POSIX Lexing with Derivatives of Regular Expressions

Fahad Ausaf   Roy Dyckhoff   Christian Urban

March 17, 2025

Abstract

Brzozowski introduced the notion of derivatives for regular expressions. They can be used for a very simple regular expression matching algorithm. Sulzmann and Lu [2] cleverly extended this algorithm in order to deal with POSIX matching, which is the underlying disambiguation strategy for regular expressions needed in lexers. In this entry we give our inductive definition of what a POSIX value is and show (i) that such a value is unique (for given regular expression and string being matched) and (ii) that Sulzmann and Lu's algorithm always generates such a value (provided that the regular expression matches the string). We also prove the correctness of an optimised version of the POSIX matching algorithm. Finally we show that (iii) our inductive definition of a POSIX value is equivalent to an alternative definition by Okui and Suzuki [1] which identifies POSIX values as least elements according to an ordering of values. All results are given also for the bounded regular expressions ($r^{\{n\}}$ and $r^{\{..n\}}$).

Contents

theory *Lexer*
  imports *Regular−Sets.Derivatives*

begin

1 Values

datatype *'a val* =
  Void
| *Atm 'a*
| *Seq 'a val 'a val*
| *Right 'a val*
| *Left 'a val*
| *Stars ('a val) list*

2 The string behind a value

fun
  flat :: *'a val* ⇒ *'a list*
where
  flat (*Void*) = []
| *flat* (*Atm c*) = [*c*]
| *flat* (*Left v*) = *flat v*
| *flat* (*Right v*) = *flat v*
| *flat* (*Seq v1 v2*) = (*flat v1*) @ (*flat v2*)
| *flat* (*Stars* []) = []
| *flat* (*Stars* (*v*#*vs*)) = (*flat v*) @ (*flat* (*Stars vs*))

abbreviation
  flats vs ≡ *concat* (*map flat vs*)

lemma *flat-Stars* [*simp*]:
 flat (*Stars vs*) = *concat* (*map flat vs*)
⟨*proof*⟩

3 Relation between values and regular expressions

inductive
  Prf :: *'a val* ⇒ *'a rexp* ⇒ *bool* (‹⊢ - : -› [*100*, *100*] *100*)
where
 ⟦⊢ *v1* : *r1*; ⊢ *v2* : *r2*⟧ ⟹ ⊢ *Seq v1 v2* : *Times r1 r2*
| ⊢ *v1* : *r1* ⟹ ⊢ *Left v1* : *Plus r1 r2*
| ⊢ *v2* : *r2* ⟹ ⊢ *Right v2* : *Plus r1 r2*
| ⊢ *Void* : *One*
| ⊢ *Atm c* : *Atom c*
| ⟦∀ *v* ∈ *set vs*. ⊢ *v* : *r* ∧ *flat v* ≠ []⟧ ⟹ ⊢ *Stars vs* : *Star r*

inductive-cases *Prf-elims*:
  ⊢ *v* : *Zero*
  ⊢ *v* : *Times r1 r2*
  ⊢ *v* : *Plus r1 r2*

$\vdash v : One$
$\vdash v : Atom\ c$
$\vdash vs : Star\ r$

lemma *Prf-flat-lang*:
assumes $\vdash v : r$ **shows** $flat\ v \in lang\ r$
$\langle proof \rangle$

lemma *Star-string*:
assumes $s \in star\ A$
shows $\exists ss.\ concat\ ss = s \wedge (\forall s \in set\ ss.\ s \in A)$
$\langle proof \rangle$

lemma *Star-val*:
assumes $\forall s{\in}set\ ss.\ \exists v.\ s = flat\ v \wedge \vdash v : r$
shows $\exists vs.\ flats\ vs = concat\ ss \wedge (\forall v{\in}set\ vs.\ \vdash v : r \wedge flat\ v \neq [])$
$\langle proof \rangle$

lemma *L-flat-Prf1*:
assumes $\vdash v : r$ **shows** $flat\ v \in lang\ r$
$\langle proof \rangle$

lemma *L-flat-Prf2*:
assumes $s \in lang\ r$ **shows** $\exists v.\ \vdash v : r \wedge flat\ v = s$
$\langle proof \rangle$

lemma *L-flat-Prf*:
$lang\ r = \{flat\ v \mid v.\ \vdash v : r\}$
$\langle proof \rangle$

4 Sulzmann and Lu functions

fun
$mkeps :: 'a\ rexp \Rightarrow 'a\ val$
where
$mkeps(One) = Void$
$\mid mkeps(Times\ r1\ r2) = Seq\ (mkeps\ r1)\ (mkeps\ r2)$
$\mid mkeps(Plus\ r1\ r2) = (if\ nullable(r1)\ then\ Left\ (mkeps\ r1)\ else\ Right\ (mkeps\ r2))$
$\mid mkeps(Star\ r) = Stars\ []$

fun $injval :: 'a\ rexp \Rightarrow 'a \Rightarrow 'a\ val \Rightarrow 'a\ val$
where
$injval\ (Atom\ d)\ c\ Void = Atm\ c$
$\mid injval\ (Plus\ r1\ r2)\ c\ (Left\ v1) = Left(injval\ r1\ c\ v1)$
$\mid injval\ (Plus\ r1\ r2)\ c\ (Right\ v2) = Right(injval\ r2\ c\ v2)$
$\mid injval\ (Times\ r1\ r2)\ c\ (Seq\ v1\ v2) = Seq\ (injval\ r1\ c\ v1)\ v2$
$\mid injval\ (Times\ r1\ r2)\ c\ (Left\ (Seq\ v1\ v2)) = Seq\ (injval\ r1\ c\ v1)\ v2$

| *injval* (*Times r1 r2*) *c* (*Right v2*) = *Seq* (*mkeps r1*) (*injval r2 c v2*)
| *injval* (*Star r*) *c* (*Seq v* (*Stars vs*)) = *Stars* ((*injval r c v*) # *vs*)

5 Mkeps, injval

lemma *mkeps-nullable*:
  assumes *nullable r*
  shows $\vdash$ *mkeps r* : *r*
⟨*proof*⟩

lemma *mkeps-flat*:
  assumes *nullable r*
  shows *flat* (*mkeps r*) = []
⟨*proof*⟩

lemma *Prf-injval-flat*:
  assumes $\vdash$ *v* : *deriv c r*
  shows *flat* (*injval r c v*) = *c* # (*flat v*)
⟨*proof*⟩

lemma *Prf-injval*:
  assumes $\vdash$ *v* : *deriv c r*
  shows $\vdash$ (*injval r c v*) : *r*
⟨*proof*⟩

6 Our Alternative Posix definition

inductive
  Posix :: *'a list* ⇒ *'a rexp* ⇒ *'a val* ⇒ *bool* (‹- ∈ - → -› [*100*, *100*, *100*] *100*)
where
  Posix-One: [] ∈ *One* → *Void*
| *Posix-Atom*: [*c*] ∈ (*Atom c*) → (*Atm c*)
| *Posix-Plus1*: *s* ∈ *r1* → *v* ⟹ *s* ∈ (*Plus r1 r2*) → (*Left v*)
| *Posix-Plus2*: ⟦*s* ∈ *r2* → *v*; *s* ∉ *lang r1*⟧ ⟹ *s* ∈ (*Plus r1 r2*) → (*Right v*)
| *Posix-Times*: ⟦*s1* ∈ *r1* → *v1*; *s2* ∈ *r2* → *v2*;
    $\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s2 \wedge (s1 @ s_3) \in lang\ r1 \wedge s_4 \in lang\ r2)$⟧ ⟹
    (*s1* @ *s2*) ∈ (*Times r1 r2*) → (*Seq v1 v2*)
| *Posix-Star1*: ⟦*s1* ∈ *r* → *v*; *s2* ∈ *Star r* → *Stars vs*; *flat v* ≠ [];
    $\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s2 \wedge (s1 @ s_3) \in lang\ r \wedge s_4 \in lang\ (Star\ r))$⟧
    ⟹ (*s1* @ *s2*) ∈ *Star r* → *Stars* (*v* # *vs*)
| *Posix-Star2*: [] ∈ *Star r* → *Stars* []

inductive-cases *Posix-elims*:
  s ∈ *Zero* → *v*
  s ∈ *One* → *v*
  s ∈ *Atom c* → *v*
  s ∈ *Plus r1 r2* → *v*
  s ∈ *Times r1 r2* → *v*

$s \in Star\ r \to v$

lemma *Posix1*:
assumes $s \in r \to v$
shows $s \in lang\ r$ $flat\ v = s$
⟨*proof*⟩

lemma *Posix1a*:
assumes $s \in r \to v$
shows $\vdash v : r$
⟨*proof*⟩

lemma *Posix-mkeps*:
assumes *nullable r*
shows $[] \in r \to mkeps\ r$
⟨*proof*⟩

lemma *Posix-determ*:
assumes $s \in r \to v1$ $s \in r \to v2$
shows $v1 = v2$
⟨*proof*⟩

lemma *Posix-injval*:
assumes $s \in (deriv\ c\ r) \to v$
shows $(c \# s) \in r \to (injval\ r\ c\ v)$
⟨*proof*⟩

7 The Lexer by Sulzmann and Lu

fun
$lexer :: 'a\ rexp \Rightarrow 'a\ list \Rightarrow ('a\ val)\ option$
where
$lexer\ r\ [] = (if\ nullable\ r\ then\ Some(mkeps\ r)\ else\ None)$
$|\ lexer\ r\ (c\#s) = (case\ (lexer\ (deriv\ c\ r)\ s)\ of$
$None \Rightarrow None$
$|\ Some(v) \Rightarrow Some(injval\ r\ c\ v))$

lemma *lexer-correct-None*:
shows $s \notin lang\ r \longleftrightarrow lexer\ r\ s = None$
⟨*proof*⟩

lemma *lexer-correct-Some*:
shows $s \in lang\ r \longleftrightarrow (\exists v.\ lexer\ r\ s = Some(v) \wedge s \in r \to v)$
⟨*proof*⟩

lemma *lexer-correctness*:
  shows $(lexer\ r\ s = Some\ v) \longleftrightarrow s \in r \to v$
  and $(lexer\ r\ s = None) \longleftrightarrow \neg(\exists\, v.\ s \in r \to v)$
$\langle proof \rangle$

end
theory *LexicalVals*
  imports *Lexer HOL−Library.Sublist*
begin

8 Sets of Lexical Values

Shows that lexical values are finite for a given regex and string.

