POSIX Lexing with Derivatives of Regular Expressions

Fahad Ausaf Roy Dyckhoff Christian Urban

December 14, 2021

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 [1] 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.

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*))

lemma *flat-Stars* [*simp*]:
 flat (*Stars vs*) = *concat* (*map flat vs*)
by (*induct vs*) (*auto*)

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*
| ⊢ *Stars* [] : *Star r*
| ⟦⊢ *v* : *r*; ⊢ *Stars vs* : *Star r*⟧ ⟹ ⊢ *Stars* (*v* # *vs*) : *Star r*

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

lemma *Prf-flat-lang*:
  assumes $\vdash v : r$ **shows** *flat* $v \in$ *lang* r
using *assms*
by(*induct v r rule*: *Prf.induct*) (*auto*)

lemma *Prf-Stars*:
  assumes $\forall v \in$ *set vs*. $\vdash v : r$
  shows $\vdash$ *Stars vs* : *Star r*
using *assms*
by(*induct vs*) (*auto intro*: *Prf.intros*)

lemma *Star-string*:
  assumes $s \in$ *star A*
  shows $\exists$ *ss*. *concat ss* $= s \wedge (\forall s \in$ *set ss*. $s \in A)$
using *assms*
by (*metis in-star-iff-concat subsetD*)

lemma *Star-val*:
  assumes $\forall s{\in}$*set ss*. $\exists v$. $s =$ *flat* $v \wedge \vdash v : r$
  shows $\exists$ *vs*. *concat* (*map flat vs*) $=$ *concat ss* $\wedge (\forall v{\in}$*set vs*. $\vdash v : r)$
using *assms*
apply(*induct ss*)
apply(*auto*)
apply (*metis empty-iff list.set*(*1*))
by (*metis concat.simps*(*2*) *list.simps*(*9*) *set-ConsD*)

lemma *L-flat-Prf1*:
  assumes $\vdash v : r$ **shows** *flat* $v \in$ *lang* r
using *assms*
by (*induct*)(*auto*)

lemma *L-flat-Prf2*:
  assumes $s \in$ *lang* r **shows** $\exists v. \vdash v : r \wedge$ *flat* $v = s$
using *assms*
apply(*induct r arbitrary*: *s*)
apply(*auto intro*: *Prf.intros*)
using *Prf.intros*(*2*) *flat.simps*(*3*) **apply** *blast*
using *Prf.intros*(*3*) *flat.simps*(*4*) **apply** *blast*
apply (*metis Prf.intros*(*1*) *concE flat.simps*(*5*))
apply(*subgoal-tac* $\exists$ *vs*::(*'a val*) *list*. *concat* (*map flat vs*) $= s \wedge (\forall v \in$ *set vs*. $\vdash v$
: $r)$)
apply(*auto*)[*1*]
apply(*rule-tac x*=*Stars vs* **in** *exI*)
apply(*simp*)
apply (*simp add*: *Prf-Stars*)
apply(*drule Star-string*)
apply(*auto*)
apply(*rule Star-val*)
apply(*auto*)

done

lemma *L-flat-Prf*:
  lang r = {*flat v* | *v*. ⊢ *v* : *r*}
using *L-flat-Prf1 L-flat-Prf2* **by** *blast*

4 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* []

fun *injval* :: *'a rexp* ⇒ *'a* ⇒ *'a val* ⇒ *'a val*
where
  injval (*Atom d*) *c Void* = *Atm d*
| *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*)

5 Mkeps, injval

lemma *mkeps-nullable*:
  assumes *nullable r*
  shows ⊢ *mkeps r* : *r*
using *assms*
by (*induct r*)
   (*auto intro*: *Prf.intros*)

lemma *mkeps-flat*:
  assumes *nullable r*
  shows *flat* (*mkeps r*) = []
using *assms*
by (*induct r*) (*auto*)

lemma *Prf-injval*:
  assumes ⊢ *v* : *deriv c r*
  shows ⊢ (*injval r c v*) : *r*
using *assms*
apply(*induct r arbitrary*: *c v rule*: *rexp.induct*)
apply(*auto intro*!: *Prf.intros mkeps-nullable elim*!: *Prf-elims split*: *if-splits*)

apply(*rotate-tac 2*)
apply(*erule Prf.cases*)
apply(*simp-all*)[7]
apply(*auto*)
apply (*metis Prf.intros*(6) *Prf.intros*(7))
by (*metis Prf.intros*(7))

lemma *Prf-injval-flat*:
  assumes $\vdash v : deriv\ c\ r$
  shows *flat* (*injval r c v*) = *c* # (*flat v*)
using *assms*
apply(*induct r arbitrary*: *v c*)
apply(*auto elim*!: *Prf-elims split*: *if-splits*)
apply(*metis mkeps-flat*)
apply(*rotate-tac 2*)
apply(*erule Prf.cases*)
apply(*simp-all*)[7]
done

