Labeled Transition Systems

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

Abstract

Labeled transition systems are ubiquitous in computer science. They are used e.g. for automata and for program graphs in program analysis. We formalize labeled transition systems with and without epsilon transitions. The main difference between formalizations of labeled transition systems is in their choice of how to represent the transition system. In the present formalization the set of nodes is a type, and a labeled transition system is represented as a locale fixing a set of transitions where each transition is a triple of respectively a start node, a label and an end node. Wimmer [Wim20] provides an overview of formalizations of graphs and transition systems.

Contents

theory *LTS* **imports** *Main "HOL−Library.Multiset_Order"* **begin**

1 LTS

1.1 Transitions

type-synonym ('*state*, '*label*) *transition* = "'*state* × '*label* × '*state*"

1.2 LTS functions

fun *trans_hd* :: "('*state*, '*label*) *transition* ⇒ '*state*" **where**
"*trans_hd* (*s1*,γ,*s2*) = *s1*"

fun *trans_tl* :: "('*state*, '*label*) *transition* ⇒ '*state*" **where**
"*trans_tl* (*s1*,γ,*s2*) = *s2*"

fun *transitions_of* :: "'*state list* ∗ '*label list* ⇒ ('*state*, '*label*) *transition multiset*" **where**
"*transitions_of* (*s1*#*s2*#*ss*, γ#*w*) = {# (*s1*, γ, *s2*) #} + *transitions_of* (*s2*#*ss*, *w*)"
| "*transitions_of* ([*s1*],_) = {#}"
| "*transitions_of* ([],_) = {#}"
| "*transitions_of* (_,[]) = {#}"

fun *transition_list* :: "'*state list* ∗ '*label list* ⇒ ('*state*, '*label*) *transition list*" **where**
"*transition_list* (*s1*#*s2*#*ss*, γ#*w*) = (*s1*, γ, *s2*) # (*transition_list* (*s2*#*ss*, *w*))"
| "*transition_list* ([*s1*],_) = []"
| "*transition_list* ([],_) = []"
| "*transition_list* (_,[]) = []"

fun *transition_list'* :: "'*state* ∗ '*label list* ∗ '*state list* ∗ '*state* ⇒ ('*state*, '*label*) *transition list*" **where**
"*transition_list'* (*p*,*w*,*ss*,*q*) = *transition_list* (*ss*, *w*)"

fun *transitions_of'* :: "'*state* ∗ '*label list* ∗ '*state list* ∗ '*state* ⇒ ('*state*, '*label*) *transition multiset*" **where**
"*transitions_of'* (*p*,*w*,*ss*,*q*) = *transitions_of* (*ss*, *w*)"

fun *transition_list_of'* **where**
"*transition_list_of'* (*p*,γ#*w*,*p'*#*p''*#*ss*,*q*) = (*p*, γ, *p''*)#(*transition_list_of'* (*p''*,*w*,*p''*#*ss*,*q*))"
| "*transition_list_of'* (*p*, [], _, *p''*) = []"
| "*transition_list_of'* (*p*, _, [], *p''*) = []"
| "*transition_list_of'* (*v*, *va* # *vc*, [*vf*], *ve*) = []"

fun *append_path_with_word* :: "('*a list* × '*b list*) ⇒ ('*a list* × '*b list*) ⇒ ('*a list* × '*b list*)" (**infix** "@´" *65*) **where**
"(*ss1*,*w1*) @´ (*ss2*,*w2*) = (*ss1*@(*tl ss2*), *w1* @ *w2*)"

fun *append_path_with_word_γ* :: "(('*a list* × '*b list*) ∗ '*b*) ⇒ ('*a list* × '*b list*) ⇒ ('*a list* × '*b list*)" (**infix** "@$^{\gamma}$" *65*) **where**
"((*ss1*,*w1*),γ) @$^{\gamma}$ (*ss2*,*w2*) = (*ss1*@*ss2*, *w1* @ [γ] @ *w2*)"

fun *append_trans_star_states* :: "('*a* × '*b list* × '*a list* × '*a*) ⇒ ('*a* × '*b list* × '*a list* × '*a*) ⇒ ('*a* × '*b list* × '*a list* × '*a*)" (**infix** "@@´" *65*) **where**
"(*p1*,*w1*,*ss1*,*q1*) @@´ (*p2*,*w2*,*ss2*,*q2*) = (*p1*, *w1* @ *w2*, *ss1*@(*tl ss2*), *q2*)"

fun *append_trans_star_states_γ* :: "(('*a* × '*b list* × '*a list* × '*a*) ∗ '*b*) ⇒ ('*a* × '*b list* × '*a list* × '*a*) ⇒ ('*a* × '*b list* × '*a list* × '*a*)" (**infix** "@@$^{\gamma}$" *65*) **where**
"((*p1*,*w1*,*ss1*,*q1*),γ) @@$^{\gamma}$ (*p2*,*w2*,*ss2*,*q2*) = (*p1*, *w1* @ [γ] @ *w2*, *ss1*@*ss2*, *q2*)"

definition *inters* :: "('*state*, '*label*) *transition set* ⇒ ('*state*, '*label*) *transition set* ⇒ (('*state* ∗ '*state*), '*label*) *transition set*" **where**
"*inters ts1 ts2* = {((*p1*, *q1*), α, (*p2*, *q2*)). (*p1*, α, *p2*) ∈ *ts1* ∧ (*q1*, α, *q2*) ∈ *ts2*}"

definition *inters_finals* :: "'*state set* ⇒ '*state set* ⇒ ('*state* ∗ '*state*) *set*" **where**
"*inters_finals finals1 finals2* = *finals1* × *finals2*"

lemma *inters_code*[*code*]:

"inters ts1 ts2 = (⋃(p1, α, p2) ∈ ts1. ⋃(q1, α′, q2) ∈ ts2. if α = α′ then {((p1, q1), α, (p2, q2))} else {})"
⟨*proof*⟩

1.3 LTS locale

locale *LTS* =
fixes *transition_relation* :: *"('state, 'label) transition set"*
begin

More definitions.

definition *step_relp* :: *"'state ⇒ 'state ⇒ bool"* (**infix** *"⇒"* *80*) **where**
"c ⇒ c′ ⟷ (∃ l. (c, l, c′) ∈ transition_relation)"

abbreviation *step_starp* :: *"'state ⇒ 'state ⇒ bool"* (**infix** *"⇒*"* *80*) **where**
"c ⇒ c′ ≡ step_relp** c c′"*

definition *step_rel* :: *"'state rel"* **where**
"step_rel = {(c, c′). step_relp c c′}"

definition *step_star* :: *"'state rel"* **where**
"step_star = {(c, c′). step_starp c c′}"

definition *post_star* :: *"'state set ⇒ 'state set"* **where**
"post_star C = {c′. ∃ c ∈ C. c ⇒ c′}"*

definition *pre_star* :: *"'state set ⇒ 'state set"* **where**
"pre_star C = {c′. ∃ c ∈ C. c′ ⇒ c}"*

inductive-set *path* :: *"'state list set"* **where**
"[s] ∈ path"
| *"(s′#ss) ∈ path ⟹ (s,l,s′) ∈ transition_relation ⟹ s#s′#ss ∈ path"*

inductive-set *trans_star* :: *"('state * 'label list * 'state) set"* **where**
trans_star_refl[*iff*]:
"(p, [], p) ∈ trans_star"
| *trans_star_step*:
"(p,γ,q′) ∈ transition_relation ⟹
(q′,w,q) ∈ trans_star ⟹
(p, γ#w, q) ∈ trans_star"

inductive-cases *trans_star_empty* [*elim*]: *"(p, [], q) ∈ trans_star"*
inductive-cases *trans_star_cons*: *"(p, γ#w, q) ∈ trans_star"*

inductive-set *trans_star_states* :: *"('state * 'label list * 'state list * 'state) set"* **where**
trans_star_states_refl[*iff*]:
"(p,[],[p],p) ∈ trans_star_states"
| *trans_star_states_step*:
"(p,γ,q′) ∈ transition_relation ⟹
(q′,w,ss,q) ∈ trans_star_states ⟹
(p, γ#w, p#ss, q) ∈ trans_star_states"

inductive-set *path_with_word* :: *"('state list * 'label list) set"* **where**
path_with_word_refl[*iff*]:
"([s],[]) ∈ path_with_word"
| *path_with_word_step*:
"(s′#ss, w) ∈ path_with_word ⟹
(s,l,s′) ∈ transition_relation ⟹
(s#s′#ss,l#w) ∈ path_with_word"

definition *start_of* :: *"('state list × 'label list) ⇒ 'state"* **where**
"start_of π = hd (fst π)"

definition *end_of* :: "('*state list* × '*label list*) ⇒ '*state*" **where**
"*end_of* π = *last* (*fst* π)"

abbreviation *path_with_word_from* :: "'*state* ⇒ ('*state list* ∗ '*label list*) *set*" **where**
"*path_with_word_from q* == {π. π ∈ *path_with_word* ∧ *start_of* π = *q*}"

abbreviation *path_with_word_to* :: "'*state* ⇒ ('*state list* ∗ '*label list*) *set*" **where**
"*path_with_word_to q* == {π. π ∈ *path_with_word* ∧ *end_of* π = *q*}"

abbreviation *path_with_word_from_to* :: "'*state* ⇒ '*state* ⇒ ('*state list* ∗ '*label list*) *set*" **where**
"*path_with_word_from_to start end* == {π. π ∈ *path_with_word* ∧ *start_of* π = *start* ∧ *end_of* π = *end*}"

inductive-set *transition_list_path* :: "('*state*, '*label*) *transition list set*" **where**
"(*q*, *l*, *q*′) ∈ *transition_relation* ⟹
[(*q*, *l*, *q*′)] ∈ *transition_list_path*"
| "(*q*, *l*, *q*′) ∈ *transition_relation* ⟹
(*q*′, *l*′, *q*′′) # *ts* ∈ *transition_list_path* ⟹
(*q*, *l*, *q*′) # (*q*′, *l*′, *q*′′) # *ts* ∈ *transition_list_path*"

lemma *singleton_path_start_end*:
assumes "([*s*], []) ∈ *LTS.path_with_word pg*"
shows "*start_of* ([*s*], []) = *end_of* ([*s*], [])"
⟨*proof*⟩

lemma *path_with_word_length*:
assumes "(*ss*, *w*) ∈ *path_with_word*"
shows "*length ss* = *length w* + *1*"
⟨*proof*⟩

