More on Lazy Lists

Stefan Friedrich

October 13, 2025

Abstract

This theory contains some useful extensions to the LList theory by Larry Paulson, including finite, infinite, and positive llists over an alphabet, as well as the new constants take and drop and the prefix order of llists. Finally, the notions of safety and liveness in the sense of [1] are defined.

Contents

1 More on llists

theory `LList2`
imports `Coinductive.Coinductive_List`
begin

1.1 Preliminaries

notation
  LCons (**infixr** <##> 65) **and**
  lappend (**infixr** <@@> 65)

lemmas llistE = llist.exhaust

1.2 Finite and infinite llists over an alphabet

inductive_set
  finlsts :: "'a set ⇒ 'a llist set" (<(_*)> [1000] 999)
  for A :: "'a set"
where
  LNil_fin [iff]: "LNil ∈ A*"
| LCons_fin [intro!]: "⟦ l ∈ A*; a ∈ A ⟧ ⟹ a ## l ∈ A*"

coinductive_set
  alllsts :: "'a set ⇒ 'a llist set" (<(_∞)> [1000] 999)
  for A :: "'a set"
where
  LNil_all [iff]: "LNil ∈ A∞"
| LCons_all [intro!]: "⟦ l ∈ A∞; a ∈ A ⟧ ⟹ a ## l ∈ A∞"

declare alllsts.cases [case_names LNil LCons, cases set: alllsts]

definition inflsts :: "'a set ⇒ 'a llist set" (<(_ω)> [1000] 999)
where "Aω ≡ A∞ - UNIV*"

definition fpslsts :: "'a set ⇒ 'a llist set" (<(_♣)> [1000] 999)
where "A♣ ≡ A* - {LNil}"

definition poslsts :: "'a set ⇒ 'a llist set" (<(_♠)> [1000] 999)
where "A♠ ≡ A∞ - {LNil}"

1.2.1 Facts about all llists

lemma alllsts_UNIV [iff]:
  "s ∈ UNIV∞"
⟨*proof*⟩

lemma alllsts_empty [simp]: "{}∞ = {LNil}"
  ⟨*proof*⟩

lemma alllsts_mono:
  assumes asubb: "A ⊆ B"
  shows "A∞ ⊆ B∞"
⟨*proof*⟩

lemmas alllstsp_mono [mono] = alllsts_mono [to_pred pred_subset_eq]

lemma LConsE [iff]: "x##xs ∈ A∞ = (x∈A ∧ xs ∈ A∞)"
  ⟨*proof*⟩

1.2.2 Facts about non-empty (positive) llists

lemma poslsts_iff [iff]:
"(s ∈ A♠) = (s ∈ A∞ ∧ s ≠ LNil)"
⟨*proof*⟩

lemma poslsts_UNIV [iff]:
"s ∈ UNIV♠ = (s ≠ LNil)"
⟨*proof*⟩

lemma poslsts_empty [simp]: "{}♠ = {}"
⟨*proof*⟩

lemma poslsts_mono:
"A ⊆ B ⟹ A♠ ⊆ B♠"
⟨*proof*⟩

1.2.3 Facts about finite llists

lemma finlsts_empty [simp]: "{}⋆ = {LNil}"
⟨*proof*⟩

lemma finsubsetall: "x ∈ A⋆ ⟹ x ∈ A∞"
⟨*proof*⟩

lemma finlsts_mono:
"A⊆B ⟹ A⋆ ⊆ B⋆"
⟨*proof*⟩

lemmas finlstsp_mono [mono] = finlsts_mono [to_pred pred_subset_eq]

lemma finlsts_induct
[case_names LNil_fin LCons_fin, induct set: finlsts, consumes 1]:
assumes xA: "x ∈ A⋆"
and lnil: "⋀l. l = LNil ⟹ P l"
and lcons: "⋀a l. ⟦l ∈ A⋆; P l; a ∈ A⟧ ⟹ P (a ## l)"
shows "P x"
⟨*proof*⟩