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 ∈ *Star r* → *v*

lemma *Posix1*:
  assumes *s* ∈ *r* → *v*
  shows *s* ∈ *lang r flat v* = *s*
using *assms*

by (*induct s r v rule*: *Posix.induct*) (*auto*)

lemma *Posix1a*:
  assumes $s \in r \to v$
  shows $\vdash v : r$
using *assms*
by (*induct s r v rule*: *Posix.induct*)(*auto intro*: *Prf.intros*)

lemma *Posix-mkeps*:
  assumes *nullable r*
  shows $[] \in r \to mkeps\ r$
using *assms*
apply(*induct r*)
apply(*auto intro*: *Posix.intros simp add*: *nullable-iff*)
apply(*subst append.simps*(*1*)[*symmetric*])
apply(*rule Posix.intros*)
apply(*auto*)
done

lemma *Posix-determ*:
  assumes $s \in r \to v1\ s \in r \to v2$
  shows $v1 = v2$
using *assms*
proof (*induct s r v1 arbitrary*: *v2 rule*: *Posix.induct*)
  case (*Posix-One v2*)
  have $[] \in One \to v2$ **by** *fact*
  then show $Void = v2$ **by** *cases auto*
next
  case (*Posix-Atom c v2*)
  have $[c] \in Atom\ c \to v2$ **by** *fact*
  then show $Atm\ c = v2$ **by** *cases auto*
next
  case (*Posix-Plus1 s r1 v r2 v2*)
  have $s \in Plus\ r1\ r2 \to v2$ **by** *fact*
  moreover
  have $s \in r1 \to v$ **by** *fact*
  then have $s \in lang\ r1$ **by** (*simp add*: *Posix1*)
  ultimately obtain v' **where** *eq*: $v2 = Left\ v'\ s \in r1 \to v'$ **by** *cases auto*
  moreover
  have *IH*: $\bigwedge v2.\ s \in r1 \to v2 \Longrightarrow v = v2$ **by** *fact*
  ultimately have $v = v'$ **by** *simp*
  then show $Left\ v = v2$ **using** *eq* **by** *simp*
next
  case (*Posix-Plus2 s r2 v r1 v2*)
  have $s \in Plus\ r1\ r2 \to v2$ **by** *fact*
  moreover

