POSIX Lexing with Derivatives of Regular Expressions

Fahad Ausaf Roy Dyckhoff Christian Urban

May 26, 2024

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** $flat\ v \in lang\ r$
⟨*proof*⟩

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

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

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

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

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

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 ⊢ *mkeps r* : *r*
⟨*proof*⟩

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

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

lemma *Prf-injval*:
  assumes ⊢ *v* : *deriv c r*
  shows ⊢ (*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*;
    ¬(∃ s_3 s_4. s_3 ≠ [] ∧ s_3 @ s_4 = *s2* ∧ (*s1* @ s_3) ∈ *lang r1* ∧ s_4 ∈ *lang r2*)⟧ ⟹
    (*s1* @ *s2*) ∈ (*Times r1 r2*) → (*Seq v1 v2*)
| *Posix-Star1*: ⟦*s1* ∈ *r* → *v*; *s2* ∈ *Star r* → *Stars vs*; *flat v* ≠ [];
    ¬(∃ s_3 s_4. s_3 ≠ [] ∧ s_3 @ s_4 = *s2* ∧ (*s1* @ s_3) ∈ *lang r* ∧ s_4 ∈ *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*

$s \in Star\ r \rightarrow v$

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

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

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

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

lemma *Posix-injval*:
assumes $s \in (deriv\ c\ r) \rightarrow v$
shows $(c\ \#\ s) \in r \rightarrow (injval\ r\ c\ v)$
⟨*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 $s \in lang\ r \longleftrightarrow (\exists\, v.\ lexer\ r\ s = Some(v) \wedge s \in r \rightarrow v)$
⟨*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 *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 $LV\ (Plus\ r1\ r2)\ s = Left\ `\ LV\ r1\ s \cup Right\ `\ LV\ r2\ s$
⟨*proof*⟩

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 $Suffixes\ (c\ \#\ s) = Suffixes\ s \cup \{c\ \#\ s\}$
⟨*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*⟩

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**
  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-Plus (*Zero*, f_1) (r_2, f_2) = (r_2, *F-RIGHT* f_2)
| *simp-Plus* (r_1, f_1) (*Zero*, f_2) = (r_1, *F-LEFT* f_1)
| *simp-Plus* (r_1, f_1) (r_2, f_2) =
  (*if* r_1 = r_2 *then* (r_1, *F-LEFT* f_1) *else* (*Plus* r_1 r_2, *F-Plus* f_1 f_2))

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

lemma *simp-Times-simps*[*simp*]:
  simp-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-Times1* (*snd p1*) (*snd p2*))
                  else (*if* (*fst p2* = *One*) *then* (*fst p1*, *F-Times2* (*snd p1*) (*snd p2*))
                  else (*Times* (*fst p1*) (*fst p2*), *F-Times* (*snd p1*) (*snd p2*))))))
⟨*proof*⟩

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

fun
  simp :: *'a rexp* ⇒ *'a rexp* ∗ (*'a val* ⇒ *'a val*)
where
  simp (*Plus r1 r2*) = *simp-Plus* (*simp r1*) (*simp r2*)
| *simp* (*Times r1 r2*) = *simp-Times* (*simp r1*) (*simp r2*)
| *simp r* = (*r*, *id*)

fun
  slexer :: *'a rexp* ⇒ *'a list* ⇒ (*'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 ⇒ 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 ∈ (fst (simp r)) → v
  shows s ∈ r → ((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* ⇒ (*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* ∈ *Pos* (*Stars vs*)}

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 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*⟩

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* ⇒ *nat list* ⇒ *'a val* ⇒ *bool* (- ⊏*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* (λ*v1 v2*. *v1* :$\sqsubseteq$*val v2*) (λ *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* :⊏*val v2 flat v1* = *flat v2*
shows *Left v1* :⊏*val Left v2*
⟨*proof*⟩

lemma *PosOrd-Left-eq*:
assumes *flat v1* = *flat v2*
shows *Left v1* :⊏*val Left v2* ⟷ *v1* :⊏*val v2*
⟨*proof*⟩

