

More on Lazy Lists

Stefan Friedrich

December 12, 2009

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 | 2 |
| 1.1 | Preliminaries | 2 |
| 1.2 | Finite and infinite llists over an alphabet | 2 |
| 1.2.1 | Facts about all llists | 3 |
| 1.2.2 | Facts about non-empty (positive) llists | 3 |
| 1.2.3 | Facts about finite llists | 4 |
| 1.2.4 | A recursion operator for finite llists | 5 |
| 1.2.5 | Facts about non-empty (positive) finite llists | 5 |
| 1.2.6 | Facts about infinite llists | 6 |
| 1.3 | Lappend | 7 |
| 1.3.1 | Simplification | 7 |
| 1.3.2 | Typing rules | 7 |
| 1.4 | Length, indexing, prefixes, and suffixes of llists | 10 |
| 1.5 | The constant llist | 17 |
| 1.6 | The prefix order of llists | 18 |
| 1.6.1 | Typing rules | 19 |
| 1.6.2 | More simplification rules | 20 |
| 1.6.3 | Finite prefixes and infinite suffixes | 21 |
| 1.7 | Safety and Liveness | 25 |

1 More on llists

```
theory LList2
imports LList
begin
```

1.1 Preliminaries

```
syntax
```

```
LCons :: "'a ⇒ 'a llist ⇒ 'a llist" (infixr "##" 65)
lappend :: "'a llist, 'a llist] => 'a llist" (infixr "@@" 65)
```

```
lemmas lappend_assoc = lappend_assoc'
```

```
lemmas llistE [case_names LNil LCons, cases type: llist]
```

```
lemma llist_split: "P (llist_case f1 f2 x) =
  ((x = LNil → P f1) ∧ (∀ a xs. x = a ## xs → P (f2 a xs)))"
  by (cases "x") auto
```

```
lemma llist_split_asm:
```

```
"P (llist_case f1 f2 x) =
  (¬ (x = LNil ∧ ¬ P f1 ∨ (∃ a llist. x = a ## llist ∧ ¬ P (f2 a llist))))"
  by (cases "x") auto
```

1.2 Finite and infinite llists over an alphabet

```
consts
```

```
inflsts :: "'a set ⇒ 'a llist set"
fplsts :: "'a set ⇒ 'a llist set"
poslsts :: "'a set ⇒ 'a llist set"
```

```
syntax (xsymbols)
```

```
inflsts :: "'a set ⇒ 'a llist set" ("(_ω)" [1000] 999)
fplsts :: "'a set ⇒ 'a llist set" ("(_♣)" [1000] 999)
poslsts :: "'a set ⇒ 'a llist set" ("(_♠)" [1000] 999)
```

```
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]
```

defs

inflsts_def: " $A^\omega \equiv A^\infty - \text{UNIV}^*$ "

poslsts_def: " $A^\spadesuit \equiv A^\infty - \{\text{LNil}\}$ "

fpslsts_def: " $A^\clubsuit \equiv A^* - \{\text{LNil}\}$ "

1.2.1 Facts about all lists

lemma neq_LNil_conv: " $(xs \neq \text{LNil}) = (\exists y \text{ ys}. xs = y \ \#\# \ \text{ys})$ "
by (cases xs) auto

lemma alllsts_UNIV [iff]:

" $s \in \text{UNIV}^\infty$ "

proof (rule alllsts.coinduct [of " $\lambda x. \text{True}$ "], simp)

fix z :: "'a llist"

show " $z = \text{LNil} \vee (\exists l \ a. z = a \ \#\# \ l \wedge (\text{True} \vee l \in \text{UNIV}^\infty) \wedge a \in \text{UNIV})$ "

by (cases "z") auto

qed

lemma alllsts_empty [simp]: " $\{\}^\infty = \{\text{LNil}\}$ "

by (auto elim: alllsts.cases)

lemma alllsts_mono:

assumes asubb: " $A \subseteq B$ "

