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

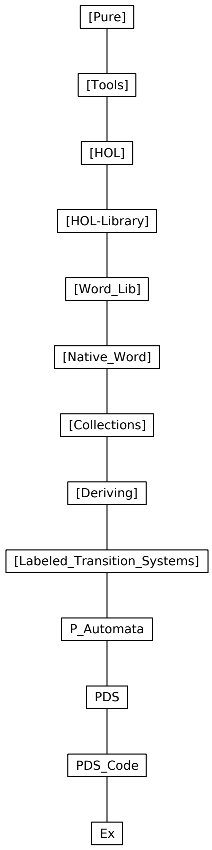

Figure 1: Theory dependency graph

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)"
  ⟨proof⟩

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)"
  ⟨proof⟩

typedef 'a mark_state = "{(Q :: 'a set, I). I ⊆ Q}"
  ⟨proof⟩
setup-lifting *type_definition_mark_state*
lift-definition *get_visited* :: "'a mark_state ⇒ 'a set" **is** *fst* ⟨proof⟩
lift-definition *get_next* :: "'a mark_state ⇒ 'a set" **is** *snd* ⟨proof⟩
lift-definition *make_mark_state* :: "'a set ⇒ 'a set ⇒ 'a mark_state" **is** "λQ J. (Q ∪ J, J)" ⟨proof⟩
lemma *get_next_get_visited*: "get_next ms ⊆ get_visited ms"
  ⟨proof⟩
lemma *get_next_set_next*[*simp*]: "get_next (make_mark_state Q J) = J"
  ⟨proof⟩
lemma *get_visited_set_next*[*simp*]: "get_visited (make_mark_state Q J) = Q ∪ J"
  ⟨proof⟩

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))"
  ⟨proof⟩
termination ⟨proof⟩

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"
  ⟨proof⟩

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⟩

lemma mark_sound: “mark ms ⟹ (∃ p ∈ get_next ms. ∃ q ∈ finals. ∃ w. (p, w, q) ∈ trans_star)”
⟨proof⟩

lemma nonempty_code[code]: “nonempty = mark (make_mark_state {} (set (map Init Enum.enum)))”
⟨proof⟩

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”
⟨proof⟩

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”
⟨proof⟩

lemma inters_trans_star1:
assumes “(p1q2, w :: 'label list, p2q2) ∈ pa.trans_star”
shows “(fst p1q2, w, fst p2q2) ∈ A1.trans_star”
⟨proof⟩

lemma inters_trans_star:
assumes “(p1q2, w :: 'label list, p2q2) ∈ pa.trans_star”
shows “(snd p1q2, w, snd p2q2) ∈ A2.trans_star”
⟨proof⟩

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”
⟨proof⟩

lemma inters_accept_iff: “accepts_aut_inters p w ⟷ A1.accepts_aut p w ∧ A2.accepts_aut p w”
⟨proof⟩

lemma lang_aut_alt:
“pa.lang_aut = {(p, w). (p, w) ∈ lang_aut_inters}”
⟨proof⟩

lemma inters_lang: “lang_aut_inters = A1.lang_aut ∩ A2.lang_aut”
⟨proof⟩

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* ≡ λ*p 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*" "(λ*p*. (*Init p*, *Init p*))"
  ⟨*proof*⟩

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*"
  ⟨*proof*⟩

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*⟩

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"*
⟨*proof*⟩

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"*
⟨*proof*⟩

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"
⟨*proof*⟩

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*⟩

lemma *inters_ε_lang_ε*: *"lang_aut_inters_ε = A1.lang_aut_ε ∩ A2.lang_aut_ε"*
⟨*proof*⟩

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* ⟨*proof*⟩

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))"
⟨*proof*⟩

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*⟩

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

instance ⟨*proof*⟩

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)"*
⟨*proof*⟩

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

lemma *inits_code[code]*: *"inits = set (map Init Enum.enum)"*
⟨*proof*⟩

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* ⟨*proof*⟩
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"*
⟨*proof*⟩

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**
"$\varepsilon == None$"

lemma *accepts_mono*[*mono*]: "*mono accepts*"
⟨*proof*⟩

lemma *accepts_cons*: "$(Init\ p,\ \gamma,\ Init\ p') \in ts \Longrightarrow accepts\ ts\ (p',\ w) \Longrightarrow accepts\ ts\ (p,\ \gamma\ \#\ w)$"
⟨*proof*⟩