lemma *path_with_word_lengths*:
assumes "(*qs* @ [*qnminus1*], *w*) ∈ *path_with_word*"
shows "*length qs* = *length w*"
⟨*proof*⟩

lemma *path_with_word_butlast*:
assumes "(*ss*, *w*) ∈ *path_with_word*"
assumes "*length ss* ≥ *2*"
shows "(*butlast ss*, *butlast w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *transition_butlast*:
assumes "(*ss*, *w*) ∈ *path_with_word*"
assumes "*length ss* ≥ *2*"
shows "(*last* (*butlast ss*), *last w*, *last ss*) ∈ *transition_relation*"
⟨*proof*⟩

lemma *path_with_word_induct_reverse* [*consumes 1*, *case_names path_with_word_refl path_with_word_step_rev*]:
"(*ss*, *w*) ∈ *path_with_word* ⟹
(⋀*s*. *P* [*s*] []) ⟹
(⋀*ss s w l s*′. (*ss* @ [*s*], *w*) ∈ *path_with_word* ⟹
P (*ss* @ [*s*]) *w* ⟹
(*s*, *l*, *s*′) ∈ *transition_relation* ⟹
P (*ss* @ [*s*, *s*′]) (*w* @ [*l*]))
⟹ *P ss w*"
⟨*proof*⟩

lemma *path_with_word_from_induct_reverse*:
"(*ss*, *w*) ∈ *path_with_word_from start* ⟹
(⋀*s*. *P* [*s*] []) ⟹
(⋀*ss s w l s*′. (*ss* @ [*s*], *w*) ∈ *path_with_word_from start* ⟹
P (*ss* @ [*s*]) *w* ⟹
(*s*, *l*, *s*′) ∈ *transition_relation* ⟹

$P\ (ss\ @\ [s,\ s'])\ (w\ @\ [l]))$
$\Longrightarrow P\ ss\ w$"
⟨*proof*⟩

inductive *transition_of* :: "('*state*, '*label*) *transition* ⇒ '*state list* ∗ '*label list* ⇒ *bool*" **where**
"*transition_of* (*s1*,γ,*s2*) (*s1*#*s2*#*ss*, γ#*w*)"
| "*transition_of* (*s1*,γ,*s2*) (*ss*, *w*) ⟹
transition_of (*s1*,γ,*s2*) (*s*#*ss*, μ#*w*)"

lemma *path_with_word_not_empty*[*simp*]: "¬([],*w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *trans_star_path_with_word*:
assumes "(*p*, *w*, *q*) ∈ *trans_star*"
shows "∃ *ss*. *hd ss* = *p* ∧ *last ss* = *q* ∧ (*ss*, *w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *trans_star_trans_star_states*:
assumes "(*p*, *w*, *q*) ∈ *trans_star*"
shows "∃ *ss*. (*p*, *w*, *ss*, *q*) ∈ *trans_star_states*"
⟨*proof*⟩

lemma *trans_star_states_trans_star*:
assumes "(*p*, *w*, *ss*, *q*) ∈ *trans_star_states*"
shows "(*p*, *w*, *q*) ∈ *trans_star*"
⟨*proof*⟩

lemma *path_with_word_trans_star*:
assumes "(*ss*, *w*) ∈ *path_with_word*"
assumes "*length ss* ≠ *0*"
shows "(*hd ss*, *w*, *last ss*) ∈ *trans_star*"
⟨*proof*⟩

lemma *path_with_word_trans_star_Cons*:
assumes "(*s1*#*ss*@[*s2*], *w*) ∈ *path_with_word*"
shows "(*s1*, *w*, *s2*) ∈ *trans_star*"
⟨*proof*⟩

lemma *path_with_word_trans_star_Singleton*:
assumes "([*s2*], *w*) ∈ *path_with_word*"
shows "(*s2*, [], *s2*) ∈ *trans_star*"
⟨*proof*⟩

lemma *trans_star_split*:
assumes "(*p''*, *u1* @ *w1*, *q*) ∈ *trans_star*"
shows "∃ *q1*. (*p''*, *u1*, *q1*) ∈ *trans_star* ∧ (*q1*, *w1*, *q*) ∈ *trans_star*"
⟨*proof*⟩

lemma *trans_star_states_append*:
assumes "(*p2*, *w2*, *w2_ss*, *q'*) ∈ *trans_star_states*"
assumes "(*q'*, *v*, *v_ss*, *q*) ∈ *trans_star_states*"
shows "(*p2*, *w2* @ *v*, *w2_ss* @ *tl v_ss*, *q*) ∈ *trans_star_states*"
⟨*proof*⟩

lemma *trans_star_states_length*:
assumes "(*p*, *u*, *u_ss*, *p1*) ∈ *trans_star_states*"
shows "*length u_ss* = *Suc* (*length u*)"
⟨*proof*⟩

lemma *trans_star_states_last*:
assumes "(*p*, *u*, *u_ss*, *p1*) ∈ *trans_star_states*"
shows "*p1* = *last u_ss*"
⟨*proof*⟩

lemma *trans_star_states_hd*:
assumes "$(q', v, v\_ss, q) \in trans\_star\_states$"
shows "$q' = hd\ v\_ss$"
⟨*proof*⟩

lemma *trans_star_states_transition_relation*:
assumes "$(p, \gamma \# w\_rest, ss, q) \in trans\_star\_states$"
shows "$\exists\, s\ \gamma'.\ (s, \gamma', q) \in transition\_relation$"
⟨*proof*⟩