shows " $A^\infty \subseteq B^\infty$ "

proof

fix x assume "x $\in A^\infty$ " thus "x $\in B^\infty$ "

proof (elim alllsts.coinduct [of " $\lambda z. z \in A^\infty$ "])

fix z assume "z $\in A^\infty$ "

thus "z = LNil $\vee (\exists l \ a. z = a \ \#\# \ l \wedge (l \in A^\infty \vee l \in B^\infty) \wedge a \in B)$ "

using asubb by (cases "z") auto

qed

qed

declare alllsts_mono [mono_set]

lemma LConsE [iff]: " $x \ \#\# \ xs \in A^\infty = (x \in A \wedge xs \in A^\infty)$ "

by (auto elim: alllsts.cases)

1.2.2 Facts about non-empty (positive) lists

lemma poslsts_iff [iff]:

" $(s \in A^\spadesuit) = (s \in A^\infty \wedge s \neq \text{LNil})$ "

by (auto simp: poslsts_def)

lemma poslsts_UNIV [iff]:

" $s \in \text{UNIV}^\spadesuit = (s \neq \text{LNil})$ "

by auto

lemma poslsts_empty [simp]: " $\{\}^\spadesuit = \{\}$ "

by auto

```

lemma poslstst_mono:
  "A ⊆ B ⇒ A+ ⊆ B+"
  by (auto dest: alllstst_mono)

```

1.2.3 Facts about finite llists

```

lemma finlstst_empty [simp]: "{}* = {LNil}"
  by (auto elim: finlstst.cases)

```

```

lemma finsubsetall: "x ∈ A* ⇒ x ∈ A∞"
  by (induct rule: finlstst.induct) auto

```

```

lemma finlstst_mono:
  "A ⊆ B ⇒ A* ⊆ B*"
  by (auto, erule finlstst.induct) auto

```

```

declare finlstst_mono [mono_set]

```

```

lemma finlstst_induct
  [case_names LNil_fin LCons_fin, induct set: finlstst, 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 [rule_format]:
  "x ∈ A* ⇒ x ∈ B∞ → x ∈ B*"
proof (induct rule: finlstst.induct)
  case LNil_fin show ?case by auto
next
  case (LCons_fin l a)
  show ?case
  proof
    assume "a##l ∈ B∞" thus "a##l ∈ B*" using LCons_fin
      by (cases "a##l") auto
  qed
qed

```

```

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

```

```

lemma finT_simp [simp]:
  "r ∈ A* ⇒ r ∈ UNIV*"
  by auto

```

1.2.4 A recursion operator for finite llists

constdefs

```

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

finlsts_rec :: "[ 'b, [ 'a, 'a llist, 'b ] ⇒ 'b ] ⇒ 'a llist ⇒ 'b"
"finlsts_rec c d r ≡ if r ∈ UNIV*
then (wfrec finlsts_pred (%f. llist_case 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* ] ⇒ x ∈ Aω"
  by (unfold inflsts_def) auto

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 ∈ Aω ⇒ x ∈ A∞"
  by (auto intro: finite_lemma finsubsetall)

lemma inflsts_mono:
  "A ⊆ B ⇒ Aω ⊆ Bω"
  by (blast dest: alllsts_mono infsubsetall)

lemma inflsts_cases [case_names LCons, cases set: inflsts, consumes 1]:
  assumes sinf: "s ∈ Aω"
  and R: "∧ a l. [ l ∈ Aω; a ∈ A; s = a ## l ] ⇒ R"
  shows "R"
proof -
  from sinf have "s ∈ A∞" "s ∉ UNIV*"
  by auto
  then obtain a l where "l ∈ Aω" and "a ∈ A" and "s = a ## l"
  by (cases "s") auto
  thus ?thesis by (rule R)
qed

lemma inflstsI2: "[ a ∈ A; t ∈ Aω ] ⇒ a ## t ∈ Aω"
  by (auto elim: finlsts.cases)

lemma infT_simp [simp]:
  "r ∈ Aω ⇒ r ∈ UNIVω"
  by auto

