

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

1	Introduction	2
2	Automata	3
2.1	P-Automaton locale	3
2.2	Intersection P-Automaton locale	4
3	Automata with epsilon	5
3.1	P-Automaton with epsilon locale	5
3.2	Intersection P-Automaton with epsilon locale	5
4	PDS	6
5	PDS with P automata	7
5.1	Saturations	8
5.2	Saturation rules	8
5.3	Pre* lemmas	10
5.4	Post* lemmas	12
5.5	Intersection Automata	14
5.6	Intersection epsilon-Automata	15
5.7	Dual search	16

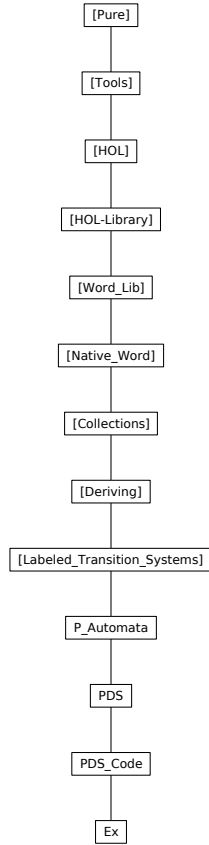


Figure 1: Theory dependency graph

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 \Rightarrow 'state"
 and *finals* :: "'state set"
 begin

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

lemma *initials_list*:
 "*initials* = set (map *Init* Enum.enum)"
 <proof>

definition *accepts_aut* :: "'ctr_loc \Rightarrow 'label list \Rightarrow bool" where
 "*accepts_aut* \equiv $\lambda p w. (\exists q \in \text{finals}. (Init\ p, w, q) \in \text{trans_star})$ "

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

definition *nonempty* where
 "*nonempty* \longleftrightarrow *lang_aut* \neq {}"

lemma *nonempty_alt*:
 "*nonempty* \longleftrightarrow ($\exists p. \exists q \in \text{finals}. \exists w. (Init\ p, w, q) \in \text{trans_star}$)"
 <proof>

typedef 'a *mark_state* = "{(Q :: 'a set, I). I \subseteq Q}"
 <proof>

setup-lifting type_definition *mark_state*

lift-definition *get_visited* :: "'a *mark_state* \Rightarrow 'a set" is fst <proof>

lift-definition *get_next* :: "'a *mark_state* \Rightarrow 'a set" is snd <proof>

lift-definition *make_mark_state* :: "'a set \Rightarrow 'a set \Rightarrow 'a *mark_state*" is " $\lambda Q J. (Q \cup J, J)$ " <proof>

lemma *get_next_get_visited*: "*get_next* ms \subseteq *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 \cup J"
 <proof>

function *mark* where

"*mark* ms \longleftrightarrow
 (let Q = *get_visited* ms; I = *get_next* ms in
 if I \cap finals \neq {} then True
 else let J = ($\bigcup (q,w,q') \in \text{transition_relation}. \text{if } q \in I \wedge q' \notin Q \text{ then } \{q'\} \text{ else } \{\}$) in
 if J = {} then False else *mark* (make_mark_state Q J))"
 <proof>

termination <proof>

declare *mark.simps*[simp del]

lemma *trapped_transitions*: " $(p, w, q) \in \text{trans_star} \implies$
 $\forall p \in Q. (\forall \gamma q. (p, \gamma, q) \in \text{transition_relation} \longrightarrow q \in Q) \implies$
 $p \in Q \implies q \in Q$ "
 <proof>

lemma *mark_complete*: " $(p, w, q) \in \text{trans_star} \implies (\text{get_visited}\ ms - \text{get_next}\ ms) \cap \text{finals} = \{\} \implies$
 $\forall p \in \text{get_visited}\ ms - \text{get_next}\ ms. \forall q \gamma. (p, \gamma, q) \in \text{transition_relation} \longrightarrow q \in \text{get_visited}\ ms \implies$ "

$p \in \text{get_visited } ms \implies q \in \text{finals} \implies \text{mark } ms$
 <proof>

lemma *mark_sound*: “ $\text{mark } ms \implies (\exists p \in \text{get_next } ms. \exists q \in \text{finals}. \exists w. (p, w, q) \in \text{trans_star})$ ”
 <proof>

lemma *nonempty_code*[code]: “ $\text{nonempty} = \text{mark } (\text{make_mark_state } \{\} (\text{set } (\text{map } \text{Init } \text{Enum.enum})))$ ”
 <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 \Rightarrow 'state”
and *finals1* :: “'state set”
and *ts2* :: “('state, 'label) transition set”
and *finals2* :: “'state set”

begin

sublocale *pa*: *P_Automaton* “*inters ts1 ts2*” “ $(\lambda p. (\text{Init } p, \text{Init } p))$ ” “*inters_finals finals1 finals2*”
 <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* = $\{(p, w). \text{accepts_aut_inters } p w\}$ ”

lemma *trans_star_inter*:
assumes “ $(p1, w, p2) \in A1.\text{trans_star}$ ”
assumes “ $(q1, w, q2) \in A2.\text{trans_star}$ ”
shows “ $((p1, q1), w :: \text{'label list}, (p2, q2)) \in \text{pa.trans_star}$ ”
 <proof>

lemma *inters_trans_star1*:
assumes “ $(p1q2, w :: \text{'label list}, p2q2) \in \text{pa.trans_star}$ ”
shows “ $(\text{fst } p1q2, w, \text{fst } p2q2) \in A1.\text{trans_star}$ ”
 <proof>

