*

**definition** *"accept_pre_star_exec_check A c = (if inits ⊆ LTS.srcs A then Some (accepts (pre_star_exec A) c) else None)"*

```
lemma while_option_finite_subset_Some: fixes C :: "'a set"
  assumes "mono f" and "!!X. X ⊆ C ⟹ f X ⊆ C" and "finite C" and X: "X ⊆ C" "X ⊆ f X"
  shows "∃ P. while_option (λA. f A ≠ A) f X = Some P"
proof(rule measure_while_option_Some[where
    f= "%A::'a set. card C − card A" and P= "%A. A ⊆ C ∧ A ⊆ f A" and s= X])
  fix A assume A: "A ⊆ C ∧ A ⊆ f A" "f A ≠ A"
  show "(f A ⊆ C ∧ f A ⊆ f (f A)) ∧ card C − card (f A) < card C − card A"
    (is "?L ∧ ?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: "∃ t. pre_star_loop s = Some t"
  unfolding pre_star_loop_def
  by (rule while_option_finite_subset_Some[where C=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' ⊆ s then s else pre_star_exec (s ∪ 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: "pre_star_loop ts = Some t"
    by blast
  obtain k where k: "t = ((λs. s ∪ pre_star1 s) ^^ k) ts" and le: "pre_star1 t ⊆ t"
    using while_option_stop2[OF t[unfolded pre_star_loop_def]] by auto
  have "(⋃{us. pre_star_rule t us}) − t ⊆ 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 ⇒ (('ctr_loc, 'noninit,
'label) state, 'label option) transition set ⇒ bool" where
  add_trans_pop:
  "(p, γ) ↪ (p', pop) ⟹
   (Init p, [γ], q) ∈ LTS_ε.trans_star_ε ts ⟹
   (Init p', ε, q) ∉ ts ⟹
   post_star_rules ts (ts ∪ {(Init p', ε, q)})"
| add_trans_swap:
  "(p, γ) ↪ (p', swap γ') ⟹
   (Init p, [γ], q) ∈ LTS_ε.trans_star_ε ts ⟹
   (Init p', Some γ', q) ∉ ts ⟹
   post_star_rules ts (ts ∪ {(Init p', Some γ', q)})"
| add_trans_push_1:
  "(p, γ) ↪ (p', push γ' γ'') ⟹
   (Init p, [γ], q) ∈ LTS_ε.trans_star_ε ts ⟹
   (Init p', Some γ', Isolated p' γ') ∉ ts ⟹
   post_star_rules ts (ts ∪ {(Init p', Some γ', Isolated p' γ')})"
| add_trans_push_2:
  "(p, γ) ↪ (p', push γ' γ'') ⟹
   (Init p, [γ], q) ∈ LTS_ε.trans_star_ε ts ⟹
   (Isolated p' γ', Some γ'', q) ∉ ts ⟹
   (Init p', Some γ', Isolated p' γ') ∈ ts ⟹
   post_star_rules ts (ts ∪ {(Isolated p' γ', Some γ'', q)})"
```

```
lemma pre_star_rule_mono:
  "pre_star_rule ts ts' ⟹ ts ⊂ ts'"
  unfolding pre_star_rule.simps by auto

lemma post_star_rules_mono:
  "post_star_rules ts ts' ⟹ ts ⊂ ts'"
proof(induction rule: post_star_rules.induct)
  case (add_trans_pop p γ p' q ts)
  then show ?case by auto
next
  case (add_trans_swap p γ p' γ' q ts)
  then show ?case by auto
next
  case (add_trans_push_1 p γ p' γ' γ'' q ts)
  then show ?case by auto
next
  case (add_trans_push_2 p γ p' γ' γ'' q ts)
  then show ?case by auto
qed

lemma pre_star_rule_card_Suc: "pre_star_rule ts ts' ⟹ card ts' = Suc (card ts)"
  unfolding pre_star_rule.simps by auto

lemma post_star_rules_card_Suc: "post_star_rules ts ts' ⟹ card ts' = Suc (card ts)"
proof(induction rule: post_star_rules.induct)
  case (add_trans_pop p γ p' q ts)
  then show ?case by auto
next
  case (add_trans_swap p γ p' γ' q ts)
  then show ?case by auto
next
  case (add_trans_push_1 p γ p' γ' γ'' q ts)
  then show ?case by auto
next
  case (add_trans_push_2 p γ p' γ' γ'' q ts)
  then show ?case by auto
qed


lemma pre_star_saturation_termination:
  "¬(∃ tts. (∀ 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:
  "¬(∃ tts. (∀ 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 "∃ ts'. saturation pre_star_rule ts ts'"
  using pre_star_rule_card_Suc saturation_exi by blast

lemma post_star_saturation_exi:
  shows "∃ 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 ⟹ A ⊆ B"
proof(induction rule: pre_star_rule.inducts)
  case (add_trans p γ p' w q rel)
  then show ?case
    by auto
qed
```

```
lemma post_star_rules_incr: "post_star_rules A B ⟹ A ⊆ B"
proof(induction rule: post_star_rules.inducts)
  case (add_trans_pop p γ p' q ts)
  then show ?case
    by auto
next
  case (add_trans_swap p γ p' γ' q ts)
  then show ?case
    by auto
next
  case (add_trans_push_1 p γ p' γ' γ'' q ts)
  then show ?case
    by auto
next
  case (add_trans_push_2 p γ p' γ' γ'' q ts)
  then show ?case
    by auto
qed

lemma saturation_rtranclp_pre_star_rule_incr: "pre_star_rule** A B ⟹ A ⊆ 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: "post_star_rules** A B ⟹ A ⊆ 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:
  "pre_star_rule** A A' ⟹ LTS.trans_star A ⊆ LTS.trans_star A'"
  using mono_def LTS_trans_star_mono saturation_rtranclp_pre_star_rule_incr by metis

lemma post_star'_incr_trans_star:
  "post_star_rules** A A' ⟹ LTS.trans_star A ⊆ LTS.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:
  "post_star_rules** A A' ⟹ LTS_ε.trans_star_ε A ⊆ LTS_ε.trans_star_ε 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:
  "saturation pre_star_rule A A' ⟹ LTS.trans_star A ⊆ LTS.trans_star A'"
  by (simp add: pre_star'_incr_trans_star saturation_def)

lemma post_star_lim'_incr_trans_star:
  "saturation post_star_rules A A' ⟹ LTS.trans_star A ⊆ LTS.trans_star A'"
  by (simp add: post_star'_incr_trans_star saturation_def)

lemma post_star_lim'_incr_trans_star_ε:
  "saturation post_star_rules A A' ⟹ LTS_ε.trans_star_ε A ⊆ LTS_ε.trans_star_ε A'"
  by (simp add: post_star'_incr_trans_star_ε saturation_def)
```

### 5.3 Pre* lemmas

```
lemma inits_srcs_iff_Ctr_Loc_srcs:
  "inits ⊆ LTS.srcs A ⟷ (∄q γ q'. (q, γ, Init q') ∈ A)"
proof
  assume "inits ⊆ LTS.srcs A"
  then show "∄q γ q'. (q, γ, Init q') ∈ A"
    by (simp add: Collect_mono_iff LTS.srcs_def inits_def)
next
  assume "∄q γ q'. (q, γ, Init q') ∈ A"
  show "inits ⊆ LTS.srcs A"
    by (metis LTS.srcs_def2 inits_def ‹∄q γ q'. (q, γ, Init q') ∈ A› mem_Collect_eq
        state.collapse(1) subsetI)
qed

lemma lemma_3_1:
  assumes "p'w ⇒* pv"
  assumes "pv ∈ 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 "∃q ∈ finals. (Init p, v, q) ∈ 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 ∈ finals ∧ (Init p'', u, q) ∈ LTS.trans_star A'"
    unfolding accepts_def by auto
  have "∃γ w1 u1. w=γ#w1 ∧ u=lbl u1@w1 ∧ (p', γ) ↪ (p'', u1)"
    using p''u_def p'w_def step.hyps(1) step_relp_def2 by auto
  then obtain γ w1 u1 where γ_w1_u1_p: "w=γ#w1 ∧ u=lbl u1@w1 ∧ (p', γ) ↪ (p'', u1)"
    by blast

  then have "∃q1. (Init p'', lbl u1, q1) ∈ LTS.trans_star A' ∧ (q1, w1, q) ∈ LTS.trans_star A'"
    using q_p LTS.trans_star_split by auto

  then obtain q1 where q1_p: "(Init p'', lbl u1, q1) ∈ LTS.trans_star A' ∧ (q1, w1, q) ∈ LTS.trans_star A'"
    by auto

  then have in_A': "(Init p', γ, q1) ∈ A'"
    using γ_w1_u1_p add_trans[of p' γ p'' u1 q1 A'] saturated_def saturation_def step.prems by metis

  then have "(Init p', γ#w1, q) ∈ LTS.trans_star A'"
    using LTS.trans_star.trans_star_step q1_p by meson
  then have t_in_A': "(Init p', w, q) ∈ LTS.trans_star A'"
    using γ_w1_u1_p by blast

  from q_p t_in_A' have "q ∈ finals ∧ (Init p', w, q) ∈ 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 "∃γ w' wa. w = γ # w' ∧ y = lbl wa @ w' ∧ (p', γ) ↪ (p1, wa)"
    using step_relp_def2[of p' w p1 y] by auto
  then obtain γ w' wa where γ_w'_wa_p: " w = γ # w' ∧ y = lbl wa @ w' ∧ (p', γ) ↪ (p1, wa)"
    by metis
  then have "(p, u @ v) ⇒* (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) ⇒* (p1, y)"
  shows "(p, u @ v) ⇒* (p1, y @ v)"
  using assms step_relp_append_aux by auto

lemma step_relp_append_empty:
  assumes "(p, u) ⇒* (p1, [])"
  shows "(p, u @ v) ⇒* (p1, v)"
  using step_relp_append[OF assms] by auto

lemma lemma_3_2_a':
  assumes "inits ⊆ LTS.srcs A"
  assumes "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(2) assms(3)
proof (induction arbitrary: p q w rule: rtranclp_induct)
  case base
  then have "(Init p, w, q) ∈ LTS.trans_star A ∧ (p, w) ⇒* (p, w)"
    by auto
  then show ?case
    by auto
next
  case (step Aiminus1 Ai)

  from step(2) obtain p1 γ p2 w2 q' where p1_γ_p2_w2_q'_p:
    "Ai = Aiminus1 ∪ {(Init p1, γ, q')}"
    "(p1, γ) ↪ (p2, w2)"
    "(Init p2, lbl w2, q') ∈ LTS.trans_star Aiminus1"
    "(Init p1, γ, q') ∉ Aiminus1"
    by (meson pre_star_rule.cases)

  define t :: "(('ctr_loc, 'noninit, 'label) state, 'label) transition"
    where "t = (Init p1, γ, q')"

  obtain ss where ss_p: "(Init p, w, ss, q) ∈ LTS.trans_star_states Ai"
    using step(4) LTS.trans_star_trans_star_states by metis

  define j where "j = count (transitions_of' (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 "(Init p, w, q) ∈ LTS.trans_star Aiminus1"
      using count_zero_remove_trans_star_states_trans_star p1_γ_p2_w2_q'_p(1) t_def by metis
    then show ?case
      using step.IH by metis
  next
    case (Suc j')
    have "∃u v u_ss v_ss.
