More on Lazy Lists

Stefan Friedrich

September 13, 2023

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)

translations
  "case p of XCONST LNil ⇒ a | x ## l ⇒ b" ⇌ "CONST case_llist a (λx l. b) p"
  "case p of XCONST LNil :: 'a ⇒ a | x ## l ⇒ b" ⇀ "CONST case_llist a (λx l. b) p"


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 -
  have "s ∈ UNIV" by blast
  thus ?thesis
  proof coinduct
    case (alllsts z)
    thus ?case by(cases z) auto
  qed
qed

lemma alllsts_empty [simp]: "{}∞ = {LNil}"
```

```
  by (auto elim: alllsts.cases)

lemma alllsts_mono:
  assumes asubb: "A ⊆ B"
  shows "A^∞ ⊆ B^∞"
proof
  fix x assume "x ∈ A^∞"
  thus "x ∈ B^∞"
  proof coinduct
    case (alllsts z)
    thus ?case using asubb by(cases z) auto
  qed
qed

lemmas alllstsp_mono [mono] = alllsts_mono [to_pred pred_subset_eq]

lemma LConsE [iff]: "x##xs ∈ A^∞ = (x∈A ∧ xs ∈ A^∞)"
  by (auto elim: alllsts.cases)
```

1.2.2 Facts about non-empty (positive) llists

```
lemma poslsts_iff [iff]:
  "(s ∈ A♠) = (s ∈ A^∞ ∧ s ≠ LNil)"
  by (simp add: poslsts_def)

lemma poslsts_UNIV [iff]:
  "s ∈ UNIV♠ = (s ≠ LNil)"
  by auto

lemma poslsts_empty [simp]: "{}♠ = {}"
  by auto

lemma poslsts_mono:
  "A ⊆ B ⟹ A♠ ⊆ B♠"
  by (auto dest: alllsts_mono)
```

1.2.3 Facts about finite llists

```
lemma finlsts_empty [simp]: "{}* = {LNil}"
  by (auto elim: finlsts.cases)

lemma finsubsetall: "x ∈ A* ⟹ x ∈ A^∞"
  by (induct rule: finlsts.induct) auto

lemma finlsts_mono:
"A⊆B ⟹ A* ⊆ B*"
  by (auto, erule finlsts.induct) auto

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"
  using xA by (induct "x") (auto intro: lnil lcons)

lemma finite_lemma:
  assumes "x ∈ A⋆"
  shows "x ∈ B∞ ⟹ x ∈ B⋆"
using assms
proof (induct)
  case LNil_fin thus ?case by auto
next
  case (LCons_fin a l)
  thus ?case using LCons_fin by (cases "a##l") auto
qed

lemma fin_finite [dest]:
assumes "r ∈ A⋆" "r ∉ UNIV⋆"
  shows "False"
proof-
  have "A ⊆ UNIV" by auto
  hence "A⋆ ⊆ UNIV⋆" by (rule finlsts_mono)
  thus ?thesis using assms by auto
qed

lemma finT_simp [simp]:
  "r ∈ A⋆ ⟹ r∈UNIV⋆"
  by auto
```

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"
  by (auto simp: finlsts_pred_def)

lemma wf_finlsts_pred: "wf finlsts_pred"
proof (rule wfI [of _ "UNIV⋆"])
  show "finlsts_pred ⊆ UNIV⋆ × UNIV⋆"
    by (auto simp: finlsts_pred_def elim: finlsts.cases)
next
  fix x::"'a llist" and P::"'a llist ⇒ bool"
  assume xfin: "x ∈ UNIV⋆" and H [unfolded finlsts_pred_def]:
    "(∀x. (∀y. (y, x) ∈ finlsts_pred ⟶ P y) ⟶ P x)"
  from  xfin show "P x"
  proof(induct x)
```

```
    case LNil_fin with H show ?case by blast
  next
    case (LCons_fin a l) with H show ?case by blast
  qed
qed

lemma finlsts_rec_LNil: "finlsts_rec c d LNil = c"
  by (auto simp: wf_finlsts_pred finlsts_rec_def wfrec)

lemma finlsts_rec_LCons:
 "r ∈ A⋆ ⟹ finlsts_rec c d (a ## r) = d a r (finlsts_rec c d r)"
  by (auto simp: wf_finlsts_pred finlsts_rec_def wfrec cut_def intro: finlsts_predI)

lemma finlsts_rec_LNil_def:
  "f ≡ finlsts_rec c d ⟹ f LNil = c"
  by (auto simp: finlsts_rec_LNil)

lemma finlsts_rec_LCons_def:
  "⟦ f ≡ finlsts_rec c d; r ∈ A⋆ ⟧ ⟹ f (a ## r) = d a r (f r)"
  by (auto simp: finlsts_rec_LCons)
```

1.2.5 Facts about non-empty (positive) finite llists

```
lemma fpslsts_iff [iff]:
  "(s ∈ A♣) = (s ∈ A⋆ ∧ s ≠ LNil)"
  by (auto simp: fpslsts_def)

lemma fpslsts_empty [simp]: "{}♣ = {}"
  by auto

lemma fpslsts_mono:
  "A ⊆ B ⟹ A♣ ⊆ B♣"
  by (auto dest: finlsts_mono)

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-
  from rfps have "r ∈ A⋆" and "r ≠ LNil" by auto
  thus ?thesis
    by (cases r, simp) (blast intro!: H)
qed
```