have $s \notin lang\ r1$ **by** *fact*
ultimately obtain v' **where** *eq*: $v2 = Right\ v'\ s \in r2 \rightarrow v'$
by *cases* (*auto simp add*: *Posix1*)
moreover
have *IH*: $\bigwedge v2.\ s \in r2 \rightarrow v2 \Longrightarrow v = v2$ **by** *fact*
ultimately have $v = v'$ **by** *simp*
then show $Right\ v = v2$ **using** *eq* **by** *simp*
next
case (*Posix-Times s1 r1 v1 s2 r2 v2 v'*)
have $(s1 \mathbin{@} s2) \in Times\ r1\ r2 \rightarrow v'$
$s1 \in r1 \rightarrow v1\ s2 \in r2 \rightarrow v2$
$\neg\ (\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3 \mathbin{@} s_4 = s2 \wedge s1 \mathbin{@} s_3 \in lang\ r1 \wedge s_4 \in lang\ r2)$ **by** *fact+*
then obtain $v1'\ v2'$ **where** $v' = Seq\ v1'\ v2'\ s1 \in r1 \rightarrow v1'\ s2 \in r2 \rightarrow v2'$
apply(*cases*) **apply** (*auto simp add*: *append-eq-append-conv2*)
using *Posix1*(*1*) **by** *fastforce+*
moreover
have *IHs*: $\bigwedge v1'.\ s1 \in r1 \rightarrow v1' \Longrightarrow v1 = v1'$
$\bigwedge v2'.\ s2 \in r2 \rightarrow v2' \Longrightarrow v2 = v2'$ **by** *fact+*
ultimately show $Seq\ v1\ v2 = v'$ **by** *simp*
next
case (*Posix-Star1 s1 r v s2 vs v2*)
have $(s1 \mathbin{@} s2) \in Star\ r \rightarrow v2$
$s1 \in r \rightarrow v\ s2 \in Star\ r \rightarrow Stars\ vs\ flat\ v \neq []$
$\neg\ (\exists s_3\ s_4.\ s_3 \neq [] \wedge s_3 \mathbin{@} s_4 = s2 \wedge s1 \mathbin{@} s_3 \in lang\ r \wedge s_4 \in lang\ (Star\ r))$
by *fact+*
then obtain $v'\ vs'$ **where** $v2 = Stars\ (v' \#\ vs')\ s1 \in r \rightarrow v'\ s2 \in (Star\ r) \rightarrow (Stars\ vs')$
apply(*cases*) **apply** (*auto simp add*: *append-eq-append-conv2*)
using *Posix1*(*1*) **apply** *fastforce*
apply (*metis Posix1*(*1*) *Posix-Star1.hyps*(*6*) *append-Nil append-Nil2*)
using *Posix1*(*2*) **by** *blast*
moreover
have *IHs*: $\bigwedge v2.\ s1 \in r \rightarrow v2 \Longrightarrow v = v2$
$\bigwedge v2.\ s2 \in Star\ r \rightarrow v2 \Longrightarrow Stars\ vs = v2$ **by** *fact+*
ultimately show $Stars\ (v \#\ vs) = v2$ **by** *auto*
next
case (*Posix-Star2 r v2*)
have $[] \in Star\ r \rightarrow v2$ **by** *fact*
then show $Stars\ [] = v2$ **by** *cases* (*auto simp add*: *Posix1*)
qed

lemma *Posix-injval*:
assumes $s \in (deriv\ c\ r) \rightarrow v$
shows $(c \#\ s) \in r \rightarrow (injval\ r\ c\ v)$
using *assms*
proof(*induct r arbitrary*: *s v rule*: *rexp.induct*)
case *Zero*

```
  have s ∈ deriv c Zero → v by fact
  then have s ∈ Zero → v by simp
  then have False by cases
  then show (c # s) ∈ Zero → (injval Zero c v) by simp
next
  case One
  have s ∈ deriv c One → v by fact
  then have s ∈ Zero → v by simp
  then have False by cases
  then show (c # s) ∈ One → (injval One c v) by simp
next
  case (Atom d)
  consider (eq) c = d | (ineq) c ≠ d by blast
  then show (c # s) ∈ (Atom d) → (injval (Atom d) c v)
  proof (cases)
    case eq
    have s ∈ deriv c (Atom d) → v by fact
    then have s ∈ One → v using eq by simp
    then have eqs: s = [] ∧ v = Void by cases simp
    show (c # s) ∈ Atom d → injval (Atom d) c v using eq eqs
    by (auto intro: Posix.intros)
  next
    case ineq
    have s ∈ deriv c (Atom d) → v by fact
    then have s ∈ Zero → v using ineq by simp
    then have False by cases
    then show (c # s) ∈ Atom d → injval (Atom d) c v by simp
  qed
next
  case (Plus r1 r2)
  have IH1: ⋀s v. s ∈ deriv c r1 → v ⟹ (c # s) ∈ r1 → injval r1 c v by fact
  have IH2: ⋀s v. s ∈ deriv c r2 → v ⟹ (c # s) ∈ r2 → injval r2 c v by fact
  have s ∈ deriv c (Plus r1 r2) → v by fact
  then have s ∈ Plus (deriv c r1) (deriv c r2) → v by simp
  then consider (left) v' where v = Left v' s ∈ deriv c r1 → v'
           | (right) v' where v = Right v' s ∉ lang (deriv c r1) s ∈ deriv c r2 →
v'
            by cases auto
  then show (c # s) ∈ Plus r1 r2 → injval (Plus r1 r2) c v
  proof (cases)
    case left
    have s ∈ deriv c r1 → v' by fact
    then have (c # s) ∈ r1 → injval r1 c v' using IH1 by simp
     then have (c # s) ∈ Plus r1 r2 → injval (Plus r1 r2) c (Left v') by (auto
intro: Posix.intros)
    then show (c # s) ∈ Plus r1 r2 → injval (Plus r1 r2) c v using left by simp
  next
    case right
    have s ∉ lang (deriv c r1) by fact
```