definition
  $LV :: 'a\ rexp \Rightarrow 'a\ list \Rightarrow ('a\ val)\ set$
where $LV\ r\ s \equiv \{v.\ \vdash v : r \wedge flat\ v = s\}$

lemma *LV-simps*:
  shows $LV\ Zero\ s = \{\}$
  and $LV\ One\ s = (if\ s = []\ then\ \{Void\}\ else\ \{\})$
  and $LV\ (Atom\ c)\ s = (if\ s = [c]\ then\ \{Atm\ c\}\ else\ \{\})$
  and $LV\ (Plus\ r1\ r2)\ s = Left\ `\ LV\ r1\ s \cup Right\ `\ LV\ r2\ s$
$\langle proof \rangle$

abbreviation
  $Prefixes\ s \equiv \{s'.\ prefix\ s'\ s\}$

abbreviation
  $Suffixes\ s \equiv \{s'.\ suffix\ s'\ s\}$

abbreviation
  $SSuffixes\ s \equiv \{s'.\ strict\text{-}suffix\ s'\ s\}$

lemma *Suffixes-cons* [*simp*]:
  shows $Suffixes\ (c\ \#\ s) = Suffixes\ s \cup \{c\ \#\ s\}$
$\langle proof \rangle$

lemma *finite-Suffixes*:
  shows $finite\ (Suffixes\ s)$
$\langle proof \rangle$

lemma *finite-SSuffixes*:
  shows $finite\ (SSuffixes\ s)$

⟨*proof*⟩

lemma *finite-Prefixes*:
  shows *finite* (*Prefixes s*)
⟨*proof*⟩

lemma *LV-STAR-finite*:
  assumes $\forall$ *s*. *finite* (*LV r s*)
  shows *finite* (*LV* (*Star r*) *s*)
⟨*proof*⟩

lemma *LV-finite*:
  shows *finite* (*LV r s*)
⟨*proof*⟩

Our POSIX values are lexical values.

lemma *Posix-LV*:
  assumes $s \in r \rightarrow v$
  shows $v \in$ *LV r s*
  ⟨*proof*⟩

lemma *Posix-Prf*:
  assumes $s \in r \rightarrow v$
  shows $\vdash v : r$
  ⟨*proof*⟩

end

theory *Simplifying*
  imports *Lexer*
begin

9 Lexer including simplifications

fun *F-RIGHT* **where**
  F-RIGHT f v = *Right* (*f v*)

fun *F-LEFT* **where**
  F-LEFT f v = *Left* (*f v*)

fun *F-Plus* **where**
  F-Plus f_1 f_2 (*Right v*) = *Right* (f_2 *v*)
| *F-Plus* f_1 f_2 (*Left v*) = *Left* (f_1 *v*)
| *F-Plus f1 f2 v* = *v*

fun *F-Times1* **where**
$\mathit{F\text{-}Times1}\ f_1\ f_2\ v = \mathit{Seq}\ (f_1\ \mathit{Void})\ (f_2\ v)$

fun *F-Times2* **where**
$\mathit{F\text{-}Times2}\ f_1\ f_2\ v = \mathit{Seq}\ (f_1\ v)\ (f_2\ \mathit{Void})$

fun *F-Times* **where**
$\mathit{F\text{-}Times}\ f_1\ f_2\ (\mathit{Seq}\ v_1\ v_2) = \mathit{Seq}\ (f_1\ v_1)\ (f_2\ v_2)$
$|\ \mathit{F\text{-}Times}\ f1\ f2\ v = v$

fun *simp-Plus* **where**
$\mathit{simp\text{-}Plus}\ (\mathit{Zero}, f_1)\ (r_2, f_2) = (r_2, \mathit{F\text{-}RIGHT}\ f_2)$
$|\ \mathit{simp\text{-}Plus}\ (r_1, f_1)\ (\mathit{Zero}, f_2) = (r_1, \mathit{F\text{-}LEFT}\ f_1)$
$|\ \mathit{simp\text{-}Plus}\ (r_1, f_1)\ (r_2, f_2) =$
$(\mathit{if}\ r_1 = r_2\ \mathit{then}\ (r_1, \mathit{F\text{-}LEFT}\ f_1)\ \mathit{else}\ (\mathit{Plus}\ r_1\ r_2, \mathit{F\text{-}Plus}\ f_1\ f_2))$

fun *simp-Times* **where**
$\mathit{simp\text{-}Times}\ (\mathit{Zero}, f_1)\ (r_2, f_2) = (\mathit{Zero}, \mathit{undefined})$
$|\ \mathit{simp\text{-}Times}\ (r_1, f_1)\ (\mathit{Zero}, f_2) = (\mathit{Zero}, \mathit{undefined})$
$|\ \mathit{simp\text{-}Times}\ (\mathit{One}, f_1)\ (r_2, f_2) = (r_2, \mathit{F\text{-}Times1}\ f_1\ f_2)$
$|\ \mathit{simp\text{-}Times}\ (r_1, f_1)\ (\mathit{One}, f_2) = (r_1, \mathit{F\text{-}Times2}\ f_1\ f_2)$
$|\ \mathit{simp\text{-}Times}\ (r_1, f_1)\ (r_2, f_2) = (\mathit{Times}\ r_1\ r_2, \mathit{F\text{-}Times}\ f_1\ f_2)$

lemma *simp-Times-simps*[*simp*]:
$\mathit{simp\text{-}Times}\ p1\ p2 = (\mathit{if}\ (\mathit{fst}\ p1 = \mathit{Zero})\ \mathit{then}\ (\mathit{Zero}, \mathit{undefined})$
$\mathit{else}\ (\mathit{if}\ (\mathit{fst}\ p2 = \mathit{Zero})\ \mathit{then}\ (\mathit{Zero}, \mathit{undefined})$
$\mathit{else}\ (\mathit{if}\ (\mathit{fst}\ p1 = \mathit{One})\ \mathit{then}\ (\mathit{fst}\ p2, \mathit{F\text{-}Times1}\ (\mathit{snd}\ p1)\ (\mathit{snd}\ p2))$
$\mathit{else}\ (\mathit{if}\ (\mathit{fst}\ p2 = \mathit{One})\ \mathit{then}\ (\mathit{fst}\ p1, \mathit{F\text{-}Times2}\ (\mathit{snd}\ p1)\ (\mathit{snd}\ p2))$
$\mathit{else}\ (\mathit{Times}\ (\mathit{fst}\ p1)\ (\mathit{fst}\ p2), \mathit{F\text{-}Times}\ (\mathit{snd}\ p1)\ (\mathit{snd}\ p2))))))$
$\langle \mathit{proof} \rangle$

lemma *simp-Plus-simps*[*simp*]:
$\mathit{simp\text{-}Plus}\ p1\ p2 = (\mathit{if}\ (\mathit{fst}\ p1 = \mathit{Zero})\ \mathit{then}\ (\mathit{fst}\ p2, \mathit{F\text{-}RIGHT}\ (\mathit{snd}\ p2))$
$\mathit{else}\ (\mathit{if}\ (\mathit{fst}\ p2 = \mathit{Zero})\ \mathit{then}\ (\mathit{fst}\ p1, \mathit{F\text{-}LEFT}\ (\mathit{snd}\ p1))$
$\mathit{else}\ (\mathit{if}\ (\mathit{fst}\ p1 = \mathit{fst}\ p2)\ \mathit{then}\ (\mathit{fst}\ p1, \mathit{F\text{-}LEFT}\ (\mathit{snd}\ p1))$
$\mathit{else}\ (\mathit{Plus}\ (\mathit{fst}\ p1)\ (\mathit{fst}\ p2), \mathit{F\text{-}Plus}\ (\mathit{snd}\ p1)\ (\mathit{snd}\ p2)))))$
$\langle \mathit{proof} \rangle$

fun
$\mathit{simp} :: {'a}\ \mathit{rexp} \Rightarrow {'a}\ \mathit{rexp} * ({'a}\ \mathit{val} \Rightarrow {'a}\ \mathit{val})$
where
$\mathit{simp}\ (\mathit{Plus}\ r1\ r2) = \mathit{simp\text{-}Plus}\ (\mathit{simp}\ r1)\ (\mathit{simp}\ r2)$
$|\ \mathit{simp}\ (\mathit{Times}\ r1\ r2) = \mathit{simp\text{-}Times}\ (\mathit{simp}\ r1)\ (\mathit{simp}\ r2)$
$|\ \mathit{simp}\ r = (r, \mathit{id})$

fun
$\mathit{slexer} :: {'a}\ \mathit{rexp} \Rightarrow {'a}\ \mathit{list} \Rightarrow ({'a}\ \mathit{val})\ \mathit{option}$
where
$\mathit{slexer}\ r\ [] = (\mathit{if}\ \mathit{nullable}\ r\ \mathit{then}\ \mathit{Some}(\mathit{mkeps}\ r)\ \mathit{else}\ \mathit{None})$

| *slexer* r $(c\#s)$ = (*let* (*rs*, *fr*) = *simp* (*deriv* c r) *in*
(*case* (*slexer* *rs* s) *of*
None ⇒ *None*
| *Some*(v) ⇒ *Some*(*injval* r c (*fr* v))))

lemma *slexer-better-simp*:
slexer r $(c\#s)$ = (*case* (*slexer* (*fst* (*simp* (*deriv* c r))) s) *of*
None ⇒ *None*
| *Some*(v) ⇒ *Some*(*injval* r c ((*snd* (*simp* (*deriv* c r))) v)))
⟨*proof*⟩

lemma *L-fst-simp*:
shows *lang* r = *lang* (*fst* (*simp* r))
⟨*proof*⟩

lemma *Posix-simp*:
assumes $s \in$ (*fst* (*simp* r)) $\to v$
shows $s \in r \to$ ((*snd* (*simp* r)) v)
⟨*proof*⟩

lemma *slexer-correctness*:
shows *slexer* r s = *lexer* r s
⟨*proof*⟩

end

theory *Positions*
imports *Lexer LexicalVals*
begin

10 An alternative definition for POSIX values by Okui & Suzuki

11 Positions in Values

fun
at :: $'a$ *val* ⇒ *nat list* ⇒ $'a$ *val*
where
at v [] = v
| *at* (*Left* v) (*0#ps*)= *at* v *ps*
| *at* (*Right* v) (*Suc 0#ps*)= *at* v *ps*
| *at* (*Seq v1 v2*) (*0#ps*)= *at* *v1* *ps*
| *at* (*Seq v1 v2*) (*Suc 0#ps*)= *at* *v2* *ps*
| *at* (*Stars vs*) (*n#ps*)= *at* (*nth vs n*) *ps*

fun *Pos* :: $'a$ *val* $\Rightarrow$ (*nat list*) *set*
where
  $Pos\ (Void) = \{[]\}$
$|\ Pos\ (Atm\ c) = \{[]\}$
$|\ Pos\ (Left\ v) = \{[]\} \cup \{0\#ps \mid ps.\ ps \in Pos\ v\}$
$|\ Pos\ (Right\ v) = \{[]\} \cup \{1\#ps \mid ps.\ ps \in Pos\ v\}$
$|\ Pos\ (Seq\ v1\ v2) = \{[]\} \cup \{0\#ps \mid ps.\ ps \in Pos\ v1\} \cup \{1\#ps \mid ps.\ ps \in Pos\ v2\}$
$|\ Pos\ (Stars\ []) = \{[]\}$
$|\ Pos\ (Stars\ (v\#vs)) = \{[]\} \cup \{0\#ps \mid ps.\ ps \in Pos\ v\} \cup \{Suc\ n\#ps \mid n\ ps.\ n\#ps \in Pos\ (Stars\ vs)\}$

lemma *Pos-stars*:
  $Pos\ (Stars\ vs) = \{[]\} \cup (\bigcup n < length\ vs.\ \{n\#ps \mid ps.\ ps \in Pos\ (vs\ !\ n)\})$
$\langle proof \rangle$

lemma *Pos-empty*:
  shows $[] \in Pos\ v$
$\langle proof \rangle$

abbreviation
  $intlen\ vs \equiv int\ (length\ vs)$

definition *pflat-len* :: $'a$ *val* $\Rightarrow$ *nat list* $=>$ *int*
where
  $pflat\text{-}len\ v\ p \equiv (if\ p \in Pos\ v\ then\ intlen\ (flat\ (at\ v\ p))\ else\ -1)$