1.2.6 Facts about infinite llists

```
lemma inflstsI [intro]:
  "⟦ x ∈ A∞; x ∈ UNIV⋆ ⟹ False ⟧ ⟹ x ∈ Aω"
unfolding inflsts_def by clarsimp

lemma inflstsE [elim]:
  "⟦ x ∈ Aω; ⟦ x ∈ A∞; x ∉ UNIV⋆ ⟧ ⟹ R ⟧ ⟹ R"
  by (unfold inflsts_def) auto
```

```
lemma inflsts_empty [simp]: "{}^ω = {}"
  by auto

lemma infsubsetall: "x ∈ A^ω ⟹ x ∈ A^∞"
  by (auto intro: finite_lemma finsubsetall)

lemma inflsts_mono:
  "A ⊆ B ⟹ A^ω ⊆ B^ω"
  by (blast dest: alllsts_mono infsubsetall)

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 -
  from sinf have "s ∈ A^∞" "s ∉ UNIV*"
    by auto
  then obtain a l where "l ∈ A^ω" and "a∈A" and "s = a ## l"
    by (cases "s") auto
  thus ?thesis by (rule R)
qed

lemma inflstsI2: "⟦a ∈ A; t ∈ A^ω⟧ ⟹ a ## t ∈ A^ω"
  by  (auto elim: finlsts.cases)

lemma infT_simp [simp]:
  "r ∈ A^ω ⟹ r∈UNIV^ω"
  by auto

lemma  alllstsE [consumes 1, case_names finite infinite]:
  "⟦ x∈A^∞; x ∈ A* ⟹ P; x ∈ A^ω ⟹ P ⟧ ⟹ P"
  by (auto intro: finite_lemma simp: inflsts_def)


lemma fin_inf_cases [case_names finite infinite]:
  "⟦ r∈UNIV* ⟹ P; r ∈ UNIV^ω ⟹ P ⟧ ⟹ P"
  by auto

lemma  fin_Int_inf: "A* ∩ A^ω = {}"
  and   fin_Un_inf: "A* ∪ A^ω = A^∞"
  by (auto intro: finite_lemma finsubsetall)

lemma notfin_inf [iff]: "(x ∉ UNIV*) = (x ∈ UNIV^ω)"
  by auto

lemma notinf_fin [iff]: "(x ∉ UNIV^ω) = (x ∈ UNIV*)"
  by auto
```

1.3 Lappend

1.3.1 Simplification

```
lemma lapp_inf [simp]:
  assumes "s ∈ A^ω"
  shows "s @@ t = s"
using assms
by(coinduction arbitrary: s)(auto elim: inflsts_cases)

lemma LNil_is_lappend_conv [iff]:
"(LNil = s @@ t) = (s = LNil ∧ t = LNil)"
  by (cases "s") auto

lemma lappend_is_LNil_conv [iff]:
  "(s @@ t = LNil) = (s = LNil ∧ t = LNil)"
  by (cases "s") auto

lemma same_lappend_eq [iff]:
 "r ∈ A* ⟹ (r @@ s = r @@ t) = (s = t)"
  by (erule finlsts.induct) simp+
```