lemma *inters_trans_star*:
assumes “ $(p1q2, w :: \text{'label list}, p2q2) \in \text{pa.trans_star}$ ”
shows “ $(\text{snd } p1q2, w, \text{snd } p2q2) \in A2.\text{trans_star}$ ”
 <proof>

lemma *inters_trans_star_iff*:
 “ $((p1, q2), w :: \text{'label list}, (p2, q2)) \in \text{pa.trans_star} \iff (p1, w, p2) \in A1.\text{trans_star} \wedge (q2, w, q2) \in A2.\text{trans_star}$ ”
 <proof>

lemma *inters_accept_iff*: “ $\text{accepts_aut_inters } p w \iff A1.\text{accepts_aut } p w \wedge A2.\text{accepts_aut } p w$ ”
 <proof>

lemma *lang_aut_alt*:
 “ $\text{pa.lang_aut} = \{(p, w). (p, w) \in \text{lang_aut_inters}\}$ ”
 <proof>

lemma *inters_lang*: “ $\text{lang_aut_inters} = A1.\text{lang_aut} \cap A2.\text{lang_aut}$ ”
 <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* ≡ 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))”
{*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 “ $(fst\ p1q2, w, fst\ p2q2) \in A1.trans_star_ε$ ”
 ⟨proof⟩

lemma *inters_trans_star_ε*:
assumes “ $(p1q2, w :: 'label\ list, p2q2) \in pa.trans_star_ε$ ”
shows “ $(snd\ p1q2, w, snd\ p2q2) \in A2.trans_star_ε$ ”
 ⟨proof⟩

lemma *inters_trans_star_ε_iff*:
 “ $((p1, q2), w :: 'label\ list, (p2, q2)) \in pa.trans_star_ε \longleftrightarrow$
 $(p1, w, p2) \in A1.trans_star_ε \wedge (q2, w, q2) \in A2.trans_star_ε$ ”
 ⟨proof⟩

lemma *inters_ε_accept_ε_iff*:
 “ $accepts_aut_inters_ε\ p\ w \longleftrightarrow A1.accepts_aut_ε\ p\ w \wedge A2.accepts_aut_ε\ p\ w$ ”
 ⟨proof⟩

lemma *inters_ε_lang_ε*: “ $lang_aut_inters_ε = A1.lang_aut_ε \cap A2.lang_aut_ε$ ”
 ⟨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* = “ $('ctr_loc \times 'label) \times ('ctr_loc \times 'label\ operation)$ ”
type-synonym (*'ctr_loc, 'label*) *conf* = “ $'ctr_loc \times 'label\ list$ ”

We define push down systems.

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

begin

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

definition *is_rule* :: “ $'ctr_loc \times 'label \Rightarrow 'ctr_loc \times 'label\ operation \Rightarrow bool$ ” (**infix** “ \hookrightarrow ” 80) **where**
 “ $p\gamma \hookrightarrow p'w \equiv (p\gamma, p'w) \in \Delta$ ”

inductive-set *transition_rel* :: “ $(('ctr_loc, 'label)\ conf \times unit \times ('ctr_loc, 'label)\ conf)\ set$ ” **where**
 “ $(p, \gamma) \hookrightarrow (p', w) \implies$
 $((p, \gamma \# w'), (), (p', (lbl\ w) @ w')) \in transition_rel$ ”

interpretation *LTS transition_rel* ⟨proof⟩

notation *step_relp* (**infix** “ \Rightarrow ” 80)

notation *step_starp* (**infix** “ \Rightarrow^* ” 80)

lemma *step_relp_def2*:
 “ $(p, \gamma w') \Rightarrow (p', ww') \longleftrightarrow (\exists \gamma\ w'\ w. \gamma w' = \gamma \# w' \wedge ww' = (lbl\ w) @ w' \wedge (p, \gamma) \hookrightarrow (p', w))$ ”
 ⟨proof⟩

end

5 PDS with P automata

type-synonym ('ctr_loc, 'label) sat_rule = “('ctr_loc, 'label) transition set \Rightarrow ('ctr_loc, 'label) transition set \Rightarrow 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 \cup Noninit ‘ final_noninits”

lemma F_not_Ext: “ $\neg(\exists f \in \text{finals}. \text{is_Isolated } f)$ ”
 <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** “ \Rightarrow ” 80)

notation step_starp (**infix** “ \Rightarrow^* ” 80)

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

lemma accepts_accepts_aut: “accepts ts (p, w) \longleftrightarrow P_Automaton.accepts_aut ts Init finals p w”
 <proof>

definition accepts_ε :: “((('ctr_loc, 'noninit, 'label) state, 'label option) transition set \Rightarrow ('ctr_loc, 'label) conf \Rightarrow bool” **where**
 “accepts_ε ts $\equiv \lambda(p,w). (\exists q \in \text{finals}. (\text{Init } p,w,q) \in \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon ts)$ ”

abbreviation ε :: “*label option*” **where**
“ $\varepsilon == \text{None}$ ”

lemma *accepts_mono*[*mono*]: “*mono accepts*”
⟨*proof*⟩

lemma *accepts_cons*: “ $(\text{Init } p, \gamma, \text{Init } p') \in ts \implies \text{accepts } ts (p', w) \implies \text{accepts } ts (p, \gamma \# w)$ ”
⟨*proof*⟩

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

lemma *lang_lang_aut*: “ $\text{lang } ts = (\lambda(s,w). (s, w)) \text{ ' } (P_Automaton.\text{lang_aut } ts \text{ Init finals})$ ”
⟨*proof*⟩

lemma *lang_aut_lang*: “ $P_Automaton.\text{lang_aut } ts \text{ Init finals} = \text{lang } ts$ ”
⟨*proof*⟩