lemma finite_lemma:
assumes "x ∈ A⋆"
shows "x ∈ B∞ ⟹ x ∈ B⋆"
⟨*proof*⟩

lemma fin_finite [dest]:
assumes "r ∈ A⋆" "r ∉ UNIV⋆"
shows "False"
⟨*proof*⟩

lemma finT_simp [simp]:
"r ∈ A⋆ ⟹ r∈UNIV⋆"
⟨*proof*⟩

1.2.4 A recursion operator for finite llists

definition finlsts_pred :: "('a llist × 'a llist) set"
where "finlsts_pred ≡ {(r,s). r ∈ UNIV⋆ ∧ (∃a. a##r = s)}"

definition finlsts_rec :: "['b, ['a, 'a llist, 'b] ⇒ 'b] ⇒ 'a llist ⇒ 'b"
where
"finlsts_rec c d r ≡ if r ∈ UNIV⋆
then (wfrec finlsts_pred (%f. case_llist c (%a r. d a r (f r))) r)
else undefined"

lemma finlsts_predI: "r ∈ A⋆ ⟹ (r, a##r) ∈ finlsts_pred"
⟨*proof*⟩

lemma wf_finlsts_pred: "wf finlsts_pred"
⟨*proof*⟩

lemma finlsts_rec_LNil: "finlsts_rec c d LNil = c"
⟨*proof*⟩

lemma finlsts_rec_LCons:
"r ∈ A⋆ ⟹ finlsts_rec c d (a ## r) = d a r (finlsts_rec c d r)"
⟨*proof*⟩

lemma finlsts_rec_LNil_def:
"f ≡ finlsts_rec c d ⟹ f LNil = c"
⟨*proof*⟩

lemma finlsts_rec_LCons_def:
"⟦ f ≡ finlsts_rec c d; r ∈ A⋆ ⟧ ⟹ f (a ## r) = d a r (f r)"
⟨*proof*⟩

1.2.5 Facts about non-empty (positive) finite llists

lemma fpslsts_iff [iff]:
"(s ∈ A♣) = (s ∈ A⋆ ∧ s ≠ LNil)"
⟨*proof*⟩

lemma fpslsts_empty [simp]: "{}♣ = {}"
⟨*proof*⟩

lemma fpslsts_mono:
"A ⊆ B ⟹ A♣ ⊆ B♣"
⟨*proof*⟩

lemma fpslsts_cases [case_names LCons, cases set: fpslsts]:
assumes rfps: "r ∈ A♣"
and H: "⋀ a rs. ⟦ r = a ## rs; a∈A; rs ∈ A⋆ ⟧ ⟹ R"
shows "R"
⟨*proof*⟩

1.2.6 Facts about infinite llists

lemma inflstsI [intro]:

"⟦ x ∈ A^∞; x ∈ UNIV^⋆ ⟹ False ⟧ ⟹ x ∈ A^ω"
⟨*proof*⟩

lemma inflstsE [elim]:
"⟦ x ∈ A^ω; ⟦ x ∈ A^∞; x ∉ UNIV^⋆ ⟧ ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma inflsts_empty [simp]: "{}^ω = {}"
⟨*proof*⟩

lemma infsubsetall: "x ∈ A^ω ⟹ x ∈ A^∞"
⟨*proof*⟩

lemma inflsts_mono:
"A ⊆ B ⟹ A^ω ⊆ B^ω"
⟨*proof*⟩

lemma inflsts_cases [case_names LCons, cases set: inflsts, consumes 1]:
assumes sinf: "s ∈ A^ω"
and R: "⋀a l. ⟦ l ∈ A^ω; a ∈ A; s = a ## l ⟧ ⟹ R"
shows "R"
⟨*proof*⟩

lemma inflstsI2: "⟦a ∈ A; t ∈ A^ω⟧ ⟹ a ## t ∈ A^ω"
⟨*proof*⟩

lemma infT_simp [simp]:
"r ∈ A^ω ⟹ r∈UNIV^ω"
⟨*proof*⟩

lemma alllstsE [consumes 1, case_names finite infinite]:
"⟦ x∈A^∞; x ∈ A^⋆ ⟹ P; x ∈ A^ω ⟹ P ⟧ ⟹ P"
⟨*proof*⟩

lemma fin_inf_cases [case_names finite infinite]:
"⟦ r∈UNIV^⋆ ⟹ P; r ∈ UNIV^ω ⟹ P ⟧ ⟹ P"
⟨*proof*⟩