1.3.2 Typing rules

```
lemma lappT:
  assumes sllist: "s ∈ A^∞"
  and tllist: "t ∈ A^∞"
  shows "s@@t ∈ A^∞"
proof -
  from assms have "lappend s t ∈ (⋃u∈A^∞. ⋃v∈A^∞. {lappend u v})" by fast
  thus ?thesis
  proof coinduct
    case (alllsts z)
    then obtain u v where ullist: "u∈A^∞" and vllist: "v∈A^∞"
      and zapp: "z=u @@ v" by auto
    thus ?case by (cases "u") (auto elim: alllsts.cases)
  qed
qed

lemma lappfin_finT: "⟦ s ∈ A*; t ∈ A* ⟧ ⟹ s@@t ∈ A*"
  by (induct rule: finlsts.induct) auto

lemma lapp_fin_fin_lemma:
  assumes rsA: "r @@ s ∈ A*"
  shows "r ∈ A*"
using rsA
proof(induct l≡"r@@s" arbitrary: r)
  case LNil_fin thus ?case by auto
next
  case (LCons_fin a l')
  show ?case
  proof (cases "r")
    case LNil thus ?thesis by auto
  next
```

```
    case (LCons x xs) with ‹a##l' = r @@ s›
    have "a = x" and "l' = xs @@ s" by auto
    with LCons_fin LCons show ?thesis by auto
  qed
qed

lemma lapp_fin_fin_iff [iff]: "(r @@ s ∈ A⋆) = (r ∈ A⋆ ∧ s ∈ A⋆)"
proof (auto intro: lappfin_finT lapp_fin_fin_lemma)
  assume rsA: "r @@ s ∈ A⋆"
  hence "r ∈ A⋆" by (rule lapp_fin_fin_lemma)
  hence "r @@ s ∈ A⋆ ⟶ s ∈ A⋆"
    by (induct "r", simp) (auto elim: finlsts.cases)
  with rsA show "s ∈ A⋆" by auto
qed

lemma lapp_all_invT:
assumes rs: "r@@s ∈ A∞"
  shows "r ∈ A∞"
proof (cases "r ∈ UNIV⋆")
  case False
  with rs show ?thesis by simp
next
  case True
  thus ?thesis using rs
    by (induct "r") auto
qed

lemma lapp_fin_infT: "⟦s ∈ A⋆; t ∈ Aω⟧ ⟹ s @@ t ∈ Aω"
  by (induct rule: finlsts.induct)
     (auto intro: inflstsI2)

lemma app_invT:
  assumes "r ∈ A⋆" shows "r @@ s ∈ Aω ⟹ s ∈ Aω"
using assms
proof (induct arbitrary: s)
  case LNil_fin thus ?case by simp
next
  case (LCons_fin a l)
  from ‹(a ## l) @@ s ∈ Aω›
  have "a ## (l @@ s) ∈ Aω" by simp
  hence "l @@ s ∈ Aω" by (auto elim: inflsts_cases)
  with LCons_fin show "s ∈ Aω" by blast
qed

lemma lapp_inv2T:
  assumes rsinf: "r @@ s ∈ Aω"
  shows "r ∈ A⋆ ∧ s ∈ Aω ∨ r ∈ Aω"
proof (rule disjCI)
  assume rnotinf: "r ∉ Aω"
  moreover from rsinf have rsall: "r@@s ∈ A∞"
    by auto
  hence "r ∈ A∞" by (rule lapp_all_invT)
  hence "r ∈ A⋆" using rnotinf by (auto elim: alllstsE)
```

```
  ultimately show "r ∈ A⋆ ∧ s ∈ Aω" using rsinf
    by (auto  intro: app_invT)
qed

lemma lapp_infT:
  "(r @@ s ∈ Aω) = (r ∈ A⋆ ∧ s ∈ Aω ∨ r ∈ Aω)"
  by (auto dest: lapp_inv2T intro: lapp_fin_infT)

lemma lapp_allT_iff:
  "(r @@ s ∈ A∞) = (r ∈ A⋆ ∧ s ∈ A∞ ∨ r ∈ Aω)"
  (is "?L = ?R")
proof
  assume ?L thus ?R by (cases rule: alllstsE) (auto simp: lapp_infT intro: finsubsetall)
next
  assume ?R thus ?L by (auto dest: finsubsetall intro: lappT)
qed
```

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*"
  by (induct rule: finlsts.induct) auto

lemma lrev_lappend [simp]:
  assumes fin: "xs ∈ UNIV*" "ys ∈ UNIV*"
  shows "lrev (xs @@ ys) = (lrev ys) @@ (lrev xs)"
  using fin
  by induct (auto simp: lrev_LCons [of _ UNIV] lappend_assoc)

lemma lrev_lrev_ident [simp]:
  assumes fin: "xs ∈ UNIV*"
  shows "lrev (lrev xs) = xs"
  using fin
proof (induct)
  case (LCons_fin a l)
  have "a ## LNil ∈ UNIV*" by auto
  thus ?case using LCons_fin
    by auto
qed simp

lemma lrev_is_LNil_conv [iff]:
  "xs ∈ UNIV* ⟹ (lrev xs = LNil) = (xs = LNil)"
  by (induct rule: finlsts.induct) auto

lemma LNil_is_lrev_conv [iff]:
"xs ∈ UNIV* ⟹ (LNil = lrev xs) = (xs = LNil)"
by (induct rule: finlsts.induct) auto

lemma lrev_is_lrev_conv [iff]:
assumes fin: "xs ∈ UNIV*" "ys ∈ UNIV*"
  shows "(lrev xs = lrev ys) = (xs = ys)"
  (is "?L = ?R")
proof
  assume L: ?L
  hence "lrev (lrev xs) = lrev (lrev ys)" by simp
  thus ?R using fin by simp
qed simp

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-
  define l where "l = lrev xs"
  with fin have "l ∈ A⋆" by simp
  hence "P (lrev l)"
  proof (induct l)
    case LNil_fin with init show ?case by simp
  next
    case (LCons_fin a l) thus ?case by (auto intro: step)
  qed
  thus ?thesis using fin l_def by simp
qed

lemma finlsts_rev_cases:
  assumes tfin: "t ∈ A⋆"
  obtains (LNil) "t = LNil"
    |    (snocl) a l where "l ∈ A⋆" "a ∈ A" "t = l @@ a ## LNil"
  using assms
  by (induct rule: lrev_induct) auto

lemma ll2f_LNil [simp]: "LNil!!x = None"
  by (cases "x") auto

lemma None_lfinite: "t!!i = None ⟹ t ∈ UNIV⋆"
proof (induct "i" arbitrary: t)
  case 0 thus ?case
    by(cases t) auto
next
  case (Suc n)
  show ?case
  proof(cases t)
    case LNil thus ?thesis by auto
  next
    case (LCons x l')
    with ‹l' !! n = None ⟹ l' ∈ UNIV⋆› ‹t !! Suc n = None›
    show ?thesis by auto
  qed
qed

lemma infinite_Some: "t ∈ A^ω ⟹ ∃a. t!!i = Some a"
  by (rule ccontr) (auto dest: None_lfinite)

lemmas infinite_idx_SomeE = exE [OF infinite_Some]

lemma Least_True [simp]:
  "(LEAST (n::nat). True) = 0"
  by (auto simp: Least_def)

lemma  ll2f_llength [simp]: "r ∈ A⋆ ⟹ r!!(llength r) = None"
  by (erule finlsts.induct) auto
```