```

```

lemma alllstE [consumes 1, case_names finite infinite]:
  "[[ x ∈ A∞; x ∈ A* ⇒ P; x ∈ Aω ⇒ P ]] ⇒ P"
  by (auto intro: finite_lemma simp: inflsts_def)

```

```

lemma fin_inf_cases [case_names finite infinite]:
  "[[ r ∈ UNIV* ⇒ P; r ∈ UNIVω ⇒ P ]] ⇒ P"
  by auto

```

```

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

```

```

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

```

```

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

```

1.3 Lappend

1.3.1 Simplification

```

lemma lapp_inf [simp]:
  "s ∈ Aω ⇒ s @@ t = s"
  by (rule llist_equalityI [of _ _ " (λu. (u@@t, u))'Aω"], auto)
  (erule_tac s = "u" in inflsts_cases, auto)

```

```

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 (rule alllstE.coinduct [of "λx. x ∈ (∪ u ∈ A∞. ∪ v ∈ A∞. {u@@v})"])
  from sllist tllist show "s @@ t ∈ (∪ u ∈ A∞. ∪ v ∈ A∞. {u @@ v})"
  by fast
next fix z assume "z ∈ (∪ u ∈ A∞. ∪ v ∈ A∞. {u @@ v})"
  then obtain u v where ullist: "u ∈ A∞" and vllist: "v ∈ A∞" and zapp: "z = u @@ v"
  by auto

```

```

    thus "z = LNil  $\vee$  ( $\exists l$  a. z = a ## l  $\wedge$  ( $l \in (\bigcup u \in A^\infty. \bigcup v \in A^\infty. \{u @@ v\}) \vee l \in A^\infty$ )  $\wedge$ 
a  $\in$  A)"
    by (cases "u") (auto elim: alllst.cases)
qed

```

```

lemma lappfin_finT: "[[ s  $\in$  A*; t  $\in$  A* ]  $\implies$  s@@t  $\in$  A*"
  by (induct rule: finlst.induct) auto

```

```

lemma lapp_fin_fin_lemma:
  assumes rsA: "r @@ s  $\in$  A*"
  shows "r  $\in$  A*"
proof-
  have " $\forall l \in A^*. \forall r. l = r @@ s \implies r \in A^*$ "
  proof rule
    fix l assume "l  $\in$  A*"
    thus " $\forall r. l = r @@ s \implies r \in A^*$ "
    proof (induct "l")
      case LNil_fin thus ?case by auto
    next
      case (LCons_fin a l') show ?case
      proof (clarify)
        fix r assume al'rs: "a##l' = r @@ s"
        show "r  $\in$  A*"
        proof (cases "r")
          case LNil thus ?thesis by auto
        next
          case (LCons x xs) with al'rs
            have "a = x" and "l' = xs @@ s"
            by auto
            with LCons_fin LCons show ?thesis
            by auto
        qed
      qed
    qed
  qed
  with rsA show ?thesis by blast
qed

```

```

lemma lapp_fin_fin_iff [iff]: "(r @@ s  $\in$  A*) = (r  $\in$  A*  $\wedge$  s  $\in$  A*)"
proof (auto intro: lappfin_finT lapp_fin_fin_lemma)
  assume rsA: "r @@ s  $\in$  A*"
  hence "r  $\in$  A*" by (rule lapp_fin_fin_lemma)
  hence "r @@ s  $\in$  A*  $\implies$  s  $\in$  A*"
  by (induct "r", simp) (auto elim: finlst.cases)
  with rsA show "s  $\in$  A*" by auto
qed

```

```

lemma lapp_all_invT:
  assumes rs: "r@@s  $\in$  A $^\infty$ "
  shows "r  $\in$  A $^\infty$ "
proof (cases "r  $\in$  UNIV*")
  case False
  with rs show ?thesis by simp