lemma *trans_star_states_path_with_word*:
assumes "$(p, w, ss, q) \in trans\_star\_states$"
shows "$(ss,w) \in path\_with\_word$"
⟨*proof*⟩

lemma *path_with_word_trans_star_states*:
assumes "$(ss,w) \in path\_with\_word$"
assumes "$p = hd\ ss$"
assumes "$q = last\ ss$"
shows "$(p, w, ss, q) \in trans\_star\_states$"
⟨*proof*⟩

lemma *append_path_with_word_path_with_word*:
assumes "$last\ \gamma 2ss = hd\ v\_ss$"
assumes "$(\gamma 2ss, \gamma 2\varepsilon) \in path\_with\_word$"
assumes "$(v\_ss, v) \in path\_with\_word$"
shows "$(\gamma 2ss, \gamma 2\varepsilon)\ @'\ (v\_ss, v) \in path\_with\_word$"
⟨*proof*⟩

lemma *hd_is_hd*:
assumes "$(p, w, ss, q) \in trans\_star\_states$"
assumes "$(p1, \gamma, q1) = hd\ (transition\_list'\ (p, w, ss, q))$"
assumes "$transition\_list'\ (p, w, ss, q) \neq []$"
shows "$p = p1$"
⟨*proof*⟩

definition *srcs* :: "'*state set*" **where**
"$srcs = \{p.\ \nexists\, q\ \gamma.\ (q, \gamma, p) \in transition\_relation\}$"

definition *sinks* :: "'*state set*" **where**
"$sinks = \{p.\ \nexists\, q\ \gamma.\ (p, \gamma, q) \in transition\_relation\}$"

definition *isolated* :: "'*state set*" **where**
"$isolated = srcs \cap sinks$"

lemma *srcs_def2*:
"$q \in srcs \longleftrightarrow (\nexists\, q'\ \gamma.\ (q', \gamma, q) \in transition\_relation)$"
⟨*proof*⟩

lemma *sinks_def2*:
"$q \in sinks \longleftrightarrow (\nexists\, q'\ \gamma.\ (q, \gamma, q') \in transition\_relation)$"
⟨*proof*⟩

lemma *isolated_no_edges*:
assumes "$(p, \gamma, q) \in transition\_relation$"
shows "$p \notin isolated \wedge q \notin isolated$"
⟨*proof*⟩

lemma *source_never_or_hd*:
assumes "$(ss, w) \in path\_with\_word$"
assumes "$p1 \in srcs$"
assumes "$t = (p1, \gamma, q1)$"

shows *"count (transitions_of (ss, w)) t = 0 ∨*
((hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1))"
⟨*proof*⟩

lemma *source_only_hd*:
assumes *"(ss, w) ∈ path_with_word"*
assumes *"p1 ∈ srcs"*
assumes *"count (transitions_of (ss, w)) t > 0"*
assumes *"t = (p1, γ, q1)"*
shows *"hd (transition_list (ss, w)) = t ∧ count (transitions_of (ss, w)) t = 1"*
⟨*proof*⟩

lemma *no_end_in_source*:
assumes *"(p, w, qq) ∈ trans_star"*
assumes *"w ≠ []"*
shows *"qq ∉ srcs"*
⟨*proof*⟩

lemma *transition_list_length_Cons*:
assumes *"length ss = Suc (length w)"*
assumes *"hd (transition_list (ss, w)) = (p, γ, q)"*
assumes *"transition_list (ss, w) ≠ []"*
shows *"∃ w' ss'. w = γ # w' ∧ ss = p # q # ss'"*
⟨*proof*⟩

lemma *transition_list_Cons*:
assumes *"(p, w, ss, q) ∈ trans_star_states"*
assumes *"hd (transition_list (ss, w)) = (p, γ, q1)"*
assumes *"transition_list (ss, w) ≠ []"*
shows *"∃ w' ss'. w = γ # w' ∧ ss = p # q1 # ss'"*
⟨*proof*⟩

lemma *nothing_after_sink*:
assumes *"([q, q']@ss, γ1#w) ∈ path_with_word"*
assumes *"q' ∈ sinks"*
shows *"ss = [] ∧ w = []"*
⟨*proof*⟩

lemma *count_transitions_of'_tails*:
assumes *"(p, γ', q'_add) ≠ (p1, γ, q')"*
shows *"count (transitions_of' (p, γ' # w, p # q'_add # ss_rest, q)) (p1, γ, q') =*
count (transitions_of' (q'_add, w, q'_add # ss_rest, q)) (p1, γ, q')"
⟨*proof*⟩

lemma *avoid_count_zero*:
assumes *"(p, w, ss, q) ∈ trans_star_states"*
assumes *"(p1, γ, q') ∉ transition_relation"*
shows *"count (transitions_of' (p, w, ss, q)) (p1, γ, q') = 0"*
⟨*proof*⟩

lemma *transition_list_append*:
assumes *"(ss,w) ∈ path_with_word"*
assumes *"(ss',w') ∈ path_with_word"*
assumes *"last ss = hd ss'"*
shows *"transition_list ((ss,w) @´ (ss',w')) = transition_list (ss,w) @ transition_list (ss',w')"*
⟨*proof*⟩

lemma *split_path_with_word_beginning''*:
assumes *"(SS, WW) ∈ path_with_word"*
assumes *"SS = (ss @ ss')"*
assumes *"length ss = Suc (length w)"*
assumes *"WW = w @ w'"*
shows *"(ss,w) ∈ path_with_word"*