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

5.1 Saturations

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

definition *saturation* :: “ $(\text{'c}, \text{'l}) \text{ sat_rule} \Rightarrow (\text{'c}, \text{'l}) \text{ transition set} \Rightarrow (\text{'c}, \text{'l}) \text{ transition set} \Rightarrow \text{bool}$ ” **where**
“ $\text{saturation rule } ts ts' \longleftrightarrow \text{rule}^{**} ts ts' \wedge \text{saturated rule } ts'$ ”

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

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

lemma *saturation_exi*:
assumes “ $\bigwedge ts ts' :: (\text{'c} :: \text{finite}, \text{'l} :: \text{finite}) \text{ transition set. rule } ts ts' \implies \text{card } ts' = \text{Suc } (\text{card } ts)$ ”
shows “ $\exists ts'. \text{saturation rule } ts ts'$ ”
⟨*proof*⟩

5.2 Saturation rules

inductive *pre_star_rule* :: “ $((\text{'ctr_loc}, \text{'noninit}, \text{'label}) \text{ state}, \text{'label}) \text{ transition set} \Rightarrow ((\text{'ctr_loc}, \text{'noninit}, \text{'label}) \text{ state}, \text{'label}) \text{ transition set} \Rightarrow \text{bool}$ ” **where**
add_trans: “ $(p, \gamma) \hookrightarrow (p', w) \implies (\text{Init } p', \text{lbl } w, q) \in LTS.\text{trans_star } ts \implies (\text{Init } p, \gamma, q) \notin ts \implies \text{pre_star_rule } ts (ts \cup \{(\text{Init } p, \gamma, q)\})$ ”

definition *pre_star1* :: “ $((\text{'ctr_loc}, \text{'noninit}, \text{'label}) \text{ state}, \text{'label}) \text{ transition set} \Rightarrow ((\text{'ctr_loc}, \text{'noninit}, \text{'label}) \text{ state}, \text{'label}) \text{ transition set}$ ” **where**
“ $\text{pre_star1 } ts = (\bigcup((p, \gamma), (p', w)) \in \Delta. \bigcup q \in LTS.\text{reach } ts (\text{Init } p') (\text{lbl } w). \{(\text{Init } p, \gamma, q)\})$ ”

lemma *pre_star1_mono*: “*mono pre_star1*”
⟨*proof*⟩

lemma *pre_star_rule_pre_star1*:
assumes “ $X \subseteq \text{pre_star1 } ts$ ”

shows “ $\text{pre_star_rule}^{**} \text{ ts } (\text{ts} \cup X)$ ”
 ⟨proof⟩

lemma $\text{pre_star_rule_pre_star1s}$: “ $\text{pre_star_rule}^{**} \text{ ts } (((\lambda s. s \cup \text{pre_star1 } s) \sim k) \text{ ts})$ ”
 ⟨proof⟩

definition “ $\text{pre_star_loop} = \text{while_option } (\lambda s. s \cup \text{pre_star1 } s \neq s) (\lambda s. s \cup \text{pre_star1 } s)$ ”

definition “ $\text{pre_star_exec} = \text{the } o \text{ pre_star_loop}$ ”

definition “ $\text{pre_star_exec_check } A = (\text{if } \text{inits} \subseteq \text{LTS.sracs } A \text{ then } \text{pre_star_loop } A \text{ else } \text{None})$ ”

definition “ $\text{accept_pre_star_exec_check } A \text{ c} = (\text{if } \text{inits} \subseteq \text{LTS.sracs } A \text{ then } \text{Some } (\text{accepts } (\text{pre_star_exec } A) \text{ c}) \text{ else } \text{None})$ ”

lemma $\text{while_option_finite_subset_Some}$: **fixes** $C :: \text{“a set”}$

assumes “ $\text{mono } f$ ” **and** “ $\forall X. X \subseteq C \implies f X \subseteq C$ ” **and** “ $\text{finite } C$ ” **and** X : “ $X \subseteq C$ ” “ $X \subseteq f X$ ”

shows “ $\exists P. \text{while_option } (\lambda A. f A \neq A) f X = \text{Some } P$ ”

⟨proof⟩

lemma $\text{pre_star_exec_terminates}$: “ $\exists t. \text{pre_star_loop } s = \text{Some } t$ ”

⟨proof⟩

lemma $\text{pre_star_exec_code}$ [code]:

“ $\text{pre_star_exec } s = (\text{let } s' = \text{pre_star1 } s \text{ in if } s' \subseteq s \text{ then } s \text{ else } \text{pre_star_exec } (s \cup s'))$ ”

⟨proof⟩

lemma $\text{saturation_pre_star_exec}$: “ $\text{saturation } \text{pre_star_rule } \text{ts } (\text{pre_star_exec } \text{ts})$ ”

⟨proof⟩

inductive post_star_rules :: “((ctr_loc , noninit , label) state , label option) $\text{transition set} \implies ((\text{ctr_loc}$, noninit , label) state , label option) $\text{transition set} \implies \text{bool}$ ” **where**

add_trans_pop:

“ $(p, \gamma) \hookrightarrow (p', \text{pop}) \implies$
 ($\text{Init } p, [\gamma], q \in \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \text{ ts} \implies$
 ($\text{Init } p', \varepsilon, q \notin \text{ts} \implies$
 $\text{post_star_rules } \text{ts } (\text{ts} \cup \{(\text{Init } p', \varepsilon, q)\})$ ”

| **add_trans_swap**:

“ $(p, \gamma) \hookrightarrow (p', \text{swap } \gamma') \implies$
 ($\text{Init } p, [\gamma], q \in \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \text{ ts} \implies$
 ($\text{Init } p', \text{Some } \gamma', q \notin \text{ts} \implies$
 $\text{post_star_rules } \text{ts } (\text{ts} \cup \{(\text{Init } p', \text{Some } \gamma', q)\})$ ”

| **add_trans_push_1**:

“ $(p, \gamma) \hookrightarrow (p', \text{push } \gamma' \gamma'') \implies$
 ($\text{Init } p, [\gamma], q \in \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \text{ ts} \implies$
 ($\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma' \notin \text{ts} \implies$
 $\text{post_star_rules } \text{ts } (\text{ts} \cup \{(\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma')\})$ ”

| **add_trans_push_2**:

“ $(p, \gamma) \hookrightarrow (p', \text{push } \gamma' \gamma'') \implies$
 ($\text{Init } p, [\gamma], q \in \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \text{ ts} \implies$
 ($\text{Isolated } p' \gamma', \text{Some } \gamma'', q \notin \text{ts} \implies$
 ($\text{Init } p', \text{Some } \gamma', \text{Isolated } p' \gamma' \in \text{ts} \implies$
 $\text{post_star_rules } \text{ts } (\text{ts} \cup \{(\text{Isolated } p' \gamma', \text{Some } \gamma'', q)\})$ ”

lemma $\text{pre_star_rule_mono}$:

“ $\text{pre_star_rule } \text{ts } \text{ts}' \implies \text{ts} \subseteq \text{ts}'$ ”

⟨proof⟩

lemma $\text{post_star_rules_mono}$:

“ $\text{post_star_rules } \text{ts } \text{ts}' \implies \text{ts} \subseteq \text{ts}'$ ”

⟨proof⟩

lemma $\text{pre_star_rule_card_Suc}$: “ $\text{pre_star_rule } \text{ts } \text{ts}' \implies \text{card } \text{ts}' = \text{Suc } (\text{card } \text{ts})$ ”

⟨proof⟩

lemma *post_star_rules_card_Suc*: “ $\text{post_star_rules } ts \ ts' \implies \text{card } ts' = \text{Suc } (\text{card } ts)$ ”
 ⟨proof⟩

lemma *pre_star_saturation_termination*:
 “ $\neg(\exists \text{tts}. (\forall i :: \text{nat}. \text{pre_star_rule } (\text{tts } i) (\text{tts } (\text{Suc } i))))$ ”
 ⟨proof⟩

lemma *post_star_saturation_termination*:
 “ $\neg(\exists \text{tts}. (\forall i :: \text{nat}. \text{post_star_rules } (\text{tts } i) (\text{tts } (\text{Suc } i))))$ ”
 ⟨proof⟩

lemma *pre_star_saturation_exi*:
shows “ $\exists \text{ts}'.$ *saturation pre_star_rule ts ts'*”
 ⟨proof⟩

lemma *post_star_saturation_exi*:
shows “ $\exists \text{ts}'.$ *saturation post_star_rules ts ts'*”
 ⟨proof⟩

lemma *pre_star_rule_incr*: “ $\text{pre_star_rule } A \ B \implies A \subseteq B$ ”
 ⟨proof⟩

lemma *post_star_rules_incr*: “ $\text{post_star_rules } A \ B \implies A \subseteq B$ ”
 ⟨proof⟩

lemma *saturation_rtranclp_pre_star_rule_incr*: “ $\text{pre_star_rule}^{**} \ A \ B \implies A \subseteq B$ ”
 ⟨proof⟩

lemma *saturation_rtranclp_post_star_rule_incr*: “ $\text{post_star_rules}^{**} \ A \ B \implies A \subseteq B$ ”
 ⟨proof⟩

lemma *pre_star'_incr_trans_star*:
 “ $\text{pre_star_rule}^{**} \ A \ A' \implies \text{LTS.trans_star } A \subseteq \text{LTS.trans_star } A'$ ”
 ⟨proof⟩

lemma *post_star'_incr_trans_star*:
 “ $\text{post_star_rules}^{**} \ A \ A' \implies \text{LTS.trans_star } A \subseteq \text{LTS.trans_star } A'$ ”
 ⟨proof⟩

lemma *post_star'_incr_trans_star_ε*:
 “ $\text{post_star_rules}^{**} \ A \ A' \implies \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \ A \subseteq \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \ A'$ ”
 ⟨proof⟩

lemma *pre_star_lim'_incr_trans_star*:
 “ $\text{saturation pre_star_rule } A \ A' \implies \text{LTS.trans_star } A \subseteq \text{LTS.trans_star } A'$ ”
 ⟨proof⟩

lemma *post_star_lim'_incr_trans_star*:
 “ $\text{saturation post_star_rules } A \ A' \implies \text{LTS.trans_star } A \subseteq \text{LTS.trans_star } A'$ ”
 ⟨proof⟩

lemma *post_star_lim'_incr_trans_star_ε*:
 “ $\text{saturation post_star_rules } A \ A' \implies \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \ A \subseteq \text{LTS}_\varepsilon.\text{trans_star}_\varepsilon \ A'$ ”
 ⟨proof⟩