lemma fin_Int_inf: "A^⋆ ∩ A^ω = {}"
and fin_Un_inf: "A^⋆ ∪ A^ω = A^∞"
⟨*proof*⟩

lemma notfin_inf [iff]: "(x ∉ UNIV^⋆) = (x ∈ UNIV^ω)"
⟨*proof*⟩

lemma notinf_fin [iff]: "(x ∉ UNIV^ω) = (x ∈ UNIV^⋆)"
⟨*proof*⟩

1.3 Lappend

1.3.1 Simplification

lemma lapp_inf [simp]:
  assumes "s ∈ A^ω"
  shows "s @@ t = s"
⟨*proof*⟩

lemma LNil_is_lappend_conv [iff]:
"(LNil = s @@ t) = (s = LNil ∧ t = LNil)"
  ⟨*proof*⟩

lemma lappend_is_LNil_conv [iff]:
  "(s @@ t = LNil) = (s = LNil ∧ t = LNil)"
  ⟨*proof*⟩

lemma same_lappend_eq [iff]:
 "r ∈ $A^\star$ ⟹ (r @@ s = r @@ t) = (s = t)"
  ⟨*proof*⟩

1.3.2 Typing rules

lemma lappT:
  assumes sllist: "s ∈ A^∞"
  and tllist: "t ∈ A^∞"
  shows "s@@t ∈ A^∞"
⟨*proof*⟩

lemma lappfin_finT: "⟦ s ∈ $A^\star$; t ∈ $A^\star$ ⟧ ⟹ s@@t ∈ $A^\star$"
  ⟨*proof*⟩

lemma lapp_fin_fin_lemma:
  assumes rsA: "r @@ s ∈ $A^\star$"
  shows "r ∈ $A^\star$"
⟨*proof*⟩

lemma lapp_fin_fin_iff [iff]: "(r @@ s ∈ $A^\star$) = (r ∈ $A^\star$ ∧ s ∈ $A^\star$)"
⟨*proof*⟩

lemma lapp_all_invT:
assumes rs: "r@@s ∈ A^∞"
  shows "r ∈ A^∞"
⟨*proof*⟩

lemma lapp_fin_infT: "⟦s ∈ $A^\star$; t ∈ A^ω⟧ ⟹ s @@ t ∈ A^ω"
  ⟨*proof*⟩

lemma app_invT:
  assumes "r ∈ $A^\star$" **shows** "r @@ s ∈ A^ω ⟹ s ∈ A^ω"
⟨*proof*⟩

lemma lapp_inv2T:
  assumes rsinf: "r @@ s ∈ A^ω"

shows "r ∈ A* ∧ s ∈ A^ω ∨ r ∈ A^ω"
⟨*proof*⟩

lemma lapp_infT:
"(r @@ s ∈ A^ω) = (r ∈ A* ∧ s ∈ A^ω ∨ r ∈ A^ω)"
⟨*proof*⟩

lemma lapp_allT_iff:
"(r @@ s ∈ A^∞) = (r ∈ A* ∧ s ∈ A^∞ ∨ r ∈ A^ω)"
(**is** "?L = ?R")
⟨*proof*⟩

1.4 Length, indexing, prefixes, and suffixes of llists

primrec ll2f :: "'a llist ⇒ nat ⇒ 'a option" (**infix** <!!> 100)
where
"l!!0 = (case l of LNil ⇒ None | x ## xs ⇒ Some x)"
| "l!!(Suc i) = (case l of LNil ⇒ None | x ## xs ⇒ xs!!i)"

primrec ltake :: "'a llist ⇒ nat ⇒ 'a llist" (**infixl** <↓> 110)
where
"l ↓ 0 = LNil"
| "l ↓ Suc i = (case l of LNil ⇒ LNil | x ## xs ⇒ x ## ltake xs i)"

primrec ldrop :: "'a llist ⇒ nat ⇒ 'a llist" (**infixl** <↑> 110)
where
"l ↑ 0 = l"
| "l ↑ Suc i = (case l of LNil ⇒ LNil | x ## xs ⇒ ldrop xs i)"

definition lset :: "'a llist ⇒ 'a set"
where "lset l ≡ ran (ll2f l)"

definition llength :: "'a llist ⇒ nat"
where "llength ≡ finlsts_rec 0 (λ a r n. Suc n)"

definition llast :: "'a llist ⇒ 'a"
where "llast ≡ finlsts_rec undefined (λ x xs l. if xs = LNil then x else l)"