```

```
        ss = u_ss@v_ss ∧ w = u@[γ]@v ∧
        (Init p,u,u_ss, Init p1) ∈ LTS.trans_star_states Aiminus1 ∧
        (Init p1,[γ],q') ∈ LTS.trans_star Ai ∧
        (q',v,v_ss,q) ∈ LTS.trans_star_states Ai ∧
        (Init p, w, ss, q) = ((Init p, u, u_ss, Init p1), γ) @@^γ (q', v, v_ss, q)"
    using split_at_first_t[of "Init p" w ss q Ai j' "Init p1" γ q' Aiminus1]
    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 ∧ w = u@[γ]@v"
    "(Init p,u,u_ss, Init p1) ∈ LTS.trans_star_states Aiminus1"
    "(Init p1,[γ],q') ∈ LTS.trans_star Ai"
    "(q',v,v_ss,q) ∈ LTS.trans_star_states Ai"
    "(Init p, w, ss, q) = ((Init p, u, u_ss, Init p1), γ) @@^γ (q', v, v_ss, q)"
    by blast
  from this(2) have "∃p'' w''. (Init p'', w'', Init p1) ∈ LTS.trans_star A ∧ (p, u) ⇒* (p'', w'')"
    using Suc(1)[of p u _ "Init p1"] step.IH step.prems(1)
    by (meson LTS.trans_star_states_trans_star LTS.trans_star_trans_star_states)
  from this this(1) have VIII: "(p, u) ⇒* (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: "∃w2_ss. (Init p2, lbl w2, w2_ss, q') ∈ LTS.trans_star_states Aiminus1"
    using LTS.trans_star_trans_star_states[of "Init p2" "lbl w2" q' Aiminus1] by auto
  then obtain w2_ss where III_2: "(Init p2, lbl w2, w2_ss, q') ∈ LTS.trans_star_states Aiminus1"
    by blast

  from III have V:
    "(Init p2, lbl w2, w2_ss, q') ∈ LTS.trans_star_states Aiminus1"
    "(q', v, v_ss, q) ∈ LTS.trans_star_states Ai"
    using III_2 ‹(q', v, v_ss, q) ∈ LTS.trans_star_states Ai› 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') ∈ 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) ∈ 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 == λx. count (transitions_of' x) t"

    have "countts (Init p, w, ss, q) = Suc j' "
      using Suc.prems(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.srcs 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.srcs 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 ∈ finals" "(Init p, w, q) ∈ LTS.trans_star A'"
    by auto
  then have "∃p' w'. (p,w) ⇒* (p',w') ∧ (Init p', w', q) ∈ 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) ⇒* (p',w')" "(Init p', w', q) ∈ LTS.trans_star A"
    by auto
  then have "(p', w') ∈ lang A"
    unfolding lang_def unfolding accepts_def using q_p(1) by auto
  then have "(p,w) ∈ pre_star (lang A)"
    unfolding pre_star_def using p'_w'_p(1) by auto
  then show "c ∈ pre_star (lang A)"
    unfolding p_def w_def by auto
qed

— Corresponds to Schwoon's theorem 3.2
theorem pre_star_rule_accepts_correct:
  assumes "inits ⊆ LTS.srcs A"
  assumes "saturation pre_star_rule A A'"
  shows "{c. accepts A' c} = pre_star (lang A)"
proof (rule; rule)
  fix c :: "'ctr_loc × 'label list"
  define p where "p = fst c"
  define w where "w = snd c"
  assume "c ∈ pre_star (lang A)"
  then have "(p,w) ∈ pre_star (lang A)"
    unfolding p_def w_def by auto
  then have "∃p' w'. (p',w') ∈ lang A ∧ (p,w) ⇒* (p',w')"
    unfolding pre_star_def by force
  then obtain p' w' where "(p',w') ∈ lang A ∧ (p,w) ⇒* (p',w')"
    by auto
  then have "∃q ∈ finals. (Init p, w, q) ∈ LTS.trans_star A'"
    using lemma_3_1 assms(2) unfolding accepts_def by force
  then have "accepts A' (p,w)"
    unfolding accepts_def by auto
  then show "c ∈ {c. accepts A' c}"
    using p_def w_def by auto
next
  fix c :: "'ctr_loc × 'label list"
  assume "c ∈ {w. accepts A' w}"
  then show "c ∈ pre_star (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 "inits ⊆ LTS.srcs A"
  assumes "saturation pre_star_rule A A'"
  shows "lang A' = pre_star (lang A)"
  using assms(1) assms(2) lang_def pre_star_rule_accepts_correct by auto

theorem pre_star_exec_accepts_correct:
  assumes "inits ⊆ LTS.srcs A"
  shows "{c. accepts (pre_star_exec A) c} = pre_star (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 "inits ⊆ LTS.srcs A"
  shows "lang (pre_star_exec A) = pre_star (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 "pre_star_exec_check A ≠ None"
```

```
  shows "{c. accepts (the (pre_star_exec_check A)) c} = pre_star (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.srcs 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.srcs 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.srcs 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.srcs 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.srcs 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 "accepts_ε A' (fst p'w, snd p'w)"
  using assms
proof (induct arbitrary: pv rule: rtranclp_induct)
  case base
  show ?case
    using assms post_star_lim'_incr_trans_star_ε
    by (auto simp: lang_ε_def accepts_ε_def)
next
  case (step p''u p'w)
  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_ε A' (p'', u)"
    using assms(2) p''_def step.hyps(3) step.prems(2) u_def by metis
  then have "∃ q. q ∈ finals ∧ (Init p'', u, q) ∈ LTS_ε.trans_star_ε A'"
    by (auto simp: accepts_ε_def)
  then obtain q where q_p: "q ∈ finals ∧ (Init p'', u, q) ∈ LTS_ε.trans_star_ε A'"
    by metis
  then have "∃ u_ε. q ∈ finals ∧ LTS_ε.ε_exp u_ε u ∧ (Init p'', u_ε, q) ∈ LTS.trans_star A'"
    using LTS_ε.trans_star_ε_iff_ε_exp_trans_star[of "Init p''" u q A'] by auto
  then obtain u_ε where II: "q ∈ finals" "LTS_ε.ε_exp u_ε u" "(Init p'', u_ε, q) ∈ LTS.trans_star A'"
    by blast
  have "∃ γ u1 w1. u=γ#u1 ∧ w=lbl w1@u1 ∧ (p'', γ) ↪ (p', w1)"
    using p''u_def p'w_def step.hyps(2) step_relp_def2 by auto
  then obtain γ u1 w1 where III: "u=γ#u1" "w=lbl w1@u1" "(p'', γ) ↪ (p', w1)"
    by blast

  have p'_inits: "Init p' ∈ inits"
    unfolding inits_def by auto
  have p''_inits: "Init p'' ∈ inits"
    unfolding inits_def by auto

  have "∃ γ_ε u1_ε. LTS_ε.ε_exp γ_ε [γ] ∧ LTS_ε.ε_exp u1_ε u1 ∧ (Init p'', γ_ε@u1_ε, q) ∈ LTS.trans_star
A'"
  proof −
    have "∃ γ_ε u1_ε. LTS_ε.ε_exp γ_ε [γ] ∧ LTS_ε.ε_exp u1_ε u1 ∧ u_ε = γ_ε @ u1_ε"
      using LTS_ε.ε_exp_split'[of u_ε γ u1] II(2) III(1) by auto
    then obtain γ_ε u1_ε where "LTS_ε.ε_exp γ_ε [γ] ∧ LTS_ε.ε_exp u1_ε u1 ∧ u_ε = γ_ε @ u1_ε"
      by auto
    then have "(Init p'', γ_ε@u1_ε , q) ∈ LTS.trans_star A'"
      using II(3) by auto
    then show ?thesis
      using ‹LTS_ε.ε_exp γ_ε [γ] ∧ LTS_ε.ε_exp u1_ε u1 ∧ u_ε = γ_ε @ u1_ε› by blast
  qed
  then obtain γ_ε u1_ε where
    iii: "LTS_ε.ε_exp γ_ε [γ]" and
    iv: "LTS_ε.ε_exp u1_ε u1" "(Init p'', γ_ε@u1_ε, q) ∈ LTS.trans_star A'"
    by blast
  then have VI: "∃ q1. (Init p'', γ_ε, q1) ∈ LTS.trans_star A' ∧ (q1, u1_ε, q) ∈ LTS.trans_star A'"
    by (simp add: LTS.trans_star_split)
  then obtain q1 where VI: "(Init p'', γ_ε, q1) ∈ LTS.trans_star A'" "(q1, u1_ε, q) ∈ LTS.trans_star A'"
    by blast