5.3 Pre* lemmas

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

lemma *lemma_3_1*:
assumes “ $p'w \Rightarrow^* pv$ ”

assumes “ $pv \in \text{lang } A$ ”
assumes “ $\text{saturation pre_star_rule } A \ A'$ ”
shows “ $\text{accepts } A' \ p'w$ ”
 ⟨*proof*⟩

lemma *word_into_init_empty_states*:
fixes $A :: ((\text{ctr_loc}, \text{noninit}, \text{label}) \text{ state}, \text{label}) \text{ transition set}$
assumes “ $(p, w, ss, \text{Init } q) \in \text{LTS.trans_star_states } A$ ”
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
shows “ $w = [] \wedge p = \text{Init } q \wedge ss = [p]$ ”
 ⟨*proof*⟩

lemma *word_into_init_empty*:
fixes $A :: ((\text{ctr_loc}, \text{noninit}, \text{label}) \text{ state}, \text{label}) \text{ transition set}$
assumes “ $(p, w, \text{Init } q) \in \text{LTS.trans_star } A$ ”
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
shows “ $w = [] \wedge p = \text{Init } q$ ”
 ⟨*proof*⟩

lemma *step_relp_append_aux*:
assumes “ $pu \Rightarrow^* p1y$ ”
shows “ $(\text{fst } pu, \text{snd } pu @ v) \Rightarrow^* (\text{fst } p1y, \text{snd } p1y @ v)$ ”
 ⟨*proof*⟩

lemma *step_relp_append*:
assumes “ $(p, u) \Rightarrow^* (p1, y)$ ”
shows “ $(p, u @ v) \Rightarrow^* (p1, y @ v)$ ”
 ⟨*proof*⟩

lemma *step_relp_append_empty*:
assumes “ $(p, u) \Rightarrow^* (p1, [])$ ”
shows “ $(p, u @ v) \Rightarrow^* (p1, v)$ ”
 ⟨*proof*⟩

lemma *lemma_3_2_a'*:
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
assumes “ $\text{pre_star_rule}^{**} \ A \ A'$ ”
assumes “ $(\text{Init } p, w, q) \in \text{LTS.trans_star } A'$ ”
shows “ $\exists p' w'. (\text{Init } p', w', q) \in \text{LTS.trans_star } A \wedge (p, w) \Rightarrow^* (p', w')$ ”
 ⟨*proof*⟩

lemma *lemma_3_2_a*:
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
assumes “ $\text{saturation pre_star_rule } A \ A'$ ”
assumes “ $(\text{Init } p, w, q) \in \text{LTS.trans_star } A'$ ”
shows “ $\exists p' w'. (\text{Init } p', w', q) \in \text{LTS.trans_star } A \wedge (p, w) \Rightarrow^* (p', w')$ ”
 ⟨*proof*⟩

theorem *pre_star_rule_subset_pre_star_lang*:
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
assumes “ $\text{pre_star_rule}^{**} \ A \ A'$ ”
shows “ $\{c. \text{accepts } A' \ c\} \subseteq \text{pre_star } (\text{lang } A)$ ”
 ⟨*proof*⟩

theorem *pre_star_rule_accepts_correct*:
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
assumes “ $\text{saturation pre_star_rule } A \ A'$ ”
shows “ $\{c. \text{accepts } A' \ c\} = \text{pre_star } (\text{lang } A)$ ”
 ⟨*proof*⟩

theorem *pre_star_rule_correct*:
assumes “ $\text{inits} \subseteq \text{LTS.sracs } A$ ”
assumes “ $\text{saturation pre_star_rule } A \ A'$ ”
shows “ $\text{lang } A' = \text{pre_star } (\text{lang } A)$ ”
 ⟨*proof*⟩

theorem *pre_star_exec_accepts_correct*:
assumes “*inits* \subseteq *LTS.srcs A*”
shows “ $\{c. \text{accepts} (\text{pre_star_exec } A) c\} = \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *pre_star_exec_lang_correct*:
assumes “*inits* \subseteq *LTS.srcs A*”
shows “ $\text{lang} (\text{pre_star_exec } A) = \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *pre_star_exec_check_accepts_correct*:
assumes “*pre_star_exec_check A* \neq *None*”
shows “ $\{c. \text{accepts} (\text{the} (\text{pre_star_exec_check } A)) c\} = \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *pre_star_exec_check_correct*:
assumes “*pre_star_exec_check A* \neq *None*”
shows “ $\text{lang} (\text{the} (\text{pre_star_exec_check } A)) = \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *accept_pre_star_exec_correct_True*:
assumes “*inits* \subseteq *LTS.srcs A*”
assumes “*accepts* (*pre_star_exec A*) *c*”
shows “ $c \in \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *accept_pre_star_exec_correct_False*:
assumes “*inits* \subseteq *LTS.srcs A*”
assumes “ $\neg \text{accepts} (\text{pre_star_exec } A) c$ ”
shows “ $c \notin \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *accept_pre_star_exec_correct_Some_True*:
assumes “*accept_pre_star_exec_check A c* = *Some True*”
shows “ $c \in \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *accept_pre_star_exec_correct_Some_False*:
assumes “*accept_pre_star_exec_check A c* = *Some False*”
shows “ $c \notin \text{pre_star} (\text{lang } A)$ ”
 $\langle \text{proof} \rangle$

theorem *accept_pre_star_exec_correct_None*:
assumes “*accept_pre_star_exec_check A c* = *None*”
shows “ $\neg \text{inits} \subseteq \text{LTS.srcs } A$ ”
 $\langle \text{proof} \rangle$