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

lemma *lang_lang_aut*: "$lang\ ts = (\lambda(s,w).\ (s,\ w))\ `\ (P\_Automaton.lang\_aut\ ts\ Init\ finals)$"
⟨*proof*⟩

lemma *lang_aut_lang*: "$P\_Automaton.lang\_aut\ ts\ Init\ finals = lang\ ts$"
⟨*proof*⟩

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

5.1 Saturations

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

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

lemma *no_infinite*:
assumes "$\bigwedge ts\ ts' :: ('c ::finite,\ 'l::finite)\ transition\ set.\ rule\ ts\ ts' \Longrightarrow card\ ts' = Suc\ (card\ ts)$"
assumes "$\forall i :: nat.\ rule\ (tts\ i)\ (tts\ (Suc\ i))$"
shows "*False*"
⟨*proof*⟩

lemma *saturation_termination*:
assumes "$\bigwedge ts\ ts' :: ('c ::finite,\ 'l::finite)\ transition\ set.\ rule\ ts\ ts' \Longrightarrow card\ ts' = Suc\ (card\ ts)$"
shows "$\neg(\exists tts.\ (\forall i :: nat.\ rule\ (tts\ i)\ (tts\ (Suc\ i))))$"
⟨*proof*⟩

lemma *saturation_exi*:
assumes "$\bigwedge ts\ ts' :: ('c ::finite,\ 'l::finite)\ transition\ set.\ rule\ ts\ ts' \Longrightarrow card\ ts' = Suc\ (card\ ts)$"
shows "$\exists ts'.\ saturation\ rule\ ts\ ts'$"
⟨*proof*⟩

5.2 Saturation rules

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

definition *pre_star1* :: "$(('ctr\_loc,\ 'noninit,\ 'label)\ state,\ 'label)\ transition\ set \Rightarrow (('ctr\_loc,\ 'noninit,\ 'label)\ state,\ 'label)\ transition\ set$" **where**
"$pre\_star1\ ts =$
$(\bigcup((p,\ \gamma),\ (p',\ w)) \in \Delta.\ \bigcup q \in LTS.reach\ ts\ (Init\ p')\ (lbl\ w).\ \{(Init\ p,\ \gamma,\ q)\})$"

lemma *pre_star1_mono*: "*mono pre_star1*"
⟨*proof*⟩

lemma *pre_star_rule_pre_star1*:
assumes "$X \subseteq pre\_star1\ ts$"

shows “pre_star_rule** ts (ts ∪ X)”
⟨proof⟩

lemma pre_star_rule_pre_star1s: “pre_star_rule** ts (((λs. s ∪ pre_star1 s) ^^ k) ts)”
⟨proof⟩

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⟩

lemma pre_star_exec_terminates: “∃ t. pre_star_loop s = Some t”
⟨proof⟩

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'))”
⟨proof⟩

lemma saturation_pre_star_exec: “saturation pre_star_rule ts (pre_star_exec ts)”
⟨proof⟩

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'”
⟨proof⟩

lemma post_star_rules_mono:
“post_star_rules ts ts' ⟹ ts ⊂ ts'”
⟨proof⟩

lemma pre_star_rule_card_Suc: “pre_star_rule ts ts' ⟹ card ts' = Suc (card ts)”
⟨proof⟩

lemma *post_star_rules_card_Suc*: "$post\_star\_rules\ ts\ ts' \Longrightarrow card\ ts' = Suc\ (card\ ts)$"
⟨*proof*⟩

lemma *pre_star_saturation_termination*:
"$\neg(\exists\, tts.\ (\forall\, i :: nat.\ pre\_star\_rule\ (tts\ i)\ (tts\ (Suc\ i))))$"
⟨*proof*⟩

lemma *post_star_saturation_termination*:
"$\neg(\exists\, tts.\ (\forall\, i :: nat.\ post\_star\_rules\ (tts\ i)\ (tts\ (Suc\ i))))$"
⟨*proof*⟩

lemma *pre_star_saturation_exi*:
shows "$\exists\, ts'.\ saturation\ pre\_star\_rule\ ts\ ts'$"
⟨*proof*⟩