  then have VI_2: "(Init p'', [γ], q1) ∈ LTS_ε.trans_star_ε A'" "(q1, u1, q) ∈ LTS_ε.trans_star_ε A'"
    by (meson LTS_ε.trans_star_ε_iff_ε_exp_trans_star iii VI(2) iv(1))+

  show ?case
```

```
  proof (cases w1)
    case pop
    then have r: "(p'', γ) ↪ (p', pop)"
      using III(3) by blast
    then have "(Init p', ε, q1) ∈ A'"
      using VI_2(1) add_trans_pop assms saturated_def saturation_def p'_inits by metis
    then have "(Init p', w, q) ∈ LTS_ε.trans_star_ε A'"
      using III(2) VI_2(2) pop LTS_ε.trans_star_ε.trans_star_ε_step_ε by fastforce
    then have "accepts_ε A' (p', w)"
      unfolding accepts_ε_def using II(1) by blast
    then show ?thesis
      using p'_def w_def by force
  next
    case (swap γ')
    then have r: "(p'', γ) ↪ (p', swap γ')"
      using III(3) by blast
    then have "(Init p', Some γ', q1) ∈ A'"
      by (metis VI_2(1) add_trans_swap assms(3) saturated_def saturation_def)
    have "(Init p', w, q) ∈ LTS_ε.trans_star_ε A'"
      using III(2) LTS_ε.trans_star_ε.trans_star_ε_step_γ VI_2(2) append_Cons append_self_conv2
        lbl.simps(3) swap ‹(Init p', Some γ', q1) ∈ A'› by fastforce
    then have "accepts_ε A' (p', w)"
      unfolding accepts_ε_def
      using II(1) by blast
    then show ?thesis
      using p'_def w_def by force
  next
    case (push γ' γ'')
    then have r: "(p'', γ) ↪ (p', push γ' γ'')"
      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 "(Init p', Some γ', Isolated p' γ') ∈ 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 "(Isolated p' γ', Some γ'', q1) ∈ A'"
      using assms(3) using saturated_def saturation_def by metis
    have "(Init p', [γ'], Isolated p' γ') ∈ LTS_ε.trans_star_ε A' ∧
          (Isolated p' γ', [γ''], q1) ∈ LTS_ε.trans_star_ε A' ∧
          (q1, u1, q) ∈ LTS_ε.trans_star_ε A'"
      by (metis LTS_ε.trans_star_ε.simps VI_2(2) ‹(Init p', Some γ', Isolated p' γ') ∈ A'›
          ‹(Isolated p' γ', Some γ'', q1) ∈ A'›)
    have "(Init p', w, q) ∈ LTS_ε.trans_star_ε A'"
      using III(2) VI_2(2) ‹(Init p', Some γ', Isolated p' γ') ∈ A'›
        ‹(Isolated p' γ', Some γ'', q1) ∈ A'› push LTS_ε.append_edge_edge_trans_star_ε by auto
    then have "accepts_ε A' (p', w)"
      unfolding accepts_ε_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) ⇒* (p',w)"
  and "(p, v) ∈ lang_ε A"
  and "saturation post_star_rules A A'"
  shows "accepts_ε A' (p', w)"
  using assms lemma_3_3' by force

lemma init_only_hd:
  assumes "(ss, w) ∈ LTS.path_with_word A"
  assumes "inits ⊆ LTS.srcs A"
```

```
  assumes "count (transitions_of (ss, w)) t > 0"
  assumes "t = (Init p1, γ, q1)"
  shows "hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1"
  using assms LTS.source_only_hd by (metis LTS.srcs_def2 inits_srcs_iff_Ctr_Loc_srcs)

lemma no_edge_to_Ctr_Loc_avoid_Ctr_Loc:
  assumes "(p, w, qq) ∈ LTS.trans_star Aiminus1"
  assumes "w ≠ []"
  assumes "inits ⊆ LTS.srcs Aiminus1"
  shows "qq ∉ inits"
  using assms LTS.no_end_in_source by (metis subset_iff)

lemma no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε:
  assumes "(p, [γ], qq) ∈ LTS_ε.trans_star_ε Aiminus1"
  assumes "inits ⊆ LTS.srcs Aiminus1"
  shows "qq ∉ 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 "∄ q γ q'. (q, γ, Init q') ∈ A"
  shows "∄ q γ q'. (q, γ, Init q') ∈ Ai"
using assms
proof (induction rule: rtranclp_induct)
  case base
  then show ?case by auto
next
  case (step Aiminus1 Ai)
  then have ind: "∄ q γ q'. (q, γ, Init q') ∈ Aiminus1"
    by auto
  from step(2) show ?case
  proof (cases rule: post_star_rules.cases)
    case (add_trans_pop p γ p' q)
    have "q ∉ inits"
      using ind no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε inits_srcs_iff_Ctr_Loc_srcs
      by (metis local.add_trans_pop(3))
    then have "∄ 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 γ p' γ' q)
    have "q ∉ inits"
      using add_trans_swap ind no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε inits_srcs_iff_Ctr_Loc_srcs
      by metis
    then have "∄ 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 γ p' γ' γ'' q)
    have "q ∉ inits"
      using add_trans_push_1 ind no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε inits_srcs_iff_Ctr_Loc_srcs
      by metis
    then have "∄ 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 γ p' γ' γ'' q)
    have "q ∉ inits"
      using add_trans_push_2 ind no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε inits_srcs_iff_Ctr_Loc_srcs
      by metis
```

```
    then have "∄qq. q = 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 "post_star_rules** A Ai"
  assumes "inits ⊆ LTS.srcs A"
  shows "inits ⊆ 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 ⊆ LTS.srcs A"
  assumes "N ⊆ LTS.sinks A"
  shows "∀p γ q. (p, γ, q) ∈ A ⟶ p ∉ N ∧ q ∉ N"
  by (metis LTS.srcs_def2 LTS.sinks_def2 assms(1) assms(2) in_mono)

lemma post_star_rules_Isolated_source_invariant':
  assumes "post_star_rules** A A'"
  assumes "isols ⊆ LTS.isolated A"
  assumes "(Init p', Some γ', Isolated p' γ') ∉ A'"
  shows "∄p γ. (p, γ, Isolated p' γ') ∈ 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 Aiminus1 Ai)
  from step(2) show ?case
  proof (cases rule: post_star_rules.cases)
    case (add_trans_pop p''' γ'' p'' q)
    then have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then have nin: "∄p γ. (p, γ, Isolated p' γ') ∈ Aiminus1"
      using local.add_trans_pop(1) step.IH step.prems(1,2) by fastforce
    then have "Isolated p' γ' ≠ q"
      using add_trans_pop(4) LTS_ε.trans_star_not_to_source_ε LTS.srcs_def2
      by (metis local.add_trans_pop(3) state.distinct(3))
    then have "∄p γ. (p, γ, Isolated p' γ') = (Init p'', ε, q)"
      by auto
    then show ?thesis
      using nin add_trans_pop(1) by auto
  next
    case (add_trans_swap p'''' γ'' p'' γ''' q)
    then have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then have nin: "∄p γ. (p, γ, Isolated p' γ') ∈ Aiminus1"
      using local.add_trans_swap(1) step.IH step.prems(1,2) by fastforce
    then have "Isolated p' γ' ≠ q"
      using LTS.srcs_def2
      by (metis state.distinct(4) LTS_ε.trans_star_not_to_source_ε local.add_trans_swap(3))
    then have "∄p γ. (p, γ, Isolated p' γ') = (Init p'', Some γ''', q)"
      by auto
    then show ?thesis
      using nin add_trans_swap(1) by auto
  next
    case (add_trans_push_1 p'''' γ'' p'' γ''' γ'''' q)
    then have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then show ?thesis
```

```
      using add_trans_push_1(1) Un_iff state.inject(2) prod.inject singleton_iff step.IH
        step.prems(1,2) by blast
  next
    case (add_trans_push_2 p'''' γ'' p'' γ''' γ'''' q)
    have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) .