definition lbutlast :: "'a llist ⇒ 'a llist"
where "lbutlast ≡ finlsts_rec LNil (λ x xs l. if xs = LNil then LNil else x##l)"

definition lrev :: "'a llist ⇒ 'a llist"
where "lrev ≡ finlsts_rec LNil (λ x xs l. l @@ x ## LNil)"

lemmas llength_LNil = llength_def [THEN finlsts_rec_LNil_def]
and llength_LCons = llength_def [THEN finlsts_rec_LCons_def]
lemmas llength_simps [simp] = llength_LNil llength_LCons

lemmas llast_LNil = llast_def [THEN finlsts_rec_LNil_def]
and llast_LCons = llast_def [THEN finlsts_rec_LCons_def]
lemmas llast_simps [simp] = llast_LNil llast_LCons

lemmas lbutlast_LNil = lbutlast_def [THEN finlsts_rec_LNil_def]

```
  and lbutlast_LCons = lbutlast_def [THEN finlsts_rec_LCons_def]
lemmas lbutlast_simps [simp] = lbutlast_LNil lbutlast_LCons

lemmas lrev_LNil = lrev_def [THEN finlsts_rec_LNil_def]
  and lrev_LCons = lrev_def [THEN finlsts_rec_LCons_def]
lemmas lrev_simps [simp] = lrev_LNil lrev_LCons

lemma lrevT [simp, intro!]:
  "xs ∈ A^* ⟹ lrev xs ∈ A^*"
  ⟨proof⟩

lemma lrev_lappend [simp]:
  assumes fin: "xs ∈ UNIV^*" "ys ∈ UNIV^*"
  shows "lrev (xs @@ ys) = (lrev ys) @@ (lrev xs)"
  ⟨proof⟩

lemma lrev_lrev_ident [simp]:
  assumes fin: "xs ∈ UNIV^*"
  shows "lrev (lrev xs) = xs"
  ⟨proof⟩

lemma lrev_is_LNil_conv [iff]:
  "xs ∈ UNIV^* ⟹ (lrev xs = LNil) = (xs = LNil)"
  ⟨proof⟩

lemma LNil_is_lrev_conv [iff]:
"xs ∈ UNIV^* ⟹ (LNil = lrev xs) = (xs = LNil)"
⟨proof⟩

lemma lrev_is_lrev_conv [iff]:
assumes fin: "xs ∈ UNIV^*" "ys ∈ UNIV^*"
  shows "(lrev xs = lrev ys) = (xs = ys)"
  (is "?L = ?R")
⟨proof⟩

lemma lrev_induct [case_names LNil snocl, consumes 1]:
  assumes fin: "xs ∈ A^*"
  and init: "P LNil"
  and step: "⋀x xs. ⟦ xs ∈ A^*; P xs; x ∈ A ⟧ ⟹ P (xs @@ x##LNil)"
  shows "P xs"
⟨proof⟩

lemma finlsts_rev_cases:
  assumes tfin: "t ∈ A^*"
  obtains (LNil) "t = LNil"
    |    (snocl) a l where "l ∈ A^*" "a ∈ A" "t = l @@ a ## LNil"
  ⟨proof⟩

lemma ll2f_LNil [simp]: "LNil!!x = None"
  ⟨proof⟩

lemma None_lfinite: "t!!i = None ⟹ t ∈ UNIV^*"
⟨proof⟩
```

lemma infinite_Some: "t ∈ A^ω ⟹ ∃a. t!!i = Some a"
⟨*proof*⟩

lemmas infinite_idx_SomeE = exE [OF infinite_Some]

lemma Least_True [simp]:
"(LEAST (n::nat). True) = 0"
⟨*proof*⟩

lemma ll2f_llength [simp]: "r ∈ $A^\star$ ⟹ r!!(llength r) = None"
⟨*proof*⟩

lemma llength_least_None:
assumes rA: "r ∈ $A^\star$"
shows "llength r = (LEAST i. r!!i = None)"
⟨*proof*⟩

lemma ll2f_lem1:
"t !! (Suc i) = Some x ⟹ ∃ y. t !! i = Some y"
⟨*proof*⟩

lemmas ll2f_Suc_Some = ll2f_lem1 [THEN exE]