```
lemma llength_least_None:
  assumes rA: "r ∈ A*"
  shows "llength r = (LEAST i. r!!i = None)"
using rA
proof induct
  case LNil_fin thus ?case by simp
next
  case (LCons_fin a l)
  hence "(LEAST i. (a ## l) !! i = None) = llength (a ## l)"
    by (auto intro!: ll2f_llength Least_Suc2)
  thus ?case by rule
qed

lemma ll2f_lem1:
 "t !! (Suc i) = Some x ⟹ ∃ y. t !! i = Some y"
proof (induct i arbitrary: x t)
  case 0 thus ?case by (auto split: llist.splits)
next
  case (Suc k) thus ?case
    by (cases t) auto
qed

lemmas ll2f_Suc_Some = ll2f_lem1 [THEN exE]

lemma ll2f_None_Suc: "t !! i = None ⟹ t !! Suc i = None"
proof (induct i arbitrary: t)
  case 0 thus ?case by (auto split: llist.split)
next
  case (Suc k) thus ?case by (cases t) auto
qed

lemma ll2f_None_le:
  "⟦ t!!j = None; j ≤ i ⟧ ⟹ t!!i = None"
proof (induct i arbitrary: t j)
  case 0 thus ?case by simp
next
  case (Suc k) thus ?case by (cases j) (auto split: llist.split)
qed

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 -
  have  "∃ y. t !! j = Some y" (is "?R")
  proof (rule ccontr)
    assume "¬ ?R"
    hence "t !! j = None" by auto
    with tisome jlei show False
      by (auto dest:  ll2f_None_le)
  qed
  thus ?thesis using H by auto
```

```
qed

lemma ltake_LNil [simp]: "LNil ↓ i = LNil"
  by (cases "i") auto

lemma ltake_LCons_Suc: "(a ## l) ↓ (Suc i) = a ## l ↓ i"
  by simp

lemma take_fin [iff]: "t ∈ A^∞ ⟹ t↓i ∈ A^*"
proof (induct i arbitrary: t)
  case 0 show ?case by auto
next
  case (Suc j) thus ?case
    by (cases "t") auto
qed

lemma ltake_fin [iff]:
  "r ↓ i ∈ UNIV^*"
  by simp

lemma llength_take [simp]: "t ∈ A^ω ⟹ llength (t↓i) = i"
proof (induct "i" arbitrary: t)
  case 0 thus ?case by simp
next
  case (Suc j)
  from ‹t ∈ A^ω› ‹⋀t. t ∈ A^ω ⟹ llength (t ↓ j) = j› show ?case
    by(cases) (auto simp: llength_LCons [of _ UNIV])
qed

lemma ltake_ldrop_id: "(x ↓ i) @@ (x ↑ i) = x"
proof (induct "i" arbitrary: x)
  case 0 thus ?case by simp
next
  case (Suc j) thus ?case
    by (cases x) auto
qed

lemma ltake_ldrop:
  "(xs ↑ m) ↓ n =(xs ↓ (n + m)) ↑ m"
proof (induct "m" arbitrary: xs)
  case 0 show ?case by simp
next
  case (Suc l) thus ?case
    by (cases "xs") auto
qed

lemma ldrop_LNil [simp]: "LNil ↑ i = LNil"
  by (cases "i") auto

lemma ldrop_add: "t ↑ (i + k) = t ↑ i ↑ k"
proof (induct "i" arbitrary: t)
  case (Suc j) thus ?case
    by (cases "t") auto
```

```
qed simp

lemma ldrop_fun: "t ↑ i !! j = t!!(i + j)"
proof (induct i arbitrary: t)
  case 0 thus ?case by simp
next
  case (Suc k) then show ?case
    by (cases "t") auto
qed

lemma ldropT[simp]: "t ∈ A^∞ ⟹ t ↑ i ∈ A^∞"
proof (induct i arbitrary: t)
  case 0 thus ?case by simp
next case (Suc j)
  thus ?case by (cases "t") auto
qed

lemma ldrop_finT[simp]: "t ∈ A^⋆ ⟹ t ↑ i ∈ A^⋆"
proof (induct i arbitrary: t)
  case 0 thus ?case by simp
next
  fix n t assume "t ∈ A^⋆" and
    "⋀t::'a llist. t ∈ A^⋆ ⟹ t ↑ n ∈ A^⋆"
  thus "t ↑ Suc n ∈ A^⋆"
    by (cases "t") auto
qed

lemma ldrop_infT[simp]: "t ∈ A^ω ⟹ t ↑ i ∈ A^ω"
proof (induct i arbitrary: t)
  case 0 thus ?case by simp
next
  case (Suc n)
  from ‹t ∈ A^ω› ‹⋀t. t ∈ A^ω ⟹ t ↑ n ∈ A^ω› show ?case
    by (cases "t") auto
qed