```

```

next
  case True
  hence "r @@ s ∈ A∞ → r ∈ A∞"
  by (induct "r") auto
  with rs show ?thesis by auto
qed

lemma lapp_fin_infT: "[s ∈ A*; t ∈ Aω] ⇒ s @@ t ∈ Aω"
  by (induct rule: finlsts.induct)
  (auto intro: inflstsI2)

lemma app_invT [rule_format]:
  "r ∈ A* ⇒ ∀s. r @@ s ∈ Aω → s ∈ Aω"
proof (induct rule: finlsts.induct)
  case LNil_fin thus ?case by simp
next case (LCons_fin l a) show ?case
  proof clarify
    fix s assume "(a ## l) @@ s ∈ Aω"
    hence "a ## (l @@ s) ∈ Aω" by simp
    hence "l @@ s ∈ Aω" by (auto elim: inflsts_cases)
    with LCons_fin show "s ∈ Aω" by blast
  qed
qed

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

consts

```
ll2f :: "'a llist ⇒ nat ⇒ 'a option" (infix "!!" 100)
ltake :: "'a llist ⇒ nat ⇒ 'a llist"
ldrop :: "'a llist ⇒ nat ⇒ 'a llist"
```

syntax (xsymbols)

```
ltake  :: "'a llist ⇒ nat ⇒ 'a llist" (infixl "↓" 110)
ldrop  :: "'a llist ⇒ nat ⇒ 'a llist" (infixl "↑" 110)
```

primrec

```
"l!!0 = (case l of LNil ⇒ None | LCons x xs ⇒ Some x)"
"l!!(Suc i) = (case l of LNil ⇒ None | x ## xs ⇒ xs!!i)"
```

primrec

```
"l ↓ 0 = LNil"
"l ↓ Suc i = (case l of LNil ⇒ LNil | x ## xs ⇒ x ## xs ↓ i)"
```

primrec

```
"l ↑ 0 = l"
"l ↑ Suc i = (case l of LNil ⇒ LNil | x ## xs ⇒ xs ↑ i)"
```

constdefs

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

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

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

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

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

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

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

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

```
lemmas lrev_LNil = lrev_def [THEN finlsts_rec_LNil_def, standard]
and lrev_LCons = lrev_def [THEN finlsts_rec_LCons_def, standard]
```

```

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, simp)
  case (LCons_fin a l)
  have "a ## LNil ∈ UNIV*" by auto
  thus ?case using LCons_fin
  by auto
qed

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-
  from fin have "lrev xs ∈ A*" by simp
  hence "P (lrev (lrev xs))"
  proof (induct "lrev xs")
    case (LNil_fin l) with init show "P (lrev l)" by simp
  next
    case (LCons_fin a l) thus ?case by (auto intro: step)
  end
end

```

```

qed
thus ?thesis using fin by simp
qed

lemma finlsts_rev_cases [case_names LNil snocl, consumes 1]:
  assumes tfin: "t ∈ A*"
  and      lnil: "t = LNil ⇒ P"
  and      lcons: "∧ a l. [ l ∈ A*; a ∈ A; t = l @@ a ## LNil ] ⇒ P"
  shows "P"
  using prems
  by (induct rule: lrev_induct) auto

lemma l12f_LNil [simp]: "LNil!!x = None"
  by (cases "x") auto

lemma None_lfinite [rule_format]: "∀t. t!!i = None → t ∈ UNIV*"
proof (induct "i")
  case 0 show ?case
  proof
    fix t show "t !! 0 = None → t ∈ UNIV*"
    by (rule llistE [of "t"]) auto
  qed
next case (Suc n)
  show ?case
  proof clarify
    fix t assume tsuc: "(t::'a llist)!!Suc n = None"
    show "t ∈ UNIV*"
    proof (rule llistE [of "t"], clarify)
      fix x l' assume "t = x ## l'"
      with Suc tsuc show "t ∈ UNIV*"
      by auto
    qed
  qed
qed
qed

lemma infinite_Some: "t ∈ Aω ⇒ ∃a. t!!i = Some a"
  by (rule ccontr) (auto dest: None_lfinite)

lemmas infinite_idx_SomeE = exE [OF infinite_Some, standard]