lemma *pflat-len-simps*:
  shows $pflat\text{-}len\ (Seq\ v1\ v2)\ (0\#p) = pflat\text{-}len\ v1\ p$
  and $pflat\text{-}len\ (Seq\ v1\ v2)\ (Suc\ 0\#p) = pflat\text{-}len\ v2\ p$
  and $pflat\text{-}len\ (Left\ v)\ (0\#p) = pflat\text{-}len\ v\ p$
  and $pflat\text{-}len\ (Left\ v)\ (Suc\ 0\#p) = -1$
  and $pflat\text{-}len\ (Right\ v)\ (Suc\ 0\#p) = pflat\text{-}len\ v\ p$
  and $pflat\text{-}len\ (Right\ v)\ (0\#p) = -1$
  and $pflat\text{-}len\ (Stars\ (v\#vs))\ (Suc\ n\#p) = pflat\text{-}len\ (Stars\ vs)\ (n\#p)$
  and $pflat\text{-}len\ (Stars\ (v\#vs))\ (0\#p) = pflat\text{-}len\ v\ p$
  and $pflat\text{-}len\ v\ [] = intlen\ (flat\ v)$
$\langle proof \rangle$

lemma *pflat-len-Stars-simps*:
  assumes $n < length\ vs$
  shows $pflat\text{-}len\ (Stars\ vs)\ (n\#p) = pflat\text{-}len\ (vs!n)\ p$
$\langle proof \rangle$

lemma *pflat-len-outside*:
  assumes $p \notin Pos\ v1$

shows *pflat-len v1 p* = −*1*
⟨*proof*⟩

12 Orderings

definition *prefix-list*:: ′*a list* ⇒ ′*a list* ⇒ *bool* (‹- ⊑*pre* -› [*60*,*59*] *60*)
where
ps1 ⊑*pre ps2* ≡ ∃ *ps*′. *ps1* @*ps*′ = *ps2*

definition *sprefix-list*:: ′*a list* ⇒ ′*a list* ⇒ *bool* (‹- ⊏*spre* -› [*60*,*59*] *60*)
where
ps1 ⊏*spre ps2* ≡ *ps1* ⊑*pre ps2* ∧ *ps1* ≠ *ps2*

inductive *lex-list* :: *nat list* ⇒ *nat list* ⇒ *bool* (‹- ⊏*lex* -› [*60*,*59*] *60*)
where
[] ⊏*lex* (*p*#*ps*)
| *ps1* ⊏*lex ps2* ⟹ (*p*#*ps1*) ⊏*lex* (*p*#*ps2*)
| *p1* < *p2* ⟹ (*p1*#*ps1*) ⊏*lex* (*p2*#*ps2*)

lemma *lex-irrfl*:
fixes *ps1 ps2* :: *nat list*
assumes *ps1* ⊏*lex ps2*
shows *ps1* ≠ *ps2*
⟨*proof*⟩

lemma *lex-simps* [*simp*]:
fixes *xs ys* :: *nat list*
shows [] ⊏*lex ys* ⟷ *ys* ≠ []
and *xs* ⊏*lex* [] ⟷ *False*
and (*x* # *xs*) ⊏*lex* (*y* # *ys*) ⟷ (*x* < *y* ∨ (*x* = *y* ∧ *xs* ⊏*lex ys*))
⟨*proof*⟩

lemma *lex-trans*:
fixes *ps1 ps2 ps3* :: *nat list*
assumes *ps1* ⊏*lex ps2 ps2* ⊏*lex ps3*
shows *ps1* ⊏*lex ps3*
⟨*proof*⟩

lemma *lex-trichotomous*:
fixes *p q* :: *nat list*
shows *p* = *q* ∨ *p* ⊏*lex q* ∨ *q* ⊏*lex p*
⟨*proof*⟩

13 POSIX Ordering of Values According to Okui & Suzuki

definition *PosOrd*:: $'a$ *val* $\Rightarrow$ *nat list* $\Rightarrow$ $'a$ *val* $\Rightarrow$ *bool* (‹- $\sqsubset$*val* - -› [*60*, *60*, *59*] *60*)
where
v1 $\sqsubset$*val p v2* $\equiv$ *pflat-len v1 p* $>$ *pflat-len v2 p* $\wedge$
($\forall$ *q* $\in$ *Pos v1* $\cup$ *Pos v2*. *q* $\sqsubset$*lex p* $\longrightarrow$ *pflat-len v1 q* $=$ *pflat-len v2 q*)

lemma *PosOrd-def2*:
shows *v1* $\sqsubset$*val p v2* $\longleftrightarrow$
pflat-len v1 p $>$ *pflat-len v2 p* $\wedge$
($\forall$ *q* $\in$ *Pos v1*. *q* $\sqsubset$*lex p* $\longrightarrow$ *pflat-len v1 q* $=$ *pflat-len v2 q*) $\wedge$
($\forall$ *q* $\in$ *Pos v2*. *q* $\sqsubset$*lex p* $\longrightarrow$ *pflat-len v1 q* $=$ *pflat-len v2 q*)
⟨*proof*⟩

definition *PosOrd-ex*:: $'a$ *val* $\Rightarrow$ $'a$ *val* $\Rightarrow$ *bool* (‹- :$\sqsubset$*val* -› [*60*, *59*] *60*)
where
v1 :$\sqsubset$*val v2* $\equiv$ $\exists$ *p*. *v1* $\sqsubset$*val p v2*

definition *PosOrd-ex-eq*:: $'a$ *val* $\Rightarrow$ $'a$ *val* $\Rightarrow$ *bool* (‹- :$\sqsubseteq$*val* -› [*60*, *59*] *60*)
where
v1 :$\sqsubseteq$*val v2* $\equiv$ *v1* :$\sqsubset$*val v2* $\vee$ *v1* $=$ *v2*

lemma *PosOrd-trans*:
assumes *v1* :$\sqsubset$*val v2 v2* :$\sqsubset$*val v3*
shows *v1* :$\sqsubset$*val v3*
⟨*proof*⟩

lemma *PosOrd-irrefl*:
assumes *v* :$\sqsubset$*val v*
shows *False*
⟨*proof*⟩

lemma *PosOrd-assym*:
assumes *v1* :$\sqsubset$*val v2*
shows $\neg$(*v2* :$\sqsubset$*val v1*)
⟨*proof*⟩

lemma *PosOrd-ordering*:
shows *ordering* (λ*v1 v2*. *v1* :$\sqsubseteq$*val v2*) (λ *v1 v2*. *v1* :$\sqsubset$*val v2*)
⟨*proof*⟩

lemma *PosOrd-order*:

shows *class.order* $(\lambda v1\ v2.\ v1 :\sqsubseteq val\ v2)$ $(\lambda\ v1\ v2.\ v1 :\sqsubset val\ v2)$
⟨*proof*⟩

lemma *PosOrd-ex-eq2*:
shows $v1 :\sqsubset val\ v2 \longleftrightarrow (v1 :\sqsubseteq val\ v2 \wedge v1 \neq v2)$
⟨*proof*⟩

lemma *PosOrdeq-trans*:
assumes $v1 :\sqsubseteq val\ v2$ $v2 :\sqsubseteq val\ v3$
shows $v1 :\sqsubseteq val\ v3$
⟨*proof*⟩

lemma *PosOrdeq-antisym*:
assumes $v1 :\sqsubseteq val\ v2$ $v2 :\sqsubseteq val\ v1$
shows $v1 = v2$
⟨*proof*⟩

lemma *PosOrdeq-refl*:
shows $v :\sqsubseteq val\ v$
⟨*proof*⟩

lemma *PosOrd-shorterE*:
assumes $v1 :\sqsubset val\ v2$
shows *length* (*flat* $v2$) $\leq$ *length* (*flat* $v1$)
⟨*proof*⟩

lemma *PosOrd-shorterI*:
assumes *length* (*flat* $v2$) $<$ *length* (*flat* $v1$)
shows $v1 :\sqsubset val\ v2$
⟨*proof*⟩

lemma *PosOrd-spreI*:
assumes *flat* $v' \sqsubset spre$ *flat* v
shows $v :\sqsubset val\ v'$
⟨*proof*⟩

lemma *pflat-len-inside*:
assumes *pflat-len* $v2\ p$ $<$ *pflat-len* $v1\ p$
shows $p \in Pos\ v1$
⟨*proof*⟩

lemma *PosOrd-Left-Right*:
assumes *flat* $v1$ = *flat* $v2$
shows *Left* $v1 :\sqsubset val$ *Right* $v2$
⟨*proof*⟩

lemma *PosOrd-LeftE*:
assumes *Left v1* $:\sqsubset$*val Left v2 flat v1* = *flat v2*
shows *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-LeftI*:
assumes *v1* $:\sqsubset$*val v2 flat v1* = *flat v2*
shows *Left v1* $:\sqsubset$*val Left v2*
⟨*proof*⟩

lemma *PosOrd-Left-eq*:
assumes *flat v1* = *flat v2*
shows *Left v1* $:\sqsubset$*val Left v2* $\longleftrightarrow$ *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-RightE*:
assumes *Right v1* $:\sqsubset$*val Right v2 flat v1* = *flat v2*
shows *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-RightI*:
assumes *v1* $:\sqsubset$*val v2 flat v1* = *flat v2*
shows *Right v1* $:\sqsubset$*val Right v2*
⟨*proof*⟩

lemma *PosOrd-Right-eq*:
assumes *flat v1* = *flat v2*
shows *Right v1* $:\sqsubset$*val Right v2* $\longleftrightarrow$ *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-SeqI1*:
assumes *v1* $:\sqsubset$*val w1 flat* (*Seq v1 v2*) = *flat* (*Seq w1 w2*)
shows *Seq v1 v2* $:\sqsubset$*val Seq w1 w2*
⟨*proof*⟩

lemma *PosOrd-SeqI2*:
assumes *v2* $:\sqsubset$*val w2 flat v2* = *flat w2*
shows *Seq v v2* $:\sqsubset$*val Seq v w2*
⟨*proof*⟩

lemma *PosOrd-Seq-eq*:
assumes *flat v2* = *flat w2*
shows (*Seq v v2*) $:\sqsubset$*val* (*Seq v w2*) $\longleftrightarrow$ *v2* $:\sqsubset$*val w2*
⟨*proof*⟩

lemma *PosOrd-StarsI*:
  assumes $v1 :\sqsubset val\ v2$ $flats\ (v1\#vs1) = flats\ (v2\#vs2)$
  shows $Stars\ (v1\#vs1) :\sqsubset val\ Stars\ (v2\#vs2)$
⟨*proof*⟩

lemma *PosOrd-StarsI2*:
  assumes $Stars\ vs1 :\sqsubset val\ Stars\ vs2$ $flats\ vs1 = flats\ vs2$
  shows $Stars\ (v\#vs1) :\sqsubset val\ Stars\ (v\#vs2)$
⟨*proof*⟩

lemma *PosOrd-Stars-appendI*:
  assumes $Stars\ vs1 :\sqsubset val\ Stars\ vs2$ $flat\ (Stars\ vs1) = flat\ (Stars\ vs2)$
  shows $Stars\ (vs\ @\ vs1) :\sqsubset val\ Stars\ (vs\ @\ vs2)$
⟨*proof*⟩

lemma *PosOrd-StarsE2*:
  assumes $Stars\ (v\ \#\ vs1) :\sqsubset val\ Stars\ (v\ \#\ vs2)$
  shows $Stars\ vs1 :\sqsubset val\ Stars\ vs2$
⟨*proof*⟩

lemma *PosOrd-Stars-appendE*:
  assumes $Stars\ (vs\ @\ vs1) :\sqsubset val\ Stars\ (vs\ @\ vs2)$
  shows $Stars\ vs1 :\sqsubset val\ Stars\ vs2$
⟨*proof*⟩

lemma *PosOrd-Stars-append-eq*:
  assumes $flats\ vs1 = flats\ vs2$
  shows $Stars\ (vs\ @\ vs1) :\sqsubset val\ Stars\ (vs\ @\ vs2) \longleftrightarrow Stars\ vs1 :\sqsubset val\ Stars\ vs2$
⟨*proof*⟩

lemma *PosOrd-almost-trichotomous*:
  shows $v1 :\sqsubset val\ v2 \lor v2 :\sqsubset val\ v1 \lor (length\ (flat\ v1) = length\ (flat\ v2))$
⟨*proof*⟩