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

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

lemma *PosOrd-Right-eq*:
assumes *flat v1* = *flat v2*
shows *Right v1* :⊏*val Right v2* ⟷ *v1* :⊏*val v2*
⟨*proof*⟩

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

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

lemma *PosOrd-Seq-eq*:
assumes *flat v2* = *flat w2*
shows (*Seq v v2*) :⊏*val* (*Seq v w2*) ⟷ *v2* :⊏*val w2*
⟨*proof*⟩

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

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

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

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

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

lemma *PosOrd-Stars-append-eq*:
assumes *flats vs1* = *flats vs2*
shows *Stars* (*vs* @ *vs1*) :⊏*val Stars* (*vs* @ *vs2*) ⟷ *Stars vs1* :⊏*val Stars vs2*
⟨*proof*⟩

lemma *PosOrd-almost-trichotomous*:
shows *v1* :⊏*val v2* ∨ *v2* :⊏*val v1* ∨ (*length* (*flat v1*) = *length* (*flat v2*))
⟨*proof*⟩

14 The Posix Value is smaller than any other lexical value

lemma *Posix-PosOrd*:
assumes *s* ∈ *r* → *v1 v2* ∈ *LV r s*
shows *v1* :⊑*val v2*
⟨*proof*⟩

lemma *Posix-PosOrd-reverse*:
assumes *s* ∈ *r* → *v1*
shows ¬(∃ *v2* ∈ *LV r s*. *v2* :⊏*val v1*)
⟨*proof*⟩

lemma *PosOrd-Posix*:
assumes *v1* ∈ *LV r s* ∀ v_2 ∈ *LV r s*. ¬ v_2 :⊏*val v1*
shows *s* ∈ *r* → *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-order-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 $(atoms\colon 'a)\ rexp =$
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*) $^{\frown\frown}$ *n*) |
lang (Upto r n) = ($\bigcup$ *i* ∈ {..*n*}. (*lang r*) $^{\frown\frown}$ *i*) |
lang (From r n) = ($\bigcup$ *i* ∈ {*n*..}. (*lang r*) $^{\frown\frown}$ *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*) $^{\frown\frown}$ *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* ⇒ *'a rexp* ⇒ *'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* ∈ *cs then One else Zero*)

fun
derivs :: *'a list* ⇒ *'a rexp* ⇒ *'a rexp*
where
derivs [] *r* = *r*
| *derivs* (*c* # *s*) *r* = *derivs s* (*deriv c r*)

lemma *deriv-pow* [*simp*]:
shows *Deriv c* (*A* ^^ *n*) = (*if n* = *0 then* {} *else* (*Deriv c A*) @@ (*A* ^^ (*n* − *1*)))
⟨*proof*⟩

lemma *lang-deriv*: *lang* (*deriv c r*) = *Deriv c* (*lang r*)
⟨*proof*⟩

lemma *lang-derivs*: *lang* (*derivs s r*) = *Derivs s* (*lang r*)
⟨*proof*⟩

A regular expression matcher:

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

lemma *matcher-correctness*: *matcher r s* ⟷ *s* ∈ *lang r*
⟨*proof*⟩

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* ⇒ *'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 ≡ *concat* (*map flat vs*)

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

lemma *flats-empty*:
  assumes (∀ *v*∈*set vs*. *flat v* = [])
  shows *flats vs* = []
⟨*proof*⟩