lemma ll2f_None_Suc: "t !! i = None ⟹ t !! Suc i = None"
⟨*proof*⟩

lemma ll2f_None_le:
"⟦ t!!j = None; j ≤ i ⟧ ⟹ t!!i = None"
⟨*proof*⟩

lemma ll2f_Some_le:
assumes jlei: "j ≤ i"
and tisome: "t !! i = Some x"
and H: "⋀ y. t !! j = Some y ⟹ Q"
shows "Q"
⟨*proof*⟩

lemma ltake_LNil [simp]: "LNil ↓ i = LNil"
⟨*proof*⟩

lemma ltake_LCons_Suc: "(a ## l) ↓ (Suc i) = a ## l ↓ i"
⟨*proof*⟩

lemma take_fin [iff]: "t ∈ A^∞ ⟹ t↓i ∈ $A^\star$"
⟨*proof*⟩

lemma ltake_fin [iff]:
"r ↓ i ∈ $UNIV^\star$"
⟨*proof*⟩

lemma llength_take [simp]: "t ∈ A^ω ⟹ llength (t↓i) = i"
⟨*proof*⟩

lemma ltake_ldrop_id: "(x ↓ i) @@ (x ↑ i) = x"
⟨*proof*⟩

lemma ltake_ldrop:
"(xs ↑ m) ↓ n =(xs ↓ (n + m)) ↑ m"
⟨*proof*⟩

lemma ldrop_LNil [simp]: "LNil ↑ i = LNil"
⟨*proof*⟩

lemma ldrop_add: "t ↑ (i + k) = t ↑ i ↑ k"
⟨*proof*⟩

lemma ldrop_fun: "t ↑ i !! j = t!!(i + j)"
⟨*proof*⟩

lemma ldropT[simp]: "t ∈ A^∞ ⟹ t ↑ i ∈ A^∞"
⟨*proof*⟩

lemma ldrop_finT[simp]: "t ∈ $A^\star$ ⟹ t ↑ i ∈ $A^\star$"
⟨*proof*⟩

lemma ldrop_infT[simp]: "t ∈ A^ω ⟹ t ↑ i ∈ A^ω"
⟨*proof*⟩

lemma lapp_suff_llength: "r ∈ $A^\star$ ⟹ (r@@s) ↑ llength r = s"
⟨*proof*⟩

lemma ltake_lappend_llength [simp]:
"r ∈ $A^\star$ ⟹ (r @@ s) ↓ llength r = r"
⟨*proof*⟩

lemma ldrop_LNil_less:
"⟦j ≤ i; t ↑ j = LNil⟧ ⟹ t ↑ i = LNil"
⟨*proof*⟩

lemma ldrop_inf_iffT [iff]: "(t ↑ i ∈ $UNIV^\omega$) = (t ∈ $UNIV^\omega$)"
⟨*proof*⟩

lemma ldrop_fin_iffT [iff]: "(t ↑ i ∈ $UNIV^\star$) = (t ∈ $UNIV^\star$)"
⟨*proof*⟩

lemma drop_nonLNil: "t↑i ≠ LNil ⟹ t ≠ LNil"
⟨*proof*⟩

lemma llength_drop_take:
"t↑i ≠ LNil ⟹ llength (t↓i) = i"
⟨*proof*⟩

lemma fps_induct [case_names LNil LCons, induct set: fpslsts, consumes 1]:
assumes fps: "l ∈ $A^\clubsuit$"
and init: "⋀a. a ∈ A ⟹ P (a##LNil)"

and step: "⋀a l. ⟦ l ∈ A♣; P l; a ∈ A ⟧ ⟹ P (a ## l)"
shows "P l"
⟨*proof*⟩

lemma lbutlast_lapp_llast:
assumes "l ∈ A♣"
shows "l = lbutlast l @@ (llast l ## LNil)"
⟨*proof*⟩

lemma llast_snoc [simp]:
assumes fin: "xs ∈ A⋆"
shows "llast (xs @@ x ## LNil) = x"
⟨*proof*⟩

lemma lbutlast_snoc [simp]:
assumes fin: "xs ∈ A⋆"
shows "lbutlast (xs @@ x ## LNil) = xs"
⟨*proof*⟩

lemma llast_lappend [simp]:
"⟦ x ∈ UNIV⋆; y ∈ UNIV⋆ ⟧ ⟹ llast (x @@ a ## y) = llast (a ## y)"
⟨*proof*⟩

lemma llast_llength:
assumes tfin: "t ∈ UNIV⋆"
shows "t ≠ LNil ⟹ t !! (llength t - (Suc 0)) = Some (llast t)"
⟨*proof*⟩

