# POSIX Lexing with Derivatives of Regular Expressions (Proof Pearl)

Fahad Ausaf[1], Roy Dyckhoff[2], and Christian Urban[1]

[1] King's College London, United Kingdom
[2] St Andrews

**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 cleverly extended this algorithm in order to deal with POSIX matching, which is the underlying disambiguation strategy for regular expressions needed in lexers. Sulzmann and Lu have made available on-line what they call a "rigorous proof" of the correctness of their algorithm w.r.t. their specification; regrettably, it appears to us to have unfillable gaps. In the first part of this paper 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. Our definitions and proof are much simpler than those by Sulzmann and Lu and can be easily formalised in Isabelle/HOL. In the second part we analyse the correctness argument by Sulzmann and Lu in more detail and explain why it seems hard to turn it into a proof rigorous enough to be accepted by a system such as Isabelle/HOL.
**Keywords:** POSIX matching, Derivatives of Regular Expressions, Isabelle/HOL

## 1 Introduction

Sulzmann and Lu [3]

there are two commonly used disambiguation strategies to create a unique matching tree: one is called *greedy* matching [1] and the other is *POSIX* matching [2,3]. For the latter there are two rough rules:

- The Longest Match Rule (or "maximal munch rule"):

  The longest initial substring matched by any regular expression is taken as next token.
- Rule Priority:

  For a particular longest initial substring, the first regular expression that can match determines the token.

In the context of lexing, POSIX is the more interesting disambiguation strategy as it produces longest matches, which is necessary for tokenising programs. For example the string *iffoo* should not match the keyword *if* and the rest, but as one string *iffoo*, which might be a variable name in a program. As another example consider the string $xy$ and the regular expression $(x + y + xy)^*$. Either the input string can be matched in two 'iterations' by the single letter-regular expressions $x$ and $y$, or directly in one iteration by $xy$. The first case corresponds to greedy matching, which first matches with the left-most symbol and only matches the next symbol in case of a mismatch. The second case is POSIX matching, which prefers the longest match. In case more than one (longest) matches exist, only then it prefers the left-most match. While POSIX matching seems natural, it turns out to be much more subtle than greedy matching in terms of implementations and in terms of proving properties about it. If POSIX matching is implemented using automata, then one has to follow transitions (according to the input string) until one finds an accepting state, record this state and look for further transition which might lead to another accepting state that represents a longer input initial substring to be matched. Only if none can be found, the last accepting state is returned.

Sulzmann and Lu's paper [3] targets POSIX regular expression matching. They write that it is known to be to be a non-trivial problem and nearly all POSIX matching implementations are "buggy" [3, Page 203]. For this they cite a study by Kuklewicz [2]. My current work is about formalising the proofs in the paper by Sulzmann and Lu. Specifically, they propose in this paper a POSIX matching algorithm and give some details of a correctness proof for this algorithm inside the paper and some more details in an appendix. This correctness proof is unformalised, meaning it is just a "pencil-and-paper" proof, not done in a theorem prover. Though, the paper and presumably the proof have been peer-reviewed. Unfortunately their proof does not give enough details such that it can be straightforwardly implemented in a theorem prover, say Isabelle. In fact, the purported proof they outline does not work in central places. We discovered this when filling in many gaps and attempting to formalise the proof in Isabelle.

**Contributions:**

## 2 Preliminaries

Strings in Isabelle/HOL are lists of characters with the empty string being represented by the empty list, written $[]$, and list-cons being written as $\_ :: \_$. By using the type char we have a supply of finitely many characters roughly corresponding to the ASCII character set. Regular exprtessions

$$r := 0 \mid 1 \mid c \mid r_1 + r_2 \mid r_1 \cdot r_2 \mid r^\star$$

# 3 POSIX Regular Expression Matching

# 4 The Argument by Sulzmmann and Lu

# 5 Conclusion

Nipkow lexer from 2000

Values

$$v := \mathit{Void} \mid \mathit{Char}\ c \mid \mathit{Left}\ v \mid \mathit{Right}\ v \mid \mathit{Seq}\ v_1\ v_2 \mid \mathit{Stars}\ vs$$