then have $c \# s \notin lang\ r1$ **by** (*simp add*: *lang-deriv Deriv-def*)
moreover
have $s \in deriv\ c\ r2 \rightarrow v'$ **by** *fact*
then have $(c \# s) \in r2 \rightarrow injval\ r2\ c\ v'$ **using** *IH2* **by** *simp*
ultimately have $(c \# s) \in Plus\ r1\ r2 \rightarrow injval\ (Plus\ r1\ r2)\ c\ (Right\ v')$
by (*auto intro*: *Posix.intros*)
then show $(c \# s) \in Plus\ r1\ r2 \rightarrow injval\ (Plus\ r1\ r2)\ c\ v$ **using** *right* **by** *simp*
qed
next
case (*Times r1 r2*)
have *IH1*: $\bigwedge s\ v.\ s \in deriv\ c\ r1 \rightarrow v \Longrightarrow (c \# s) \in r1 \rightarrow injval\ r1\ c\ v$ **by** *fact*
have *IH2*: $\bigwedge s\ v.\ s \in deriv\ c\ r2 \rightarrow v \Longrightarrow (c \# s) \in r2 \rightarrow injval\ r2\ c\ v$ **by** *fact*
have $s \in deriv\ c\ (Times\ r1\ r2) \rightarrow v$ **by** *fact*
then consider
(*left-nullable*) *v1 v2 s1 s2* **where**
$v = Left\ (Seq\ v1\ v2)\ \ s = s1 @ s2$
$s1 \in deriv\ c\ r1 \rightarrow v1\ s2 \in r2 \rightarrow v2\ nullable\ r1$
$\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s2 \wedge s1 @ s_3 \in lang\ (deriv\ c\ r1) \wedge s_4 \in lang\ r2)$
| (*right-nullable*) *v1 s1 s2* **where**
$v = Right\ v1\ s = s1 @ s2$
$s \in deriv\ c\ r2 \rightarrow v1\ nullable\ r1\ s1 @ s2 \notin lang\ (Times\ (deriv\ c\ r1)\ r2)$
| (*not-nullable*) *v1 v2 s1 s2* **where**
$v = Seq\ v1\ v2\ s = s1 @ s2$
$s1 \in deriv\ c\ r1 \rightarrow v1\ s2 \in r2 \rightarrow v2\ \neg nullable\ r1$
$\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s2 \wedge s1 @ s_3 \in lang\ (deriv\ c\ r1) \wedge s_4 \in lang\ r2)$
by (*force split*: *if-splits elim*!: *Posix-elims simp add*: *lang-deriv Deriv-def*)
then show $(c \# s) \in Times\ r1\ r2 \rightarrow injval\ (Times\ r1\ r2)\ c\ v$
proof (*cases*)
case *left-nullable*
have $s1 \in deriv\ c\ r1 \rightarrow v1$ **by** *fact*
then have $(c \# s1) \in r1 \rightarrow injval\ r1\ c\ v1$ **using** *IH1* **by** *simp*
moreover
have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s2 \wedge s1 @ s_3 \in lang\ (deriv\ c\ r1) \wedge s_4 \in lang\ r2)$ **by** *fact*
then have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s2 \wedge (c \# s1) @ s_3 \in lang\ r1 \wedge s_4 \in lang\ r2)$
by (*simp add*: *lang-deriv Deriv-def*)
ultimately have $((c \# s1) @ s2) \in Times\ r1\ r2 \rightarrow Seq\ (injval\ r1\ c\ v1)\ v2$ **using** *left-nullable* **by** (*rule-tac Posix.intros*)
then show $(c \# s) \in Times\ r1\ r2 \rightarrow injval\ (Times\ r1\ r2)\ c\ v$ **using** *left-nullable* **by** *simp*
next
case *right-nullable*
have *nullable r1* **by** *fact*
then have $[] \in r1 \rightarrow (mkeps\ r1)$ **by** (*rule Posix-mkeps*)
moreover