14 The Posix Value is smaller than any other lexical value

lemma *Posix-PosOrd*:
  assumes $s \in r \to v1$ $v2 \in LV\ r\ s$
  shows $v1 :\sqsubseteq val\ v2$
⟨*proof*⟩

lemma *Posix-PosOrd-reverse*:
  assumes $s \in r \to v1$
  shows $\neg(\exists\, v2 \in LV\ r\ s.\ v2 :\sqsubset val\ v1)$
⟨*proof*⟩

lemma *PosOrd-Posix*:
assumes $v1 \in LV\ r\ s\ \forall\, v_2 \in LV\ r\ s.\ \neg\ v_2\ :\sqsubset val\ v1$
shows $s \in r \rightarrow v1$
⟨*proof*⟩

lemma *Least-existence*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists\, vmin \in LV\ r\ s.\ \forall\, v \in LV\ r\ s.\ vmin\ :\sqsubseteq val\ v$
⟨*proof*⟩

lemma *Least-existence1*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists!\ v_{min} \in LV\ r\ s.\ \forall\, v \in LV\ r\ s.\ v_{min}\ :\sqsubseteq val\ v$
⟨*proof*⟩

lemma *Least-existence2*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists! vmin \in LV\ r\ s.\ lexer\ r\ s = Some\ vmin \wedge (\forall\, v \in LV\ r\ s.\ vmin\ :\sqsubseteq val\ v)$
⟨*proof*⟩

lemma *Least-existence1-pre*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists! vmin \in LV\ r\ s.\ \forall\, v \in (LV\ r\ s \cup \{v'.\ flat\ v' \sqsubset spre\ s\}).\ vmin\ :\sqsubseteq val\ v$
⟨*proof*⟩

lemma *PosOrd-partial*:
shows $partial\text{-}order\text{-}on\ UNIV\ \{(v1,\ v2).\ v1\ :\sqsubseteq val\ v2\}$
⟨*proof*⟩

lemma *PosOrd-wf*:
shows $wf\ \{(v1,\ v2).\ v1\ :\sqsubset val\ v2 \wedge v1 \in LV\ r\ s \wedge v2 \in LV\ r\ s\}$
⟨*proof*⟩

unused-thms

end

15 Extended Regular Expressions 3

theory *Regular-Exps3*
imports *Regular−Sets.Regular-Set*
begin

datatype (*atoms*: *'a*) *rexp* =
is-Zero: *Zero* |
is-One: *One* |
Atom 'a |

Plus (*'a rexp*) (*'a rexp*) |
Times (*'a rexp*) (*'a rexp*) |
Star (*'a rexp*) |
NTimes (*'a rexp*) *nat* |
Upto (*'a rexp*) *nat* |
From (*'a rexp*) *nat* |
Rec string (*'a rexp*) |
Charset (*'a set*)

fun *lang* :: *'a rexp* => *'a lang* **where**
lang Zero = {} |
lang One = {[]} |
lang (*Atom a*) = {[*a*]} |
lang (*Plus r s*) = (*lang r*) *Un* (*lang s*) |
lang (*Times r s*) = *conc* (*lang r*) (*lang s*) |
lang (*Star r*) = *star*(*lang r*) |
lang (*NTimes r n*) = ((*lang r*) ^^ *n*) |
lang (*Upto r n*) = ($\bigcup$ *i* ∈ {..*n*}. (*lang r*) ^^ *i*) |
lang (*From r n*) = ($\bigcup$ *i* ∈ {*n*..}. (*lang r*) ^^ *i*) |
lang (*Rec l r*) = *lang r* |
lang (*Charset cs*) = {[*c*] | *c* . *c* ∈ *cs*}

primrec *nullable* :: *'a rexp* ⇒ *bool* **where**
nullable Zero = *False* |
nullable One = *True* |
nullable (*Atom c*) = *False* |
nullable (*Plus r1 r2*) = (*nullable r1* ∨ *nullable r2*) |
nullable (*Times r1 r2*) = (*nullable r1* ∧ *nullable r2*) |
nullable (*Star r*) = *True* |
nullable (*NTimes r n*) = (*if n = 0 then True else nullable r*) |
nullable (*Upto r n*) = *True* |
nullable (*From r n*) = (*if n = 0 then True else nullable r*) |
nullable (*Rec l r*) = *nullable r* |
nullable (*Charset cs*) = *False*

lemma *pow-empty-iff*:
  shows [] ∈ (*lang r*) ^^ *n* ⟷ (*if n = 0 then True else* [] ∈ (*lang r*))
  ⟨*proof*⟩

lemma *nullable-iff*:
  shows *nullable r* ⟷ [] ∈ *lang r*
  ⟨*proof*⟩

end

16 Derivatives of Extended Regular Expressions

theory *Derivatives3*

imports *Regular-Exps3*
begin

This theory is based on work by Brozowski.

16.1 Brzozowski's derivatives of regular expressions

fun
  $deriv :: 'a \Rightarrow 'a\ rexp \Rightarrow 'a\ rexp$
where
  $deriv\ c\ (Zero) = Zero$
$|$ $deriv\ c\ (One) = Zero$
$|$ $deriv\ c\ (Atom\ c') = (if\ c = c'\ then\ One\ else\ Zero)$
$|$ $deriv\ c\ (Plus\ r1\ r2) = Plus\ (deriv\ c\ r1)\ (deriv\ c\ r2)$
$|$ $deriv\ c\ (Times\ r1\ r2) =$
  $(if\ nullable\ r1\ then\ Plus\ (Times\ (deriv\ c\ r1)\ r2)\ (deriv\ c\ r2)\ else\ Times\ (deriv\ c\ r1)\ r2)$
$|$ $deriv\ c\ (Star\ r) = Times\ (deriv\ c\ r)\ (Star\ r)$
$|$ $deriv\ c\ (NTimes\ r\ n) = (if\ n = 0\ then\ Zero\ else\ Times\ (deriv\ c\ r)\ (NTimes\ r\ (n - 1)))$
$|$ $deriv\ c\ (Upto\ r\ n) = (if\ n = 0\ then\ Zero\ else\ Times\ (deriv\ c\ r)\ (Upto\ r\ (n - 1)))$
$|$ $deriv\ c\ (From\ r\ n) = (if\ n = 0\ then\ Times\ (deriv\ c\ r)\ (Star\ r)\ else\ Times\ (deriv\ c\ r)\ (From\ r\ (n - 1)))$
$|$ $deriv\ c\ (Rec\ l\ r) = deriv\ c\ r$
$|$ $deriv\ c\ (Charset\ cs) = (if\ c \in cs\ then\ One\ else\ Zero)$

fun
  $derivs :: 'a\ list \Rightarrow 'a\ rexp \Rightarrow 'a\ rexp$
where
  $derivs\ []\ r = r$
$|$ $derivs\ (c\ \#\ s)\ r = derivs\ s\ (deriv\ c\ r)$

lemma *deriv-pow* [*simp*]:
  shows $Deriv\ c\ (A\ \hat{}\hat{}\ n) = (if\ n = 0\ then\ \{\}\ else\ (Deriv\ c\ A)\ @@\ (A\ \hat{}\hat{}\ (n - 1)))$
  $\langle proof \rangle$

lemma *lang-deriv*: $lang\ (deriv\ c\ r) = Deriv\ c\ (lang\ r)$
  $\langle proof \rangle$

lemma *lang-derivs*: $lang\ (derivs\ s\ r) = Derivs\ s\ (lang\ r)$
$\langle proof \rangle$

A regular expression matcher:

definition $matcher :: 'a\ rexp \Rightarrow 'a\ list \Rightarrow bool$ **where**
$matcher\ r\ s = nullable\ (derivs\ s\ r)$

lemma *matcher-correctness*: *matcher r s* $\longleftrightarrow$ *s* $\in$ *lang r*
$\langle proof \rangle$

end

theory *Lexer3*
imports *Derivatives3*
begin

17 Values

datatype *'a val* =
Void
| *Atm 'a*
| *Seq 'a val 'a val*
| *Right 'a val*
| *Left 'a val*
| *Stars ('a val) list*
| *Recv string 'a val*

18 The string behind a value

fun
flat :: *'a val* $\Rightarrow$ *'a list*
where
flat (*Void*) = []
| *flat* (*Atm c*) = [*c*]
| *flat* (*Left v*) = *flat v*
| *flat* (*Right v*) = *flat v*
| *flat* (*Seq v1 v2*) = (*flat v1*) @ (*flat v2*)
| *flat* (*Stars* []) = []
| *flat* (*Stars* (*v#vs*)) = (*flat v*) @ (*flat* (*Stars vs*))
| *flat* (*Recv l v*) = *flat v*

abbreviation
flats vs $\equiv$ *concat* (*map flat vs*)

lemma *flat-Stars* [*simp*]:
flat (*Stars vs*) = *concat* (*map flat vs*)
$\langle proof \rangle$

lemma *flats-empty*:
assumes ($\forall$ *v*$\in$*set vs*. *flat v* = [])
shows *flats vs* = []
$\langle proof \rangle$