lemma lapp_suff_llength: "r ∈ A^⋆ ⟹ (r@@s) ↑ llength r = s"
  by (induct rule: finlsts.induct) auto

lemma ltake_lappend_llength [simp]:
  "r ∈ A^⋆ ⟹ (r @@ s) ↓ llength r = r"
  by (induct rule: finlsts.induct) auto

lemma ldrop_LNil_less:
  "⟦j ≤ i; t ↑ j = LNil⟧ ⟹ t ↑ i = LNil"
proof (induct i arbitrary: j t)
  case 0 thus ?case by auto
next case (Suc n) thus ?case
    by (cases j, simp) (cases t, simp_all)
qed

lemma ldrop_inf_iffT [iff]: "(t ↑ i ∈ UNIV^ω)  =  (t ∈ UNIV^ω)"
proof
```

```
  show "t↑i ∈ UNIV^ω ⟹ t ∈ UNIV^ω"
    by (rule ccontr) (auto dest: ldrop_finT)
qed auto

lemma ldrop_fin_ifffT [iff]: "(t ↑ i ∈ UNIV*) = (t ∈ UNIV*)"
  by auto

lemma drop_nonLNil: "t↑i ≠ LNil ⟹ t ≠ LNil"
  by (auto)

lemma llength_drop_take:
  "t↑i ≠ LNil ⟹ llength (t↓i) = i"
proof (induct i arbitrary: t)
  case 0 show ?case by simp
next
  case (Suc j) thus ?case by (cases t) (auto simp: llength_LCons [of _ UNIV])
qed

lemma fps_induct [case_names LNil LCons, induct set: fpslsts, consumes 1]:
  assumes fps: "l ∈ A♣"
  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-
  from fps have "l ∈ A*" and "l ≠ LNil" by auto
  thus ?thesis
    by (induct, simp) (cases, auto intro: init step)
qed

lemma lbutlast_lapp_llast:
assumes "l ∈ A♣"
  shows "l = lbutlast l @@ (llast l ## LNil)"
  using assms by induct auto

lemma llast_snoc [simp]:
  assumes fin: "xs ∈ A*"
  shows "llast (xs @@ x ## LNil) = x"
  using fin
proof induct
  case LNil_fin thus ?case by simp
next
  case (LCons_fin a l)
  have "x ## LNil ∈ UNIV*" by auto
  with LCons_fin show ?case
    by (auto simp: llast_LCons [of _ UNIV])
qed

lemma lbutlast_snoc [simp]:
  assumes fin: "xs ∈ A*"
  shows "lbutlast (xs @@ x ## LNil) = xs"
  using fin
proof induct
  case LNil_fin thus ?case by simp
```

```
next
  case (LCons_fin a l)
  have "x ## LNil ∈ UNIV*" by auto
  with LCons_fin show ?case
    by (auto simp: lbutlast_LCons [of _ UNIV])
qed

lemma llast_lappend [simp]:
"⟦ x ∈ UNIV*; y ∈ UNIV* ⟧ ⟹ llast (x @@ a ## y) = llast (a ## y)"
proof (induct rule: finlsts.induct)
  case LNil_fin thus ?case by simp
next case (LCons_fin l b)
  hence "l @@ a ## y ∈ UNIV*" by auto
  thus ?case using LCons_fin
    by (auto simp: llast_LCons [of _ UNIV])
qed

lemma llast_llength:
  assumes tfin: "t ∈ UNIV*"
  shows "t ≠ LNil ⟹ t !! (llength t - (Suc 0)) = Some (llast t)"
  using tfin
proof induct
  case (LNil_fin l) thus ?case by auto
next
  case (LCons_fin a l) note consal = this thus ?case
  proof (cases l)
    case LNil_fin thus ?thesis using consal by simp
  next
    case (LCons_fin aa la)
    thus ?thesis using consal by simp
  qed
qed
```

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"
  by (auto simp: lconst_def intro: iterates)

lemma lconst_LNil [iff]: "lconst a ≠ LNil"
  by (clarify,frule subst [OF lconst_unfold]) simp

lemma lconstT:
  assumes aA: "a ∈ A"
  shows "lconst a ∈ A^ω"
proof (rule inflstsI)
  show "lconst a ∈ A^∞"
  proof (rule alllsts.coinduct [of "λx. x = lconst a"], simp_all)
    have "lconst a = a ## lconst a"
      by (rule lconst_unfold)
    with aA
```

```
    show "∃l aa. lconst a = aa ## l ∧ (l = lconst a ∨ l ∈ A∞) ∧ aa ∈ A"
      by blast
  qed
next assume lconst: "lconst a ∈ UNIV*"
  moreover have "⋀l. l ∈ UNIV* ⟹ lconst a ≠ l"
  proof-
    fix l::"'a llist" assume "l∈UNIV*"
    thus "lconst a ≠ l"
    proof (rule finlsts_induct, simp_all)
      fix a' l' assume
        al': "lconst a ≠ l'" and
        l'A: "l' ∈ UNIV*"
      from al' show  "lconst a ≠ a' ## l'"
      proof (rule contrapos_np)
        assume notal: "¬ lconst a ≠ a' ## l'"
        hence "lconst a = a' ## l'" by simp
        hence "a ## lconst a = a' ## l'"
          by (rule subst [OF lconst_unfold])
        thus "lconst a = l'" by auto
      qed
    qed
  qed
  ultimately show "False" using aA by auto
qed
```

