POSIX Lexing with Derivatives of Regular Expressions (Proof Pearl)

Fahad Ausaf, Roy Dyckhoff and Christian Urban

King's College London, University of St Andrews

Brzozowski's Derivatives of Regular Expressions

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

chars:

$$\mathbf{0}\backslash c \overset{\text{def}}{=} \mathbf{0}$$

$$\mathbf{1}\backslash c \overset{\text{def}}{=} \mathbf{0}$$

$$d\backslash c \overset{\text{def}}{=} \textit{if } d = c \textit{ then } \mathbf{1} \textit{ else } \mathbf{0}$$

$$r_1 + r_2\backslash c \overset{\text{def}}{=} r_1\backslash c + r_2\backslash c$$

$$\begin{aligned} r_1 \cdot r_2\backslash c \overset{\text{def}}{=} & \textit{ if nullable } r_1 \\ & \textit{ then } r_1\backslash c \cdot r_2 + r_2\backslash c \textit{ else } r_1\backslash c \cdot r_2 \end{aligned}$$

$$r^*\backslash c \overset{\text{def}}{=} r\backslash c \cdot r^*$$

strings:

$$r\backslash[] \overset{\text{def}}{=} r$$

$$r\backslash c :: s \overset{\text{def}}{=} (r\backslash c)\backslash s$$

Brzozowski's Matcher

Does r_1 match string abc?

$$r_1 \xrightarrow{\_\backslash a} r_2 \xrightarrow{\_\backslash b} r_3 \xrightarrow{\_\backslash c} r_4 \quad \mathit{nullable}?$$

Brzozowski's Matcher

Does r_1 match string abc?

$$r_1 \xrightarrow{\_\backslash a} r_2 \xrightarrow{\_\backslash b} r_3 \xrightarrow{\_\backslash c} r_4 \quad \mathit{nullable}?$$

It leads to an elegant functional program:

$$\mathit{matches}(r, s) \overset{\text{def}}{=} \mathit{nullable}(r \backslash s)$$

Brzozowski's Matcher

Does r_1 match string abc?

$$r_1 \xrightarrow{\_\backslash a} r_2 \xrightarrow{\_\backslash b} r_3 \xrightarrow{\_\backslash c} r_4 \quad \mathit{nullable}?$$

It leads to an elegant functional program:

$$\mathit{matches}(r, s) \overset{\text{def}}{=} \mathit{nullable}(r\backslash s)$$

It is an easy exercise to formally prove (e.g. Coq, HOL, Isabelle):

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

Brzozowski's Matcher

Does r_1 match string abc?

$$r_1 \xrightarrow{\_\backslash a} r_2 \xrightarrow{\_\backslash b} r_3 \xrightarrow{\_\backslash c} r_4 \quad nullable?$$

It leads to an elegant functional program:

$$matches(r, s) \overset{\text{def}}{=} nullable(r \backslash s)$$

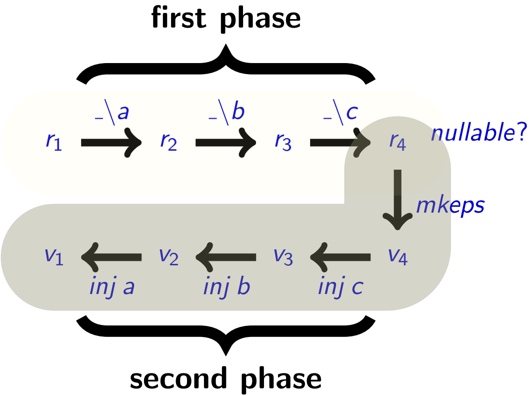
It is an easy exercise to formally prove (e.g. Coq, HOL, Isabelle):

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

But Brzozowski's matcher gives only a yes/no-answer.

Sulzmann and Lu's Matcher

Sulzmann and Lu added a second phase in order to answer **how** the regular expression matched the string.



There are several possible answers for **how**: POSIX, GREEDY, ...

Regular Expressions and Values

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

$$
\begin{array}{lcl}
r & ::= & \mathbf{0} \\
  & \mid & \mathbf{1} \\
  & \mid & c \\
  & \mid & r_1 \cdot r_2 \\
  & \mid & r_1 + r_2 \\
  &   & \\
  & \mid & r^*
\end{array}
\qquad\qquad
\begin{array}{lcl}
v & ::= & \\
  & \mid & \mathit{Empty} \\
  & \mid & \mathit{Char}(c) \\
  & \mid & \mathit{Seq}(v_1 \cdot v_2) \\
  & \mid & \mathit{Left}(v) \\
  & \mid & \mathit{Right}(v) \\
  & \mid & [v_1, \ldots, v_n]
\end{array}
$$

POSIX Matching (needed for Lexing)

Longest Match Rule: The longest initial substring matched by any regular expression is taken as the next token.

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

For example: $r_{keywords} + r_{identifiers}$:

- `i f f o o ␣ b l a`
- `i f ␣ b l a`

Problems with POSIX

Grathwohl, Henglein and Rasmussen wrote:

> *"The POSIX strategy is more complicated than the greedy because of the dependence on information about the length of matched strings in the various subexpressions."*

Also Kuklewicz maintains a unit-test repository for POSIX matching, which indicates that most POSIX mathcers are buggy.

`http://www.haskell.org/haskellwiki/Regex_Posix`

“Correctness” by Sulzmann and Lu

- Sulzmann & Lu’s idea is to order all possible answer such that they can prove the correct answer is the maximum
- The idea is taken from a GREEDY algorithm (and it works there)

“Correctness” by Sulzmann and Lu

- Sulzmann & Lu’s idea is to order all possible answer such that they can prove the correct answer is the maximum
- The idea is taken from a GREEDY algorithm (and it works there)

- **But** we made no progress in formalising Sulzmann & Lu’s idea, because
  - transitivity, existence of maxima etc all fail to turn into real proofs
  - the reason: the ordering works only if ....
  - though we did find mistakes:

> “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.”

"Correctness" by Sulzmann and Lu

- Sulzmann & Lu's idea is to order all possible answer such that they can prove the correct answer is the maximum
- The idea is taken from a GREEDY algorithm (and it works there)

- **But** we made no progress in formalising Sulzmann & Lu's idea, because
  - transitivity, existence of maxima etc all fail to turn into real proofs
  - the reason: the ordering works only if ....
  - though we did find mistakes:

> "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."

"Correctness" by Sulzmann and Lu

- Sulzmann & Lu's idea is to order all possible answer such that they can prove the correct answer is the maximum
- The idea is taken from a GREEDY algorithm (and it works there)

- **But** we made no progress in formalising Sulzmann & Lu's idea, because
  - transitivity, existence of maxima etc all fail to turn into real proofs
  - the reason: the ordering works only if ....
  - though we did find mistakes:

> "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 and Lu Matcher

We want to match the string *abc* using r_1

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

Sulzmann and 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 and 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 and 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 and 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 and 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 and 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 and 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 and Lu Matcher

We want to match the string *abc* using r_1

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

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

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

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 $s \in r \rightarrow v$ (our specification)

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

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

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

...

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)