The language of a regular expression

$$\begin{array}{lcl} L\,(0) & \stackrel{\text{def}}{=} & \varnothing \\ L\,(1) & \stackrel{\text{def}}{=} & \{[]\} \\ L\,(c) & \stackrel{\text{def}}{=} & \{[c]\} \\ L\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & (L\,r_1) \mathbin{@} (L\,r_2) \\ L\,(r_1 + r_2) & \stackrel{\text{def}}{=} & (L\,r_1) \cup (L\,r_2) \\ L\,(r^\star) & \stackrel{\text{def}}{=} & (L\,r)\star \end{array}$$

The nullable function

$$\begin{array}{lcl} \mathit{nullable}\,(0) & \stackrel{\text{def}}{=} & \mathit{False} \\ \mathit{nullable}\,(1) & \stackrel{\text{def}}{=} & \mathit{True} \\ \mathit{nullable}\,(c) & \stackrel{\text{def}}{=} & \mathit{False} \\ \mathit{nullable}\,(r_1 + r_2) & \stackrel{\text{def}}{=} & \mathit{nullable}\ r_1 \vee \mathit{nullable}\ r_2 \\ \mathit{nullable}\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \mathit{nullable}\ r_1 \wedge \mathit{nullable}\ r_2 \\ \mathit{nullable}\,(r^\star) & \stackrel{\text{def}}{=} & \mathit{True} \end{array}$$

The derivative function for characters and strings

$$\begin{array}{lcl} \mathit{der}\ c\ (0) & \stackrel{\text{def}}{=} & 0 \\ \mathit{der}\ c\ (1) & \stackrel{\text{def}}{=} & 0 \\ \mathit{der}\ c\ (c') & \stackrel{\text{def}}{=} & \mathit{if}\ c = c'\ \mathit{then}\ 1\ \mathit{else}\ 0 \\ \mathit{der}\ c\ (r_1 + r_2) & \stackrel{\text{def}}{=} & (\mathit{der}\ c\ r_1) + (\mathit{der}\ c\ r_2) \\ \mathit{der}\ c\ (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \mathit{if}\ \mathit{nullable}\ r_1\ \mathit{then}\ ((\mathit{der}\ c\ r_1) \cdot r_2) + (\mathit{der}\ c\ r_2)\ \mathit{else} \\ & & (\mathit{der}\ c\ r_1) \cdot r_2 \\ \mathit{der}\ c\ (r^\star) & \stackrel{\text{def}}{=} & (\mathit{der}\ c\ r) \cdot (r^\star) \\ \\ \mathit{ders}\ []\ r & \stackrel{\text{def}}{=} & r \\ \mathit{ders}\ (c :: s)\ r & \stackrel{\text{def}}{=} & \mathit{ders}\ s\ (\mathit{der}\ c\ r) \end{array}$$

The *flat* function for values

$$\begin{array}{lcl}
|\mathit{Void}| & \stackrel{\text{def}}{=} & [] \\
|\mathit{Char}\ c| & \stackrel{\text{def}}{=} & [c] \\
|\mathit{Left}\ v| & \stackrel{\text{def}}{=} & |v| \\
|\mathit{Right}\ v| & \stackrel{\text{def}}{=} & |v| \\
|\mathit{Seq}\ v_1\ v_2| & \stackrel{\text{def}}{=} & |v_1| \mathbin{@} |v_2| \\
|\mathit{Stars}\ []| & \stackrel{\text{def}}{=} & [] \\
|\mathit{Stars}\ (v :: vs)| & \stackrel{\text{def}}{=} & |v| \mathbin{@} |\mathit{Stars}\ vs|
\end{array}$$

The *mkeps* function

$$\begin{array}{lcl}
\mathit{mkeps}\ (1) & \stackrel{\text{def}}{=} & \mathit{Void} \\
\mathit{mkeps}\ (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \mathit{Seq}\ (\mathit{mkeps}\ r_1)\ (\mathit{mkeps}\ r_2) \\
\mathit{mkeps}\ (r_1 + r_2) & \stackrel{\text{def}}{=} & \mathit{if\ nullable}\ r_1\ \mathit{then\ Left}\ (\mathit{mkeps}\ r_1)\ \mathit{else\ Right}\ (\mathit{mkeps}\ r_2) \\
\mathit{mkeps}\ (r^\star) & \stackrel{\text{def}}{=} & \mathit{Stars}\ []
\end{array}$$