⟨*proof*⟩

lemma *split_path_with_word_end'*:
assumes "(*SS*,*WW*) ∈ *path_with_word*"
assumes "*SS* = (*ss* @ *ss'*)"
assumes "*length ss'* = *Suc* (*length w'*)"
assumes "*WW* = *w* @ *w'*"
shows "(*ss'*,*w'*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *split_path_with_word_end*:
assumes "(*ss* @ *ss'*,*w* @ *w'*) ∈ *path_with_word*"
assumes "*length ss'* = *Suc* (*length w'*)"
shows "(*ss'*,*w'*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *split_path_with_word_beginning'*:
assumes "(*ss* @ *ss'*,*w* @ *w'*) ∈ *path_with_word*"
assumes "*length ss* = *Suc* (*length w*)"
shows "(*ss*,*w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *split_path_with_word_beginning*:
assumes "(*ss*, *w*) @´ (*ss'*, *w'*) ∈ *path_with_word*"
assumes "*length ss* = *Suc* (*length w*)"
shows "(*ss*,*w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *path_with_word_remove_last'*:
assumes "(*SS*, *W*) ∈ *path_with_word*"
assumes "*SS* = *ss* @ [*s*, *s'*]"
assumes "*W* = *w* @ [*l*]"
shows "(*ss* @ [*s*], *w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *path_with_word_remove_last*:
assumes "(*ss* @ [*s*, *s'*], *w* @ [*l*]) ∈ *path_with_word*"
shows "(*ss* @ [*s*], *w*) ∈ *path_with_word*"
⟨*proof*⟩

lemma *transition_list_append_edge*:
assumes "(*ss* @ [*s*, *s'*], *w* @ [*l*]) ∈ *path_with_word*"
shows "*transition_list* (*ss* @ [*s*, *s'*], *w* @ [*l*]) = *transition_list* (*ss* @ [*s*], *w*) @ [(*s*, *l*, *s'*)]"
⟨*proof*⟩

end

1.4 More LTS lemmas

lemma *hd_transition_list_append_path_with_word*:
assumes "*hd* (*transition_list* (*ss*, *w*)) = (*p1*, *γ*, *q1*)"
assumes "*transition_list* (*ss*, *w*) ≠ []"
shows "([*p1*, *q1*], [*γ*]) @´ (*tl ss*, *tl w*) = (*ss*, *w*)"
⟨*proof*⟩

lemma *counting*:
"*count* (*transitions_of'* ((*hdss1*,*ww1*,*ss1*,*lastss1*))) (*s1*, *γ*, *s2*) =
count (*transitions_of* ((*ss1*,*ww1*))) (*s1*, *γ*, *s2*)"
⟨*proof*⟩

lemma *count_append_path_with_word_γ*:
assumes "*length ss1* = *Suc* (*length ww1*)"
assumes "*ss2* ≠ []"
shows "*count* (*transitions_of* (((*ss1*,*ww1*),*γ'*) @^*γ* (*ss2*,*ww2*))) (*s1*, *γ*, *s2*) =

count (transitions_of (ss1,ww1)) (s1, γ, s2) +
(if s1 = last ss1 ∧ s2 = hd ss2 ∧ γ = γ′ then 1 else 0) +
count (transitions_of (ss2,ww2)) (s1, γ, s2)"
⟨*proof*⟩

lemma *count_append_path_with_word*:
assumes *"length ss1 = Suc (length ww1)"*
assumes *"ss2 ≠ []"*
assumes *"last ss1 = hd ss2"*
shows *"count (transitions_of (((ss1, ww1)) @′ (ss2, ww2))) (s1, γ, s2) =*
count (transitions_of (ss1, ww1)) (s1, γ, s2) +
count (transitions_of (ss2, ww2)) (s1, γ, s2)"
⟨*proof*⟩

lemma *count_append_trans_star_states_γ_length*:
assumes *"length (ss1) = Suc (length (ww1))"*
assumes *"ss2 ≠ []"*
shows *"count (transitions_of′ (((hdss1,ww1,ss1,lastss1),γ′) @@^γ (hdss2,ww2,ss2,lastss2))) (s1, γ, s2) =*
count (transitions_of′ (hdss1,ww1,ss1,lastss1)) (s1, γ, s2) +
(if s1 = last ss1 ∧ s2 = hd ss2 ∧ γ = γ′ then 1 else 0) +
count (transitions_of′ (hdss2,ww2,ss2,lastss2)) (s1, γ, s2)"
⟨*proof*⟩

lemma *count_append_trans_star_states_γ*:
assumes *"(hdss1,ww1,ss1,lastss1) ∈ LTS.trans_star_states A"*
assumes *"(hdss2,ww2,ss2,lastss2) ∈ LTS.trans_star_states A"*
shows *"count (transitions_of′ (((hdss1,ww1,ss1,lastss1),γ′) @@^γ (hdss2,ww2,ss2,lastss2))) (s1, γ, s2) =*
count (transitions_of′ (hdss1,ww1,ss1,lastss1)) (s1, γ, s2) +
(if s1 = last ss1 ∧ s2 = hd ss2 ∧ γ = γ′ then 1 else 0) +
count (transitions_of′ (hdss2,ww2,ss2,lastss2)) (s1, γ, s2)"
⟨*proof*⟩