19 Relation between values and regular expressions

inductive
  Prf :: $'a\ val \Rightarrow 'a\ rexp \Rightarrow bool$ (‹⊢ - : -› [*100*, *100*] *100*)
where
 $[\![\vdash v1 : r1; \vdash v2 : r2]\!] \Longrightarrow \vdash Seq\ v1\ v2 : Times\ r1\ r2$
$|\ \vdash v1 : r1 \Longrightarrow \vdash Left\ v1 : Plus\ r1\ r2$
$|\ \vdash v2 : r2 \Longrightarrow \vdash Right\ v2 : Plus\ r1\ r2$
$|\ \vdash Void : One$
$|\ \vdash Atm\ c : Atom\ c$
$|\ [\![\forall v \in set\ vs.\ \vdash v : r \wedge flat\ v \neq []]\!] \Longrightarrow \vdash Stars\ vs : Star\ r$
$|\ [\![\forall v \in set\ vs1.\ \vdash v : r \wedge flat\ v \neq [];$
  $\forall v \in set\ vs2.\ \vdash v : r \wedge flat\ v = [];$
  $length\ (vs1\ @\ vs2) = n]\!] \Longrightarrow \vdash Stars\ (vs1\ @\ vs2) : NTimes\ r\ n$
$|\ [\![\forall v \in set\ vs.\ \vdash v : r \wedge flat\ v \neq []; length\ vs \leq n]\!] \Longrightarrow \vdash Stars\ vs : Upto\ r\ n$
$|\ [\![\forall v \in set\ vs1.\ \vdash v : r \wedge flat\ v \neq [];$
  $\forall v \in set\ vs2.\ \vdash v : r \wedge flat\ v = [];$
  $length\ (vs1\ @\ vs2) = n]\!] \Longrightarrow \vdash Stars\ (vs1\ @\ vs2) : From\ r\ n$
$|\ [\![\forall v \in set\ vs.\ \vdash v : r \wedge flat\ v \neq []; length\ vs > n]\!] \Longrightarrow \vdash Stars\ vs : From\ r\ n$
$|\ \vdash v : r \Longrightarrow \vdash Recv\ l\ v : Rec\ l\ r$
$|\ c \in cs \Longrightarrow \vdash Atm\ c : Charset\ cs$

inductive-cases *Prf-elims*:
  $\vdash v : Zero$
  $\vdash v : Times\ r1\ r2$
  $\vdash v : Plus\ r1\ r2$
  $\vdash v : One$
  $\vdash v : Atom\ c$
  $\vdash v : Star\ r$
  $\vdash v : NTimes\ r\ n$
  $\vdash v : Upto\ r\ n$
  $\vdash v : From\ r\ n$
  $\vdash v : Rec\ l\ r$
  $\vdash v : Charset\ cs$

lemma *Prf-NTimes-empty*:
  assumes $\forall v \in set\ vs.\ \vdash v : r \wedge flat\ v = []$
  and $length\ vs = n$
  shows $\vdash Stars\ vs : NTimes\ r\ n$
  ⟨*proof*⟩

lemma *Times-decomp*:
  assumes $s \in A\ @@\ B$
  shows $\exists s1\ s2.\ s = s1\ @\ s2 \wedge s1 \in A \wedge s2 \in B$
  ⟨*proof*⟩

lemma *pow-string*:

assumes $s \in A \frown\frown n$
shows $\exists ss.\ concat\ ss = s \wedge (\forall s \in set\ ss.\ s \in A) \wedge length\ ss = n$
$\langle proof \rangle$

lemma *pow-Prf*:
assumes $\forall v{\in}set\ vs.\ \vdash v : r \wedge flat\ v \in A$
shows $flats\ vs \in A \frown\frown (length\ vs)$
$\langle proof \rangle$

lemma *Star-string*:
assumes $s \in star\ A$
shows $\exists ss.\ concat\ ss = s \wedge (\forall s \in set\ ss.\ s \in A)$
$\langle proof \rangle$

lemma *Star-val*:
assumes $\forall s{\in}set\ ss.\ \exists v.\ s = flat\ v \wedge \vdash v : r$
shows $\exists vs.\ flats\ vs = concat\ ss \wedge (\forall v{\in}set\ vs.\ \vdash v : r \wedge flat\ v \neq [])$
$\langle proof \rangle$

lemma *Aux*:
assumes $\forall s{\in}set\ ss.\ s = []$
shows $concat\ ss = []$
$\langle proof \rangle$

lemma *pow-cstring*:
assumes $s \in A \frown\frown n$
shows $\exists ss1\ ss2.\ concat\ (ss1\ @\ ss2) = s \wedge length\ (ss1\ @\ ss2) = n \wedge$
$(\forall s \in set\ ss1.\ s \in A \wedge s \neq []) \wedge (\forall s \in set\ ss2.\ s \in A \wedge s = [])$
$\langle proof \rangle$

lemma *flats-cval*:
assumes $\forall s{\in}set\ ss.\ \exists v.\ s = flat\ v \wedge \vdash v : r$
shows $\exists vs1\ vs2.\ flats\ vs1 = concat\ ss \wedge length\ (vs1\ @\ vs2) = length\ ss \wedge$
$(\forall v{\in}set\ vs1.\ \vdash v : r \wedge flat\ v \neq []) \wedge$
$(\forall v{\in}set\ vs2.\ \vdash v : r \wedge flat\ v = [])$
$\langle proof \rangle$

lemma *flats-cval2*:
assumes $\forall s{\in}set\ ss.\ \exists v.\ s = flat\ v \wedge \vdash v : r$
shows $\exists vs.\ flats\ vs = concat\ ss \wedge length\ vs \leq length\ ss \wedge (\forall v{\in}set\ vs.\ \vdash v : r \wedge$
$flat\ v \neq [])$
$\langle proof \rangle$

lemma *Prf-flat-lang*:
assumes $\vdash v : r$ **shows** $flat\ v \in lang\ r$
$\langle proof \rangle$

lemma *L-flat-Prf2*:

assumes $s \in lang\ r$
shows $\exists v.\ \vdash v : r \wedge flat\ v = s$
$\langle proof \rangle$

lemma *L-flat-Prf*:
$lang\ r = \{flat\ v \mid v.\ \vdash v : r\}$
$\langle proof \rangle$

20 Sulzmann and Lu functions

fun
$mkeps :: 'a\ rexp \Rightarrow 'a\ val$
where
$mkeps(One) = Void$
$\mid mkeps(Times\ r1\ r2) = Seq\ (mkeps\ r1)\ (mkeps\ r2)$
$\mid mkeps(Plus\ r1\ r2) = (if\ nullable(r1)\ then\ Left\ (mkeps\ r1)\ else\ Right\ (mkeps\ r2))$
$\mid mkeps(Star\ r) = Stars\ []$
$\mid mkeps(Upto\ r\ n) = Stars\ []$
$\mid mkeps(NTimes\ r\ n) = Stars\ (replicate\ n\ (mkeps\ r))$
$\mid mkeps(From\ r\ n) = Stars\ (replicate\ n\ (mkeps\ r))$
$\mid mkeps(Rec\ l\ r) = Recv\ l\ (mkeps\ r)$

fun $injval :: 'a\ rexp \Rightarrow 'a \Rightarrow 'a\ val \Rightarrow 'a\ val$
where
$injval\ (Atom\ d)\ c\ Void = Atm\ c$
$\mid injval\ (Plus\ r1\ r2)\ c\ (Left\ v1) = Left(injval\ r1\ c\ v1)$
$\mid injval\ (Plus\ r1\ r2)\ c\ (Right\ v2) = Right(injval\ r2\ c\ v2)$
$\mid injval\ (Times\ r1\ r2)\ c\ (Seq\ v1\ v2) = Seq\ (injval\ r1\ c\ v1)\ v2$
$\mid injval\ (Times\ r1\ r2)\ c\ (Left\ (Seq\ v1\ v2)) = Seq\ (injval\ r1\ c\ v1)\ v2$
$\mid injval\ (Times\ r1\ r2)\ c\ (Right\ v2) = Seq\ (mkeps\ r1)\ (injval\ r2\ c\ v2)$
$\mid injval\ (Star\ r)\ c\ (Seq\ v\ (Stars\ vs)) = Stars\ ((injval\ r\ c\ v)\ \#\ vs)$
$\mid injval\ (NTimes\ r\ n)\ c\ (Seq\ v\ (Stars\ vs)) = Stars\ ((injval\ r\ c\ v)\ \#\ vs)$
$\mid injval\ (Upto\ r\ n)\ c\ (Seq\ v\ (Stars\ vs)) = Stars\ ((injval\ r\ c\ v)\ \#\ vs)$
$\mid injval\ (From\ r\ n)\ c\ (Seq\ v\ (Stars\ vs)) = Stars\ ((injval\ r\ c\ v)\ \#\ vs)$
$\mid injval\ (Rec\ l\ r)\ c\ v = Recv\ l\ (injval\ r\ c\ v)$
$\mid injval\ (Charset\ cs)\ c\ Void = Atm\ c$

21 Mkeps, injval

lemma *mkeps-flat*:
assumes $nullable(r)$
shows $flat\ (mkeps\ r) = []$
$\langle proof \rangle$

lemma *mkeps-nullable*:
assumes $nullable\ r$
shows $\vdash mkeps\ r : r$
$\langle proof \rangle$

lemma *Prf-injval-flat*:
assumes $\vdash v : deriv\ c\ r$
shows $flat\ (injval\ r\ c\ v) = c\ \#\ (flat\ v)$
$\langle proof \rangle$

lemma *Prf-injval*:
assumes $\vdash v : deriv\ c\ r$
shows $\vdash (injval\ r\ c\ v) : r$
$\langle proof \rangle$

22 Our Alternative Posix definition

inductive
Posix :: $'a\ list \Rightarrow 'a\ rexp \Rightarrow 'a\ val \Rightarrow bool$ (‹- ∈ - → -› $[100, 100, 100]\ 100$)
where
Posix-One: $[] \in One \rightarrow Void$
| *Posix-Atom*: $[c] \in (Atom\ c) \rightarrow (Atm\ c)$
| *Posix-Plus1*: $s \in r1 \rightarrow v \Longrightarrow s \in (Plus\ r1\ r2) \rightarrow (Left\ v)$
| *Posix-Plus2*: $[\![s \in r2 \rightarrow v;\ s \notin lang\ r1]\!] \Longrightarrow s \in (Plus\ r1\ r2) \rightarrow (Right\ v)$
| *Posix-Times*: $[\![s1 \in r1 \rightarrow v1;\ s2 \in r2 \rightarrow v2;$
$\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (s1\ @\ s_3) \in lang\ r1 \wedge s_4 \in lang\ r2)]\!] \Longrightarrow$
$(s1\ @\ s2) \in (Times\ r1\ r2) \rightarrow (Seq\ v1\ v2)$
| *Posix-Star1*: $[\![s1 \in r \rightarrow v;\ s2 \in Star\ r \rightarrow Stars\ vs;\ flat\ v \neq [];$
$\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (s1\ @\ s_3) \in lang\ r \wedge s_4 \in lang\ (Star\ r))]\!]$
$\Longrightarrow (s1\ @\ s2) \in Star\ r \rightarrow Stars\ (v\ \#\ vs)$
| *Posix-Star2*: $[] \in Star\ r \rightarrow Stars\ []$
| *Posix-NTimes1*: $[\![s1 \in r \rightarrow v;\ s2 \in NTimes\ r\ n \rightarrow Stars\ vs;\ flat\ v \neq [];$
$\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (s1\ @\ s_3) \in lang\ r \wedge s_4 \in lang\ (NTimes\ r\ n))]\!]$
$\Longrightarrow (s1\ @\ s2) \in NTimes\ r\ (n + 1) \rightarrow Stars\ (v\ \#\ vs)$
| *Posix-NTimes2*: $[\![\forall v \in set\ vs.\ [] \in r \rightarrow v;\ length\ vs = n]\!]$
$\Longrightarrow [] \in NTimes\ r\ n \rightarrow Stars\ vs$
| *Posix-Upto1*: $[\![s1 \in r \rightarrow v;\ s2 \in Upto\ r\ n \rightarrow Stars\ vs;\ flat\ v \neq [];$
$\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (s1\ @\ s_3) \in lang\ r \wedge s_4 \in lang\ (Upto\ r\ n))]\!]$
$\Longrightarrow (s1\ @\ s2) \in Upto\ r\ (n + 1) \rightarrow Stars\ (v\ \#\ vs)$
| *Posix-Upto2*: $[] \in Upto\ r\ n \rightarrow Stars\ []$
| *Posix-From2*: $[\![\forall v \in set\ vs.\ [] \in r \rightarrow v;\ length\ vs = n]\!]$
$\Longrightarrow [] \in From\ r\ n \rightarrow Stars\ vs$
| *Posix-From1*: $[\![s1 \in r \rightarrow v;\ s2 \in From\ r\ (n - 1) \rightarrow Stars\ vs;\ flat\ v \neq [];\ 0 < n;$
$\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (s1\ @\ s_3) \in lang\ r \wedge s_4 \in lang\ (From\ r\ (n - 1)))]\!]$
$\Longrightarrow (s1\ @\ s2) \in From\ r\ n \rightarrow Stars\ (v\ \#\ vs)$
| *Posix-From3*: $[\![s1 \in r \rightarrow v;\ s2 \in Star\ r \rightarrow Stars\ vs;\ flat\ v \neq [];$
$\neg(\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (s1\ @\ s_3) \in lang\ r \wedge s_4 \in lang\ (Star\ r))]\!]$
$\Longrightarrow (s1\ @\ s2) \in From\ r\ 0 \rightarrow Stars\ (v\ \#\ vs)$
| *Posix-Rec*: $s \in r \rightarrow v \Longrightarrow s \in (Rec\ l\ r) \rightarrow (Recv\ l\ v)$
| *Posix-Cset*: $c \in cs \Longrightarrow [c] \in (Charset\ cs) \rightarrow (Atm\ c)$

