POSIX Lexing with Derivatives of Regular Expressions (Proof Pearl)

Fahad Ausaf
King's College London

joint work with Roy Dyckhoff and Christian Urban

Regular Expressions

$$
\begin{array}{rcll}
r & ::= & \varnothing & \text{null} \\
 & \mid & \epsilon & \text{empty string} \\
 & \mid & c & \text{character} \\
 & \mid & r_1 \cdot r_2 & \text{sequence} \\
 & \mid & r_1 + r_2 & \text{alternative / choice} \\
 & \mid & r^* & \text{star (zero or more)}
\end{array}
$$

The Derivative of a Rexp

If r matches the string $c::s$, what is a regular expression that matches just s?

$der\ c\ r$ gives the answer, Brzozowski (1964), Owens (2005) "...have been lost in the sands of time..."

Correctness

It is a relative easy exercise in a theorem prover:

$$matches(r, s) \text{ if and only if } s \in L(r)$$

where $matches(r, s) \stackrel{\text{def}}{=} nullable(ders(r, s))$

POSIX Regex Matching

Two rules:

- Longest match rule ("maximal munch rule"): The longest initial substring matched by any regular expression is taken as the next token.

i	f	f	o	o	␣	b	l	a

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

i	f	␣	b	l	a

POSIX Regex Matching

Two rules:

- Longest match rule ("maximal munch rule"): The longest initial substring matched by any regular expression is taken as the next token.

i	f	f	o	o	␣	b	l	a

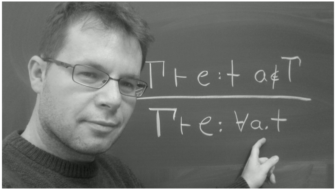
- Rule priority: For a particular longest initial substring, the first regular expression that can match determines the token.

i	f	␣	b	l	a

Kuklewicz: most POSIX matchers are buggy
http://www.haskell.org/haskellwiki/Regex_Posix

POSIX Regex Matching

- Sulzmann & Lu came up with a beautiful idea for how to extend the simple regular expression matcher to POSIX matching/lexing (FLOPS 2014)



Martin Sulzmann

- the idea: define an inverse operation to the derivatives

Regexes and Values

Regular expressions and their corresponding values (for *how* a regular expression matched a string):

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
r & ::= & \varnothing & v & ::= & \\
 & | & \epsilon & & & \mathit{Empty} \\
 & | & c & & | & \mathit{Char}(c) \\
 & | & r_1 \cdot r_2 & & | & \mathit{Seq}(v_1, v_2) \\
 & | & r_1 + r_2 & & | & \mathit{Left}(v) \\
 & & & & | & \mathit{Right}(v) \\
 & | & r^* & & | & [] \\
 & & & & | & [v_1, \ldots v_n]
\end{array}
$$

Regexes and Values

Regular expressions and their corresponding values (for *how* a regular expression matched a string):

$$
\begin{array}{rcl}
r & ::= & \varnothing \\
  & \mid & \epsilon \\
  & \mid & c \\
  & \mid & r_1 \cdot r_2 \\
  & \mid & r_1 + r_2 \\
  & & \\
  & \mid & r^*
\end{array}
\qquad\qquad
\begin{array}{rcl}
v & ::= & \\
  & & \mathit{Empty} \\
  & \mid & \mathit{Char}(c) \\
  & \mid & \mathit{Seq}(v_1, v_2) \\
  & \mid & \mathit{Left}(v) \\
  & \mid & \mathit{Right}(v) \\
  & \mid & [] \\
  & \mid & [v_1, \ldots\, v_n]
\end{array}
$$

There is also a notion of a string behind a value: $|v|$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$r_1 \xrightarrow{der\ a} r_2$$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$r_1 \xrightarrow{der\,a} r_2 \xrightarrow{der\,b} r_3$$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$r_1 \xrightarrow{der\,a} r_2 \xrightarrow{der\,b} r_3 \xrightarrow{der\,c} r_4$$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$r_1 \xrightarrow{der\, a} r_2 \xrightarrow{der\, b} r_3 \xrightarrow{der\, c} r_4 \quad nullable?$$