1.5 The constant llist

definition lconst :: "'a ⇒ 'a llist" **where**
"lconst a ≡ iterates (λx. x) a"

lemma lconst_unfold: "lconst a = a ## lconst a"
⟨*proof*⟩

lemma lconst_LNil [iff]: "lconst a ≠ LNil"
⟨*proof*⟩

lemma lconstT:
assumes aA: "a ∈ A"
shows "lconst a ∈ Aω"
⟨*proof*⟩

1.6 The prefix order of llists

instantiation llist :: (type) order
begin

definition
llist_le_def: "(s :: 'a llist) ≤ t ⟷ (∃d. t = s @@ d)"

definition

llist_less_def: "(s :: 'a llist) < t ⟷ (s ≤ t ∧ s ≠ t)"

lemma not_LCons_le_LNil [iff]:
"¬ (a##l) ≤ LNil"
⟨*proof*⟩

lemma LNil_le [iff]:"LNil ≤ s"
⟨*proof*⟩

lemma le_LNil [iff]: "(s ≤ LNil) = (s = LNil)"
⟨*proof*⟩

lemma llist_inf_le:
"s ∈ A^ω ⟹ (s≤t) = (s=t)"
⟨*proof*⟩

lemma le_LCons [iff]: "(x ## xs ≤ y ## ys) = (x = y ∧ xs ≤ ys)"
⟨*proof*⟩

lemma llist_le_refl [iff]:
"(s:: 'a llist) ≤ s"
⟨*proof*⟩

lemma llist_le_trans [trans]:
fixes r:: "'a llist"
shows "r ≤ s ⟹ s ≤ t ⟹ r ≤ t"
⟨*proof*⟩

lemma llist_le_anti_sym:
fixes s:: "'a llist"
assumes st: "s ≤ t"
and ts: "t ≤ s"
shows "s = t"
⟨*proof*⟩

lemma llist_less_le_not_le:
fixes s :: "'a llist"
shows "(s < t) = (s ≤ t ∧ ¬ t ≤ s)"
⟨*proof*⟩

instance
⟨*proof*⟩

end

1.6.1 Typing rules

lemma llist_le_finT [simp]:
"r≤s ⟹ s ∈ $A^\star$ ⟹ r ∈ $A^\star$"
⟨*proof*⟩

lemma llist_less_finT [iff]:
"r<s ⟹ s ∈ $A^\star$ ⟹ r ∈ $A^\star$"

⟨*proof*⟩

1.6.2 More simplification rules

lemma LNil_less_LCons [iff]: "LNil < a ## t"
⟨*proof*⟩

lemma not_less_LNil [iff]:
"¬ r < LNil"
⟨*proof*⟩

lemma less_LCons [iff]:
" (a ## r < b ## t) = (a = b ∧ r < t)"
⟨*proof*⟩

lemma llength_mono [iff]:
assumes"r ∈ A*"
shows "s<r ⟹ llength s < llength r"
⟨*proof*⟩

lemma le_lappend [iff]: "r ≤ r @@ s"
⟨*proof*⟩

lemma take_inf_less:
"t ∈ UNIV^ω ⟹ t ↓ i < t"
⟨*proof*⟩

lemma lapp_take_less:
assumes iless: "i < llength r"
shows "(r @@ s) ↓ i < r"
⟨*proof*⟩

1.6.3 Finite prefixes and infinite suffixes

definition finpref :: "'a set ⇒ 'a llist ⇒ 'a llist set"
where "finpref A s ≡ {r. r ∈ A* ∧ r ≤ s}"

definition suff :: "'a set ⇒ 'a llist ⇒ 'a llist set"
where "suff A s ≡ {r. r ∈ A^∞ ∧ s ≤ r}"

definition infsuff :: "'a set ⇒ 'a llist ⇒ 'a llist set"
where "infsuff A s ≡ {r. r ∈ A^ω ∧ s ≤ r}"

definition prefix_closed :: "'a llist set ⇒ bool"
where "prefix_closed A ≡ ∀ t ∈ A. ∀ s ≤ t. s ∈ A"

definition pprefix_closed :: "'a llist set ⇒ bool"
where "pprefix_closed A ≡ ∀ t ∈ A. ∀ s. s ≤ t ∧ s ≠ LNil ⟶ s ∈ A"

definition suffix_closed :: "'a llist set ⇒ bool"
where "suffix_closed A ≡ ∀ t ∈ A. ∀ s. t ≤ s ⟶ s ∈ A"

lemma finpref_LNil [simp]:

"finpref A LNil = {LNil}"
⟨*proof*⟩

lemma finpref_fin: "x ∈ finpref A s ⟹ x ∈ A⋆"
⟨*proof*⟩

lemma finpref_mono2: "s ≤ t ⟹ finpref A s ⊆ finpref A t"
⟨*proof*⟩

lemma suff_LNil [simp]:
"suff A LNil = A∞"
⟨*proof*⟩

lemma suff_all: "x ∈ suff A s ⟹ x ∈ A∞"
⟨*proof*⟩

lemma suff_mono2: "s ≤ t ⟹ suff A t ⊆ suff A s"
⟨*proof*⟩

lemma suff_appE:
assumes rA: "r ∈ A⋆"
and tsuff: "t ∈ suff A r"
obtains s **where** "s ∈ A∞" "t = r@@s"
⟨*proof*⟩

lemma LNil_suff [iff]: "(LNil ∈ suff A s) = (s = LNil)"
⟨*proof*⟩

lemma finpref_suff [dest]:
"⟦ r ∈ finpref A t; t∈A∞ ⟧ ⟹ t ∈ suff A r"
⟨*proof*⟩

lemma suff_finpref:
"⟦ t ∈ suff A r; r∈A⋆ ⟧ ⟹ r ∈ finpref A t"
⟨*proof*⟩

lemma suff_finpref_iff:
"⟦ r∈A⋆; t∈A∞ ⟧ ⟹ (r ∈ finpref A t) = (t ∈ suff A r)"
⟨*proof*⟩

lemma infsuff_LNil [simp]:
"infsuff A LNil = Aω"
⟨*proof*⟩

lemma infsuff_inf: "x ∈ infsuff A s ⟹ x ∈ Aω"
⟨*proof*⟩

lemma infsuff_mono2: "s ≤ t ⟹ infsuff A t ⊆ infsuff A s"
⟨*proof*⟩

lemma infsuff_appE:
assumes rA: "r ∈ A⋆"
and tinfsuff: "t ∈ infsuff A r"

obtains s **where** "s ∈ A^ω" "t = r@@s"
⟨*proof*⟩

lemma finpref_infsuff [dest]:
"⟦ r ∈ finpref A t; t∈A^ω ⟧ ⟹ t ∈ infsuff A r"
⟨*proof*⟩

lemma infsuff_finpref:
"⟦ t ∈ infsuff A r; r∈$A^\star$ ⟧ ⟹ r ∈ finpref A t"
⟨*proof*⟩

lemma infsuff_finpref_iff [iff]:
"⟦ r∈$A^\star$; t∈A^ω ⟧ ⟹ (t ∈ finpref A r) = (r ∈ infsuff A t)"
⟨*proof*⟩

lemma prefix_lemma:
assumes xinf: "x ∈ A^ω"
and yinf: "y ∈ A^ω"
and R: "⋀ s. ⟦ s ∈ $A^\star$; s ≤ x⟧ ⟹ s ≤ y"
shows "x = y"
⟨*proof*⟩

lemma inf_neqE:
"⟦ x ∈ A^ω; y ∈ A^ω; x ≠ y;
⋀s. ⟦ s∈$A^\star$; s ≤ x; ¬ s ≤ y⟧ ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma pref_locally_linear:
fixes s::"'a llist"
assumes sx: "s ≤ x"
and tx: "t ≤ x"
shows "s ≤ t ∨ t ≤ s"
⟨*proof*⟩

definition pfinpref :: "'a set ⇒ 'a llist ⇒ 'a llist set"
where "pfinpref A s ≡ finpref A s - {LNil}"

lemma pfinpref_iff [iff]:
"(x ∈ pfinpref A s) = (x ∈ finpref A s ∧ x ≠ LNil)"
⟨*proof*⟩