19 Relation between values and regular expressions

inductive
  Prf :: *'a val* ⇒ *'a rexp* ⇒ *bool* (⊢ - : - [*100*, *100*] *100*)

where
$[\![\vdash v1 : r1; \vdash v2 : r2]\!] \Longrightarrow \vdash Seq\ v1\ v2 : Times\ r1\ r2$
$\mid \vdash v1 : r1 \Longrightarrow \vdash Left\ v1 : Plus\ r1\ r2$
$\mid \vdash v2 : r2 \Longrightarrow \vdash Right\ v2 : Plus\ r1\ r2$
$\mid \vdash Void : One$
$\mid \vdash Atm\ c : Atom\ c$
$\mid [\![\forall v \in set\ vs. \vdash v : r \wedge flat\ v \neq []]\!] \Longrightarrow \vdash Stars\ vs : Star\ r$
$\mid [\![\forall v \in set\ vs1. \vdash v : r \wedge flat\ v \neq [];$
$\forall v \in set\ vs2. \vdash v : r \wedge flat\ v = [];$
$length\ (vs1 \mathbin{@} vs2) = n]\!] \Longrightarrow \vdash Stars\ (vs1 \mathbin{@} vs2) : NTimes\ r\ n$
$\mid [\![\forall v \in set\ vs. \vdash v : r \wedge flat\ v \neq []; length\ vs \leq n]\!] \Longrightarrow \vdash Stars\ vs : Upto\ r\ n$
$\mid [\![\forall v \in set\ vs1. \vdash v : r\ \wedge flat\ v \neq [];$
$\forall v \in set\ vs2. \vdash v : r \wedge flat\ v = [];$
$length\ (vs1 \mathbin{@} vs2) = n]\!] \Longrightarrow \vdash Stars\ (vs1 \mathbin{@} vs2) : From\ r\ n$
$\mid [\![\forall v \in set\ vs. \vdash v : r\ \wedge flat\ v \neq []; length\ vs > n]\!] \Longrightarrow \vdash Stars\ vs : From\ r\ n$
$\mid \vdash v : r \Longrightarrow \vdash Recv\ l\ v : Rec\ l\ r$
$\mid c \in cs \Longrightarrow \vdash Atm\ c : Charset\ cs$

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 $s \in A \mathbin{@@} B$
shows $\exists s1\ s2.\ s = s1 \mathbin{@} s2 \wedge s1 \in A \wedge s2 \in B$
⟨*proof*⟩

lemma *pow-string*:
assumes $s \in A \mathbin{\frown\frown} n$
shows $\exists ss.\ concat\ ss = s \wedge (\forall s \in set\ ss.\ s \in A) \wedge length\ ss = n$
⟨*proof*⟩

lemma *pow-Prf*:

assumes $\forall\, v{\in}set\ vs.\ \vdash v : r \wedge flat\ v \in A$
shows $flats\ vs \in A \frown (length\ vs)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *L-flat-Prf*:

lang r = {*flat v* | *v*. ⊢ *v* : *r*}
⟨*proof*⟩

20 Sulzmann and Lu functions

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

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

21 Mkeps, injval

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

lemma *mkeps-nullable*:
assumes *nullable r*
shows ⊢ *mkeps r* : *r*
⟨*proof*⟩

lemma *Prf-injval-flat*:
assumes ⊢ *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*⟩

22 Our Alternative Posix definition

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

inductive-cases *Posix-elims*:
$s \in Zero \rightarrow v$
$s \in One \rightarrow v$
$s \in Atom\ c \rightarrow v$
$s \in Plus\ r1\ r2 \rightarrow v$

$s \in Times\ r1\ r2 \to v$
$s \in Star\ r \to v$
$s \in NTimes\ r\ n \to v$
$s \in Upto\ r\ n \to v$
$s \in From\ r\ n \to v$
$s \in Rec\ l\ r \to v$
$s \in Charset\ cs \to v$

lemma *Posix1*:
assumes $s \in r \to v$
shows $s \in lang\ r\ flat\ v = s$
⟨*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 *List-eq-zipI*:
assumes $\forall (v1, v2) \in set\ (zip\ vs1\ vs2).\ v1 = v2$
and *length vs1* = *length vs2*
shows *vs1* = *vs2*
⟨*proof*⟩

Our Posix definition determines a unique value.