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"
  by (unfold llist_le_def) auto

lemma LNil_le [iff]:"LNil ≤ s"
  by (auto simp: llist_le_def)

lemma le_LNil [iff]: "(s ≤ LNil) = (s = LNil)"
  by (auto simp: llist_le_def)

lemma llist_inf_le:
  "s ∈ Aω  ⟹ (s≤t) = (s=t)"
  by (unfold llist_le_def) auto

lemma le_LCons [iff]: "(x ## xs ≤ y ## ys) = (x = y ∧ xs ≤ ys)"
  by (unfold llist_le_def) auto
```

```
lemma llist_le_refl [iff]:
  "(s:: 'a llist) ≤ s"
  by (unfold llist_le_def) (rule exI [of _ "LNil"], simp)

lemma llist_le_trans [trans]:
  fixes r:: "'a llist"
  shows "r ≤ s ⟹ s ≤ t ⟹ r ≤ t"
  by (auto simp: llist_le_def lappend_assoc)

lemma llist_le_anti_sym:
  fixes s:: "'a llist"
  assumes st: "s ≤ t"
  and ts: "t ≤ s"
  shows "s = t"
proof-
  have "s ∈ UNIV∞" by auto
  thus ?thesis
  proof (cases rule: alllstsE)
    case finite
    hence "∀ t.  s ≤ t ∧ t ≤ s ⟶ s = t"
    proof (induct rule: finlsts.induct)
      case LNil_fin thus ?case by auto
    next
      case (LCons_fin l a) show ?case
      proof
        fix t from LCons_fin show  "a ## l ≤ t ∧ t ≤ a ## l ⟶ a ## l = t"
          by (cases "t") blast+
      qed
    qed
    thus ?thesis using st ts by blast
  next case infinite thus ?thesis using st by (simp add: llist_inf_le)
  qed
qed

lemma llist_less_le_not_le:
  fixes s :: "'a llist"
  shows "(s < t) = (s ≤ t ∧ ¬ t ≤ s)"
  by (auto simp add: llist_less_def dest: llist_le_anti_sym)

instance
  by standard
    (assumption | rule llist_le_refl
      llist_le_trans llist_le_anti_sym llist_less_le_not_le)+

end
```

1.6.1 Typing rules

```
lemma llist_le_finT [simp]:
 "r≤s ⟹ s ∈ A⋆ ⟹ r ∈ A⋆"
proof-
  assume rs: "r≤s" and sfin: "s ∈ A⋆"
  from sfin have "∀r. r≤s ⟶ r∈A⋆"
```

```
  proof (induct "s")
    case LNil_fin thus ?case by auto
  next
    case (LCons_fin a l) show  ?case
    proof (clarify)
      fix r assume ral: "r ≤ a ## l"
      thus "r ∈ A⋆" using LCons_fin
        by (cases r) auto
    qed
  qed
  with rs show ?thesis by auto
qed

lemma llist_less_finT [iff]:
 "r<s ⟹ s ∈ A⋆ ⟹ r ∈ A⋆"
  by (auto simp: less_le)
```

1.6.2 More simplification rules

```
lemma LNil_less_LCons [iff]: "LNil < a ## t"
  by (simp add: less_le)

lemma not_less_LNil [iff]:
  "¬ r < LNil"
  by (auto simp: less_le)

lemma less_LCons [iff]:
  " (a ## r < b ## t) = (a = b ∧ r < t)"
  by (auto simp: less_le)

lemma llength_mono [iff]:
  assumes"r ∈ A⋆"
  shows "s<r ⟹ llength s < llength r"
  using assms
proof(induct "r" arbitrary: s)
  case LNil_fin thus ?case by simp
next
  case (LCons_fin a l)
  thus ?case
    by (cases s) (auto simp: llength_LCons [of _ UNIV])
qed

lemma le_lappend [iff]: "r ≤ r @@ s"
  by (auto simp: llist_le_def)

lemma take_inf_less:
  "t ∈ UNIVω ⟹ t ↓ i < t"
proof (induct i arbitrary: t)
  case 0 thus ?case by (auto elim: inflsts_cases)
next
  case (Suc i)
  from ‹t ∈ UNIVω› show ?case
  proof (cases "t")
```