lemma *count_append_trans_star_states_length*:
assumes *"length (ss1) = Suc (length (ww1))"*
assumes *"ss2 ≠ []"*
assumes *"last ss1 = hd ss2"*
shows *"count (transitions_of′ (((hdss1,ww1,ss1,lastss1)) @@′ (hdss2,ww2,ss2,lastss2))) (s1, γ, s2) =*
count (transitions_of′ (hdss1,ww1,ss1,lastss1)) (s1, γ, s2) +
count (transitions_of′ (hdss2,ww2,ss2,lastss2)) (s1, γ, s2)"
⟨*proof*⟩

lemma *count_append_trans_star_states*:
assumes *"(hdss1,ww1,ss1,lastss1) ∈ LTS.trans_star_states A"*
assumes *"(lastss1,ww2,ss2,lastss2) ∈ LTS.trans_star_states A"*
shows *"count (transitions_of′ (((hdss1,ww1,ss1,lastss1)) @@′ (lastss1,ww2,ss2,lastss2))) (s1, γ, s2) =*
count (transitions_of′ (hdss1,ww1,ss1,lastss1)) (s1, γ, s2) +
count (transitions_of′ (lastss1,ww2,ss2,lastss2)) (s1, γ, s2)"
⟨*proof*⟩

context fixes Δ :: *"('state, 'label) transition set"* **begin**
fun *reach* **where**
"reach p [] = {p}"
| *"reach p (γ#w) =*
(⋃ q′ ∈ (⋃ (p′,γ′,q′) ∈ Δ. if p′ = p ∧ γ′ = γ then {q′} else {}).
reach q′ w)"
end
lemma *trans_star_imp_exec*: *"(p,w,q) ∈ LTS.trans_star Δ ⟹ q ∈ reach Δ p w"*
⟨*proof*⟩
lemma *reach_imp*: *"q ∈ reach Δ p w ⟹ (p,w,q) ∈ LTS.trans_star Δ"*
⟨*proof*⟩
lemma *trans_star_code*[*code_unfold*]: *"(p,w,q) ∈ LTS.trans_star Δ ⟷ q ∈ reach Δ p w"*
⟨*proof*⟩

lemma *subset_srcs_code*[*code_unfold*]:
"$X \subseteq LTS.srcs\ A \longleftrightarrow (\forall\, q \in X.\ q \notin snd\ `\ snd\ `\ A)$"
⟨*proof*⟩

lemma *LTS_trans_star_mono*:
"*mono LTS.trans_star*"
⟨*proof*⟩

lemma *count_next_0*:
assumes "*count* (*transitions_of* ($s\ \#\ s'\ \#\ ss$, $l\ \#\ w$)) ($p1$, γ, q') = *0*"
shows "*count* (*transitions_of* ($s'\ \#\ ss$, w)) ($p1$, γ, q') = *0*"
⟨*proof*⟩

lemma *count_next_hd*:
assumes "*count* (*transitions_of* ($s\ \#\ s'\ \#\ ss$, $l\ \#\ w$)) ($p1$, γ, q') = *0*"
shows "$(s, l, s') \neq (p1, \gamma, q')$"
⟨*proof*⟩

lemma *count_empty_zero*: "*count* (*transitions_of'* (p, [], [*p_add*], *p_add*)) ($p1$, γ, q') = *0*"
⟨*proof*⟩

lemma *count_zero_remove_path_with_word*:
assumes "$(ss, w) \in LTS.path\_with\_word\ Ai$"
assumes "*0* = *count* (*transitions_of* (ss, w)) ($p1$, γ, q')"
assumes "$Ai = Aiminus1 \cup \{(p1, \gamma, q')\}$"
shows "$(ss, w) \in LTS.path\_with\_word\ Aiminus1$"
⟨*proof*⟩

lemma *count_zero_remove_path_with_word_trans_star_states*:
assumes "$(p, w, ss, q) \in LTS.trans\_star\_states\ Ai$"
assumes "*0* = *count* (*transitions_of'* (p, w, ss, q)) ($p1$, γ, q')"
assumes "$Ai = Aiminus1 \cup \{(p1, \gamma, q')\}$"
shows "$(p, w, ss, q) \in LTS.trans\_star\_states\ Aiminus1$"
⟨*proof*⟩

lemma *count_zero_remove_trans_star_states_trans_star*:
assumes "$(p, w, ss, q) \in LTS.trans\_star\_states\ Ai$"
assumes "*0* = *count* (*transitions_of'* (p, w, ss, q)) ($p1$, γ, q')"
assumes "$Ai = Aiminus1 \cup \{(p1, \gamma, q')\}$"
shows "$(p, w, q) \in LTS.trans\_star\ Aiminus1$"
⟨*proof*⟩