have $s \in deriv\ c\ r2 \rightarrow v1$ **by** *fact*
then have $(c\ \#\ s) \in r2 \rightarrow (injval\ r2\ c\ v1)$ **using** *IH2* **by** *simp*
moreover
have $s1\ @\ s2 \notin lang\ (Times\ (deriv\ c\ r1)\ r2)$ **by** *fact*
then have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = c\ \#\ s \wedge []\ @\ s_3 \in lang\ r1 \wedge s_4 \in lang\ r2)$
using *right-nullable*
apply (*auto simp add*: *lang-deriv Deriv-def append-eq-Cons-conv*)
by (*metis concI mem-Collect-eq*)
ultimately have $([]\ @\ (c\ \#\ s)) \in Times\ r1\ r2 \rightarrow Seq\ (mkeps\ r1)\ (injval\ r2\ c\ v1)$
by(*rule Posix.intros*)
then show $(c\ \#\ s) \in Times\ r1\ r2 \rightarrow injval\ (Times\ r1\ r2)\ c\ v$ **using** *right-nullable* **by** *simp*
next
case *not-nullable*
have $s1 \in deriv\ c\ r1 \rightarrow v1$ **by** *fact*
then have $(c\ \#\ s1) \in r1 \rightarrow injval\ r1\ c\ v1$ **using** *IH1* **by** *simp*
moreover
have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge s1\ @\ s_3 \in lang\ (deriv\ c\ r1) \wedge s_4 \in lang\ r2)$ **by** *fact*
then have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge (c\ \#\ s1)\ @\ s_3 \in lang\ r1 \wedge s_4 \in lang\ r2)$ **by** (*simp add*: *lang-deriv Deriv-def*)
ultimately have $((c\ \#\ s1)\ @\ s2) \in Times\ r1\ r2 \rightarrow Seq\ (injval\ r1\ c\ v1)\ v2$ **using** *not-nullable*
by (*rule-tac Posix.intros*) (*simp-all*)
then show $(c\ \#\ s) \in Times\ r1\ r2 \rightarrow injval\ (Times\ r1\ r2)\ c\ v$ **using** *not-nullable* **by** *simp*
qed
next
case (*Star r*)
have *IH*: $\bigwedge s\ v.\ s \in deriv\ c\ r \rightarrow v \Longrightarrow (c\ \#\ s) \in r \rightarrow injval\ r\ c\ v$ **by** *fact*
have $s \in deriv\ c\ (Star\ r) \rightarrow v$ **by** *fact*
then consider
(*cons*) *v1 vs s1 s2* **where**
$v = Seq\ v1\ (Stars\ vs)\ s = s1\ @\ s2$
$s1 \in deriv\ c\ r \rightarrow v1\ s2 \in (Star\ r) \rightarrow (Stars\ vs)$
$\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3\ @\ s_4 = s2 \wedge s1\ @\ s_3 \in lang\ (deriv\ c\ r) \wedge s_4 \in lang\ (Star\ r))$
apply(*auto elim*!: *Posix-elims*(*1*−*5*) *simp add*: *lang-deriv Deriv-def intro*: *Posix.intros*)
apply(*rotate-tac 3*)
apply(*erule-tac Posix-elims*(*6*))
apply (*simp add*: *Posix.intros*(*6*))
using *Posix.intros*(*7*) **by** *blast*
then show $(c\ \#\ s) \in Star\ r \rightarrow injval\ (Star\ r)\ c\ v$
proof (*cases*)
case *cons*
have $s1 \in deriv\ c\ r \rightarrow v1$ **by** *fact*

then have $(c \# s1) \in r \rightarrow injval\ r\ c\ v1$ **using** *IH* **by** *simp*
moreover
have $s2 \in Star\ r \rightarrow Stars\ vs$ **by** *fact*
moreover
have $(c \# s1) \in r \rightarrow injval\ r\ c\ v1$ **by** *fact*
then have $flat\ (injval\ r\ c\ v1) = (c \# s1)$ **by** (*rule Posix1*)
then have $flat\ (injval\ r\ c\ v1) \neq []$ **by** *simp*
moreover
have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 \mathbin{@} s_4 = s2 \wedge s1 \mathbin{@} s_3 \in lang\ (deriv\ c\ r) \wedge s_4 \in lang\ (Star\ r))$ **by** *fact*
then have $\neg\ (\exists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 \mathbin{@} s_4 = s2 \wedge (c \# s1) \mathbin{@} s_3 \in lang\ r \wedge s_4 \in lang\ (Star\ r))$
by (*simp add*: *lang-deriv Deriv-def*)
ultimately
have $((c \# s1) \mathbin{@} s2) \in Star\ r \rightarrow Stars\ (injval\ r\ c\ v1 \# vs)$ **by** (*rule Posix.intros*)
then show $(c \# s) \in Star\ r \rightarrow injval\ (Star\ r)\ c\ v$ **using** *cons* **by**(*simp*)
qed
qed