    then have nin: "∄p γ. (p, γ, Isolated p' γ') ∈ Aiminus1"
      using local.add_trans_push_2(1) step.IH step.prems(1) by fastforce
    then have "Isolated p' γ' ≠ q"
      using LTS.srcs_def2 local.add_trans_push_2(3)
      by (metis state.disc(1,3) LTS_ε.trans_star_not_to_source_ε)
    then have "∄p γ. (p, γ, Isolated p' γ') = (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 ⊆ LTS.isolated A"
  assumes "(Init p', Some γ', Isolated p' γ') ∉ A'"
  shows "Isolated p' γ' ∈ LTS.srcs A'"
  by (meson LTS.srcs_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 ⊆ LTS.isolated A"
  assumes "(Init p', Some γ', Isolated p' γ') ∉ A'"
  shows "∄p γ. (Isolated p' γ', γ, p) ∈ 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 Aiminus1 Ai)
  from step(2) show ?case
  proof (cases rule: post_star_rules.cases)
    case (add_trans_pop p''' γ'' p'' q)
    then have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then have nin: "∄p γ. (Isolated p' γ', γ, p) ∈ Aiminus1"
      using local.add_trans_pop(1) step.IH step.prems(1,2) by fastforce
    then have "Isolated p' γ' ≠ q"
      using add_trans_pop(4)
        LTS_ε.trans_star_not_to_source_ε[of "Init p'''" "[γ'']" q Aiminus1 "Isolated p' γ'"]
        post_star_rules_Isolated_source_invariant local.add_trans_pop(1) step.hyps(1) step.prems(1,2)
        UnI1 local.add_trans_pop(3) by (metis (full_types) state.distinct(3))
    then have "∄p γ. (p, γ, Isolated p' γ') = (Init p'', ε, q)"
      by auto
    then show ?thesis
      using nin add_trans_pop(1) by auto
  next
    case (add_trans_swap p'''' γ'' p'' γ''' q)
    then have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then have nin: "∄p γ. (Isolated p' γ', γ, p) ∈ Aiminus1"
      using local.add_trans_swap(1) step.IH step.prems(1,2) by fastforce
    then have "Isolated p' γ' ≠ q"
      using LTS_ε.trans_star_not_to_source_ε[of "Init p''''" "[γ'']" q Aiminus1]
        local.add_trans_swap(3) post_star_rules_Isolated_source_invariant[of _ Aiminus1 p' γ'] UnCI
```

```
      local.add_trans_swap(1) step.hyps(1) step.prems(1,2) state.simps(7) by metis
    then have "∄p γ. (p, γ, Isolated p' γ') = (Init p'', Some γ''', q)"
      by auto
    then show ?thesis
      using nin add_trans_swap(1) by auto
  next
    case (add_trans_push_1 p'''' γ'' p'' γ''' γ''''' q)
    then have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then show ?thesis
      using add_trans_push_1(1) Un_iff state.inject prod.inject singleton_iff step.IH
        step.prems(1,2) by blast
  next
    case (add_trans_push_2 p'''' γ'' p'' γ''' γ'''' q)
    have "(Init p', Some γ', Isolated p' γ') ∉ Ai"
      using step.prems(2) by blast
    then have nin: "∄p γ. (Isolated p' γ', γ, p) ∈ Aiminus1"
      using local.add_trans_push_2(1) step.IH step.prems(1,2) by fastforce
    then have "Isolated p' γ' ≠ q"
      using state.disc(3)
        LTS_ε.trans_star_not_to_source_ε[of "Init p''''" "[γ'']" q Aiminus1 "Isolated p' γ'"]
        local.add_trans_push_2(3)
      using post_star_rules_Isolated_source_invariant[of _ Aiminus1 p' γ'] UnCI
        local.add_trans_push_2(1) step.hyps(1) step.prems(1,2) state.disc(1) by metis
    then have "∄p γ. (Isolated p' γ', γ, p) = (Init p'', ε, q)"
      by auto
    then show ?thesis
      using nin add_trans_push_2(1)
      using local.add_trans_push_2 step.prems(2) by auto
  qed
qed

lemma post_star_rules_Isolated_sink_invariant:
  assumes "post_star_rules** A A'"
  assumes "isols ⊆ LTS.isolated A"
  assumes "(Init p', Some γ', Isolated p' γ') ∉ A'"
  shows "Isolated p' γ' ∈ 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_constains_successors_states:
  assumes "post_star_rules** A A'"
  assumes "inits ⊆ LTS.srcs A"
  assumes "isols ⊆ LTS.isolated A"
  assumes "(Init p, w, ss, q) ∈ LTS.trans_star_states A'"
  shows "(¬is_Isolated q ⟶ (∃p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p',w') ⇒* (p, LTS_ε.remove_ε
w))) ∧
        (is_Isolated q ⟶ (the_Ctr_Loc q, [the_Label q]) ⇒* (p, LTS_ε.remove_ε w))"
  using assms
proof (induction arbitrary: p q w ss rule: rtranclp_induct)
  case base
  {
    assume ctr_loc: "is_Init q ∨ is_Noninit q"
    then have "(Init p, LTS_ε.remove_ε w, q) ∈ LTS_ε.trans_star_ε A"
      using base LTS_ε.trans_star_states_trans_star_ε by metis
    then have "∃p' w'. (p', w', q) ∈ LTS_ε.trans_star_ε A"
      by auto
    then have ?case
      using ctr_loc ‹(Init p, LTS_ε.remove_ε w, q) ∈ LTS_ε.trans_star_ε A› 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.prems(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 Aiminus1 Ai)
    from step(2) have "∃ p1 γ p2 w2 q1. Ai = Aiminus1 ∪ {(p1, γ, q1)} ∧ (p1, γ, q1) ∉ Aiminus1"
      by (cases rule: post_star_rules.cases) auto
    then obtain p1 γ q1 where p1_γ_p2_w2_q'_p:
      "Ai = Aiminus1 ∪ {(p1, γ, q1)}"
      "(p1, γ, q1) ∉ Aiminus1"
      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 Aiminus1"
        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 ⊆ LTS.srcs Ai"
    using init_Ai by auto
  moreover
  have "0 < count (transitions_of (ss, w)) t"
    by (metis Suc.prems(1) transitions_of'.simps zero_less_Suc)
  moreover
  have "t = (Init p1, ε, q1)"
    using t_def by auto
  moreover
  have "Init p1 ∈ inits"
    by (simp add: inits_def)
  ultimately
  show "hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1"
    using init_only_hd[of ss w Ai t p1 ε q1] by auto
qed

have "transition_list (ss, w) ≠ []"
  by (metis LTS.trans_star_states_path_with_word LTS.path_with_word.simps Suc.prems(1)
      Suc.prems(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], [ε]) @´ (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, [ε], [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) ∈ LTS.trans_star_states Ai"
    using Suc.prems(2) by blast
  moreover
  have "t = hd (transition_list' (Init p, w, ss, q))"
    using ‹hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1›
    by fastforce
  moreover
  have "transition_list' (Init p, w, ss, q) ≠ []"
    by (simp add: ‹transition_list (ss, w) ≠ []›)
  moreover
  have "t = (Init p1, ε, 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 ∧ count (transitions_of (ss, w)) t = 1›
  by force
have "(Init p1, [ε], [Init p1, q1], q1) ∈ LTS.trans_star_states Ai"
proof -
  have "(Init p1, ε, q1) ∈ Ai"
    using local.add_trans_pop(1) by auto
  moreover
  have "(Init p1, ε, q1) ∉ Aiminus1"
    by (simp add: local.add_trans_pop)
  ultimately
  show "(Init p1, [ε], [Init p1, q1], q1) ∈ 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) ∈ LTS.trans_star_states Aiminus1"
proof -
  from Suc(3) have "(ss, w) ∈ LTS.path_with_word Ai"
    by (meson LTS.trans_star_states_path_with_word)
```

```
      then have tl_ss_w_Ai: "(tl ss, tl w) ∈ LTS.path_with_word Ai"
        by (metis LTS.path_with_word.simps ‹transition_list (ss, w) ≠ []› list.sel(3)