lemma *split_at_first_t*:
assumes "$(p, w, ss, q) \in LTS.trans\_star\_states\ Ai$"
assumes "$Suc\ j' = count\ (transitions\_of'\ (p, w, ss, q))\ (p1, \gamma, q')$"
assumes "$(p1, \gamma, q') \notin Aiminus1$"
assumes "$Ai = Aiminus1 \cup \{(p1, \gamma, q')\}$"
shows "$\exists\, u\ v\ u\_ss\ v\_ss.$
$ss = u\_ss\ @\ v\_ss \wedge$
$w = u\ @\ [\gamma]\ @\ v \wedge$
$(p, u, u\_ss, p1) \in LTS.trans\_star\_states\ Aiminus1 \wedge$
$(p1, [\gamma], q') \in LTS.trans\_star\ Ai \wedge$
$(q', v, v\_ss, q) \in LTS.trans\_star\_states\ Ai \wedge$
$(p, w, ss, q) = ((p, u, u\_ss, p1),\gamma)\ @@^{\gamma}\ (q', v, v\_ss,q)$"
⟨*proof*⟩

lemma *trans_star_states_mono*:
assumes "$(p, w, ss, q) \in LTS.trans\_star\_states\ A1$"
assumes "$A1 \subseteq A2$"
shows "$(p, w, ss, q) \in LTS.trans\_star\_states\ A2$"
⟨*proof*⟩

lemma *count_combine_trans_star_states_append*:
assumes "*ss = u_ss @ v_ss ∧ w = u @ [γ] @ v*"
assumes "*t = (p1, γ, q′)*"
assumes "*(p, u, u_ss, p1) ∈ LTS.trans_star_states A*"
assumes "*(q′, v, v_ss, q) ∈ LTS.trans_star_states B*"
shows "*count (transitions_of′ (p, w, ss, q)) t =*
count (transitions_of′ (p, u, u_ss, p1)) t +
1 +
count (transitions_of′ (q′, v, v_ss, q)) t"
⟨*proof*⟩

lemma *count_combine_trans_star_states*:
assumes "*t = (p1, γ, q′)*"
assumes "*(p, u, u_ss, p1) ∈ LTS.trans_star_states A*"
assumes "*(q′, v, v_ss, q) ∈ LTS.trans_star_states B*"
shows "*count (transitions_of′ (((p, u, u_ss, p1),γ) @@^γ (q′, v, v_ss, q))) t =*
count (transitions_of′ (p, u, u_ss, p1)) t + 1 + count (transitions_of′ (q′, v, v_ss, q)) t"
⟨*proof*⟩

lemma *transition_list_reversed_simp*:
assumes "*length ss = length w*"
shows "*transition_list (ss @ [s, s′], w @ [l]) = (transition_list (ss@[s],w)) @ [(s,l,s′)]*"
⟨*proof*⟩

lemma *LTS_trans_star_mono′*:
"*mono LTS.trans_star_states*"
⟨*proof*⟩

lemma *path_with_word_mono′*:
assumes "*(ss, w) ∈ LTS.path_with_word A1*"
assumes "*A1 ⊆ A2*"
shows "*(ss, w) ∈ LTS.path_with_word A2*"
⟨*proof*⟩

lemma *LTS_path_with_word_mono*:
"*mono LTS.path_with_word*"
⟨*proof*⟩

1.5 Reverse transition system

fun *rev_edge* :: "*('n,'v) transition ⇒ ('n,'v) transition*" **where**
"*rev_edge (q_s,α,q_o) = (q_o, α, q_s)*"

lemma *rev_edge_rev_edge_id*[*simp*]: "*rev_edge (rev_edge x) = x*"
⟨*proof*⟩

fun *rev_path_with_word* :: "*'n list ∗ 'v list ⇒ 'n list ∗ 'v list*" **where**
"*rev_path_with_word (es,ls) = (rev es, rev ls)*"

definition *rev_edge_list* :: "*('n,'v) transition list ⇒ ('n,'v) transition list*" **where**
"*rev_edge_list ts = rev (map rev_edge ts)*"

context *LTS* **begin**

interpretation *rev_LTS*: *LTS* "*(rev_edge ‘ transition_relation)*"
⟨*proof*⟩

lemma *rev_path_in_rev_pg*:
assumes "*(ss, w) ∈ path_with_word*"
shows "*(rev ss, rev w) ∈ rev_LTS.path_with_word*"
⟨*proof*⟩

lemma *transition_list_rev_edge_list*:

assumes *"(ss,w) ∈ path_with_word"*
shows *"transition_list (rev ss, rev w) = rev_edge_list (transition_list (ss, w))"*
⟨*proof*⟩

end

2 LTS with epsilon

2.1 LTS functions

context begin

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

definition *inters_ε* :: *"('state, 'label option) transition set ⇒ ('state, 'label option) transition set ⇒ (('state * 'state), 'label option) transition set"* **where**
"inters_ε ts1 ts2 =
{((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}"

end

2.2 LTS with epsilon locale

locale *LTS_ε = LTS transition_relation* **for** *transition_relation* :: *"('state, 'label option) transition set"*
begin