lemma *post_star_saturation_exi*:
shows "$\exists\, ts'.\ saturation\ post\_star\_rules\ ts\ ts'$"
⟨*proof*⟩

lemma *pre_star_rule_incr*: "$pre\_star\_rule\ A\ B \Longrightarrow A \subseteq B$"
⟨*proof*⟩

lemma *post_star_rules_incr*: "$post\_star\_rules\ A\ B \Longrightarrow A \subseteq B$"
⟨*proof*⟩

lemma *saturation_rtranclp_pre_star_rule_incr*: "$pre\_star\_rule^{**}\ A\ B \Longrightarrow A \subseteq B$"
⟨*proof*⟩

lemma *saturation_rtranclp_post_star_rule_incr*: "$post\_star\_rules^{**}\ A\ B \Longrightarrow A \subseteq B$"
⟨*proof*⟩

lemma *pre_star'_incr_trans_star*:
"$pre\_star\_rule^{**}\ A\ A' \Longrightarrow LTS.trans\_star\ A \subseteq LTS.trans\_star\ A'$"
⟨*proof*⟩

lemma *post_star'_incr_trans_star*:
"$post\_star\_rules^{**}\ A\ A' \Longrightarrow LTS.trans\_star\ A \subseteq LTS.trans\_star\ A'$"
⟨*proof*⟩

lemma *post_star'_incr_trans_star_ε*:
"$post\_star\_rules^{**}\ A\ A' \Longrightarrow LTS\_\varepsilon.trans\_star\_\varepsilon\ A \subseteq LTS\_\varepsilon.trans\_star\_\varepsilon\ A'$"
⟨*proof*⟩

lemma *pre_star_lim'_incr_trans_star*:
"$saturation\ pre\_star\_rule\ A\ A' \Longrightarrow LTS.trans\_star\ A \subseteq LTS.trans\_star\ A'$"
⟨*proof*⟩

lemma *post_star_lim'_incr_trans_star*:
"$saturation\ post\_star\_rules\ A\ A' \Longrightarrow LTS.trans\_star\ A \subseteq LTS.trans\_star\ A'$"
⟨*proof*⟩

lemma *post_star_lim'_incr_trans_star_ε*:
"$saturation\ post\_star\_rules\ A\ A' \Longrightarrow LTS\_\varepsilon.trans\_star\_\varepsilon\ A \subseteq LTS\_\varepsilon.trans\_star\_\varepsilon\ A'$"
⟨*proof*⟩

5.3 Pre* lemmas

lemma *inits_srcs_iff_Ctr_Loc_srcs*:
"$inits \subseteq LTS.srcs\ A \longleftrightarrow (\nexists\, q\ \gamma\ q'.\ (q,\ \gamma,\ Init\ q') \in A)$"
⟨*proof*⟩

lemma *lemma_3_1*:
assumes "$p'w \Rightarrow^{*}\ pv$"

assumes *"pv ∈ lang A"*
assumes *"saturation pre_star_rule A A′"*
shows *"accepts A′ p′w"*
⟨*proof*⟩

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*⟩

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"*
⟨*proof*⟩

lemma *step_relp_append_aux*:
assumes *"pu ⇒* p1y"*
shows *"(fst pu, snd pu @ v) ⇒* (fst p1y, snd p1y @ v)"*
⟨*proof*⟩

lemma *step_relp_append*:
assumes *"(p, u) ⇒* (p1, y)"*
shows *"(p, u @ v) ⇒* (p1, y @ v)"*
⟨*proof*⟩

lemma *step_relp_append_empty*:
assumes *"(p, u) ⇒* (p1, [])"*
shows *"(p, u @ v) ⇒* (p1, v)"*
⟨*proof*⟩

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′)"*
⟨*proof*⟩

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′)"*
⟨*proof*⟩
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*⟩
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*⟩
theorem *pre_star_rule_correct*:
assumes *"inits ⊆ LTS.srcs A"*
assumes *"saturation pre_star_rule A A′"*
shows *"lang A′ = pre_star (lang A)"*
⟨*proof*⟩

theorem *pre_star_exec_accepts_correct*:
  assumes "*inits* ⊆ *LTS.srcs A*"
  shows "{*c. accepts* (*pre_star_exec A*) *c*} = *pre_star* (*lang A*)"
  ⟨*proof*⟩

theorem *pre_star_exec_lang_correct*:
  assumes "*inits* ⊆ *LTS.srcs A*"
  shows "*lang* (*pre_star_exec A*) = *pre_star* (*lang A*)"
  ⟨*proof*⟩