The *inj* function

$$\begin{array}{lcl}
\mathit{inj}\ (d)\ c\ \mathit{Void} & \stackrel{\text{def}}{=} & \mathit{Char}\ d \\
\mathit{inj}\ (r_1 + r_2)\ c\ (\mathit{Left}\ v_1) & \stackrel{\text{def}}{=} & \mathit{Left}\ (\mathit{inj}\ r_1\ c\ v_1) \\
\mathit{inj}\ (r_1 + r_2)\ c\ (\mathit{Right}\ v_2) & \stackrel{\text{def}}{=} & \mathit{Right}\ (\mathit{inj}\ r_2\ c\ v_2) \\
\mathit{inj}\ (r_1 \cdot r_2)\ c\ (\mathit{Seq}\ v_1\ v_2) & \stackrel{\text{def}}{=} & \mathit{Seq}\ (\mathit{inj}\ r_1\ c\ v_1)\ v_2 \\
\mathit{inj}\ (r_1 \cdot r_2)\ c\ (\mathit{Left}\ (\mathit{Seq}\ v_1\ v_2)) & \stackrel{\text{def}}{=} & \mathit{Seq}\ (\mathit{inj}\ r_1\ c\ v_1)\ v_2 \\
\mathit{inj}\ (r_1 \cdot r_2)\ c\ (\mathit{Right}\ v_2) & \stackrel{\text{def}}{=} & \mathit{Seq}\ (\mathit{mkeps}\ r_1)\ (\mathit{inj}\ r_2\ c\ v_2) \\
\mathit{inj}\ (r^\star)\ c\ (\mathit{Seq}\ v\ (\mathit{Stars}\ vs)) & \stackrel{\text{def}}{=} & \mathit{Stars}\ ((\mathit{inj}\ r\ c\ v) :: vs)
\end{array}$$

The inhabitation relation:

$$\frac{\vdash v_1 : r_1 \qquad \vdash v_2 : r_2}{\vdash \mathit{Seq}\ v_1\ v_2 : (r_1 \cdot r_2)}$$

$$\frac{\vdash v_1 : r_1}{\vdash (\mathit{Left}\ v_1) : (r_1 + r_2)} \qquad \frac{\vdash v_2 : r_1}{\vdash (\mathit{Right}\ v_2) : (r_2 + r_1)}$$

$$\frac{}{\vdash \mathit{Void} : (1)} \qquad \frac{}{\vdash (\mathit{Char}\ c) : (c)}$$

$$\frac{}{\vdash \mathit{Stars}\ [] : (r^\star)} \qquad \frac{\vdash v : r \qquad \vdash \mathit{Stars}\ vs : (r^\star)}{\vdash \mathit{Stars}\ (v :: vs) : (r^\star)}$$

We have also introduced a slightly restricted version of this relation where the last rule is restricted so that $|v| \neq []$. This relation for *non-problematic* is written $\models v : r$.

Our Posix relation $s \in r \to v$

$$\frac{}{[] \in (1) \to \mathit{Void}} \qquad \frac{}{[c] \in (c) \to (\mathit{Char}\ c)}$$

$$\frac{s \in r_1 \to v}{s \in (r_1 + r_2) \to (\mathit{Left}\ v)} \qquad \frac{s \in r_2 \to v \qquad s \notin (L\ r_1)}{s \in (r_1 + r_2) \to (\mathit{Right}\ v)}$$

$$\frac{\begin{array}{c} s_1 \in r_1 \to v_1 \qquad s_2 \in r_2 \to v_2 \\ \nexists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 \mathbin{@} s_4 = s_2 \wedge s_1 \mathbin{@} s_3 \in (L\ r_1) \wedge s_4 \in (L\ r_2) \end{array}}{(s_1 \mathbin{@} s_2) \in (r_1 \cdot r_2) \to \mathit{Seq}\ v_1\ v_2}$$