```
    case (LCons a l) with Suc show ?thesis
      by auto
  qed
qed

lemma lapp_take_less:
  assumes iless: "i < llength r"
  shows "(r @@ s) ↓ i < r"
proof (cases "r ∈ UNIV*")
  case True
  thus ?thesis using iless
  proof(induct i arbitrary: r)
    case 0 thus ?case by (cases "r") auto
  next
    case (Suc j)
    from ‹r ∈ UNIV*› ‹Suc j < llength r› ‹⋀r. ⟦r ∈ UNIV*; j < llength r⟧ ⟹ lappend
r s ↓ j < r›
    show ?case by (cases) auto
  qed
next
  case False thus ?thesis by (simp add: take_inf_less)
qed
```

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 ∈ Aω ∧ 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}"
  by (auto simp: finpref_def)

lemma finpref_fin: "x ∈ finpref A s ⟹ x ∈ A*"
  by (auto simp: finpref_def)

lemma finpref_mono2: "s ≤ t ⟹ finpref A s ⊆ finpref A t"
  by (unfold finpref_def) (auto dest: llist_le_trans)
```

```
lemma suff_LNil [simp]:
  "suff A LNil = A∞"
  by (simp add: suff_def)

lemma suff_all: "x ∈ suff A s ⟹ x ∈ A∞"
  by (auto simp: suff_def)

lemma suff_mono2: "s ≤ t ⟹ suff A t ⊆ suff A s"
  by (unfold suff_def) (auto dest: llist_le_trans)

lemma suff_appE:
  assumes rA: "r ∈ A⋆"
  and  tsuff: "t ∈ suff A r"
  obtains s where "s ∈ A∞" "t = r@@s"
proof-
  from tsuff obtain s where
    tA: "t ∈ A∞" and trs: "t = r @@ s"
    by (auto simp: suff_def llist_le_def)
  from rA trs tA have "s ∈ A∞"
    by (auto simp: lapp_allT_iff)
  thus ?thesis using trs
    by (rule that)
qed

lemma LNil_suff [iff]: "(LNil ∈ suff A s) = (s = LNil)"
  by (auto simp: suff_def)

lemma finpref_suff [dest]:
  "⟦ r ∈ finpref A t; t∈A∞ ⟧ ⟹ t ∈ suff A r"
  by (auto simp: finpref_def suff_def)

lemma suff_finpref:
  "⟦ t ∈ suff A r; r∈A⋆ ⟧ ⟹ r ∈ finpref A t"
  by (auto simp: finpref_def suff_def)

lemma suff_finpref_iff:
  "⟦ r∈A⋆; t∈A∞ ⟧ ⟹ (r ∈ finpref A t) = (t ∈ suff A r)"
  by (auto simp: finpref_def suff_def)

lemma infsuff_LNil [simp]:
  "infsuff A LNil = Aω"
  by (simp add: infsuff_def)

lemma infsuff_inf: "x ∈ infsuff A s ⟹ x ∈ Aω"
  by (auto simp: infsuff_def)

lemma infsuff_mono2: "s ≤ t ⟹ infsuff A t ⊆ infsuff A s"
  by (unfold infsuff_def) (auto dest: llist_le_trans)

lemma infsuff_appE:
  assumes   rA: "r ∈ A⋆"
  and tinfsuff: "t ∈ infsuff A r"
  obtains s where "s ∈ Aω" "t = r@@s"
```

```
proof-
  from tinfsuff obtain s where
    tA: "t ∈ A^ω" and trs: "t = r @@ s"
    by (auto simp: infsuff_def llist_le_def)
  from rA trs tA have "s ∈ A^ω"
    by (auto dest: app_invT)
  thus ?thesis using trs
    by (rule that)
qed

lemma finpref_infsuff [dest]:
  "⟦ r ∈ finpref A t; t∈A^ω ⟧ ⟹ t ∈ infsuff A r"
  by (auto simp: finpref_def infsuff_def)

lemma infsuff_finpref:
  "⟦ t ∈ infsuff A r; r∈A^⋆ ⟧ ⟹ r ∈ finpref A t"
  by (auto simp: finpref_def infsuff_def)

lemma infsuff_finpref_iff [iff]:
  "⟦ r∈A^⋆; t∈A^ω ⟧ ⟹ (t ∈ finpref A r) = (r ∈ infsuff A t)"
  by (auto simp: finpref_def infsuff_def)

lemma prefix_lemma:
  assumes xinf: "x ∈ A^ω"
  and yinf: "y ∈ A^ω"
  and R: "⋀ s. ⟦ s ∈ A^⋆; s ≤ x⟧ ⟹ s ≤ y"
  shows "x = y"
proof-
  let ?r = "λx y. x∈A^ω ∧ y∈A^ω ∧ finpref A x ⊆ finpref A y"
  have "?r x y" using xinf yinf
    by (auto simp: finpref_def intro: R)
  thus ?thesis
  proof (coinduct rule: llist.coinduct_strong)
    case (Eq_llist a b)
    hence ainf: "a ∈ A^ω"
      and binf: "b ∈ A^ω" and pref: "finpref A a ⊆ finpref A b" by auto
    from ainf show ?case
    proof cases
      case (LCons a' l')
      note acons = this with binf show ?thesis
      proof (cases b)
        case (LCons b' l'')
        with acons pref have "a' = b'" "finpref A l' ⊆ finpref A l''"
          by (auto simp: finpref_def)
        thus ?thesis using acons LCons by auto
      qed
    qed
  qed
qed

lemma inf_neqE:
"⟦ x ∈  A^ω; y ∈ A^ω; x ≠ y;
  ⋀s. ⟦ s∈A^⋆; s ≤ x; ¬ s ≤ y⟧ ⟹ R ⟧ ⟹ R"
```

```
  by (auto intro!: prefix_lemma)

lemma pref_locally_linear:
  fixes s::"'a llist"
  assumes sx: "s ≤ x"
  and   tx: "t ≤ x"
  shows "s ≤ t ∨ t ≤ s"
proof-
  have "s ∈ UNIV∞" by auto
  thus ?thesis
  proof (cases rule: alllstsE)
    case infinite with sx tx show ?thesis
      by (auto simp: llist_inf_le)
  next
    case finite
    thus ?thesis using sx tx
    proof (induct "s" arbitrary: x t)
      case LNil_fin thus ?case by simp
    next
      case (LCons_fin a l)
      note alx = ‹a ## l ≤ x›
      note tx = ‹t ≤ x›
      show ?case
      proof(rule disjCI)
        assume tal: "¬ t ≤ a ## l"
        show "LCons a l ≤ t"
        proof (cases t)
          case LNil thus ?thesis using tal by auto
        next case (LCons b ts) note tcons = this show ?thesis
          proof (cases x)
            case LNil thus ?thesis using alx by auto
          next
            case (LCons c xs)
            from alx  LCons have ac: "a = c" and lxs: "l ≤ xs"
              by auto
            from tx tcons LCons have bc: "b = c" and tsxs: "ts ≤ xs"
              by auto
            from tcons tal ac bc have tsl: "¬ ts ≤ l"
              by auto
            from LCons_fin lxs tsxs tsl have "l ≤ ts"
              by auto
            with tcons ac bc show ?thesis
              by auto
          qed
        qed
      qed
    qed
  qed
qed

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)"
  by (auto simp: pfinpref_def)
```