            transition_list.simps(2))
      from t_hd_once have zero_p1_ε_q1: "0 = count (transitions_of (tl ss, tl w)) (Init p1, ε, q1)"
        using count_append_path_with_word_γ[of "[hd ss]" "[]" "tl ss" "hd w" "tl w" "Init p1" ε q1, simplified]
          ‹(Init p1, [ε], [Init p1, q1], q1) ∈ LTS.trans_star_states Ai›
          ‹transition_list (ss, w) ≠ []› Suc.prems(2) VII
          LTS.transition_list_Cons[of "Init p" w ss q Ai ε q1] by (auto simp: t_def)
      have Ai_Aiminus1: "Ai = Aiminus1 ∪ {(Init p1, ε, q1)}"
        using local.add_trans_pop(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" ε q1 Aiminus1]
      have "(tl ss, tl w) ∈ LTS.path_with_word Aiminus1"
        by auto
      moreover
      have "hd (tl ss) = q1"
        using Suc.prems(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.prems(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.prems(2)
            last_ConsR list.collapse)
    qed
    have "w = ε # (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 (tl w)"
      by (metis LTS_ε.remove_ε_def removeAll.simps(2))

    have "∃γ2'. LTS_ε.ε_exp γ2' [γ2] ∧ (Init p2, γ2', q1) ∈ LTS.trans_star Aiminus1"
      using add_trans_pop
      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 IIII_1: "(Init p2, γ2', ss2, q1) ∈ LTS.trans_star_states Aiminus1"
      by blast
    have IIII_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 IIII: "(Init p2, γ2' @ tl w, ss2 @ (tl (tl ss)), q) ∈ LTS.trans_star_states Aiminus1"
      using IIII_1 IIII_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 IIII
      using step.IH step.prems(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 (γ2′ @ tl w)) ⇒* (p, LTS_ε.remove_ε (tl w))”
      proof −
        have γ2′_γ2: “LTS_ε.remove_ε γ2′ = [γ2]”
            by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_def ‹LTS_ε.ε_exp γ2′ [γ2] ∧ (Init p2, γ2′, q1) ∈
LTS.trans_star Aiminus1›)
        have “(p′,w′) ⇒* (p2, LTS_ε.remove_ε (γ2′ @ tl w))”
          using steps by auto
        moreover
        have rule: “(p2, γ2) ↪ (p, pop)”
          using VI VII by auto
        from steps have steps′: “(p′, w′) ⇒* (p2, γ2 # (LTS_ε.remove_ε (tl w)))”
          using γ2′_γ2
          by (metis Cons_eq_appendI LTS_ε.remove_ε_append_dist self_append_conv2)
        from rule steps′ have “(p2, γ2 # (LTS_ε.remove_ε (tl w))) ⇒* (p, LTS_ε.remove_ε (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_ε.remove_ε (γ2′ @ tl w)) ⇒* (p, LTS_ε.remove_ε (tl w))”
          by (simp add: LTS_ε.remove_ε_append_dist γ2′_γ2)
        ultimately
        show “(p′,w′) ⇒* (p2, LTS_ε.remove_ε (γ2′ @ tl w)) ∧
              (p2, LTS_ε.remove_ε (γ2′ @ tl w)) ⇒* (p, LTS_ε.remove_ε (tl w))”
          by auto
      qed
      then have “(p′,w′) ⇒* (p, LTS_ε.remove_ε (tl w)) ∧ (Init p′, w′, q) ∈ LTS_ε.trans_star_ε A”
        using VIII by force
      then have “∃p′ w′. (Init p′, w′, q) ∈ LTS_ε.trans_star_ε A ∧ (p′, w′) ⇒* (p, LTS_ε.remove_ε w)”
        using w_tl_ε by auto
      then show ?thesis
        using ctr_q ‹p = p1› by blast
    next
      assume “is_Isolated q”
      from V have “(the_Ctr_Loc q, [the_Label q]) ⇒* (p2, γ2#(LTS_ε.remove_ε w))”
        by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_append_dist LTS_ε.remove_ε_def
            ‹LTS_ε.ε_exp γ2′ [γ2] ∧ (Init p2, γ2′, q1) ∈ LTS.trans_star Aiminus1› ‹is_Isolated q›
            append_Cons append_self_conv2 w_tl_ε)

      then have “(the_Ctr_Loc q, [the_Label q]) ⇒* (p1, LTS_ε.remove_ε w)”
        using VI by (metis append_Nil lbl.simps(1) rtranclp.simps step_relp_def2)
      then have “(the_Ctr_Loc q, [the_Label q]) ⇒* (p, LTS_ε.remove_ε w)”
        using VII by auto
      then show ?thesis
        using ‹is_Isolated q› by blast
    qed
  next
    case (add_trans_swap p2 γ2 p1 γ′ q1)
    note III = add_trans_swap(3)
    note VI = add_trans_swap(2)
    have t_def: “t = (Init p1, Some γ′, q1)”
      using local.add_trans_swap(1) local.add_trans_swap p1_γ_p2_w2_q′_p(1) t_def by blast
    have init_Ai: “inits ⊆ LTS.srcs 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 ∧ 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 ⊆ LTS.srcs Ai”
```

```
    using init_Ai by auto
  moreover
  have "0 < count (transitions_of (ss, w)) t"
    by (metis Suc.prems(1) transitions_of'.simps zero_less_Suc)
  moreover
  have "t = (Init p1, Some γ', q1)"
    using t_def
    by auto
  moreover
  have "Init p1 ∈ inits"
    using inits_def by force
  ultimately
  show "hd (transition_list (ss, w)) = t ∧ 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) ≠ []"
  by (metis LTS.trans_star_states_path_with_word LTS.path_with_word.simps Suc.prems(1,2)
      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], [Some γ']) @´ (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 γ'], [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) ∈ LTS.trans_star_states Ai"
    using Suc.prems(2) by blast
  moreover
  have "t = hd (transition_list' (Init p, w, ss, q))"
    using ‹hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1› by fastforce
  moreover
  have "transition_list' (Init p, w, ss, q) ≠ []"
    by (simp add: ‹transition_list (ss, w) ≠ []›)
  moreover
  have "t = (Init p1, Some γ', 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 ∧ count (transitions_of (ss, w)) t = 1› by force
have "(Init p1, [Some γ'], [Init p1, q1], q1) ∈ LTS.trans_star_states Ai"
proof -
  have "(Init p1, Some γ', q1) ∈ Ai"
    using local.add_trans_swap(1) by auto
  moreover
  have "(Init p1, Some γ', q1) ∉ Aiminus1"
    using local.add_trans_swap(4) by blast
  ultimately
  show "(Init p1, [Some γ'], [Init p1, q1], q1) ∈ 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) ∈ LTS.trans_star_states Aiminus1"
proof -
  from Suc(3) have "(ss, w) ∈ LTS.path_with_word Ai"
    by (meson LTS.trans_star_states_path_with_word)
  then have tl_ss_w_Ai: "(tl ss, tl w) ∈ LTS.path_with_word Ai"
    by (metis LTS.path_with_word.simps ‹transition_list (ss, w) ≠ []› list.sel(3)
        transition_list.simps(2))
  from t_hd_once have zero_p1_ε_q1: "0 = count (transitions_of (tl ss, tl w)) (Init p1, Some γ', q1)"
     using count_append_path_with_word_γ[of "[hd ss]" "[]" "tl ss" "hd w" "tl w" "Init p1" "Some γ'" q1,
```

*simplified*]
‹(*Init p1*, [*Some* γ′], [*Init p1*, *q1*], *q1*) ∈ *LTS.trans_star_states Ai*› ‹*transition_list* (*ss*, *w*) ≠ []›
*Suc.prems*(*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.prems*(*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.prems*(*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.prems*(*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** *IIII_1*: "(*Init p2*, γ2′, *ss2*, *q1*) ∈ *LTS.trans_star_states Aiminus1*"