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*)"
  ⟨*proof*⟩

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*)"
  ⟨*proof*⟩

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*)"
  ⟨*proof*⟩

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*)"
  ⟨*proof*⟩

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*⟩

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*⟩

theorem *accept_pre_star_exec_correct_None*:
  assumes "*accept_pre_star_exec_check A c* = *None*"
  shows "¬*inits* ⊆ *LTS.srcs A*"
  ⟨*proof*⟩

5.4 Post* lemmas

lemma *lemma_3_3′*:
  assumes "*pv* $\Rightarrow^*$ *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*)"
  ⟨*proof*⟩

lemma *lemma_3_3*:
  assumes "(*p*,*v*) $\Rightarrow^*$ (*p′*,*w*)"
  and "(*p*, *v*) ∈ *lang_ε A*"
  and "*saturation post_star_rules A A′*"
  shows "*accepts_ε A′* (*p′*, *w*)"
  ⟨*proof*⟩

lemma *init_only_hd*:
  assumes "$(ss,\ w) \in LTS.path\_with\_word\ A$"
  assumes "$inits \subseteq LTS.srcs\ A$"
  assumes "$count\ (transitions\_of\ (ss,\ w))\ t > 0$"
  assumes "$t = (Init\ p1,\ \gamma,\ q1)$"
  shows "$hd\ (transition\_list\ (ss,\ w)) = t \wedge count\ (transitions\_of\ (ss,\ w))\ t = 1$"
  ⟨*proof*⟩

lemma *no_edge_to_Ctr_Loc_avoid_Ctr_Loc*:
  assumes "$(p,\ w,\ qq) \in LTS.trans\_star\ Aiminus1$"
  assumes "$w \neq []$"
  assumes "$inits \subseteq LTS.srcs\ Aiminus1$"
  shows "$qq \notin inits$"
  ⟨*proof*⟩

lemma *no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε*:
  assumes "$(p,\ [\gamma],\ qq) \in LTS\_\varepsilon.trans\_star\_\varepsilon\ Aiminus1$"
  assumes "$inits \subseteq LTS.srcs\ Aiminus1$"
  shows "$qq \notin inits$"
  ⟨*proof*⟩

lemma *no_edge_to_Ctr_Loc_post_star_rules′*:
  assumes "$post\_star\_rules^{**}\ A\ Ai$"
  assumes "$\nexists\, q\ \gamma\ q'.\ (q,\ \gamma,\ Init\ q') \in A$"
  shows "$\nexists\, q\ \gamma\ q'.\ (q,\ \gamma,\ Init\ q') \in Ai$"
⟨*proof*⟩

lemma *no_edge_to_Ctr_Loc_post_star_rules*:
  assumes "$post\_star\_rules^{**}\ A\ Ai$"
  assumes "$inits \subseteq LTS.srcs\ A$"
  shows "$inits \subseteq LTS.srcs\ Ai$"
  ⟨*proof*⟩

lemma *source_and_sink_isolated*:
  assumes "$N \subseteq LTS.srcs\ A$"
  assumes "$N \subseteq LTS.sinks\ A$"
  shows "$\forall\, p\ \gamma\ q.\ (p,\ \gamma,\ q) \in A \longrightarrow p \notin N \wedge q \notin N$"
  ⟨*proof*⟩

lemma *post_star_rules_Isolated_source_invariant′*:
  assumes "$post\_star\_rules^{**}\ A\ A'$"
  assumes "$isols \subseteq LTS.isolated\ A$"
  assumes "$(Init\ p',\ Some\ \gamma',\ Isolated\ p'\ \gamma') \notin A'$"
  shows "$\nexists\, p\ \gamma.\ (p,\ \gamma,\ Isolated\ p'\ \gamma') \in A'$"
  ⟨*proof*⟩

lemma *post_star_rules_Isolated_source_invariant*:
  assumes "$post\_star\_rules^{**}\ A\ A'$"
  assumes "$isols \subseteq LTS.isolated\ A$"
  assumes "$(Init\ p',\ Some\ \gamma',\ Isolated\ p'\ \gamma') \notin A'$"
  shows "$Isolated\ p'\ \gamma' \in LTS.srcs\ A'$"
  ⟨*proof*⟩