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*⟩

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 ⇒ None
| Some(v) ⇒ Some(injval r c v))

lemma *lexer-correct-None*:
shows *s ∉ lang r ⟷ lexer r s = None*
⟨*proof*⟩

lemma *lexer-correct-Some*:
shows *s ∈ lang r ⟷ (∃ v. lexer r s = Some(v) ∧ s ∈ r → v)*
⟨*proof*⟩

lemma *lexer-correctness*:
shows *(lexer r s = Some v) ⟷ s ∈ r → v*
and *(lexer r s = None) ⟷ ¬(∃ v. s ∈ r → 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 ⇒ 'a list ⇒ ('a val) set
where *LV r s ≡ {v. ⊢ v : r ∧ 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 ∪ Right ‘ LV r2 s*
and *LV (NTimes r 0) s = (if s = [] then {Stars []} else {})*
and *LV (Rec l r) s = {Recv l v | v. v ∈ LV r s}*
and *LV (Charset cs) s = (if length s = 1 ∧ (hd s) ∈ 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 $s \in r \rightarrow v$
  shows $v \in LV\ r\ s$
  ⟨*proof*⟩

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

end

theory *Simplifying3*
  imports *Lexer3*
begin

25 Lexer including simplifications

fun *F-RIGHT* **where**
  $F\text{-}RIGHT\ f\ v = Right\ (f\ v)$

fun *F-LEFT* **where**
  $F\text{-}LEFT\ f\ v = Left\ (f\ v)$

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

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

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

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

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-Times (Zero, f1) (r2, f2) = (Zero, undefined)
| simp-Times (r1, f1) (Zero, f2) = (Zero, undefined)
| simp-Times (One, f1) (r2, f2) = (r2, F-Times1 f1 f2)
| simp-Times (r1, f1) (One, f2) = (r1, F-Times2 f1 f2)
| simp-Times (r1, f1) (r2, f2) = (Times r1 r2, F-Times f1 f2)

lemma simp-Times-simps[simp]:
  simp-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-Times1 (snd p1) (snd p2))
                     else (if (fst p2 = One) then (fst p1, F-Times2 (snd p1) (snd p2))
                      else (Times (fst p1) (fst p2), F-Times (snd p1) (snd p2))))))
⟨proof⟩

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

fun
  simp :: 'a rexp ⇒ 'a rexp * ('a val ⇒ 'a val)
where
  simp (Plus r1 r2) = simp-Plus (simp r1) (simp r2)
| simp (Times r1 r2) = simp-Times (simp r1) (simp r2)
| simp r = (r, id)

fun
  slexer :: 'a rexp ⇒ 'a list ⇒ ('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 ⇒ 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 $s \in (fst\ (simp\ r)) \rightarrow v$
shows $s \in r \rightarrow ((snd\ (simp\ r))\ v)$
⟨*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
$Pos\ (Void) = \{[]\}$
| $Pos\ (Atm\ c) = \{[]\}$
| $Pos\ (Left\ v) = \{[]\} \cup \{0\#ps \mid ps.\ ps \in Pos\ v\}$
| $Pos\ (Right\ v) = \{[]\} \cup \{1\#ps \mid ps.\ ps \in Pos\ v\}$
| $Pos\ (Seq\ v1\ v2) = \{[]\} \cup \{0\#ps \mid ps.\ ps \in Pos\ v1\} \cup \{1\#ps \mid ps.\ ps \in Pos\ v2\}$
| $Pos\ (Stars\ []) = \{[]\}$
| $Pos\ (Stars\ (v\#vs)) = \{[]\} \cup \{0\#ps \mid ps.\ ps \in Pos\ v\} \cup \{Suc\ n\#ps \mid n\ ps.\ n\#ps \in Pos\ (Stars\ vs)\}$
| $Pos\ (Recv\ l\ v) = \{[]\} \cup \{ps\ .\ ps \in Pos\ v\}$

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

lemma *Pos-empty*:
shows $[] \in Pos\ v$
⟨*proof*⟩

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 $pflat\text{-}len\ (Recv\ l\ v)\ p = pflat\text{-}len\ v\ p$
and $pflat\text{-}len\ v\ [] = intlen\ (flat\ v)$
⟨*proof*⟩