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^⋆. ∃ s ∈ A^ω. t @@ s ∈ P"

definition possafety :: "'a set ⇒ 'a llist set ⇒ bool"
where "possafety A P ≡ ∀ t ∈ A^♠. (∀ 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"
  by (unfold safety_def) blast

lemma safetyD:
  "⟦ safety A P;  t ∈ A^∞;
    ⋀r. r ∈ finpref A t ⟹ ∃ s ∈ A^∞. r @@ s ∈ P
  ⟧ ⟹ t ∈ P"
  by (unfold safety_def) blast

lemma safetyE:
  "⟦ safety A P;
    ∀ t ∈ A^∞. (∀ r ∈ finpref A t. ∃ s ∈ A^∞. r @@ s ∈ P) ⟶ t ∈ P ⟹ R
   ⟧ ⟹ R"
  by (unfold safety_def) blast

lemma safety_prefix_closed:
  "safety UNIV P ⟹ prefix_closed P"
  by (auto dest!: safetyD
       simp: prefix_closed_def finpref_def llist_le_def lappend_assoc)
    blast

lemma livenessI:
  "(⋀s. s∈ A^⋆ ⟹ ∃ t ∈ A^∞. s @@ t ∈ P) ⟹ liveness A P"
  by (auto simp: liveness_def)
```

```
lemma livenessE:
  "⟦ liveness A P; ⋀t. ⟦  t ∈ A∞; s @@ t ∈ P⟧ ⟹ R; s ∉ A* ⟹ R⟧ ⟹ R"
  by (auto simp: liveness_def)

lemma possafetyI:
  "(⋀t. ⟦t ∈  A♠; ∀ r ∈ pfinpref A t. ∃ s ∈ A∞. r @@ s ∈ P⟧ ⟹ t ∈ P)
  ⟹ possafety A P"
  by (unfold possafety_def) blast

lemma possafetyD:
  "⟦ possafety A P;  t ∈ A♠;
    ⋀r. r ∈ pfinpref A t ⟹ ∃ s ∈ A∞. r @@ s ∈ P
  ⟧ ⟹ t ∈ P"
  by (unfold possafety_def) blast

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

lemma possafety_pprefix_closed:
  assumes psafety: "possafety UNIV P"
  shows "pprefix_closed P"
unfolding pprefix_closed_def
proof(intro ballI allI impI, erule conjE)
  fix t s assume tP: "t ∈ P" and st: "s ≤ t" and spos: "s ≠ LNil"
  from psafety show "s ∈ P"
  proof (rule possafetyD)
    from spos show  "s ∈ UNIV♠" by auto
  next fix r assume "r ∈ pfinpref UNIV s"
    then obtain u where scons: "s = r @@ u"
      by (auto simp: pfinpref_def finpref_def llist_le_def)
    with st obtain v where "t = r @@ u @@ v"
      by (auto simp: lappend_assoc llist_le_def)
    with tP show "∃s∈UNIV∞. r @@ s ∈ P" by auto
  qed
qed

lemma poslivenessI:
  "(⋀s. s∈ A♣ ⟹ ∃ t ∈ A∞. s @@ t ∈ P) ⟹ posliveness A P"
  by (auto simp: posliveness_def)

lemma poslivenessE:
  "⟦ posliveness A P; ⋀t. ⟦  t ∈ A∞; s @@ t ∈ P⟧ ⟹ R; s ∉ A♣ ⟹ R⟧ ⟹ R"
  by (auto simp: posliveness_def)

end
```

References

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