lemma *post_star_rules_Isolated_sink_invariant′*:
  assumes "$post\_star\_rules^{**}\ A\ A'$"
  assumes "$isols \subseteq LTS.isolated\ A$"
  assumes "$(Init\ p',\ Some\ \gamma',\ Isolated\ p'\ \gamma') \notin A'$"
  shows "$\nexists\, p\ \gamma.\ (Isolated\ p'\ \gamma',\ \gamma,\ p) \in A'$"
  ⟨*proof*⟩

lemma *post_star_rules_Isolated_sink_invariant*:
  assumes "$post\_star\_rules^{**}\ A\ A'$"
  assumes "$isols \subseteq LTS.isolated\ A$"

assumes *“(Init p′, Some γ′, Isolated p′ γ′) ∉ A′”*
shows *“Isolated p′ γ′ ∈ LTS.sinks A′”*
⟨*proof*⟩
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))”*
⟨*proof*⟩
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))”*
⟨*proof*⟩
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))”*
⟨*proof*⟩
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*⟩
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*⟩
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)”*
⟨*proof*⟩

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”
⟨*proof*⟩

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"
⟨proof⟩

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)"
⟨proof⟩

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"
⟨proof⟩

5.6 Intersection epsilon-Automata

context *PDS_with_P_automata* **begin**

interpretation *LTS transition_rel* ⟨proof⟩
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) transition 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)"
⟨proof⟩

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⟩

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"
⟨proof⟩

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"
⟨proof⟩

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"
⟨proof⟩

lemma *inters_ε_accept_ε_iff*:
"accepts_ε_inters (inters_ε ts1 ts2) c ⟷ accepts_ε ts1 c ∧ accepts_ε ts2 c"
⟨proof⟩

lemma *inters_ε_lang_ε*: "lang_ε_inters (inters_ε ts1 ts2) = lang_ε ts1 ∩ lang_ε ts2"
⟨proof⟩

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⟩

lemma *dual2*:
"post_star (lang_ε A1) ∩ pre_star (lang A2) ≠ {} ⟷ (∃c1 ∈ lang_ε A1. ∃c2 ∈ lang A2. c1 ⇒* c2)"
⟨proof⟩

lemma *LTS_ε_of_LTS_Some*: "(p, Some γ, q') ∈ LTS_ε_of_LTS A2' ⟷ (p, γ, q') ∈ A2'"
⟨proof⟩

lemma *LTS_ε_of_LTS_None*: "(p, None, q') ∉ LTS_ε_of_LTS A2'"
⟨proof⟩

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'"
⟨proof⟩

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')"
⟨proof⟩

lemma *accepts_ε_LTS_ε_of_LTS_iff_accepts*: "accepts_ε (LTS_ε_of_LTS A2') (p, w) ⟷ accepts A2' (p, w)"
⟨proof⟩

lemma *lang_ε_LTS_ε_of_LTS_is_lang*: "lang_ε (LTS_ε_of_LTS A2') = lang A2'"
⟨proof⟩

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⟩

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*⟩

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"*
  ⟨*proof*⟩

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"*
  ⟨*proof*⟩

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"*
  ⟨*proof*⟩

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))*
((λ*p. (Init p, Init p)))*
(inters_finals (finals initial_F_ctr_loc initial_F_ctr_loc_st)
(finals final_F_ctr_loc final_F_ctr_loc_st))"
⟨*proof*⟩

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

lemma *check_None*: "*check* Δ *I IF IF_st F FF FF_st* = *None* $\longleftrightarrow$ $\neg$ *(inits* $\subseteq$ *LTS.srcs F)*"
⟨*proof*⟩

lemma *check_Some*: "*check* Δ *I IF IF_st F FF FF_st* = *Some b* $\longleftrightarrow$
(inits $\subseteq$ *LTS.srcs F* $\wedge$ *b* = ($\exists$ *p w p' w'*.
(p, w) $\in$ *language IF IF_st I* $\wedge$
(p', w') $\in$ *language FF FF_st F* $\wedge$
step_starp Δ *(p, w) (p', w')))*"
⟨*proof*⟩

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.