**by** *blast*
**have** *IIII_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** *IIII*: "(*Init p2*, γ2′ @ *tl w*, *ss2* @ (*tl* (*tl ss*)), *q*) ∈ *LTS.trans_star_states Aiminus1*"
**using** *IIII_1 IIII_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** *IIII*
**using** *step.IH step.prems*(*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 (γ2' @ tl w)) ⇒* (p, γ' # LTS_ε.remove_ε (tl w))"
    proof -
      have γ2'_γ2: "LTS_ε.remove_ε γ2' = [γ2]"
           by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_def ‹LTS_ε.ε_exp γ2' [γ2] ∧ (Init p2, γ2', q1) ∈
LTS.trans_star Aiminus1›)
      have "(p',w') ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl w))"
        using steps by auto
      moreover
      have rule: "(p2, γ2) ↪ (p, swap γ')"
        using VI VII by auto
      from steps have steps': "(p', w') ⇒* (p2, γ2 # (LTS_ε.remove_ε (tl w)))"
        using γ2'_γ2
        by (metis Cons_eq_appendI LTS_ε.remove_ε_append_dist self_append_conv2)
      from rule steps' have "(p2,  γ2 # (LTS_ε.remove_ε (tl w))) ⇒* (p, γ' # 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 (γ2' @ tl w)) ⇒* (p, γ' # LTS_ε.remove_ε (tl w))"
        by (simp add: LTS_ε.remove_ε_append_dist γ2'_γ2)
      ultimately
      show "(p',w') ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl w)) ∧
           (p2, LTS_ε.remove_ε (γ2' @ tl w)) ⇒* (p, γ' # LTS_ε.remove_ε (tl w))"
        by auto
    qed
    then have "(p',w') ⇒* (p, γ' # LTS_ε.remove_ε (tl w)) ∧ (Init p', w', q) ∈ LTS_ε.trans_star_ε A"
      using VIII by force
    then have "∃p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p, LTS_ε.remove_ε w)"
      using LTS_ε.remove_ε_Cons_tl by (metis ‹w = Some γ' # tl w›)
    then show ?thesis
      using ctr_q ‹p = p1› by blast
  next
    assume "is_Isolated q"
    from V this have "(the_Ctr_Loc q, [the_Label q]) ⇒* (p2, LTS_ε.remove_ε (γ2' @ tl w))"
      by auto
    then have "(the_Ctr_Loc q, [the_Label q]) ⇒* (p2, γ2#(LTS_ε.remove_ε (tl w)))"
      by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_append_dist LTS_ε.remove_ε_def
          ‹LTS_ε.ε_exp γ2' [γ2] ∧ (Init p2, γ2', q1) ∈ LTS.trans_star Aiminus1› append_Cons
          append_self_conv2)
    then have "(the_Ctr_Loc q, [the_Label q]) ⇒* (p1, γ' # 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]) ⇒* (p, γ' # LTS_ε.remove_ε (tl w))"
      using VII by auto
    then show ?thesis
      using ‹is_Isolated q›
      by (metis LTS_ε.remove_ε_Cons_tl w_tl_ε)
  qed
next
  case (add_trans_push_1 p2 γ2 p1 γ1 γ'' q1')
  then have t_def: "t = (Init p1, Some γ1, Isolated p1 γ1)"
    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
    by (meson r_into_rtranclp)
  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 ⊆ LTS.srcs Ai"
      using init_Ai by auto
```

```
      moreover
      have "0 < count (transitions_of (ss, w)) t"
        by (metis Suc.prems(1) transitions_of'.simps zero_less_Suc)
      moreover
      have "t = (Init p1, Some γ1, Isolated p1 γ1)"
        using t_def by auto
      moreover
      have "Init p1 ∈ inits"
        using inits_def by fastforce
      ultimately
      show "hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1"
        using init_only_hd[of ss w Ai t] by auto
    qed
    have "transition_list (ss, w) ≠ []"
      by (metis LTS.trans_star_states_path_with_word LTS.path_with_word.simps Suc.prems(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 "(Init p, w, ss, q) ∈ LTS.trans_star_states Ai"
        using Suc.prems(2) by blast
      moreover
      have "t = hd (transition_list' (Init p, w, ss, q))"
        using ‹hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1› by fastforce
      moreover
      have "transition_list' (Init p, w, ss, q) ≠ []"
        by (simp add: ‹transition_list (ss, w) ≠ []›)
      moreover
      have "t = (Init p1, Some γ1, Isolated p1 γ1)"
        using t_def by auto
      ultimately
      show "p = p1"
        using LTS.hd_is_hd by fastforce
    qed
    from add_trans_push_1(4) have "∄p γ. (Isolated p1 γ1, γ, p) ∈ Aiminus1"
      using post_star_rules_Isolated_sink_invariant[of A Aiminus1 p1 γ1] step.hyps(1)
        step.prems(1,2,3) unfolding LTS.sinks_def by blast
    then have "∄p γ. (Isolated p1 γ1, γ, p) ∈ Ai"
      using local.add_trans_push_1(1) by blast
    then have ss_w_short: "ss = [Init p1, Isolated p1 γ1] ∧ w = [Some γ1]"
      using Suc.prems(2) VII ‹hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1› t_def
      LTS.nothing_after_sink[of "Init p1" "Isolated p1 γ1" "tl (tl ss)" "Some γ1" "tl w" Ai] ‹transition_list (ss,
w) ≠ []›
      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 = Isolated p1 γ1"
      using LTS.trans_star_states_last Suc.prems(2) by fastforce
    have "(p1, [γ1]) ⇒* (p, LTS_ε.remove_ε w)"
      using ss_w_short unfolding LTS_ε.remove_ε_def
      using VII by force
    have "(the_Ctr_Loc q, [the_Label q]) ⇒* (p, LTS_ε.remove_ε w)"
      by (simp add: ‹(p1, [γ1]) ⇒* (p, LTS_ε.remove_ε w)› q_ext)
    then show ?thesis
      using q_ext by auto
  next
    case (add_trans_push_2 p2 γ2 p1 γ1 γ'' q')
    note IX = add_trans_push_2(3)
    note XIII = add_trans_push_2(2)
    have t_def: "t = (Isolated p1 γ1, Some γ'', q')"
      using local.add_trans_push_2(1,4) 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
    by (meson r_into_rtranclp)

  from Suc(2,3) split_at_first_t[of "Init p" w ss q Ai j "Isolated p1 γ1" "Some γ''" q' Aiminus1] t_def
  have "∃ u v u_ss v_ss.
          ss = u_ss @ v_ss ∧
          w = u @ [Some γ''] @ v ∧
          (Init p, u, u_ss, Isolated p1 γ1) ∈ LTS.trans_star_states Aiminus1 ∧
          (Isolated p1 γ1, [Some γ''], q') ∈ LTS.trans_star Ai ∧ (q', v, v_ss, q) ∈ 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 γ''] @ v" and
      X_1: "(Init p, u, u_ss, Isolated p1 γ1) ∈ LTS.trans_star_states Aiminus1" and
      out_trans: "(Isolated p1 γ1, [Some γ''], q') ∈ LTS.trans_star Ai" and
      path: "(q', v, v_ss, q) ∈ LTS.trans_star_states Ai"
    by auto
  from step(3)[of p u u_ss "Isolated p1 γ1"] X_1 have
    "(¬is_Isolated (Isolated p1 γ1) ⟶
      (∃ p' w'. (Init p', w', Isolated p1 γ1) ∈ LTS_ε.trans_star_ε A ∧ (p', w') ⇒* (p, LTS_ε.remove_ε u))) ∧
     (is_Isolated (Isolated p1 γ1) ⟶
      (the_Ctr_Loc (Isolated p1 γ1), [the_Label (Isolated p1 γ1)]) ⇒* (p, LTS_ε.remove_ε u))"
    using step.prems(1,2,3) by auto
  then have "(the_Ctr_Loc (Isolated p1 γ1), [the_Label (Isolated p1 γ1)]) ⇒* (p, LTS_ε.remove_ε u)"
    by auto
  then have p1_γ1_p_u: "(p1, [γ1]) ⇒* (p, LTS_ε.remove_ε u)"
    by auto
    from IX have "∃γ2ε γ2ss. LTS_ε.ε_exp γ2ε [γ2] ∧ (Init p2, γ2ε, γ2ss, q') ∈ LTS.trans_star_states
Aiminus1"
    by (meson LTS.trans_star_trans_star_states LTS_ε.trans_star_ε_ε_exp_trans_star)
 then obtain γ2ε γ2ss where XI_1: "LTS_ε.ε_exp γ2ε [γ2] ∧ (Init p2, γ2ε, γ2ss, q') ∈ LTS.trans_star_states
Aiminus1"
    by blast
  have "(q', v, v_ss, q) ∈ LTS.trans_star_states Ai"
    using path .