7 The Lexer by Sulzmann and Lu

fun
lexer :: *'a rexp* ⇒ *'a list* ⇒ (*'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 $s \notin lang\ r \longleftrightarrow lexer\ r\ s = None$
apply(*induct s arbitrary*: *r*)
apply(*simp add*: *nullable-iff*)
apply(*drule-tac x*=*deriv a r* **in** *meta-spec*)
apply(*auto simp add*: *lang-deriv Deriv-def*)
done

lemma *lexer-correct-Some*:
shows $s \in lang\ r \longleftrightarrow (\exists\, v.\ lexer\ r\ s = Some(v) \wedge s \in r \rightarrow v)$
apply(*induct s arbitrary*: *r*)
apply(*auto simp add*: *Posix-mkeps nullable-iff*)[*1*]
apply(*drule-tac x*=*deriv a r* **in** *meta-spec*)
apply(*simp add*: *lang-deriv Deriv-def*)
apply(*rule iffI*)
apply(*auto intro*: *Posix-injval simp add*: *Posix1*(*1*))
done

lemma *lexer-correctness*:
  shows $(lexer\ r\ s = Some\ v) \longleftrightarrow s \in r \rightarrow v$
  and $(lexer\ r\ s = None) \longleftrightarrow \neg(\exists\, v.\ s \in r \rightarrow v)$
apply(*auto*)
using *lexer-correct-None lexer-correct-Some* **apply** *fastforce*
using *Posix1*(*1*) *Posix-determ lexer-correct-Some* **apply** *blast*
using *Posix1*(*1*) *lexer-correct-None* **apply** *blast*
using *lexer-correct-None lexer-correct-Some* **by** *blast*

end

theory *Simplifying*
  imports *Lexer*
begin

8 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) = (*Plus* $r_1\ r_2$, *F-Plus* $f_1\ f_2$)

fun *simp-Times* **where**
  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) = $(Times\ r_1\ r_2,\ F\text{-}Times\ f_1\ f_2)$

lemma *simp-Times-simps*[*simp*]:
  simp-Times p1 p2 = (*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*))))
by (*induct p1 p2 rule*: *simp-Times.induct*) (*auto*)

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 (*Plus* (*fst p1*) (*fst p2*), *F-Plus* (*snd p1*) (*snd p2*))))
by (*induct p1 p2 rule*: *simp-Plus.induct*) (*auto*)

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*)))
by (*auto split*: *prod.split option.split*)

lemma *L-fst-simp*:
  shows *lang r* = *lang* (*fst* (*simp r*))
by (*induct r*) (*auto*)

lemma *Posix-simp*:
  assumes *s* ∈ (*fst* (*simp r*)) → *v*
  shows *s* ∈ *r* → ((*snd* (*simp r*)) *v*)
using *assms*
proof(*induct r arbitrary*: *s v rule*: *rexp.induct*)
  case (*Plus r1 r2 s v*)
  have *IH1*: ⋀*s v*. *s* ∈ *fst* (*simp r1*) → *v* ⟹ *s* ∈ *r1* → *snd* (*simp r1*) *v* **by** *fact*