inductive-cases *Posix-elims*:
$s \in Zero \to v$
$s \in One \to v$
$s \in Atom\ c \to v$
$s \in Plus\ r1\ r2 \to v$
$s \in Times\ r1\ r2 \to v$
$s \in Star\ r \to v$
$s \in NTimes\ r\ n \to v$
$s \in Upto\ r\ n \to v$
$s \in From\ r\ n \to v$
$s \in Rec\ l\ r \to v$
$s \in Charset\ cs \to v$

lemma *Posix1*:
assumes $s \in r \to v$
shows $s \in lang\ r$ $flat\ v = s$
$\langle proof \rangle$

lemma *Posix1a*:
assumes $s \in r \to v$
shows $\vdash v : r$
$\langle proof \rangle$

lemma *Posix-mkeps*:
assumes *nullable r*
shows $[] \in r \to mkeps\ r$
$\langle proof \rangle$

lemma *List-eq-zipI*:
assumes $\forall (v1, v2) \in set\ (zip\ vs1\ vs2).\ v1 = v2$
and $length\ vs1 = length\ vs2$
shows $vs1 = vs2$
$\langle proof \rangle$

Our Posix definition determines a unique value.

lemma *Posix-determ*:
assumes $s \in r \to v1$ $s \in r \to v2$
shows $v1 = v2$
$\langle proof \rangle$

lemma *Posix-injval*:
assumes $s \in (deriv\ c\ r) \to v$
shows $(c \# s) \in r \to (injval\ r\ c\ v)$
$\langle proof \rangle$

23 The Lexer by Sulzmann and Lu

fun
  lexer :: $'a$ *rexp* $\Rightarrow$ $'a$ *list* $\Rightarrow$ ($'a$ *val*) *option*
where
  lexer r [] = (*if nullable r then Some*(*mkeps r*) *else None*)
| *lexer r* ($c\#s$) = (*case* (*lexer* (*deriv c r*) *s*) *of*
          None $\Rightarrow$ *None*
        | *Some*(*v*) $\Rightarrow$ *Some*(*injval r c v*))

lemma *lexer-correct-None*:
  shows $s \notin lang\ r \longleftrightarrow lexer\ r\ s = None$
⟨*proof*⟩

lemma *lexer-correct-Some*:
  shows $s \in lang\ r \longleftrightarrow (\exists\, v.\ lexer\ r\ s = Some(v) \wedge s \in r \rightarrow v)$
⟨*proof*⟩

lemma *lexer-correctness*:
  shows $(lexer\ r\ s = Some\ v) \longleftrightarrow s \in r \rightarrow v$
  and $(lexer\ r\ s = None) \longleftrightarrow \neg(\exists\, v.\ s \in r \rightarrow v)$
⟨*proof*⟩

end
theory *LexicalVals3*
  imports *Lexer3 HOL−Library.Sublist*
begin

24 Sets of Lexical Values

Shows that lexical values are finite for a given regex and string.

definition
  LV :: $'a$ *rexp* $\Rightarrow$ $'a$ *list* $\Rightarrow$ ($'a$ *val*) *set*
where $LV\ r\ s \equiv \{v.\ \vdash v : r \wedge flat\ v = s\}$

lemma *LV-simps*:
  shows $LV\ Zero\ s = \{\}$
  and $LV\ One\ s = (if\ s = []\ then\ \{Void\}\ else\ \{\})$
  and $LV\ (Atom\ c)\ s = (if\ s = [c]\ then\ \{Atm\ c\}\ else\ \{\})$
  and $LV\ (Plus\ r1\ r2)\ s = Left\ `\ LV\ r1\ s \cup Right\ `\ LV\ r2\ s$
  and $LV\ (NTimes\ r\ 0)\ s = (if\ s = []\ then\ \{Stars\ []\}\ else\ \{\})$
  and $LV\ (Rec\ l\ r)\ s = \{Recv\ l\ v \mid v.\ v \in LV\ r\ s\}$
  and $LV\ (Charset\ cs)\ s = (if\ length\ s = 1 \wedge (hd\ s) \in cs\ then\ \{Atm\ (hd\ s)\}\ else$
$\{\})$
⟨*proof*⟩

abbreviation
  Prefixes s ≡ {*s'*. *prefix s' s*}

abbreviation
  Suffixes s ≡ {*s'*. *suffix s' s*}

abbreviation
  SSuffixes s ≡ {*s'*. *strict-suffix s' s*}

lemma *Suffixes-cons* [*simp*]:
  shows *Suffixes* (*c* # *s*) = *Suffixes s* ∪ {*c* # *s*}
⟨*proof*⟩

lemma *finite-Suffixes*:
  shows *finite* (*Suffixes s*)
⟨*proof*⟩

lemma *finite-SSuffixes*:
  shows *finite* (*SSuffixes s*)
⟨*proof*⟩

lemma *finite-Prefixes*:
  shows *finite* (*Prefixes s*)
⟨*proof*⟩

lemma *LV-STAR-finite*:
  assumes ∀ *s*. *finite* (*LV r s*)
  shows *finite* (*LV* (*Star r*) *s*)
⟨*proof*⟩

definition
  Stars-Cons V Vs ≡ {*Stars* (*v* # *vs*) | *v vs*. *v* ∈ *V* ∧ *Stars vs* ∈ *Vs*}

definition
  Stars-Append Vs1 Vs2 ≡ {*Stars* (*vs1* @ *vs2*) | *vs1 vs2*. *Stars vs1* ∈ *Vs1* ∧ *Stars vs2* ∈ *Vs2*}

fun *Stars-Pow* :: (*'a val*) *set* ⇒ *nat* ⇒ (*'a val*) *set*
where
  Stars-Pow Vs 0 = {*Stars* []}
| *Stars-Pow Vs* (*Suc n*) = *Stars-Cons Vs* (*Stars-Pow Vs n*)

lemma *finite-Stars-Cons*:
  assumes *finite V finite Vs*
  shows *finite* (*Stars-Cons V Vs*)
  ⟨*proof*⟩

lemma *finite-Stars-Append*:
  assumes *finite Vs1 finite Vs2*
  shows *finite* (*Stars-Append Vs1 Vs2*)
  ⟨*proof*⟩

lemma *finite-Stars-Pow*:
  assumes *finite Vs*
  shows *finite* (*Stars-Pow Vs n*)
⟨*proof*⟩

lemma *LV-NTimes-5*:
  LV (*NTimes r n*) *s* ⊆ *Stars-Append* (*LV* (*Star r*) *s*) (⋃*i*≤*n*. *LV* (*NTimes r i*) [])
⟨*proof*⟩

lemma *LV-NTIMES-3*:
  shows *LV* (*NTimes r* (*Suc n*)) [] =
    (λ(*v*, *vs*). *Stars* (*v*#*vs*)) ‘ (*LV r* [] × (*Stars* −‘ (*LV* (*NTimes r n*) [])))
⟨*proof*⟩

lemma *finite-NTimes-empty*:
  assumes ⋀*s*. *finite* (*LV r s*)
  shows *finite* (*LV* (*NTimes r n*) [])
  ⟨*proof*⟩

lemma *LV-From-5*:
  shows *LV* (*From r n*) *s* ⊆ *Stars-Append* (*LV* (*Star r*) *s*) (⋃*i*≤*n*. *LV* (*From r i*) [])
⟨*proof*⟩

lemma *LV-FROMNTIMES-3*:
  shows *LV* (*From r* (*Suc n*)) [] =
    (λ(*v*,*vs*). *Stars* (*v*#*vs*)) ‘ (*LV r* [] × (*Stars* −‘ (*LV* (*From r n*) [])))
⟨*proof*⟩

lemma *LV-From-empty*:
 LV (*From r n*) [] = *Stars-Pow* (*LV r* []) *n*
  ⟨*proof*⟩

lemma *finite-From-empty*:
  assumes ∀ *s*. *finite* (*LV r s*)
  shows *finite* (*LV* (*From r n*) *s*)
  ⟨*proof*⟩

lemma *subseteq-Upto-Star*:
  shows *LV* (*Upto r n*) *s* ⊆ *LV* (*Star r*) *s*
  ⟨*proof*⟩

lemma *LV-finite*:
  shows *finite* (*LV r s*)
⟨*proof*⟩

Our POSIX values are lexical values.