1.7 Safety and Liveness

definition infsafety :: "'a set ⇒ 'a llist set ⇒ bool"
where "infsafety A P ≡ ∀ t ∈ A^ω. (∀ r ∈ finpref A t. ∃ s ∈ A^ω. r @@ s ∈ P) ⟶ t ∈ P"

definition infliveness :: "'a set ⇒ 'a llist set ⇒ bool"
where "infliveness A P ≡ ∀ t ∈ $A^\star$. ∃ s ∈ A^ω. t @@ s ∈ P"

definition possafety :: "'a set ⇒ 'a llist set ⇒ bool"
where "possafety A P ≡ ∀ t ∈ $A^\spadesuit$. (∀ r ∈ pfinpref A t. ∃ s ∈ A^∞. r @@ s ∈ P) ⟶ t ∈ P"

definition posliveness :: "'a set ⇒ 'a llist set ⇒ bool"
where "posliveness A P ≡ ∀ t ∈ A♣. ∃ s ∈ A∞. t @@ s ∈ P"

definition safety :: "'a set ⇒ 'a llist set ⇒ bool"
where "safety A P ≡ ∀ t ∈ A∞. (∀ r ∈ finpref A t. ∃ s ∈ A∞. r @@ s ∈ P) ⟶ t ∈ P"

definition liveness :: "'a set ⇒ 'a llist set ⇒ bool"
where "liveness A P ≡ ∀ t ∈ A*. ∃ s ∈ A∞. t @@ s ∈ P"

lemma safetyI:
"(⋀t. ⟦t ∈ A∞; ∀ r ∈ finpref A t. ∃ s ∈ A∞. r @@ s ∈ P⟧ ⟹ t ∈ P)
⟹ safety A P"
⟨*proof*⟩

lemma safetyD:
"⟦ safety A P; t ∈ A∞;
⋀r. r ∈ finpref A t ⟹ ∃ s ∈ A∞. r @@ s ∈ P
⟧ ⟹ t ∈ P"
⟨*proof*⟩

lemma safetyE:
"⟦ safety A P;
∀ t ∈ A∞. (∀ r ∈ finpref A t. ∃ s ∈ A∞. r @@ s ∈ P) ⟶ t ∈ P ⟹ R
⟧ ⟹ R"
⟨*proof*⟩

lemma safety_prefix_closed:
"safety UNIV P ⟹ prefix_closed P"
⟨*proof*⟩

lemma livenessI:
"(⋀s. s∈ A* ⟹ ∃ t ∈ A∞. s @@ t ∈ P) ⟹ liveness A P"
⟨*proof*⟩

lemma livenessE:
"⟦ liveness A P; ⋀t. ⟦ t ∈ A∞; s @@ t ∈ P⟧ ⟹ R; s ∉ A* ⟹ R⟧ ⟹ R"
⟨*proof*⟩

lemma possafetyI:
"(⋀t. ⟦t ∈ A♠; ∀ r ∈ pfinpref A t. ∃ s ∈ A∞. r @@ s ∈ P⟧ ⟹ t ∈ P)
⟹ possafety A P"
⟨*proof*⟩

lemma possafetyD:
"⟦ possafety A P; t ∈ A♠;
⋀r. r ∈ pfinpref A t ⟹ ∃ s ∈ A∞. r @@ s ∈ P
⟧ ⟹ t ∈ P"
⟨*proof*⟩

lemma possafetyE:
"⟦ possafety A P;
∀ t ∈ A♠. (∀ r ∈ pfinpref A t. ∃ s ∈ A∞. r @@ s ∈ P) ⟶ t ∈ P ⟹ R

⟧ ⟹ R"
⟨*proof*⟩

lemma possafety_pprefix_closed:
assumes psafety: "possafety UNIV P"
shows "pprefix_closed P"
⟨*proof*⟩

lemma poslivenessI:
"(⋀s. s∈ $A^{\clubsuit}$ ⟹ ∃ t ∈ A^{∞}. s @@ t ∈ P) ⟹ posliveness A P"
⟨*proof*⟩

lemma poslivenessE:
"⟦ posliveness A P; ⋀t. ⟦ t ∈ A^{∞}; s @@ t ∈ P⟧ ⟹ R; s ∉ $A^{\clubsuit}$ ⟹ R⟧ ⟹ R"
⟨*proof*⟩

end

References

[1] B. Alpern and F. B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):181–185, Oct. 1985.