$$\frac{\begin{array}{c} s_1 \in r \to v \qquad s_2 \in (r^\star) \to \mathit{Stars}\ vs \\ |v| \neq [] \qquad \nexists\, s_3\ s_4.\ s_3 \neq [] \wedge s_3 \mathbin{@} s_4 = s_2 \wedge s_1 \mathbin{@} s_3 \in (L\ r) \wedge s_4 \in (L\ (r^\star)) \end{array}}{(s_1 \mathbin{@} s_2) \in (r^\star) \to \mathit{Stars}\ (v :: vs)}$$

$$\frac{}{[] \in (r^\star) \to \mathit{Stars}\ []}$$

Our version of Sulzmann's ordering relation

$$\frac{v_1 \succ_{r_1} v_1' \qquad v_1 \neq v_1'}{\mathit{Seq}\ v_1\ v_2 \succ_{(r_1 \cdot r_2)} \mathit{Seq}\ v_1'\ v_2'} \qquad \frac{v_2 \succ_{r_2} v_2'}{\mathit{Seq}\ v_1\ v_2 \succ_{(r_1 \cdot r_2)} \mathit{Seq}\ v_1\ v_2'}$$

$$\frac{\mathit{len}\ (|v_1|) \leq \mathit{len}\ (|v_2|)}{(\mathit{Left}\ v_2) \succ_{(r_1 + r_2)} (\mathit{Right}\ v_1)} \qquad \frac{\mathit{len}\ (|v_2|) < \mathit{len}\ (|v_1|)}{(\mathit{Right}\ v_1) \succ_{(r_1 + r_2)} (\mathit{Left}\ v_2)}$$

$$\frac{v_2 \succ_{r_2} v_2'}{(\mathit{Right}\ v_2) \succ_{(r_1 + r_2)} (\mathit{Right}\ v_2')} \qquad \frac{v_1 \succ_{r_1} v_1'}{(\mathit{Left}\ v_1) \succ_{(r_1 + r_2)} (\mathit{Left}\ v_1')}$$

$$\frac{}{\mathit{Void} \succ_{(1)} \mathit{Void}} \qquad \frac{}{(\mathit{Char}\ c) \succ_{(c)} (\mathit{Char}\ c)}$$

$$\frac{|\mathit{Stars}\ (v :: vs)| = []}{\mathit{Stars}\ [] \succ_{(r^\star)} \mathit{Stars}\ (v :: vs)} \qquad \frac{|\mathit{Stars}\ (v :: vs)| \neq []}{\mathit{Stars}\ (v :: vs) \succ_{(r^\star)} \mathit{Stars}\ []}$$

$$\frac{v_1 \succ_r v_2 \qquad v_1 \neq v_2}{\mathit{Stars}\ (v_1 :: vs_1) \succ_{(r^\star)} \mathit{Stars}\ (v_2 :: vs_2)}$$

$$\frac{\mathit{Stars}\ vs_1 \succ_{(r^\star)} \mathit{Stars}\ vs_2}{\mathit{Stars}\ (v :: vs_1) \succ_{(r^\star)} \mathit{Stars}\ (v :: vs_2)} \qquad \frac{}{\mathit{Stars}\ [] \succ_{(r^\star)} \mathit{Stars}\ []}$$

A prefix of a string s

$$s_1 \sqsubseteq s_2 \stackrel{\mathit{def}}{=} \exists s3.\ s_1 \mathbin{@} s3 = s_2$$

Values and non-problematic values

$$\mathit{Values}\ r\ s \stackrel{\mathit{def}}{=} \{v \mid \vdash v : r \wedge (|v|) \sqsubseteq s\}$$

The point is that for a given $s$ and $r$ there are only finitely many non-problematic values.

Some lemmas we have proved:

$(L\ r) = \{|v| \mid \vdash v : r\}$
$(L\ r) = \{|v| \mid \models v : r\}$
*If* $nullable\ r$ *then* $\vdash mkeps\ r : r$.
*If* $nullable\ r$ *then* $|mkeps\ r| = []$.
*If* $\vdash v : der\ c\ r$ *then* $\vdash (inj\ r\ c\ v) : r$.
*If* $\vdash v : der\ c\ r$ *then* $|inj\ r\ c\ v| = c :: (|v|)$.
*If* $nullable\ r$ *then* $[] \in r \rightarrow mkeps\ r$.
*If* $s \in r \rightarrow v$ *then* $|v| = s$.
*If* $s \in r \rightarrow v$ *then* $\models v : r$.
*If* $s \in r \rightarrow v_1$ *and* $s \in r \rightarrow v_2$ *then* $v_1 = v_2$.