have *IH2*: $\bigwedge s\ v.\ s \in fst\ (simp\ r2) \to v \Longrightarrow s \in r2 \to snd\ (simp\ r2)\ v$ **by** *fact*
have *as*: $s \in fst\ (simp\ (Plus\ r1\ r2)) \to v$ **by** *fact*
consider (*Zero-Zero*) $fst\ (simp\ r1) = Zero\ fst\ (simp\ r2) = Zero$
| (*Zero-NZero*) $fst\ (simp\ r1) = Zero\ fst\ (simp\ r2) \neq Zero$
| (*NZero-Zero*) $fst\ (simp\ r1) \neq Zero\ fst\ (simp\ r2) = Zero$
| (*NZero-NZero*) $fst\ (simp\ r1) \neq Zero\ fst\ (simp\ r2) \neq Zero$ **by** *auto*
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$
proof(*cases*)
case (*Zero-Zero*)
with *as* **have** $s \in Zero \to v$ **by** *simp*
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$ **by** (*rule Posix-elims*(*1*))
next
case (*Zero-NZero*)
with *as* **have** $s \in fst\ (simp\ r2) \to v$ **by** *simp*
with *IH2* **have** $s \in r2 \to snd\ (simp\ r2)\ v$ **by** *simp*
moreover
from *Zero-NZero* **have** $fst\ (simp\ r1) = Zero$ **by** *simp*
then have $lang\ (fst\ (simp\ r1)) = \{\}$ **by** *simp*
then have $lang\ r1 = \{\}$ **using** *L-fst-simp* **by** *auto*
then have $s \notin lang\ r1$ **by** *simp*
ultimately have $s \in Plus\ r1\ r2 \to Right\ (snd\ (simp\ r2)\ v)$ **by** (*rule Posix-Plus2*)
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$
using *Zero-NZero* **by** *simp*
next
case (*NZero-Zero*)
with *as* **have** $s \in fst\ (simp\ r1) \to v$ **by** *simp*
with *IH1* **have** $s \in r1 \to snd\ (simp\ r1)\ v$ **by** *simp*
then have $s \in Plus\ r1\ r2 \to Left\ (snd\ (simp\ r1)\ v)$ **by** (*rule Posix-Plus1*)
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$ **using** *NZero-Zero* **by** *simp*
next
case (*NZero-NZero*)
with *as* **have** $s \in Plus\ (fst\ (simp\ r1))\ (fst\ (simp\ r2)) \to v$ **by** *simp*
then consider (*Left*) *v1* **where** $v = Left\ v1\ s \in (fst\ (simp\ r1)) \to v1$
| (*Right*) *v2* **where** $v = Right\ v2\ s \in (fst\ (simp\ r2)) \to v2\ s \notin lang\ (fst\ (simp\ r1))$
by (*erule-tac Posix-elims*(*4*))
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$
proof(*cases*)
case (*Left*)
then have $v = Left\ v1\ s \in r1 \to (snd\ (simp\ r1)\ v1)$ **using** *IH1* **by** *simp-all*
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$ **using** *NZero-NZero*
by (*simp-all add*: *Posix-Plus1*)
next
case (*Right*)
then have $v = Right\ v2\ s \in r2 \to (snd\ (simp\ r2)\ v2)\ s \notin lang\ r1$ **using** *IH2 L-fst-simp* **by** *auto*
then show $s \in Plus\ r1\ r2 \to snd\ (simp\ (Plus\ r1\ r2))\ v$ **using** *NZero-NZero*

```
        by (simp-all add: Posix-Plus2)
      qed
    qed
next
  case (Times r1 r2 s v)
  have IH1: ⋀s v. s ∈ fst (simp r1) → v ⟹ s ∈ r1 → snd (simp r1) v by fact
  have IH2: ⋀s v. s ∈ fst (simp r2) → v ⟹ s ∈ r2 → snd (simp r2) v by fact
  have as: s ∈ fst (simp (Times r1 r2)) → v by fact
  consider (One-One) fst (simp r1) = One fst (simp r2) = One
         | (One-NOne) fst (simp r1) = One fst (simp r2) ≠ One
         | (NOne-One) fst (simp r1) ≠ One fst (simp r2) = One
         | (NOne-NOne) fst (simp r1) ≠ One fst (simp r2) ≠ One by auto
  then show s ∈ Times r1 r2 → snd (simp (Times r1 r2)) v
  proof(cases)
    case (One-One)
    with as have b: s ∈ One → v by simp
    from b have s ∈ r1 → snd (simp r1) v using IH1 One-One by simp
    moreover
    from b have c: s = [] v = Void using Posix-elims(2) by auto
    moreover
    have [] ∈ One → Void by (simp add: Posix-One)
    then have [] ∈ fst (simp r2) → Void using One-One by simp
    then have [] ∈ r2 → snd (simp r2) Void using IH2 by simp
    ultimately have ([] @ []) ∈ Times r1 r2 → Seq (snd (simp r1) Void) (snd
(simp r2) Void)
      using Posix-Times by blast