Sulzmann & Lu Matcher

We want to match the string *abc* using r_1:

$$\begin{tikzcd}
r_1 \arrow[r, "der\,a"] & r_2 \arrow[r, "der\,b"] & r_3 \arrow[r, "der\,c"] & r_4 \arrow[d] & nullable? \\
 & & & v_4 &
\end{tikzcd}$$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$
\begin{tikzcd}
r_1 \arrow[r, "der\ a"] & r_2 \arrow[r, "der\ b"] & r_3 \arrow[r, "der\ c"] & r_4 \arrow[d] & nullable? \\
& & v_3 & v_4 \arrow[l, "inj\ c"'] &
\end{tikzcd}
$$

Sulzmann & Lu Matcher

We want to match the string *abc* using r_1:

$$
\begin{tikzcd}
r_1 \arrow[r, "der\,a"] & r_2 \arrow[r, "der\,b"] & r_3 \arrow[r, "der\,c"] & r_4 \arrow[d] & nullable? \\
 & v_2 & v_3 \arrow[l, "inj\,b"'] & v_4 \arrow[l, "inj\,c"'] &
\end{tikzcd}
$$

Sulzmann & Lu Matcher

We want to match the string *abc* using r_1:

$$\begin{tikzcd}
r_1 \arrow[r, "der\ a"] & r_2 \arrow[r, "der\ b"] & r_3 \arrow[r, "der\ c"] & r_4 \arrow[d] & nullable? \\
v_1 & v_2 \arrow[l, "inj\ a"'] & v_3 \arrow[l, "inj\ b"'] & v_4 \arrow[l, "inj\ c"'] &
\end{tikzcd}$$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$\begin{tikzcd}
r_1 \arrow[r, "der\,a"] \arrow[d] & r_2 \arrow[r, "der\,b"] \arrow[d] & r_3 \arrow[r, "der\,c"] \arrow[d] & r_4 \arrow[d, "mkeps"] & nullable? \\
v_1 & v_2 \arrow[l, "inj\,a"] & v_3 \arrow[l, "inj\,b"] & v_4 \arrow[l, "inj\,c"] &
\end{tikzcd}$$

Sulzmann & Lu Matcher

We want to match the string abc using r_1:

$$\begin{tikzcd}
r_1 \arrow[r, "der\ a"] \arrow[d] & r_2 \arrow[r, "der\ b"] \arrow[d] & r_3 \arrow[r, "der\ c"] \arrow[d] & r_4 \arrow[d, "mkeps"] & nullable? \\
v_1 & v_2 \arrow[l, "inj\ a"] & v_3 \arrow[l, "inj\ b"] & v_4 \arrow[l, "inj\ c"] &
\end{tikzcd}$$

The original ideas of Sulzmann and Lu are the *mkeps* and *inj* functions (ommitted here).

Sulzmann & Lu Paper

- I have no doubt the algorithm is correct — the problem is I do not believe their proof.

> "How could I miss this? Well, I was rather careless when stating this Lemma :)
>
> Great example how formal machine checked proofs (and proof assistants) can help to spot flawed reasoning steps."

Sulzmann & Lu Paper

- I have no doubt the algorithm is correct — the problem is I do not believe their proof.

> "How could I miss this? Well, I was rather careless when stating this Lemma :)
>
> Great example how formal machine checked proofs (and proof assistants) can help to spot flawed reasoning steps."

> "Well, I don't think there's any flaw. The issue is how to come up with a mechanical proof. In my world mathematical proof = mechanical proof doesn't necessarily hold."

Sulzmann & Lu Paper

- I have no doubt the algorithm is correct — the problem is I do not believe their proof.

"How could I miss this? Well, I was rather careless

Lemma 3 (Projection and Injection). *Let r be a regular expression, l a letter and v a parse tree.*

1. *If $\vdash v : r$ and $|v| = lw$ for some word w, then $\vdash proj_{(r,l)}\ v : r\backslash l$.*
2. *If $\vdash v : r\backslash l$ then $(proj_{(r,l)} \circ inj_{r\backslash l})\ v = v$.*
3. *If $\vdash v : r$ and $|v| = lw$ for some word w, then $(inj_{r\backslash l} \circ proj_{(r,l)})\ v = v$.*

MS:BUG[**Come accross this issue when going back to our constructive reg-ex work**] Consider $\vdash$ [*Right* (), *Left* a] : $(a + \epsilon)^*$. However, $proj_{((a+\epsilon)^*,a)}$ [*Right* (), *Left* a] fails! The point is that *proj* only works correctly if applied on POSIX parse trees.

MS:Possible fixes We only ever apply *proj* on Posix parse trees.

For convenience, we write "$\vdash v : r$ is POSIX" where we mean that $\vdash v : r$ holds and v is the POSIX parse tree of r for word $|v|$.