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

lemma l12f_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)"
proof-
  from rA show ?thesis

```

```

proof (induct "r")
  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!: l12f_llength Least_Suc2)
  thus ?case by rule
qed
qed

lemma l12f_lem1:
  " $\bigwedge x t. t !! (\text{Suc } i) = \text{Some } x \implies \exists y. t !! i = \text{Some } y$ "
proof (induct i)
  case 0 thus ?case by (auto split: llist_split llist_split_asm)
next
  case (Suc k) thus ?case
    by (cases t) auto
qed

lemmas l12f_Suc_Some = l12f_lem1 [THEN exE, standard]

lemma l12f_None_Suc: " $\bigwedge t. t !! i = \text{None} \implies t !! \text{Suc } i = \text{None}$ "
proof (induct i)
  case 0 thus ?case by (auto split: llist_split)
next
  case (Suc k) thus ?case by (cases t) auto
qed

lemma l12f_None_le:
  " $\bigwedge t j. [\![t !! j = \text{None}; j \leq i]\!] \implies t !! i = \text{None}$ "
proof (induct i)
  case 0 thus ?case by simp
next
  case (Suc k) thus ?case by (cases j) (auto split: llist_split)
qed

lemma l12f_Some_le:
  assumes jlei: " $j \leq i$ "
  and tisome: " $t !! i = \text{Some } x$ "
  and H: " $\bigwedge y. t !! j = \text{Some } y \implies Q$ "
  shows "Q"
proof -
  have " $\exists y. t !! j = \text{Some } y$ " (is "?R")
  proof (rule ccontr)
    assume " $\neg ?R$ "
    hence " $t !! j = \text{None}$ " by auto
    with tisome jlei show False
      by (auto dest: l12f_None_le)
  qed
  thus ?thesis using H by auto
qed

lemma ltake_LNil [simp]: " $\text{LNil} \downarrow i = \text{LNil}$ "

```

```

    by (cases "i") auto

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

lemma take_fin [iff]: "∧t. t ∈ A∞ ⇒ t↓i ∈ A*"
proof (induct i)
  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 [rule_format, simp]: "∀ t ∈ Aω. llength (t↓i) = i"
proof (induct "i")
  case 0 thus ?case by simp
next
  case (Suc j) show ?case
  proof
    fix t assume tinf: "t ∈ Aω"
    thus "llength (t ↓ Suc j) = Suc j" using Suc
      by (cases "t") (auto simp: llength_LCons [of _ UNIV])
  qed
qed

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

lemma ltake_ldrop:
  "∧xs. (xs ↑ m) ↓ n = (xs ↓ (n + m)) ↑ m"
proof (induct "m")
  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. t ↑ (i + k) = t ↑ i ↑ k"
proof (induct "i", simp)
  case (Suc j) thus ?case

```

```

    by (cases "t") auto
qed

lemma ldrop_fun: " $\bigwedge t. t \uparrow i \implies j = t!!(i + j)$ "
proof (induct i)
  case 0 thus ?case by simp
next
  case (Suc k) then show ?case
    by (cases "t") auto
qed

lemma ldropT[simp]: " $\bigwedge t. t \in A^\infty \implies t \uparrow i \in A^\infty$ "
proof (induct i)
  case 0 thus ?case by simp
next case (Suc j)
  thus ?case by (cases "t") auto
qed

lemma ldrop_finT[simp]: " $\bigwedge t. t \in A^* \implies t \uparrow i \in A^*$ "
proof (induct i)
  case 0 thus ?case by simp
next
  fix n t assume "t  $\in A^*$ " and
    " $\bigwedge t::'a \text{ llist}. t \in A^* \implies t \uparrow n \in A^*$ "
  thus "t  $\uparrow$  Suc n  $\in A^*$ "
    by (cases "t") auto
qed