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

lemma *pflat-len-outside*:
assumes $p \notin Pos\ v1$
shows $pflat\text{-}len\ v1\ p = -1$
⟨*proof*⟩

28 Orderings

definition *prefix-list*:: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$ (- $\sqsubseteq pre$ - [60,59] 60)
where
$ps1 \sqsubseteq pre\ ps2 \equiv \exists ps'.\ ps1\ @ps' = ps2$

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* ⇒ *nat list* ⇒ *'a val* ⇒ *bool* (- ⊏*val* - - [*60*, *60*, *59*] *60*)
where
  $v1 \sqsubset val\ p\ v2 \equiv pflat\text{-}len\ v1\ p > pflat\text{-}len\ v2\ p \wedge$
  $(\forall q \in Pos\ v1 \cup Pos\ v2.\ q \sqsubset lex\ p \longrightarrow pflat\text{-}len\ v1\ q = pflat\text{-}len\ v2\ q)$

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* ⇒ *'a val* ⇒ *bool* (- :⊏*val* - [*60*, *59*] *60*)

where
v1 :⊏*val v2* ≡ ∃ *p*. *v1* ⊏*val p v2*

definition *PosOrd-ex-eq*:: *'a val* ⇒ *'a val* ⇒ *bool* (- :⊑*val* - [*60*, *59*] *60*)
where
v1 :⊑*val v2* ≡ *v1* :⊏*val v2* ∨ *v1* = *v2*

lemma *PosOrd-trans*:
assumes *v1* :⊏*val v2 v2* :⊏*val v3*
shows *v1* :⊏*val v3*
⟨*proof*⟩

lemma *PosOrd-irrefl*:
assumes *v* :⊏*val v*
shows *False*
⟨*proof*⟩

lemma *PosOrd-assym*:
assumes *v1* :⊏*val v2*
shows ¬(*v2* :⊏*val v1*)
⟨*proof*⟩

lemma *PosOrd-ordering*:
shows *ordering* (λ*v1 v2*. *v1* :⊑*val v2*) (λ *v1 v2*. *v1* :⊏*val v2*)
⟨*proof*⟩

lemma *PosOrd-order*:
shows *class.order* (λ*v1 v2*. *v1* :⊑*val v2*) (λ *v1 v2*. *v1* :⊏*val v2*)
⟨*proof*⟩

lemma *PosOrd-ex-eq2*:
shows *v1* :⊏*val v2* ⟷ (*v1* :⊑*val v2* ∧ *v1* ≠ *v2*)
⟨*proof*⟩

lemma *PosOrdeq-trans*:
assumes *v1* :⊑*val v2 v2* :⊑*val v3*
shows *v1* :⊑*val v3*
⟨*proof*⟩

lemma *PosOrdeq-antisym*:
assumes *v1* :⊑*val v2 v2* :⊑*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 $pflat\text{-}len\ v2\ p < pflat\text{-}len\ v1\ p$
shows $p \in Pos\ v1$
⟨*proof*⟩

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

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

lemma *PosOrd-Right-eq*:
assumes *flat v1* = *flat v2*
shows *Right v1* :⊏*val Right v2* ⟷ *v1* :⊏*val v2*
⟨*proof*⟩

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

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

lemma *PosOrd-Seq-eq*:
assumes *flat v2* = *flat w2*
shows (*Seq v v2*) :⊏*val* (*Seq v w2*) ⟷ *v2* :⊏*val w2*
⟨*proof*⟩

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

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

lemma *PosOrd-Stars-appendI*:
assumes *Stars vs1* :⊏*val Stars vs2 flat* (*Stars vs1*) = *flat* (*Stars vs2*)
shows *Stars* (*vs* @ *vs1*) :⊏*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 $v1 :\sqsubset_{val} v2 \vee v2 :\sqsubset_{val} v1 \vee (length\ (flat\ v1) = length\ (flat\ v2))$
⟨*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 $s \in r \to v1$
⟨*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-order-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

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.