Lemma 2 follows from the following statement.

necessarily hold."

The Proof Idea by Sulzmann & Lu

- introduce an inductively defined ordering relation $v \succ_r v'$ which captures the idea of POSIX matching
- the algorithm returns the maximum of all possible values that are possible for a regular expression.

The Proof Idea by Sulzmann & Lu

- introduce an inductively defined ordering relation $v \succ_r v'$ which captures the idea of POSIX matching
- the algorithm returns the maximum of all possible values that are possible for a regular expression.

- the idea is from a paper by Cardelli & Frisch about GREEDY matching (GREEDY $=$ preferring instant gratification to delayed repletion):
- e.g. given $(a + (b + ab))^*$ and string ab

$$\begin{array}{ll} \text{GREEDY:} & [Left(a), Right(Left(b))] \\ \text{POSIX:} & [Right(Right(Seq(a,b)))] \end{array}$$

$$\frac{}{\vdash Empty : \epsilon} \qquad \frac{}{\vdash Char(c) : c}$$

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

$$\frac{\vdash v : r_1}{\vdash Left(v) : r_1 + r_2} \qquad \frac{\vdash v : r_2}{\vdash Right(v) : r_1 + r_2}$$

$$\frac{}{\vdash [] : r^*} \qquad \frac{\vdash v_1 : r \quad \ldots \quad \vdash v_n : r}{\vdash [v_1, \ldots, v_n] : r^*}$$

Problems

- Sulzmann: ...Let's assume v is not a *POSIX* value, then there must be another one ...contradiction.

Problems

- Sulzmann: ...Let's assume v is not a *POSIX* value, then there must be another one ...contradiction.

- Exists?

$$L(r) \neq \varnothing \;\Rightarrow\; \exists v.\, POSIX(v, r)$$

Problems

- Sulzmann: ...Let's assume v is not a *POSIX* value, then there must be another one ...contradiction.

- Exists?

$$L(r) \neq \varnothing \;\Rightarrow\; \exists v.\, POSIX(v, r)$$

- in the sequence case $Seq(v_1, v_2) \succ_{r_1 \cdot r_2} Seq(v'_1, v'_2)$, the induction hypotheses require $|v_1| = |v'_1|$ and $|v_2| = |v'_2|$, but you only know

$$|v_1| \mathbin{@} |v_2| = |v'_1| \mathbin{@} |v'_2|$$

Problems

- Sulzmann: ...Let's assume v is not a *POSIX* value, then there must be another one ...contradiction.

- Exists?

$$L(r) \neq \varnothing \;\Rightarrow\; \exists v.\, POSIX(v, r)$$

- in the sequence case $Seq(v_1, v_2) \succ_{r_1 \cdot r_2} Seq(v'_1, v'_2)$, the induction hypotheses require $|v_1| = |v'_1|$ and $|v_2| = |v'_2|$, but you only know

$$|v_1| \,@\, |v_2| = |v'_1| \,@\, |v'_2|$$

- although one begins with the assumption that the two values have the same flattening, this cannot be maintained as one descends into the induction (alternative, sequence)

Our Solution

- a direct definition of what a POSIX value is, using the relation $s \in r \to v$ (specification):

$$\frac{}{[] \in \epsilon \to \mathit{Empty}} \qquad \frac{}{c \in c \to \mathit{Char}(c)}$$

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

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

...

Properties

It is almost trival to prove:

- Uniqueness

  If $s \in r \rightarrow v_1$ and $s \in r \rightarrow v_2$ then $v_1 = v_2$.

- Correctness

  $lexer(r,s) = v$ if and only if $s \in r \rightarrow v$

Properties

It is almost trival to prove:

- Uniqueness

  If $s \in r \to v_1$ and $s \in r \to v_2$ then $v_1 = v_2$.

- Correctness

  $lexer(r, s) = v$ if and only if $s \in r \to v$

You can now start to implement optimisations and derive correctness proofs for them. But we still do not know whether

$$s \in r \to v$$

is a POSIX value according to Sulzmann & Lu's definition (biggest value for s and r)

Conclusion

- we replaced the POSIX definition of Sulzmann & Lu by a new definition (ours is inspired by work of Vansummeren, 2006)
- their proof contained small gaps (acknowledged) but had also fundamental flaws
- now, its a nice exercise for theorem proving
- some optimisations need to be applied to the algorithm in order to become fast enough
- can be used for lexing, is a small beautiful functional program

Questions?