    then show s ∈ Times r1 r2 → snd (simp (Times r1 r2)) v using c One-One
by simp
  next
    case (One-NOne)
    with as have b: s ∈ fst (simp r2) → v by simp
    from b have s ∈ r2 → snd (simp r2) v using IH2 One-NOne by simp
    moreover
    have [] ∈ One → Void by (simp add: Posix-One)
    then have [] ∈ fst (simp r1) → Void using One-NOne by simp
    then have [] ∈ r1 → snd (simp r1) Void using IH1 by simp
    moreover
    from One-NOne(1) have lang (fst (simp r1)) = {[]} by simp
    then have lang r1 = {[]} by (simp add: L-fst-simp[symmetric])
    ultimately have ([] @ s) ∈ Times r1 r2 → Seq (snd (simp r1) Void) (snd
(simp r2) v)
      by(rule-tac Posix-Times) auto
    then show s ∈ Times r1 r2 → snd (simp (Times r1 r2)) v using One-NOne
by simp
  next
    case (NOne-One)
      with as have s ∈ fst (simp r1) → v by simp
      with IH1 have s ∈ r1 → snd (simp r1) v by simp
    moreover
```

```
      have [] ∈ One → Void by (simp add: Posix-One)
      then have [] ∈ fst (simp r2) → Void using NOne-One by simp
      then have [] ∈ r2 → snd (simp r2) Void using IH2 by simp
    ultimately have (s @ []) ∈ Times r1 r2 → Seq (snd (simp r1) v) (snd (simp
r2) Void)
      by(rule-tac Posix-Times) auto
    then show s ∈ Times r1 r2 → snd (simp (Times r1 r2)) v using NOne-One
by simp
  next
    case (NOne-NOne)
    with as have s ∈ Times (fst (simp r1)) (fst (simp r2)) → v by simp
    then obtain s1 s2 v1 v2 where eqs: s = s1 @ s2 v = Seq v1 v2
                  s1 ∈ (fst (simp r1)) → v1 s2 ∈ (fst (simp r2)) → v2
                   ¬ (∃ s3 s4. s3 ≠ [] ∧ s3 @ s4 = s2 ∧ s1 @ s3 ∈ lang r1 ∧ s4 ∈
lang r2)
              by (erule-tac Posix-elims(5)) (auto simp add: L-fst-simp[symmetric])

    then have s1 ∈ r1 → (snd (simp r1) v1) s2 ∈ r2 → (snd (simp r2) v2)
      using IH1 IH2 by auto
      then show s ∈ Times r1 r2 → snd (simp (Times r1 r2)) v using eqs
NOne-NOne
      by(auto intro: Posix-Times)
  qed
qed (simp-all)


lemma slexer-correctness:
  shows slexer r s = lexer r s
proof(induct s arbitrary: r)
  case Nil
  show slexer r [] = lexer r [] by simp
next
  case (Cons c s r)
  have IH: ⋀r. slexer r s = lexer r s by fact
  show slexer r (c # s) = lexer r (c # s)
   proof (cases s ∈ lang (deriv c r))
    case True
     assume a1: s ∈ lang (deriv c r)
     then obtain v1 where a2: lexer (deriv c r) s = Some v1 s ∈ deriv c r →
v1
       using lexer-correct-Some by auto
     from a1 have s ∈ lang (fst (simp (deriv c r))) using L-fst-simp[symmetric]
by auto
      then obtain v2 where a3: lexer (fst (simp (deriv c r))) s = Some v2 s ∈
(fst (simp (deriv c r))) → v2
        using lexer-correct-Some by auto
      then have a4: slexer (fst (simp (deriv c r))) s = Some v2 using IH by
simp
       from a3(2) have s ∈ deriv c r → (snd (simp (deriv c r))) v2 using
```

Posix-simp **by** *auto*
with *a2(2)* **have** *v1* = (*snd* (*simp* (*deriv c r*))) *v2* **using** *Posix-determ* **by** *auto*
with *a2(1)* *a4* **show** *slexer r* (*c* # *s*) = *lexer r* (*c* # *s*) **by** (*auto split*: *prod.split*)
next
case *False*
assume *b1*: *s* ∉ *lang* (*deriv c r*)
then have *lexer* (*deriv c r*) *s* = *None* **using** *lexer-correct-None* **by** *auto*
moreover
from *b1* **have** *s* ∉ *lang* (*fst* (*simp* (*deriv c r*))) **using** *L-fst-simp*[*symmetric*] **by** *auto*
then have *lexer* (*fst* (*simp* (*deriv c r*))) *s* = *None* **using** *lexer-correct-None* **by** *auto*
then have *slexer* (*fst* (*simp* (*deriv c r*))) *s* = *None* **using** *IH* **by** *simp*
ultimately show *slexer r* (*c* # *s*) = *lexer r* (*c* # *s*)
by (*simp del*: *slexer.simps add*: *slexer-better-simp*)
qed
qed

end

References

[1] 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.