5.4 Post* lemmas

lemma *lemma_3_3'*:
assumes “ $pv \Rightarrow^* p'w$ ”
and “ $(\text{fst } pv, \text{snd } pv) \in \text{lang}_\varepsilon A$ ”
and “*saturation post_star_rules A A'*”
shows “ $\text{accepts}_\varepsilon A' (\text{fst } p'w, \text{snd } p'w)$ ”
 $\langle \text{proof} \rangle$

lemma *lemma_3_3*:
assumes “ $(p, v) \Rightarrow^* (p', w)$ ”
and “ $(p, v) \in \text{lang}_\varepsilon A$ ”
and “*saturation post_star_rules A A'*”
shows “ $\text{accepts}_\varepsilon A' (p', w)$ ”
 $\langle \text{proof} \rangle$

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 “ $(p, w, qq) \in LTS.trans_star\ A_{minus1}$ ”

assumes “ $w \neq []$ ”

assumes “ $inits \subseteq LTS.srcs\ A_{minus1}$ ”

shows “ $qq \notin inits$ ”

<proof>

lemma *no_edge_to_Ctr_Loc_avoid_Ctr_Loc_ε*:

assumes “ $(p, [\gamma], qq) \in LTS_{\epsilon}.trans_star_{\epsilon}\ A_{minus1}$ ”

assumes “ $inits \subseteq LTS.srcs\ A_{minus1}$ ”

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') $\notin A'$ ”
shows “Isolated p' $\gamma' \in LTS.sinks A'$ ”
 ⟨proof⟩

lemma rtrancpl_post_star_rules_constains_successors_states:
assumes “post_star_rules** A A'”
assumes “inits $\subseteq LTS.sracs A$ ”
assumes “isols $\subseteq LTS.isolated A$ ”
assumes “(Init p, w, ss, q) $\in LTS.trans_star_states A'$ ”
shows “($\neg is_Isolated q \longrightarrow (\exists p' w'. (Init p', w', q) \in LTS_\\epsilon.trans_star_\\epsilon A \wedge (p', w') \Rightarrow^* (p, LTS_\\epsilon.remove_\\epsilon w))) \wedge$
 (is_Isolated q $\longrightarrow (the_Ctr_Loc q, [the_Label q]) \Rightarrow^* (p, LTS_\\epsilon.remove_\\epsilon w)$)”
 ⟨proof⟩

lemma rtrancpl_post_star_rules_constains_successors:
assumes “post_star_rules** A A'”
assumes “inits $\subseteq LTS.sracs A$ ”
assumes “isols $\subseteq LTS.isolated A$ ”
assumes “(Init p, w, q) $\in LTS.trans_star A'$ ”
shows “($\neg is_Isolated q \longrightarrow (\exists p' w'. (Init p', w', q) \in LTS_\\epsilon.trans_star_\\epsilon A \wedge (p', w') \Rightarrow^* (p, LTS_\\epsilon.remove_\\epsilon w))) \wedge$
 (is_Isolated q $\longrightarrow (the_Ctr_Loc q, [the_Label q]) \Rightarrow^* (p, LTS_\\epsilon.remove_\\epsilon w)$)”
 ⟨proof⟩

lemma post_star_rules_saturation_constains_successors:
assumes “saturation post_star_rules A A'”
assumes “inits $\subseteq LTS.sracs A$ ”
assumes “isols $\subseteq LTS.isolated A$ ”
assumes “(Init p, w, q) $\in LTS.trans_star A'$ ”
shows “($\neg is_Isolated q \longrightarrow (\exists p' w'. (Init p', w', q) \in LTS_\\epsilon.trans_star_\\epsilon A \wedge (p', w') \Rightarrow^* (p, LTS_\\epsilon.remove_\\epsilon w))) \wedge$
 (is_Isolated q $\longrightarrow (the_Ctr_Loc q, [the_Label q]) \Rightarrow^* (p, LTS_\\epsilon.remove_\\epsilon w)$)”
 ⟨proof⟩

theorem post_star_rules_subset_post_star_lang:
assumes “post_star_rules** A A'”
assumes “inits $\subseteq LTS.sracs A$ ”
assumes “isols $\subseteq LTS.isolated A$ ”
shows “{c. accepts_\\epsilon A' c} $\subseteq post_star (lang_\\epsilon A)$ ”
 ⟨proof⟩

theorem post_star_rules_accepts_\\epsilon_correct:
assumes “saturation post_star_rules A A'”
assumes “inits $\subseteq LTS.sracs A$ ”
assumes “isols $\subseteq LTS.isolated A$ ”
shows “{c. accepts_\\epsilon A' c} = post_star (lang_\\epsilon A)”
 ⟨proof⟩

theorem post_star_rules_correct:
assumes “saturation post_star_rules A A'”
assumes “inits $\subseteq LTS.sracs A$ ”
assumes “isols $\subseteq LTS.isolated A$ ”
shows “lang_\\epsilon A' = post_star (lang_\\epsilon A)”
 ⟨proof⟩

end

5.5 Intersection Automata

definition accepts_inters :: “((ctr_loc, 'noninit, 'label) state * (ctr_loc, 'noninit, 'label) state, 'label) transition set
 $\Rightarrow ((ctr_loc, 'noninit, 'label) state * (ctr_loc, 'noninit, 'label) state) set \Rightarrow (ctr_loc, 'label) conf \Rightarrow bool$ ” **where**
 “accepts_inters ts finals $\equiv \lambda(p, w). (\exists qq \in finals. ((Init p, Init p), w, qq) \in LTS.trans_star ts)$ ”

lemma accepts_inters_accepts_aut_inters:
assumes “ts12 = inters ts1 ts2”
assumes “finals12 = inters_finals finals1 finals2”
shows “accepts_inters ts12 finals12 (p, w) \longleftrightarrow
 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 \Rightarrow ((‘ctr_loc, ‘noninit, ‘label) state * (‘ctr_loc, ‘noninit, ‘label) state) set \Rightarrow (‘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 “($\lambda(p,w).$ (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) \longleftrightarrow