lemma *Posix-LV*:
  assumes $s \in r \to v$
  shows $v \in LV\ r\ s$
  ⟨*proof*⟩

lemma *Posix-Prf*:
  assumes $s \in r \to v$
  shows $\vdash v : r$
  ⟨*proof*⟩

end

theory *Simplifying3*
  imports *Lexer3*
begin

25 Lexer including simplifications

fun *F-RIGHT* **where**
  F-RIGHT f v = *Right* (*f v*)

fun *F-LEFT* **where**
  F-LEFT f v = *Left* (*f v*)

fun *F-Plus* **where**
  F-Plus $f_1\ f_2$ (*Right v*) = *Right* ($f_2\ v$)
| *F-Plus* $f_1\ f_2$ (*Left v*) = *Left* ($f_1\ v$)
| *F-Plus f1 f2 v* = *v*

fun *F-Times1* **where**
  F-Times1 $f_1\ f_2\ v$ = *Seq* (f_1 *Void*) ($f_2\ v$)

fun *F-Times2* **where**
  F-Times2 $f_1\ f_2\ v$ = *Seq* ($f_1\ v$) (f_2 *Void*)

fun *F-Times* **where**
  F-Times $f_1\ f_2$ (*Seq* $v_1\ v_2$) = *Seq* ($f_1\ v_1$) ($f_2\ v_2$)
| *F-Times f1 f2 v* = *v*

fun *simp-Plus* **where**

$simp\text{-}Plus\ (Zero,\ f_1)\ (r_2,\ f_2) = (r_2,\ F\text{-}RIGHT\ f_2)$
$|\ simp\text{-}Plus\ (r_1,\ f_1)\ (Zero,\ f_2) = (r_1,\ F\text{-}LEFT\ f_1)$
$|\ simp\text{-}Plus\ (r_1,\ f_1)\ (r_2,\ f_2) =$
$(if\ r_1 = r_2\ then\ (r_1,\ F\text{-}LEFT\ f_1)\ else\ (Plus\ r_1\ r_2,\ F\text{-}Plus\ f_1\ f_2))$

fun *simp-Times* **where**
$simp\text{-}Times\ (Zero,\ f_1)\ (r_2,\ f_2) = (Zero,\ undefined)$
$|\ simp\text{-}Times\ (r_1,\ f_1)\ (Zero,\ f_2) = (Zero,\ undefined)$
$|\ simp\text{-}Times\ (One,\ f_1)\ (r_2,\ f_2) = (r_2,\ F\text{-}Times1\ f_1\ f_2)$
$|\ simp\text{-}Times\ (r_1,\ f_1)\ (One,\ f_2) = (r_1,\ F\text{-}Times2\ f_1\ f_2)$
$|\ simp\text{-}Times\ (r_1,\ f_1)\ (r_2,\ f_2) = (Times\ r_1\ r_2,\ F\text{-}Times\ f_1\ f_2)$

lemma *simp-Times-simps*[*simp*]:
$simp\text{-}Times\ p1\ p2 = (if\ (fst\ p1 = Zero)\ then\ (Zero,\ undefined)$
$else\ (if\ (fst\ p2 = Zero)\ then\ (Zero,\ undefined)$
$else\ (if\ (fst\ p1 = One)\ then\ (fst\ p2,\ F\text{-}Times1\ (snd\ p1)\ (snd\ p2))$
$else\ (if\ (fst\ p2 = One)\ then\ (fst\ p1,\ F\text{-}Times2\ (snd\ p1)\ (snd\ p2))$
$else\ (Times\ (fst\ p1)\ (fst\ p2),\ F\text{-}Times\ (snd\ p1)\ (snd\ p2))))))$
⟨*proof*⟩

lemma *simp-Plus-simps*[*simp*]:
$simp\text{-}Plus\ p1\ p2 = (if\ (fst\ p1 = Zero)\ then\ (fst\ p2,\ F\text{-}RIGHT\ (snd\ p2))$
$else\ (if\ (fst\ p2 = Zero)\ then\ (fst\ p1,\ F\text{-}LEFT\ (snd\ p1))$
$else\ (if\ (fst\ p1 = fst\ p2)\ then\ (fst\ p1,\ F\text{-}LEFT\ (snd\ p1))$
$else\ (Plus\ (fst\ p1)\ (fst\ p2),\ F\text{-}Plus\ (snd\ p1)\ (snd\ p2)))))$
⟨*proof*⟩

fun
$simp :: 'a\ rexp \Rightarrow 'a\ rexp * ('a\ val \Rightarrow 'a\ val)$
where
$simp\ (Plus\ r1\ r2) = simp\text{-}Plus\ (simp\ r1)\ (simp\ r2)$
$|\ simp\ (Times\ r1\ r2) = simp\text{-}Times\ (simp\ r1)\ (simp\ r2)$
$|\ simp\ r = (r,\ id)$

fun
$slexer :: 'a\ rexp \Rightarrow 'a\ list \Rightarrow ('a\ val)\ option$
where
$slexer\ r\ [] = (if\ nullable\ r\ then\ Some(mkeps\ r)\ else\ None)$
$|\ slexer\ r\ (c\#s) = (let\ (rs,\ fr) = simp\ (deriv\ c\ r)\ in$
$(case\ (slexer\ rs\ s)\ of$
$None \Rightarrow None$
$|\ Some(v) \Rightarrow Some(injval\ r\ c\ (fr\ v))))$

lemma *slexer-better-simp*:
$slexer\ r\ (c\#s) = (case\ (slexer\ (fst\ (simp\ (deriv\ c\ r)))\ s)\ of$
$None \Rightarrow None$
$|\ Some(v) \Rightarrow Some(injval\ r\ c\ ((snd\ (simp\ (deriv\ c\ r)))\ v)))$
⟨*proof*⟩

lemma *L-fst-simp*:
  shows *lang r* = *lang* (*fst* (*simp r*))
⟨*proof*⟩

lemma *Posix-simp*:
  assumes $s \in (fst\ (simp\ r)) \to v$
  shows $s \in r \to ((snd\ (simp\ r))\ v)$
⟨*proof*⟩

lemma *slexer-correctness*:
  shows *slexer r s* = *lexer r s*
⟨*proof*⟩

end

theory *Positions3*
  imports *Lexer3 LexicalVals3*
begin

26 An alternative definition for POSIX values by Okui & Suzuki

27 Positions in Values

fun
  at :: $'a\ val \Rightarrow nat\ list \Rightarrow 'a\ val$
where
  at v [] = *v*
| *at* (*Left v*) (*0#ps*)= *at v ps*
| *at* (*Right v*) (*Suc 0#ps*)= *at v ps*
| *at* (*Seq v1 v2*) (*0#ps*)= *at v1 ps*
| *at* (*Seq v1 v2*) (*Suc 0#ps*)= *at v2 ps*
| *at* (*Stars vs*) (*n#ps*) = *at* (*nth vs n*) *ps*
| *at* (*Recv l v*) *ps* = *at v ps*

fun *Pos* :: $'a\ val \Rightarrow (nat\ list)\ set$
where
  Pos (*Void*) = {[]}
| *Pos* (*Atm c*) = {[]}
| *Pos* (*Left v*) = {[]} ∪ {*0#ps* | *ps. ps* ∈ *Pos v*}
| *Pos* (*Right v*) = {[]} ∪ {*1#ps* | *ps. ps* ∈ *Pos v*}
| *Pos* (*Seq v1 v2*) = {[]} ∪ {*0#ps* | *ps. ps* ∈ *Pos v1*} ∪ {*1#ps* | *ps. ps* ∈ *Pos v2*}
| *Pos* (*Stars* []) = {[]}
| *Pos* (*Stars* (*v#vs*)) = {[]} ∪ {*0#ps* | *ps. ps* ∈ *Pos v*} ∪ {*Suc n#ps* | *n ps. n#ps*

$\in$ *Pos* (*Stars vs*)}
| *Pos* (*Recv l v*) = {[]} ∪ {*ps* . *ps* ∈ *Pos v*}

lemma *Pos-stars*:
Pos (*Stars vs*) = {[]} ∪ (⋃ *n* < *length vs*. {*n*#*ps* | *ps*. *ps* ∈ *Pos* (*vs* ! *n*)})
⟨*proof*⟩

lemma *Pos-empty*:
shows [] ∈ *Pos v*
⟨*proof*⟩

abbreviation
intlen vs ≡ *int* (*length vs*)

definition *pflat-len* :: *'a val* ⇒ *nat list* => *int*
where
pflat-len v p ≡ (*if p* ∈ *Pos v then intlen* (*flat* (*at v p*)) *else* −*1*)

lemma *pflat-len-simps*:
shows *pflat-len* (*Seq v1 v2*) (*0*#*p*) = *pflat-len v1 p*
and *pflat-len* (*Seq v1 v2*) (*Suc 0*#*p*) = *pflat-len v2 p*
and *pflat-len* (*Left v*) (*0*#*p*) = *pflat-len v p*
and *pflat-len* (*Left v*) (*Suc 0*#*p*) = −*1*
and *pflat-len* (*Right v*) (*Suc 0*#*p*) = *pflat-len v p*
and *pflat-len* (*Right v*) (*0*#*p*) = −*1*
and *pflat-len* (*Stars* (*v*#*vs*)) (*Suc n*#*p*) = *pflat-len* (*Stars vs*) (*n*#*p*)
and *pflat-len* (*Stars* (*v*#*vs*)) (*0*#*p*) = *pflat-len v p*
and *pflat-len* (*Recv l v*) *p* = *pflat-len v p*
and *pflat-len v* [] = *intlen* (*flat v*)
⟨*proof*⟩

lemma *pflat-len-Stars-simps*:
assumes *n* < *length vs*
shows *pflat-len* (*Stars vs*) (*n*#*p*) = *pflat-len* (*vs*!*n*) *p*
⟨*proof*⟩

lemma *pflat-len-outside*:
assumes *p* ∉ *Pos v1*
shows *pflat-len v1 p* = −*1*
⟨*proof*⟩

28 Orderings

definition *prefix-list*:: *'a list* ⇒ *'a list* ⇒ *bool* (‹- ⊑*pre* -› [*60*,*59*] *60*)
where
ps1 ⊑*pre ps2* ≡ ∃ *ps'*. *ps1* @*ps'* = *ps2*

definition *sprefix-list*:: $'a$ *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *bool* (‹- $\sqsubset$*spre* -› [*60*,*59*] *60*)
where
ps1 $\sqsubset$*spre* *ps2* $\equiv$ *ps1* $\sqsubseteq$*pre* *ps2* $\wedge$ *ps1* $\neq$ *ps2*

inductive *lex-list* :: *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *bool* (‹- $\sqsubset$*lex* -› [*60*,*59*] *60*)
where
[] $\sqsubset$*lex* (*p*#*ps*)
| *ps1* $\sqsubset$*lex* *ps2* $\Longrightarrow$ (*p*#*ps1*) $\sqsubset$*lex* (*p*#*ps2*)
| *p1* < *p2* $\Longrightarrow$ (*p1*#*ps1*) $\sqsubset$*lex* (*p2*#*ps2*)

lemma *lex-irrfl*:
fixes *ps1 ps2* :: *nat list*
assumes *ps1* $\sqsubset$*lex* *ps2*
shows *ps1* $\neq$ *ps2*
⟨*proof*⟩

lemma *lex-simps* [*simp*]:
fixes *xs ys* :: *nat list*
shows [] $\sqsubset$*lex* *ys* $\longleftrightarrow$ *ys* $\neq$ []
and *xs* $\sqsubset$*lex* [] $\longleftrightarrow$ *False*
and (*x* # *xs*) $\sqsubset$*lex* (*y* # *ys*) $\longleftrightarrow$ (*x* < *y* $\vee$ (*x* = *y* $\wedge$ *xs* $\sqsubset$*lex* *ys*))
⟨*proof*⟩

lemma *lex-trans*:
fixes *ps1 ps2 ps3* :: *nat list*
assumes *ps1* $\sqsubset$*lex* *ps2* *ps2* $\sqsubset$*lex* *ps3*
shows *ps1* $\sqsubset$*lex* *ps3*
⟨*proof*⟩

lemma *lex-trichotomous*:
fixes *p q* :: *nat list*
shows *p* = *q* $\vee$ *p* $\sqsubset$*lex* *q* $\vee$ *q* $\sqsubset$*lex* *p*
⟨*proof*⟩