lemma ldrop_infT[simp]: " $\bigwedge t. t \in A^\omega \implies t \uparrow i \in A^\omega$ "
proof (induct i)
  case 0 thus ?case by simp
next
  fix n t assume "t  $\in A^\omega$ " and
    " $\bigwedge t::'a \text{ llist}. t \in A^\omega \implies t \uparrow n \in A^\omega$ "
  thus "t  $\uparrow$  Suc n  $\in A^\omega$ "
    by (cases "t") auto
qed

lemma lapp_suff_llength: "r  $\in A^* \implies (r@@s) \uparrow \text{llength } r = s$ "
  by (erule finlsts.induct) auto

lemma ltake_lappend_llength [simp]:
  "r  $\in A^* \implies (r @@ s) \downarrow \text{llength } r = r$ "
  by (erule finlsts.induct) auto

lemma ldrop_LNil_less:
  " $\bigwedge j t. \llbracket j \leq i; t \uparrow j = \text{LNil} \rrbracket \implies t \uparrow i = \text{LNil}$ "
proof (induct i)
  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 ∈ UNIVω) = (t ∈ UNIVω)"
proof (auto)
  show "t↑i ∈ UNIVω ⇒ t ∈ UNIVω"
  by (rule ccontr) (auto dest: ldrop_finT)
qed

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

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

lemma llength_drop_take:
  "∧t. t↑i ≠ LNil ⇒ llength (t↓i) = i"
proof (induct i)
  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: fpslst, 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 prems 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

constdefs
  lconst :: "'a ⇒ 'a llist"
  "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 ∈ Aω"
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 "∃! 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

```

lconst_le_def: "(s :: 'a llist) ≤ t ↔ (∃d. t = s @@ d)"

```

definition

```

lconst_less_def: "(s :: 'a llist) < t ↔ (s ≤ t ∧ s ≠ t)"

```

lemma not_LCons_le_LNil [iff]:

```

"¬ (a##l) ≤ LNil"

```

```

by (unfold lconst_le_def) auto

```

lemma LNil_le [iff]: "LNil ≤ s"

```

by (auto simp: lconst_le_def)

```

lemma le_LNil [iff]: "(s ≤ LNil) = (s = LNil)"

```

by (auto simp: lconst_le_def)

```

lemma lconst_inf_le:

```

"s ∈ Aω ⇒ (s ≤ t) = (s = t)"

```

```

by (unfold lconst_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_antisym:
  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_antisym)

instance by default
  (assumption | rule llist_le_refl
  llist_le_trans llist_le_antisym llist_less_le_not_le)+

end

```

1.6.1 Typing rules

```

lemma llist_le_finT [rule_format, 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 [rule_format, 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 [rule_format, iff]:
  assumes "r ∈ A*"
  shows "∀ s. s < r → llength s < llength r"
  using prems
proof (induct "r")
  case LNil_fin thus ?case by simp
next
  case (LCons_fin a l) show ?case
  proof (clarify)
    fix s assume sless: "s < a ## l"
    with LCons_fin show "llength s < llength (a ## l)"
      by (cases s) (auto simp: llength_LCons [of _ UNIV])
  qed
qed

```

```

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

```

```

lemma take_inf_less:
  "∧ t. t ∈ UNIVω ⇒ t ↓ i < t"
proof (induct i)

```

```

    case 0 thus ?case by (auto elim: inflsts_cases)
next
  case (Suc i) assume "t ∈ UNIVω"
  thus ?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
  have "∀r ∈ UNIV*. i < llength r → (r @@ s) ↓ i < r"
  proof(induct i)
    case 0 thus ?case
    proof clarify
      fix r::"'a llist" assume "0 < llength r"
      thus "(r @@ s) ↓ 0 < r"
        by (cases "r") auto
    qed
  next
    case (Suc j) show ?case
    proof clarify
      fix r :: "'a llist" assume "r ∈ UNIV*" "Suc j < llength r"
      thus "(r @@ s) ↓ Suc j < r" using Suc
        by (cases r) auto
    qed
  qed
  with iless True show ?thesis by auto
next case False thus ?thesis by (simp add: take_inf_less)
qed