PDS_with_P_automata.*accepts* *final_initss1* *final_noninits1* *ts1* (p,w) \wedge

PDS_with_P_automata.*accepts* *final_initss2* *final_noninits2* *ts2* (p,w)”

<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* \cap

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** “ \Rightarrow ” 80)

notation *step_starp* (**infix** “ \Rightarrow^* ” 80)

definition *accepts_ε_inters* :: “((‘ctr_loc, ‘noninit, ‘label) state * (‘ctr_loc, ‘noninit, ‘label) state, ‘label option) transition set \Rightarrow (‘ctr_loc, ‘label) conf \Rightarrow bool” **where**

“*accepts_ε_inters* *ts* \equiv $\lambda(p,w).$ ($\exists q1 \in$ *finals*. $\exists q2 \in$ *finals*. ((*Init* *p*, *Init* *p*),w,(*q1*,*q2*)) \in *LTS_ε*.*trans_star_ε* *ts*)”

definition *lang_ε_inters* :: “((‘ctr_loc, ‘noninit, ‘label) state * (‘ctr_loc, ‘noninit, ‘label) state, ‘label option) transition set \Rightarrow (‘ctr_loc, ‘label) conf set” **where**

“*lang_ε_inters* *ts* = {*c*. *accepts_ε_inters* *ts* *c*}”

lemma *trans_star_trans_star_ε_inter*:

assumes “*LTS_ε*.*ε_exp* *w1* *w*”

assumes “*LTS_ε*.*ε_exp* *w2* *w*”

assumes “(*p1*, *w1*, *p2*) \in *LTS*.*trans_star* *ts1*”

assumes “(*q1*, *w2*, *q2*) \in *LTS*.*trans_star* *ts2*”

shows “((*p1*,*q1*), *w* :: ‘label list, (*p2*,*q2*)) \in *LTS_ε*.*trans_star_ε* (*inters_ε* *ts1* *ts2*)”

<proof>

lemma *trans_star_ε_inter*:

assumes “(*p1*, *w* :: ‘label list, *p2*) \in *LTS_ε*.*trans_star_ε* *ts1*”

assumes “(*q1*, *w*, *q2*) \in *LTS_ε*.*trans_star_ε* *ts2*”

shows “((*p1*, *q1*), *w*, (*p2*, *q2*)) \in *LTS_ε*.*trans_star_ε* (*inters_ε* *ts1* *ts2*)”

<proof>

lemma *inters_trans_star_ε1*:

assumes “ $(p1q2, w :: \text{'label list}, p2q2) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} (\text{inters}_{\varepsilon} ts1 ts2)$ ”
shows “ $(fst\ p1q2, w, fst\ p2q2) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} ts1$ ”
 ⟨proof⟩

lemma inters_trans_star_ε:
assumes “ $(p1q2, w :: \text{'label list}, p2q2) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} (\text{inters}_{\varepsilon} ts1 ts2)$ ”
shows “ $(snd\ p1q2, w, snd\ p2q2) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} ts2$ ”
 ⟨proof⟩

lemma inters_trans_star_ε_iff:
 “ $((p1,q2), w :: \text{'label list}, (p2,q2)) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} (\text{inters}_{\varepsilon} ts1 ts2) \longleftrightarrow$
 $(p1, w, p2) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} ts1 \wedge (q2, w, q2) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} ts2$ ”
 ⟨proof⟩

lemma inters_ε_accept_ε_iff:
 “ $\text{accepts}_{\varepsilon} \text{inters} (\text{inters}_{\varepsilon} ts1 ts2) c \longleftrightarrow \text{accepts}_{\varepsilon} ts1 c \wedge \text{accepts}_{\varepsilon} ts2 c$ ”
 ⟨proof⟩

lemma inters_ε_lang_ε: “ $\text{lang}_{\varepsilon} \text{inters} (\text{inters}_{\varepsilon} ts1 ts2) = \text{lang}_{\varepsilon} ts1 \cap \text{lang}_{\varepsilon} ts2$ ”
 ⟨proof⟩

5.7 Dual search

lemma dual1:
 “ $\text{post_star} (\text{lang}_{\varepsilon} A1) \cap \text{pre_star} (\text{lang} A2) = \{c. \exists c1 \in \text{lang}_{\varepsilon} A1. \exists c2 \in \text{lang} A2. c1 \Rightarrow^* c \wedge c \Rightarrow^* c2\}$ ”
 ⟨proof⟩

lemma dual2:
 “ $\text{post_star} (\text{lang}_{\varepsilon} A1) \cap \text{pre_star} (\text{lang} A2) \neq \{\} \longleftrightarrow (\exists c1 \in \text{lang}_{\varepsilon} A1. \exists c2 \in \text{lang} A2. c1 \Rightarrow^* c2)$ ”
 ⟨proof⟩

lemma LTS_ε_of_LTS_Some: “ $(p, \text{Some } \gamma, q') \in LTS_{\varepsilon} \text{of_LTS } A2' \longleftrightarrow (p, \gamma, q') \in A2'$ ”
 ⟨proof⟩

lemma LTS_ε_of_LTS_None: “ $(p, \text{None}, q') \notin LTS_{\varepsilon} \text{of_LTS } A2'$ ”
 ⟨proof⟩

lemma trans_star_ε_LTS_ε_of_LTS_trans_star:
assumes “ $(p,w,q) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} (LTS_{\varepsilon} \text{of_LTS } A2')$ ”
shows “ $(p,w,q) \in LTS.\text{trans_star } A2'$ ”
 ⟨proof⟩

