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*)
⟨*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*
| ⊢ *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** $flat\ v \in lang\ r$
⟨*proof*⟩

lemma *Prf-Stars*:
  assumes $\forall v \in set\ vs. \vdash v : r$
  shows $\vdash Stars\ vs : Star\ r$
⟨*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 $\exists vs.\ concat\ (map\ flat\ vs) = concat\ ss \wedge (\forall v{\in}set\ vs. \vdash v : r)$
⟨*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* ⇒ *'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*
⟨*proof*⟩

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

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

lemma *Prf-injval-flat*:
  assumes ⊢ *v* : *deriv c r*
  shows *flat* (*injval r c v*) = *c* # (*flat v*)
⟨*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 *Simplifying*
  imports *Lexer*
begin

8 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**
  $simp\text{-}Plus\ (Zero, f_1)\ (r_2, f_2) = (r_2, F\text{-}RIGHT\ f_2)$
| $simp\text{-}Plus\ (r_1, f_1)\ (Zero, f_2) = (r_1, F\text{-}LEFT\ f_1)$
| $simp\text{-}Plus\ (r_1, f_1)\ (r_2, f_2) = (Plus\ r_1\ r_2, F\text{-}Plus\ f_1\ f_2)$

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

lemma *simp-Times-simps*[*simp*]:

simp-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*))))
⟨*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 (*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

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.