This is the main theorem that lets us prove that the algorithm is correct according to $s \in r \rightarrow v$:
*If* $s \in der\ c\ r \rightarrow v$ *then* $(c :: s) \in r \rightarrow (inj\ r\ c\ v)$.

**Proof** The proof is by induction on the definition of *der*. Other inductions would go through as well. The interesting case is for $r_1 \cdot r_2$. First we analyse the case where $nullable\ r_1$. We have by induction hypothesis

$$(IH1) \quad \forall s\ v.\ \text{if } s \in der\ c\ r_1 \rightarrow v \text{ then } (c :: s) \in r_1 \rightarrow (inj\ r_1\ c\ v)$$
$$(IH2) \quad \forall s\ v.\ \text{if } s \in der\ c\ r_2 \rightarrow v \text{ then } (c :: s) \in r_2 \rightarrow (inj\ r_2\ c\ v)$$

and have

$$s \in (((der\ c\ r_1) \cdot r_2) + (der\ c\ r_2)) \rightarrow v$$

There are two cases what $v$ can be: (1) $Left\ v'$ and (2) $Right\ v'$.

(1) We know $s \in ((der\ c\ r_1) \cdot r_2) \rightarrow v'$ holds, from which we can infer that there are $s_1, s_2, v_1, v_2$ with

$$s_1 \in der\ c\ r_1 \rightarrow v_1 \qquad \text{and} \qquad s_2 \in r_2 \rightarrow v_2$$

and also

$$\nexists s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s_2 \wedge s_1 @ s_3 \in (L\ (der\ c\ r_1)) \wedge s_4 \in (L\ r_2)$$

and have to prove

$$(c :: s_1 @ s_2) \in (r_1 \cdot r_2) \rightarrow Seq\ (inj\ r_1\ c\ v_1)\ v_2$$

The two requirements $(c :: s_1) \in r_1 \rightarrow (inj\ r_1\ c\ v_1)$ and $s_2 \in r_2 \rightarrow v_2$ can be proved by the induction hypothese (IH1) and the fact above.
This leaves to prove

$$\nexists s_3\ s_4.\ s_3 \neq [] \wedge s_3 @ s_4 = s_2 \wedge c :: s_1 @ s_3 \in (L\ r_1) \wedge s_4 \in (L\ r_2)$$

which holds because $c :: s_1 @ s_3 \in (L\ r_1)$ implies $s_1 @ s_3 \in (L\ (der\ c\ r_1))$

(2) This case is similar.

The final case is that $\neg$ *nullable* $r_1$ holds. This case again similar to the cases above.

## References


1. A. Frisch and L. Cardelli. Greedy Regular Expression Matching. In *Proc. of the 31st International Conference on Automata, Languages and Programming (ICALP)*, volume 3142 of *LNCS*, pages 618–629, 2004.
2. C. Kuklewicz. Regex Posix. https://wiki.haskell.org/Regex_Posix.
3. 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.


## 6 Roy's Rules

$$Void \lhd \epsilon \qquad Char\ c \lhd Lit\ c$$

$$\frac{v_1 \lhd r_1}{Left\ v_1 \lhd r_1 + r_2} \qquad \frac{v_2 \lhd r_2 \qquad |v_2| \notin L(r_1)}{Right\ v_2 \lhd r_1 + r_2}$$

$$\frac{v_1 \lhd r_1 \qquad v_2 \lhd r_2 \qquad s \in L(r_1 \backslash |v_1|) \wedge |v_2| \backslash s \notin L(r_2) \Rightarrow s = []}{(v_1, v_2) \lhd r_1 \cdot r_2}$$

$$\frac{v \lhd r \qquad vs \lhd r^* \qquad |v| \neq []}{(v :: vs) \lhd r^*} \qquad [] \lhd r^*$$