lemma trans_star_trans_star_ε_LTS_ε_of_LTS:
assumes “ $(p,w,q) \in LTS.\text{trans_star } A2'$ ”
shows “ $(p,w,q) \in LTS_{\varepsilon}.\text{trans_star}_{\varepsilon} (LTS_{\varepsilon} \text{of_LTS } A2')$ ”
 ⟨proof⟩

lemma accepts_ε_LTS_ε_of_LTS_iff_accepts: “ $\text{accepts}_{\varepsilon} (LTS_{\varepsilon} \text{of_LTS } A2') (p, w) \longleftrightarrow \text{accepts } A2' (p, w)$ ”
 ⟨proof⟩

lemma lang_ε_LTS_ε_of_LTS_is_lang: “ $\text{lang}_{\varepsilon} (LTS_{\varepsilon} \text{of_LTS } A2') = \text{lang } A2'$ ”
 ⟨proof⟩

theorem dual_star_correct_early_termination:
assumes “ $\text{inits} \subseteq LTS.\text{srcs } A1$ ”
assumes “ $\text{inits} \subseteq LTS.\text{srcs } A2$ ”
assumes “ $\text{isols} \subseteq LTS.\text{isolated } A1$ ”
assumes “ $\text{isols} \subseteq LTS.\text{isolated } A2$ ”
assumes “ $\text{post_star_rules}^{**} A1 A1'$ ”
assumes “ $\text{pre_star_rule}^{**} A2 A2'$ ”
assumes “ $\text{lang}_{\varepsilon} \text{inters} (\text{inters}_{\varepsilon} A1' (LTS_{\varepsilon} \text{of_LTS } A2')) \neq \{\}$ ”
shows “ $\exists c1 \in \text{lang}_{\varepsilon} A1. \exists c2 \in \text{lang } A2. c1 \Rightarrow^* c2$ ”
 ⟨proof⟩


```

theorem dual_star_correct_saturation:
  assumes "inits  $\subseteq$  LTS.sracs A1"
  assumes "inits  $\subseteq$  LTS.sracs A2"
  assumes "isols  $\subseteq$  LTS.isolated A1"
  assumes "isols  $\subseteq$  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'))  $\neq$  {}  $\longleftrightarrow$  ( $\exists$  c1  $\in$  lang_ε A1.  $\exists$  c2  $\in$  lang A2. c1
 $\Rightarrow^*$  c2)"
  <proof>

```

```

theorem dual_star_correct_early_termination_configs:
  assumes "inits  $\subseteq$  LTS.sracs A1"
  assumes "inits  $\subseteq$  LTS.sracs A2"
  assumes "isols  $\subseteq$  LTS.isolated A1"
  assumes "isols  $\subseteq$  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'))  $\neq$  {}"
  shows "c1  $\Rightarrow^*$  c2"
  <proof>

```

```

theorem dual_star_correct_saturation_configs:
  assumes "inits  $\subseteq$  LTS.sracs A1"
  assumes "inits  $\subseteq$  LTS.sracs A2"
  assumes "isols  $\subseteq$  LTS.isolated A1"
  assumes "isols  $\subseteq$  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'))  $\neq$  {}  $\longleftrightarrow$  c1  $\Rightarrow^*$  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  $\Delta$  final_automaton))
  (( $\lambda p$ . (Init p, Init p)))
  (inters_finals (finals initial_F_ctr_loc initial_F_ctr_loc_st)
    (finals final_F_ctr_loc final_F_ctr_loc_st))"
  <proof>

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

lemma check_None: "check  $\Delta$  I IF IF_st F FF FF_st = None  $\longleftrightarrow$   $\neg$  (inits  $\subseteq$  LTS.sracs F)"
  <proof>

lemma check_Some: "check  $\Delta$  I IF IF_st F FF FF_st = Some b  $\longleftrightarrow$ 
  (inits  $\subseteq$  LTS.sracs 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  $\Delta$  (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, Jiri 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 Gjøøl Jensen, Dan Kristiansen, Stefan Schmid, Morten Konggaard Schou, Bernhard Clemens Schrenk, and Jiri 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 Gjøøl Jensen, Stefan Schmid, Morten Konggaard Schou, Jiri Srba, Juan Vanerio, and Ingo van Duijn. Faster pushdown reachability analysis with applications in network verification. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 170–186. Springer, 2021.
- [Lam09] Peter Lammich. Formalization of dynamic pushdown networks in Isabelle/HOL, 2009. <https://www21.in.tum.de/~lammich/isabelle/dpn-document.pdf>.
- [LMW09] Peter Lammich, Markus Müller-Olm, and Alexander Wenner. Predecessor sets of dynamic pushdown networks with tree-regular constraints. In Ahmed Bouajjani and Oded Maler, editors, *CAV 2009*, volume 5643 of *LNCS*, pages 525–539. Springer, 2009.
- [Sch02a] Stefan Schwoon. *Model checking pushdown systems*. PhD thesis, Technical University Munich, Germany, 2002. <https://d-nb.info/96638976X/34>.
- [Sch02b] Stefan Schwoon. Moped. 2002. <http://www2.informatik.uni-stuttgart.de/fmi/szs/tools/moped/>.
- [SSE05] Dejavuth Suwimonteerabuth, Stefan Schwoon, and Javier Esparza. jMoped: A Java bytecode checker based on Moped. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS 2005*, volume 3440 of *LNCS*, pages 541–545. Springer, 2005.
- [SSST22] Anders Schlichtkrull, Morten Konggaard Schou, Jiri 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.