29 POSIX Ordering of Values According to Okui & Suzuki

definition *PosOrd*:: $'a$ *val* $\Rightarrow$ *nat list* $\Rightarrow$ $'a$ *val* $\Rightarrow$ *bool* (‹- $\sqsubset$*val* - -› [*60*, *60*, *59*] *60*)
where
v1 $\sqsubset$*val* *p* *v2* $\equiv$ *pflat-len v1 p* > *pflat-len v2 p* $\wedge$
($\forall$ *q* $\in$ *Pos v1* $\cup$ *Pos v2*. *q* $\sqsubset$*lex* *p* $\longrightarrow$ *pflat-len v1 q* = *pflat-len v2 q*)

lemma *PosOrd-def2*:
shows *v1* $\sqsubset$*val* *p* *v2* $\longleftrightarrow$

$pflat\text{-}len\ v1\ p > pflat\text{-}len\ v2\ p \wedge$
$(\forall q \in Pos\ v1.\ q \sqsubset lex\ p \longrightarrow pflat\text{-}len\ v1\ q = pflat\text{-}len\ v2\ q) \wedge$
$(\forall q \in Pos\ v2.\ q \sqsubset lex\ p \longrightarrow pflat\text{-}len\ v1\ q = pflat\text{-}len\ v2\ q)$
⟨*proof*⟩

definition *PosOrd-ex*:: $'a\ val \Rightarrow 'a\ val \Rightarrow bool$ (‹- :$\sqsubset$*val* -› [60, 59] 60)
where
$v1 :\sqsubset val\ v2 \equiv \exists p.\ v1 \sqsubset val\ p\ v2$

definition *PosOrd-ex-eq*:: $'a\ val \Rightarrow 'a\ val \Rightarrow bool$ (‹- :$\sqsubseteq$*val* -› [60, 59] 60)
where
$v1 :\sqsubseteq val\ v2 \equiv v1 :\sqsubset val\ v2 \vee v1 = v2$

lemma *PosOrd-trans*:
assumes $v1 :\sqsubset val\ v2\ v2 :\sqsubset val\ v3$
shows $v1 :\sqsubset val\ v3$
⟨*proof*⟩

lemma *PosOrd-irrefl*:
assumes $v :\sqsubset val\ v$
shows *False*
⟨*proof*⟩

lemma *PosOrd-assym*:
assumes $v1 :\sqsubset val\ v2$
shows $\neg(v2 :\sqsubset val\ v1)$
⟨*proof*⟩

lemma *PosOrd-ordering*:
shows $ordering\ (\lambda v1\ v2.\ v1 :\sqsubseteq val\ v2)\ (\lambda\ v1\ v2.\ v1 :\sqsubset val\ v2)$
⟨*proof*⟩

lemma *PosOrd-order*:
shows $class.order\ (\lambda v1\ v2.\ v1 :\sqsubseteq val\ v2)\ (\lambda\ v1\ v2.\ v1 :\sqsubset val\ v2)$
⟨*proof*⟩

lemma *PosOrd-ex-eq2*:
shows $v1 :\sqsubset val\ v2 \longleftrightarrow (v1 :\sqsubseteq val\ v2 \wedge v1 \neq v2)$
⟨*proof*⟩

lemma *PosOrdeq-trans*:
assumes $v1 :\sqsubseteq val\ v2\ v2 :\sqsubseteq val\ v3$
shows $v1 :\sqsubseteq val\ v3$
⟨*proof*⟩

lemma *PosOrdeq-antisym*:
  assumes *v1* :$\sqsubseteq$*val v2 v2* :$\sqsubseteq$*val v1*
  shows *v1* = *v2*
⟨*proof*⟩

lemma *PosOrdeq-refl*:
  shows *v* :$\sqsubseteq$*val v*
⟨*proof*⟩

lemma *PosOrd-shorterE*:
  assumes *v1* :$\sqsubset$*val v2*
  shows *length* (*flat v2*) ≤ *length* (*flat v1*)
⟨*proof*⟩

lemma *PosOrd-shorterI*:
  assumes *length* (*flat v2*) < *length* (*flat v1*)
  shows *v1* :$\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-spreI*:
  assumes *flat v′* $\sqsubset$*spre flat v*
  shows *v* :$\sqsubset$*val v′*
⟨*proof*⟩

lemma *pflat-len-inside*:
  assumes *pflat-len v2 p* < *pflat-len v1 p*
  shows *p* ∈ *Pos v1*
⟨*proof*⟩

lemma *PosOrd-Rec-eq*:
  assumes *flat v1* = *flat v2*
  shows *Recv l v1* :$\sqsubset$*val Recv l v2* ⟷ *v1* :$\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-Left-Right*:
  assumes *flat v1* = *flat v2*
  shows *Left v1* :$\sqsubset$*val Right v2*
⟨*proof*⟩

lemma *PosOrd-LeftE*:
  assumes *Left v1* :$\sqsubset$*val Left v2 flat v1* = *flat v2*
  shows *v1* :$\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-LeftI*:
  assumes *v1* :$\sqsubset$*val v2 flat v1* = *flat v2*
  shows *Left v1* :$\sqsubset$*val Left v2*

⟨*proof*⟩

lemma *PosOrd-Left-eq*:
  assumes *flat v1 = flat v2*
  shows *Left v1* $:\sqsubset$*val Left v2* $\longleftrightarrow$ *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-RightE*:
  assumes *Right v1* $:\sqsubset$*val Right v2 flat v1 = flat v2*
  shows *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-RightI*:
  assumes *v1* $:\sqsubset$*val v2 flat v1 = flat v2*
  shows *Right v1* $:\sqsubset$*val Right v2*
⟨*proof*⟩

lemma *PosOrd-Right-eq*:
  assumes *flat v1 = flat v2*
  shows *Right v1* $:\sqsubset$*val Right v2* $\longleftrightarrow$ *v1* $:\sqsubset$*val v2*
⟨*proof*⟩

lemma *PosOrd-SeqI1*:
  assumes *v1* $:\sqsubset$*val w1 flat (Seq v1 v2) = flat (Seq w1 w2)*
  shows *Seq v1 v2* $:\sqsubset$*val Seq w1 w2*
⟨*proof*⟩

lemma *PosOrd-SeqI2*:
  assumes *v2* $:\sqsubset$*val w2 flat v2 = flat w2*
  shows *Seq v v2* $:\sqsubset$*val Seq v w2*
⟨*proof*⟩

lemma *PosOrd-Seq-eq*:
  assumes *flat v2 = flat w2*
  shows *(Seq v v2)* $:\sqsubset$*val (Seq v w2)* $\longleftrightarrow$ *v2* $:\sqsubset$*val w2*
⟨*proof*⟩

lemma *PosOrd-StarsI*:
  assumes *v1* $:\sqsubset$*val v2 flats (v1#vs1) = flats (v2#vs2)*
  shows *Stars (v1#vs1)* $:\sqsubset$*val Stars (v2#vs2)*
⟨*proof*⟩

lemma *PosOrd-StarsI2*:
  assumes *Stars vs1* $:\sqsubset$*val Stars vs2 flats vs1 = flats vs2*

shows *Stars* $(v\#vs1)$ $:\sqsubset val$ *Stars* $(v\#vs2)$
⟨*proof*⟩

lemma *PosOrd-Stars-appendI*:
assumes *Stars vs1* $:\sqsubset val$ *Stars vs2 flat* (*Stars vs1*) = *flat* (*Stars vs2*)
shows *Stars* (*vs* @ *vs1*) $:\sqsubset val$ *Stars* (*vs* @ *vs2*)
⟨*proof*⟩

lemma *PosOrd-StarsE2*:
assumes *Stars* (*v* # *vs1*) $:\sqsubset val$ *Stars* (*v* # *vs2*)
shows *Stars vs1* $:\sqsubset val$ *Stars vs2*
⟨*proof*⟩

lemma *PosOrd-Stars-appendE*:
assumes *Stars* (*vs* @ *vs1*) $:\sqsubset val$ *Stars* (*vs* @ *vs2*)
shows *Stars vs1* $:\sqsubset val$ *Stars vs2*
⟨*proof*⟩

lemma *PosOrd-Stars-append-eq*:
assumes *flats vs1* = *flats vs2*
shows *Stars* (*vs* @ *vs1*) $:\sqsubset val$ *Stars* (*vs* @ *vs2*) $\longleftrightarrow$ *Stars vs1* $:\sqsubset val$ *Stars vs2*
⟨*proof*⟩

lemma *PosOrd-Stars-equalsI*:
assumes *flats vs1* = *flats vs2 length vs1* = *length vs2*
and *list-all2* ($\lambda v1\ v2.\ v1 :\sqsubseteq val\ v2$) *vs1 vs2*
shows *Stars vs1* $:\sqsubseteq val$ *Stars vs2*
⟨*proof*⟩

lemma *PosOrd-almost-trichotomous*:
shows $v1 :\sqsubset val\ v2 \lor v2 :\sqsubset val\ v1 \lor (length\ (flat\ v1) = length\ (flat\ v2))$
⟨*proof*⟩

30 The Posix Value is smaller than any other lexical value

lemma *Posix-PosOrd*:
assumes $s \in r \to v1$ $v2 \in LV\ r\ s$
shows $v1 :\sqsubseteq val\ v2$
⟨*proof*⟩

lemma *Posix-PosOrd-reverse*:
assumes $s \in r \to v1$
shows $\neg(\exists\, v2 \in LV\ r\ s.\ v2 :\sqsubset val\ v1)$
⟨*proof*⟩

lemma *PosOrd-Posix*:

assumes $v1 \in LV\ r\ s\ \forall\, v_2 \in LV\ r\ s.\ \neg\ v_2 :\sqsubset val\ v1$
shows $s \in r \to v1$
$\langle proof \rangle$

lemma *Least-existence*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists\, vmin \in LV\ r\ s.\ \forall\, v \in LV\ r\ s.\ vmin :\sqsubseteq val\ v$
$\langle proof \rangle$

lemma *Least-existence1*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists!\ v_{min} \in LV\ r\ s.\ \forall\, v \in LV\ r\ s.\ v_{min} :\sqsubseteq val\ v$
$\langle proof \rangle$

lemma *Least-existence2*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists!vmin \in LV\ r\ s.\ lexer\ r\ s = Some\ vmin \wedge (\forall\, v \in LV\ r\ s.\ vmin :\sqsubseteq val\ v)$
$\langle proof \rangle$

lemma *Least-existence1-pre*:
assumes $LV\ r\ s \neq \{\}$
shows $\exists!vmin \in LV\ r\ s.\ \forall\, v \in (LV\ r\ s \cup \{v'.\ flat\ v' \sqsubset spre\ s\}).\ vmin :\sqsubseteq val\ v$
$\langle proof \rangle$

lemma *PosOrd-partial*:
shows $partial\text{-}order\text{-}on\ UNIV\ \{(v1,\ v2).\ v1 :\sqsubseteq val\ v2\}$
$\langle proof \rangle$

lemma *PosOrd-wf*:
shows $wf\ \{(v1,\ v2).\ v1 :\sqsubset val\ v2 \wedge v1 \in LV\ r\ s \wedge v2 \in LV\ r\ s\}$
$\langle proof \rangle$

unused-thms

end

References

[1] S. Okui and T. Suzuki. Disambiguation in Regular Expression Matching via Position Automata with Augmented Transitions. In *Proc. of the 15th International Conference on Implementation and Application of Automata (CIAA)*, volume 6482 of *LNCS*, pages 231–240, 2010.

[2] M. Sulzmann and K. Lu. POSIX Regular Expression Parsing with Derivatives. In *Proc. of the 12th International Conference on Functional and Logic Programming (FLOPS)*, volume 8475 of *LNCS*, pages 203–220, 2014.