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

inductive-set *trans_star_ε* :: *"('state * 'label list * 'state) set"* **where**
trans_star_ε_refl[*iff*]: *"(p, [], p) ∈ trans_star_ε"*
| *trans_star_ε_step_γ*: *"(p, Some γ, q') ∈ transition_relation ⟹ (q',w,q) ∈ trans_star_ε*
⟹ (p, γ#w, q) ∈ trans_star_ε"
| *trans_star_ε_step_ε*: *"(p, ε, q') ∈ transition_relation ⟹ (q',w,q) ∈ trans_star_ε*
⟹ (p, w, q) ∈ trans_star_ε"

inductive-cases *trans_star_ε_empty* [*elim*]: *"(p, [], q) ∈ trans_star_ε"*
inductive-cases *trans_star_cons_ε*: *"(p, γ#w, q) ∈ trans_star"*

definition *remove_ε* :: *"'label option list ⇒ 'label list"* **where**
"remove_ε w = map the (removeAll ε w)"

definition *ε_exp* :: *"'label option list ⇒ 'label list ⇒ bool"* **where**
"ε_exp w' w ⟷ map the (removeAll ε w') = w"

lemma *trans_star_trans_star_ε*:
assumes *"(p, w, q) ∈ trans_star"*
shows *"(p, map the (removeAll ε w), q) ∈ trans_star_ε"*
⟨*proof*⟩

lemma *trans_star_ε_ε_exp_trans_star*:
assumes *"(p, w, q) ∈ trans_star_ε"*
shows *"∃ w'. ε_exp w' w ∧ (p, w', q) ∈ trans_star"*
⟨*proof*⟩

lemma *trans_star_ε_iff_ε_exp_trans_star*:
"(p, w, q) ∈ trans_star_ε ⟷ (∃ w'. ε_exp w' w ∧ (p, w', q) ∈ trans_star)"
⟨*proof*⟩

lemma *ε_exp_split'*:

assumes *"ε_exp u_ε (γ1 # u1)"*
shows *"∃γ1_ε u1_ε. ε_exp γ1_ε [γ1] ∧ ε_exp u1_ε u1 ∧ u_ε = γ1_ε @ u1_ε"*
⟨*proof*⟩

lemma *remove_ε_append_dist*:
"remove_ε (w @ w') = remove_ε w @ remove_ε w'"
⟨*proof*⟩

lemma *remove_ε_Cons_tl*:
assumes *"remove_ε w = remove_ε (Some γ' # tl w)"*
shows *"γ' # remove_ε (tl w) = remove_ε w"*
⟨*proof*⟩

lemma *trans_star_states_trans_star_ε*:
assumes *"(p, w, ss, q) ∈ trans_star_states"*
shows *"(p, LTS_ε.remove_ε w, q) ∈ trans_star_ε"*
⟨*proof*⟩

lemma *no_edge_to_source_ε*:
assumes *"(p, [γ], qq) ∈ trans_star_ε"*
shows *"qq ∉ srcs"*
⟨*proof*⟩

lemma *trans_star_not_to_source_ε*:
assumes *"(p''', w, q) ∈ trans_star_ε"*
assumes *"p''' ≠ q"*
assumes *"q' ∈ srcs"*
shows *"q' ≠ q"*
⟨*proof*⟩

lemma *append_edge_edge_trans_star_ε*:
assumes *"(p1, Some γ', p2) ∈ transition_relation"*
assumes *"(p2, Some γ'', q1) ∈ transition_relation"*
assumes *"(q1, u1, q) ∈ trans_star_ε"*
shows *"(p1, [γ', γ''] @ u1, q) ∈ trans_star_ε"*
⟨*proof*⟩

inductive-set *trans_star_states_ε* :: *"('state * 'label list * 'state list * 'state) set"* **where**
trans_star_states_ε_refl[*iff*]:
"(p,[],[p],p) ∈ trans_star_states_ε"
| *trans_star_states_ε_step_γ*:
"(p,Some γ,q') ∈ transition_relation ⟹
(q',w,ss,q) ∈ trans_star_states_ε ⟹
(p, γ#w, p#ss, q) ∈ trans_star_states_ε"
| *trans_star_states_ε_step_ε*:
"(p, ε ,q') ∈ transition_relation ⟹
(q',w,ss,q) ∈ trans_star_states_ε ⟹
(p, w, p#ss, q) ∈ trans_star_states_ε"

inductive-set *path_with_word_ε* :: *"('state list * 'label list) set"* **where**
path_with_word_ε_refl[*iff*]:
"([s],[]) ∈ path_with_word_ε"
| *path_with_word_ε_step_γ*:
"(s'#ss, w) ∈ path_with_word_ε ⟹
(s,Some l,s') ∈ transition_relation ⟹
(s#s'#ss,l#w) ∈ path_with_word_ε"
| *path_with_word_ε_step_ε*:
"(s'#ss, w) ∈ path_with_word_ε ⟹
(s,ε,s') ∈ transition_relation ⟹
(s#s'#ss,w) ∈ path_with_word_ε"

lemma *ε_exp_Some_length*:

assumes *"ε_exp (Some α # w1') w"*
shows *"0 < length w"*
⟨*proof*⟩

lemma *ε_exp_Some_hd*:
assumes *"ε_exp (Some α # w1') w"*
shows *"hd w = α"*
⟨*proof*⟩

lemma *exp_empty_empty*:
assumes *"ε_exp [] w"*
shows *"w = []"*
⟨*proof*⟩

end

2.3 More LTS lemmas

lemma *LTS_ε_trans_star_ε_mono*:
"mono LTS_ε.trans_star_ε"
⟨*proof*⟩

definition *ε_edge_of_edge* **where**
"ε_edge_of_edge = (λ(a, l, b). (a, Some l, b))"

definition *LTS_ε_of_LTS* **where**
"LTS_ε_of_LTS transition_relation = ε_edge_of_edge ' transition_relation"

end

References

[Wim20] Simon Wimmer. Archive of graph formalizations. 2020. https://github.com/wimmers/archive-of-graph-formalizations.