```

1.6.3 Finite prefixes and infinite suffixes

```

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

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

  infsuff :: "'a set ⇒ 'a llist ⇒ 'a llist set"
  "infsuff A s ≡ {r. r ∈ Aω ∧ s ≤ r}"

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

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

```

```

suffix_closed :: "'a llist set  $\Rightarrow$  bool"
"suffix_closed A  $\equiv \forall t \in A. \forall s. t \leq s \longrightarrow s \in A$ "

lemma finpref_LNil [simp]:
  "finpref A LNil = {LNil}"
  by (auto simp: finpref_def)

lemma finpref_fin: "x  $\in$  finpref A s  $\implies$  x  $\in$  A*"
  by (auto simp: finpref_def)

lemma finpref_mono2: "s  $\leq$  t  $\implies$  finpref A s  $\subseteq$  finpref A t"
  by (unfold finpref_def) (auto dest: llist_le_trans)

lemma suff_LNil [simp]:
  "suff A LNil = A $^\infty$ "
  by (simp add: suff_def)

lemma suff_all: "x  $\in$  suff A s  $\implies$  x  $\in$  A $^\infty$ "
  by (auto simp: suff_def)

lemma suff_mono2: "s  $\leq$  t  $\implies$  suff A t  $\subseteq$  suff A s"
  by (unfold suff_def) (auto dest: llist_le_trans)

lemma suff_appE:
  assumes rA: "r  $\in$  A*"
  and tsuff: "t  $\in$  suff A r"
  and H: " $\bigwedge s. [s \in A^\infty; t = r@@s] \implies R$ "
  shows "R"
proof-
  from tsuff obtain s where
    tA: "t  $\in$  A $^\infty$ " and trs: "t = r @@ s"
  by (auto simp: suff_def llist_le_def)
  from rA trs tA have "s  $\in$  A $^\infty$ "
  by (auto simp: lapp_allT_iff)
  thus ?thesis using trs
  by (rule H)
qed

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

lemma finpref_suff [dest]:
  "[[ r  $\in$  finpref A t; t  $\in$  A $^\infty$  ]  $\implies$  t  $\in$  suff A r"
  by (auto simp: finpref_def suff_def)

lemma suff_finpref:
  "[[ t  $\in$  suff A r; r  $\in$  A* ]  $\implies$  r  $\in$  finpref A t"
  by (auto simp: finpref_def suff_def)

lemma suff_finpref_iff:
  "[[ r  $\in$  A*; t  $\in$  A $^\infty$  ]  $\implies$  (r  $\in$  finpref A t) = (t  $\in$  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 tinsuff: "t ∈ infsuff A r"
  and H: "∧s. [ s ∈ Aω; t = r@@s ] ⇒ R"
  shows "R"
proof-
  from tinsuff obtain s where
    tA: "t ∈ Aω" and trs: "t = r @@ s"
  by (auto simp: infsuff_def llist_le_def)
  from rA trs tA have "s ∈ Aω"
  by (auto dest: app_invT)
  thus ?thesis using trs
  by (rule H)
qed

lemma finpref_infsuff [dest]:
  "[ r ∈ finpref A t; t ∈ Aω ] ⇒ 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 ∈ Aω ] ⇒ (t ∈ finpref A r) = (r ∈ infsuff A t)"
  by (auto simp: finpref_def infsuff_def)

lemma prefix_lemma:
  assumes xinf: "x ∈ Aω"
  and yinf: "y ∈ Aω"
  and R: "∧s. [ s ∈ A*; s ≤ x ] ⇒ s ≤ y"
  shows "x = y"
proof-
  let ?r = "{(x, y). x ∈ Aω ∧ y ∈ Aω ∧ finpref A x ⊆ finpref A y}"
  show ?thesis
proof (rule llist_equalityI [of _ _ ?r])
  show "(x, y) ∈ ?r" using xinf yinf
  by (auto simp: finpref_def intro: R)
next show "?r ⊆ llistD_Fun (?r ∪ range (λx. (x, x)))"
proof clarify
  fix a b assume ainf: "a ∈ Aω" and binf: "b ∈ Aω" and
  pref: "finpref A a ⊆ finpref A b"
  thus "(a, b) ∈ llistD_Fun (?r ∪ range (λx. (x, x)))"