  have ind:
    "(¬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)))"
  proof -
    have γ2ss_len: "length γ2ss = Suc (length γ2ε)"
      by (meson LTS.trans_star_states_length XI_1)

    have v_ss_empty: "v_ss ≠ []"
      by (metis LTS.trans_star_states.simps path list.distinct(1))

    have γ2ss_last: "last γ2ss = 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 γ1 = 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 γ1), Some γ'') @@^γ (q', v, v_ss, q)))
            (Isolated p1 γ1, Some γ'', q') =
            count (transitions_of' (Init p, u, u_ss, Isolated p1 γ1)) (Isolated p1 γ1, Some γ'', q') +
            (if Isolated p1 γ1 = last u_ss ∧ q' = hd v_ss ∧ Some γ'' = Some γ'' then 1 else 0) +
            count (transitions_of' (q', v, v_ss, q)) (Isolated p1 γ1, Some γ'', 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' where 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 "γ2 #(LTS_ε.remove_ε v) = LTS_ε.remove_ε (γ2ε @ v)"
          using XI_1
          by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_append_dist LTS_ε.remove_ε_def append_Cons
              self_append_conv2)
        moreover
        from XIII have "(p2, γ2 #(LTS_ε.remove_ε v)) ⇒* (p1, γ1#γ''#LTS_ε.remove_ε v)"
          by (metis PDS.transition_rel.intros append_Cons append_Nil lbl.simps(3) r_into_rtranclp
              step_relp_def)
        ultimately
        show "(p2, LTS_ε.remove_ε (γ2ε @ v)) ⇒* (p1, γ1#γ''#LTS_ε.remove_ε v)"
          by auto
      qed
    have p1_γ1γ''v_p_uv: "(p1, γ1#γ''#LTS_ε.remove_ε v) ⇒* (p, (LTS_ε.remove_ε u) @ (γ''#LTS_ε.remove_ε
v))"
      by (metis p1_γ1_p_u append_Cons append_Nil step_relp_append)
    have "(p, (LTS_ε.remove_ε u) @ (γ''#LTS_ε.remove_ε v)) = (p, LTS_ε.remove_ε w)"
      by (metis (no_types, lifting) Cons_eq_append_conv LTS_ε.remove_ε_Cons_tl
          LTS_ε.remove_ε_append_dist w_split hd_Cons_tl list.inject list.sel(1) list.simps(3)
          self_append_conv2)
    then show ?thesis
      using True p1_γ1γ''v_p_uv p2_γ2εv_p1_γ1_γ''_v p'_w'_p by fastforce
  next
    case False
    then have q_nlq_p2_γ2εv: "(the_Ctr_Loc q, [the_Label q]) ⇒* (p2, LTS_ε.remove_ε (γ2ε @ v))"
      using ind state.exhaust_disc
      by blast
    have p2_γ2εv_p1_γ1γ''v: "(p2, LTS_ε.remove_ε (γ2ε @ v)) ⇒* (p1, γ1 # γ'' # LTS_ε.remove_ε v)"
      by (metis (mono_tags) LTS_ε.ε_exp_def LTS_ε.remove_ε_append_dist LTS_ε.remove_ε_def XIII
          XI_1 append_Cons append_Nil lbl.simps(3) r_into_rtranclp step_relp_def2)

    have p1_γ1γ''_v_p_uγ''v: "(p1, γ1 # γ'' # LTS_ε.remove_ε v) ⇒* (p, LTS_ε.remove_ε u @ γ'' #
LTS_ε.remove_ε v)"
      by (metis p1_γ1_p_u append_Cons append_Nil step_relp_append)

    have "(p, LTS_ε.remove_ε u @ γ'' # LTS_ε.remove_ε v) = (p, LTS_ε.remove_ε w)"
      by (metis LTS_ε.remove_ε_Cons_tl LTS_ε.remove_ε_append_dist append_Cons append_Nil w_split
          hd_Cons_tl list.distinct(1) list.inject)

    then show ?thesis
      using False p1_γ1γ''_v_p_uγ''v p2_γ2εv_p1_γ1γ''v q_nlq_p2_γ2εv
      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 ⊆ LTS.srcs A"
  assumes "isols ⊆ LTS.isolated A"
  assumes "(Init p, w, q) ∈ LTS.trans_star A'"
  shows "(¬is_Isolated q ⟶ (∃p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p',w') ⇒* (p, LTS_ε.remove_ε
w))) ∧
        (is_Isolated q ⟶ (the_Ctr_Loc q, [the_Label q]) ⇒* (p, LTS_ε.remove_ε 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 ⊆ LTS.srcs A"
  assumes "isols ⊆ LTS.isolated A"
  assumes "(Init p, w, q) ∈ LTS.trans_star A'"
  shows "(¬is_Isolated q ⟶ (∃p' w'. (Init p', w', q) ∈ LTS_ε.trans_star_ε A ∧ (p',w') ⇒* (p, LTS_ε.remove_ε
w))) ∧
         (is_Isolated q ⟶ (the_Ctr_Loc q, [the_Label q]) ⇒* (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 ⊆ LTS.srcs A"
  assumes "isols ⊆ LTS.isolated A"
  shows "{c. accepts_ε A' c} ⊆ 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 ∈ {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 ∈ finals" "(Init p, w, q) ∈ LTS_ε.trans_star_ε A'"
    unfolding accepts_ε_def by auto
  then obtain w' where w'_def: "LTS_ε.ε_exp w' w ∧ (Init p, w', q) ∈ LTS.trans_star A'"
    by (meson LTS_ε.trans_star_ε_iff_ε_exp_trans_star)
  then have path: "(Init p, w', q) ∈ 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) ∈ LTS_ε.trans_star_ε A ∧ (p', w'a) ⇒* (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) ∈ LTS_ε.trans_star_ε A ∧ (p', w'a) ⇒* (p, w)"
    using w'_def
    by (metis LTS_ε.ε_exp_def LTS_ε.remove_ε_def
        ‹LTS_ε.ε_exp w' w ∧ (Init p, w', q) ∈ LTS.trans_star A'›)
  then have "(p,w) ∈ post_star (lang_ε A)"
    using ‹q ∈ finals› unfolding LTS.post_star_def accepts_ε_def lang_ε_def by fastforce
  then show "c ∈ 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 ⊆ LTS.srcs A"
  assumes "isols ⊆ 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 ∈ post_star (lang_ε A)"
  then obtain p' w' where "(p', w') ⇒* (p, w) ∧ (p', w') ∈ 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 ∈ {c. accepts_ε A' c}"
    by auto
next
  fix c :: "('ctr_loc, 'label) conf"
  assume "c ∈ {c. accepts_ε A' c}"
```

**then show** *"c ∈ post_star (lang_ε 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 ⊆ LTS.srcs A"*
  **assumes** *"isols ⊆ LTS.isolated A"*
  **shows** *"lang_ε A' = post_star (lang_ε A)"*
  **using** *assms lang_ε_def post_star_rules_accepts_ε_correct* **by** *presburger*

**end**

## 5.5 Intersection Automata

**definition** *accepts_inters* :: *"(('ctr_loc, 'noninit, 'label) state * ('ctr_loc, 'noninit, 'label) state, 'label) transition set ⇒ (('ctr_loc, 'noninit, 'label) state * ('ctr_loc, 'noninit, 'label) state) set ⇒ ('ctr_loc, 'label) conf ⇒ bool"* **where**
  *"accepts_inters ts finals ≡ λ(p,w). (∃ qq ∈ finals. ((Init p, Init p),w,qq) ∈ LTS.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) ⟷*
    *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* :: *"(('ctr_loc, 'noninit, 'label) state * ('ctr_loc, 'noninit, 'label) state, 'label) transition set ⇒ (('ctr_loc, 'noninit, 'label) state * ('ctr_loc, 'noninit, 'label) state) set ⇒ ('ctr_loc, 'label) 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** *"(λ(p,w). (p, w)) ' 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) ⟷*
    *PDS_with_P_automata.accepts final_initss1 final_noninits1 ts1 (p,w) ∧*
    *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 ∩*
    *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 "⇒" 80)
notation step_starp (infix "⇒*" 80)

definition accepts_ε_inters :: "(('ctr_loc, 'noninit, 'label) state * ('ctr_loc, 'noninit, 'label) state, 'label option)
transition set ⇒ ('ctr_loc, 'label) conf ⇒ bool" where
  "accepts_ε_inters ts ≡ λ(p,w). (∃q1 ∈ finals. ∃q2 ∈ finals. ((Init p, Init p),w,(q1,q2)) ∈ LTS_ε.trans_star_ε
ts)"

definition lang_ε_inters :: "(('ctr_loc, 'noninit, 'label) state * ('ctr_loc, 'noninit, 'label) state, 'label option) tran-
sition set ⇒ ('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) ∈ LTS.trans_star ts1"
  assumes "(q1, w2, q2) ∈ LTS.trans_star ts2"
  shows "((p1,q1), w :: 'label list, (p2,q2)) ∈ 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 "∃α w1' w2' β. w1=Some α#w1' ∧ w2=Some β#w2'")
    case True
    from True obtain α β 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.prems(1) less.prems(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) ∈ LTS.trans_star ts1"
      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) ∈ LTS.trans_star ts2"
      using less True'(2) by (metis LTS_ε.trans_star_cons_ε)
    have ind: "((p', 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'(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) ∈ 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 "((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.prems(3) less.prems(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.prems(1,2) less_numeral_extra(3)
          list.simps(8) list.size(3) removeAll.simps(1))
  qed
qed
```

```
    qed
  qed
qed

lemma trans_star_ε_inter:
  assumes "(p1, w :: 'label list, p2) ∈ LTS_ε.trans_star_ε ts1"
  assumes "(q1, w, q2) ∈ LTS_ε.trans_star_ε ts2"
  shows "((p1, q1), w, (p2, q2)) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)"
proof -
  have "∃w1'. LTS_ε.ε_exp w1' w ∧ (p1, w1', p2) ∈ LTS.trans_star ts1"
    using assms by (simp add: LTS_ε.trans_star_ε_ε_exp_trans_star)
  then obtain w1' where "LTS_ε.ε_exp w1' w ∧ (p1, w1', p2) ∈ LTS.trans_star ts1"
    by auto
  moreover
  have "∃w2'. LTS_ε.ε_exp w2' w ∧ (q1, w2', q2) ∈ LTS.trans_star ts2"
    using assms by (simp add: LTS_ε.trans_star_ε_ε_exp_trans_star)
  then obtain w2' where "LTS_ε.ε_exp w2' w ∧ (q1, w2', q2) ∈ LTS.trans_star ts2"
    by auto
  ultimately
  show ?thesis
    using trans_star_trans_star_ε_inter by metis
qed

lemma inters_trans_star_ε1:
  assumes "(p1q2, w :: 'label list, p2q2) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2)"
  shows "(fst p1q2, w, fst p2q2) ∈ LTS_ε.trans_star_ε ts1"
  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: "(fst q', w, fst q) ∈ LTS_ε.trans_star_ε ts1"
    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) ∈ 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}"
    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) ∈ LTS_ε.trans_star_ε ts2"
    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)) ∈ LTS_ε.trans_star_ε (inters_ε ts1 ts2) ⟷
   (p1, w, p2) ∈ LTS_ε.trans_star_ε ts1 ∧ (q2, w, q2) ∈ LTS_ε.trans_star_ε ts2"
  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_ε_inters (inters_ε ts1 ts2) c ⟷ accepts_ε ts1 c ∧ accepts_ε ts2 c"
proof
  assume "accepts_ε_inters (inters_ε ts1 ts2) c"
  then show "accepts_ε ts1 c ∧ accepts_ε ts2 c"
    using accepts_ε_def accepts_ε_inters_def inters_trans_star_ε inters_trans_star_ε1 by fastforce
next
  assume asm: "accepts_ε ts1 c ∧ accepts_ε ts2 c"
  define p where "p = fst c"
  define w where "w = snd c"

  from asm have "accepts_ε ts1 (p,w) ∧ accepts_ε ts2 (p,w)"
    using p_def w_def by auto
  then have "(∃ q∈finals. (Init p, w, q) ∈ LTS_ε.trans_star_ε ts1) ∧
             (∃ q∈finals. (Init p, w, q) ∈ LTS_ε.trans_star_ε ts2)"
    unfolding accepts_ε_def by auto
  then show "accepts_ε_inters (inters_ε ts1 ts2) c"
    using accepts_ε_inters_def p_def trans_star_ε_inter w_def by fastforce
qed

lemma inters_ε_lang_ε: "lang_ε_inters (inters_ε ts1 ts2) = lang_ε ts1 ∩ lang_ε ts2"
  unfolding lang_ε_inters_def lang_ε_def using inters_ε_accept_ε_iff by auto
```

## 5.7 Dual search

```
lemma dual1:
  "post_star (lang_ε A1) ∩ pre_star (lang A2) = {c. ∃ c1 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c ∧ c ⇒* c2}"
proof -
  have "post_star (lang_ε A1) ∩ pre_star (lang A2) = {c. c ∈ post_star (lang_ε A1) ∧ c ∈ pre_star (lang A2)}"
    by auto
  moreover
  have "... = {c. (∃ c1 ∈ lang_ε A1. c1 ⇒* c) ∧ (∃ c2 ∈ lang A2. c ⇒* c2)}"
    unfolding post_star_def pre_star_def by auto
  moreover
  have "... = {c. ∃ c1 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c ∧ c ⇒* c2}"
    by auto
  ultimately
  show ?thesis by metis
qed

lemma dual2:
  "post_star (lang_ε A1) ∩ pre_star (lang A2) ≠ {} ⟷ (∃ c1 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c2)"
proof (rule)
  assume "post_star (lang_ε A1) ∩ pre_star (lang A2) ≠ {}"
  then show "∃ c1∈lang_ε A1. ∃ c2∈lang A2. c1 ⇒* c2"
    by (auto simp: pre_star_def post_star_def intro: rtranclp_trans)
next
  assume "∃ c1∈lang_ε A1. ∃ c2∈lang A2. c1 ⇒* c2"
  then show "post_star (lang_ε A1) ∩ pre_star (lang A2) ≠ {}"
```

```
    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** *“∃ c1 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c2”*
**proof** −
**have** *“{c. accepts_ε A1′ c} ⊆ post_star (lang_ε A1)”*
**using** *assms* **using** *post_star_rules_subset_post_star_lang* **by** *auto*
**then have** *A1′_correct*: *“lang_ε A1′ ⊆ post_star (lang_ε A1)”*
**unfolding** *lang_ε_def* **by** *auto*

**have** *“{c. accepts A2′ c} ⊆ pre_star (lang A2)”*
**using** *pre_star_rule_subset_pre_star_lang[of A2 A2′] assms* **by** *auto*
**then have** *A2′_correct*: *“lang A2′ ⊆ pre_star (lang A2)”*
**unfolding** *lang_def* **by** *auto*

**have** *“lang_ε_inters (inters_ε A1′ (LTS_ε_of_LTS A2′)) = lang_ε A1′ ∩ lang_ε (LTS_ε_of_LTS A2′)”*
**using** *inters_ε_lang_ε[of A1′ “(LTS_ε_of_LTS A2′)”]* **by** *auto*
**moreover**
**have** *“... = lang_ε A1′ ∩ lang A2′”*
**using** *lang_ε_LTS_ε_of_LTS_is_lang* **by** *auto*
**moreover**
**have** *“... ⊆ post_star (lang_ε A1) ∩ pre_star (lang A2)”*
**using** *A1′_correct A2′_correct* **by** *auto*
**ultimately**
**have** *inters_correct*: *“lang_ε_inters (inters_ε A1′ (LTS_ε_of_LTS A2′)) ⊆ post_star (lang_ε A1) ∩ pre_star (lang A2)”*
**by** *metis*

**from** *assms* **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*

**qed**

**theorem** *dual_star_correct_saturation*:
**assumes** *“inits ⊆ LTS.srcs A1”*
**assumes** *“inits ⊆ LTS.srcs A2”*
**assumes** *“isols ⊆ LTS.isolated A1”*
**assumes** *“isols ⊆ LTS.isolated A2”*
**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 ∈ lang_ε A1. ∃ c2 ∈ lang A2. c1 ⇒* c2)”*
**proof** −
**have** *“{c. accepts_ε A1′ c} = post_star (lang_ε A1)”*
**using** *post_star_rules_accepts_ε_correct[of A1 A1′] assms* **by** *auto*
**then have** *A1′_correct*: *“lang_ε A1′ = post_star (lang_ε A1)”*
**unfolding** *lang_ε_def* **by** *auto*

**have** *“{c. accepts A2′ c} = pre_star (lang A2)”*
**using** *pre_star_rule_accepts_correct[of A2 A2′] assms* **by** *auto*
**then have** *A2′_correct*: *“lang A2′ = pre_star (lang A2)”*
**unfolding** *lang_def* **by** *auto*

**have** *“lang_ε_inters (inters_ε A1′ (LTS_ε_of_LTS A2′)) = lang_ε A1′ ∩ lang_ε (LTS_ε_of_LTS A2′)”*
**using** *inters_ε_lang_ε[of A1′ “(LTS_ε_of_LTS A2′)”]* **by** *auto*
**moreover**
**have** *“... = lang_ε A1′ ∩ lang A2′”*
**using** *lang_ε_LTS_ε_of_LTS_is_lang* **by** *auto*
**moreover**
**have** *“... = post_star (lang_ε A1) ∩ pre_star (lang A2)”*
**using** *A1′_correct A2′_correct* **by** *auto*
**ultimately**
**have** *inters_correct*: *“lang_ε_inters (inters_ε A1′ (LTS_ε_of_LTS A2′)) = post_star (lang_ε A1) ∩ pre_star (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 Δ :: "('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 Δ final_automaton))
    ((λp. (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 ⊆ 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 ⟷ ¬ (inits ⊆ LTS.srcs F)"
  unfolding check_def by auto

lemma check_Some: "check Δ I IF IF_st F FF FF_st = Some b ⟷
  (inits ⊆ LTS.srcs F ∧ b = (∃p w p' w'.
     (p, w) ∈ language IF IF_st I ∧
     (p', w') ∈ language FF FF_st F ∧
     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!: bexI[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+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+20] Peter Gjøl 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+21] Peter Gjøl 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] Dejvuth 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+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.