```

```

proof (cases a)
  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'  $\subseteq$  finpref A l''"
        by (auto simp: finpref_def)
      thus ?thesis using acons LCons
        by auto
    qed
  qed
qed
qed
qed
qed

```

```

lemma inf_neqE:
  "[ $x \in A^\omega; y \in A^\omega; x \neq y;$ 
   $\bigwedge s. [s \in A^*; s \leq x; \neg s \leq y] \implies R$  ]  $\implies R$ "
  by (auto intro!: prefix_lemma)

```

```

lemma pref_locally_linear:
  fixes s::"a llist"
  assumes sx: "s  $\leq$  x"
  and tx: "t  $\leq$  x"
  shows "s  $\leq$  t  $\vee$  t  $\leq$  s"
proof-
  have "s  $\in$  UNIV $^\infty$ " by auto
  thus ?thesis
  proof (cases rule: alllstE)
    case infinite with sx tx show ?thesis
      by (auto simp: llist_inf_le)
  next
    case finite hence " $\forall x t. s \leq x \wedge t \leq x \longrightarrow s \leq t \vee t \leq s$ "
      proof (induct "s")
        case LNil_fin thus ?case by simp
      next
        case (LCons_fin a l) show ?case
          proof clarify
            fix x t assume alx: "a ## l  $\leq$  x"
              and tx: "t  $\leq$  x" and tal: " $\neg t \leq a ## l$ "
            show "a ## l  $\leq$  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  $\leq$  xs"
                    by auto
                  from tx tcons LCons have bc: "b = c" and tsxs: "ts  $\leq$  xs"
                    by auto
                  from tcons tal ac bc have tsl: " $\neg ts \leq l$ "
                    by auto

```

```

      from LCons_fin lxs tsxs tsl have "1 ≤ ts"
      by auto
      with tcons ac bc show ?thesis
      by auto
    qed
  qed
  qed
  qed
  thus ?thesis using sx tx by auto
  qed
  qed

```

constdefs

```

pfinpref :: "'a set ⇒ 'a llist ⇒ 'a llist set"
"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

constdefs

```

infsafety :: "'a set ⇒ 'a llist set ⇒ bool"
"infsafety A P ≡ ∀ t ∈ Aω. (∀ r ∈ finpref A t. ∃ s ∈ Aω. r @@ s ∈ P) → t ∈ P"

```

```

infliveness :: "'a set ⇒ 'a llist set ⇒ bool"
"infliveness A P ≡ ∀ t ∈ A*. ∃ s ∈ Aω. t @@ s ∈ P"

```

```

possafety :: "'a set ⇒ 'a llist set ⇒ bool"
"possafety A P ≡ ∀ t ∈ A♣. (∀ r ∈ pfinpref A t. ∃ s ∈ A∞. r @@ s ∈ P) → t ∈ P"

```

```

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

```

```

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

```

```

liveness :: "'a set ⇒ 'a llist set ⇒ bool"
"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"
proof (unfold pprefix_closed_def, clarify)
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 scon: "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:

"($\bigwedge s. s \in A^\clubsuit \implies \exists t \in A^\infty. s @@ t \in P$) \implies posliveness A P"
by (auto simp: posliveness_def)

lemma poslivenessE:

"(\llbracket posliveness A P; $\bigwedge t. \llbracket t \in A^\infty; s @@ t \in P \rrbracket \implies R$; $s \notin A^\clubsuit \implies R \rrbracket \implies R$ "
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.