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?

$$
\begin{array}{lll}
\text{chars:} & \mathbf{0}\backslash c & \stackrel{\text{def}}{=} \mathbf{0} \\
& \mathbf{1}\backslash c & \stackrel{\text{def}}{=} \mathbf{0} \\
& d\backslash c & \stackrel{\text{def}}{=} \textit{if } d = c \textit{ then } \mathbf{1} \textit{ else } \mathbf{0} \\
& r_1 + r_2\backslash c & \stackrel{\text{def}}{=} r_1\backslash c \; + \; r_2\backslash c \\
& r_1 \cdot r_2\backslash c & \stackrel{\text{def}}{=} \textit{if nullable } r_1 \\
& & \quad \textit{then } r_1\backslash c \cdot r_2 \; + \; r_2\backslash c \textit{ else } r_1\backslash c \cdot r_2 \\
& r^*\backslash c & \stackrel{\text{def}}{=} r\backslash c \; \cdot \; r^* \\
\\
\text{strings:} & r\backslash [] & \stackrel{\text{def}}{=} r \\
& r\backslash c::s & \stackrel{\text{def}}{=} (r\backslash c)\backslash s
\end{array}
$$

Brzozowski's Matcher

Does r_1 match string abc?

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

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:

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

Brzozowski's Matcher

Does r_1 match string abc?

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

It leads to an elegant functional program:

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

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

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

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:

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

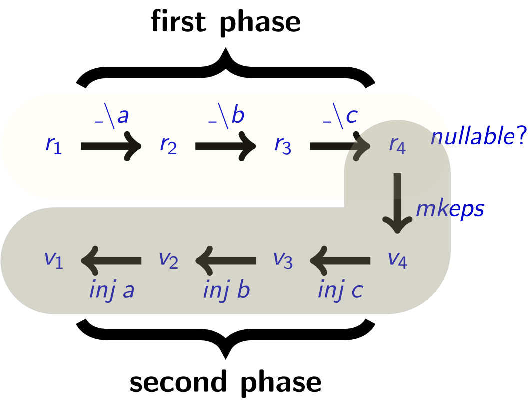
It is an easy exercise to formally prove (e.g. Coq, HOL, Isabelle):

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

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, . . .

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: $r_{keywords} + r_{identifiers}$ (fix graphics below)

`iffoo_bla`

`if_bla`

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`

Regular Expressions and Values

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

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

There is also a notion of a string behind a value $\mid v \mid$

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

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

Sulzmann and Lu Matcher

We want to match the string *abc* using r_1

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

Sulzmann and Lu Matcher

We want to match the string *abc* using r_1

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

Sulzmann and Lu Matcher

We want to match the string abc using r_1

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

Sulzmann and Lu Matcher

We want to match the string *abc* using r_1

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

Nullable, Mkeps and Injection Functions

Nullable Function

$$
\begin{aligned}
nullable\ (\mathbf{0}) &\stackrel{\text{def}}{=} False \\
nullable\ (\mathbf{1}) &\stackrel{\text{def}}{=} True \\
nullable\ (c) &\stackrel{\text{def}}{=} False \\
nullable\ (r_1 + r_2) &\stackrel{\text{def}}{=} nullable\ r_1 \vee nullable\ r_2 \\
nullable\ (r_1 \cdot r_2) &\stackrel{\text{def}}{=} nullable\ r_1 \wedge nullable\ r_2 \\
nullable\ (r^*) &\stackrel{\text{def}}{=} True
\end{aligned}
$$

Mkeps Function

$$
\begin{aligned}
mkeps\ (\mathbf{1}) &\stackrel{\text{def}}{=} () \\
mkeps\ (r_1 \cdot r_2) &\stackrel{\text{def}}{=} Seq\ (mkeps\ r_1)\ (mkeps\ r_2) \\
mkeps\ (r_1 + r_2) &\stackrel{\text{def}}{=} if\ nullable\ r_1\ then\ Left\ (mkeps\ r_1) \\
&\quad\ \ else\ Right\ (mkeps\ r_2) \\
mkeps\ (r^*) &\stackrel{\text{def}}{=} Stars\ []
\end{aligned}
$$

Injection Function

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

POSIX Ordering Relation by Sulzmann & Lu

- Introduce an inductive 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 Frisch & Cardelli about GREEDY matching (GREEDY = preferring instant gratification to delayed repletion):
- e.g. given $(a + (b + ab))^*$ and string ab

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

POSIX Ordering Relation by Sulzmann & Lu

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

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

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

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

Problems

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

Problems

- 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."

Our Solution

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

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

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

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

...

Properties

It is almost trival to prove:

- Uniqueness

$$\text{If } s \in r \rightarrow v_1 \text{ and } s \in r \rightarrow v_2 \text{ then } v_1 = v_2$$

- Correctness

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

Properties

It is almost trival to prove:

- Uniqueness

$$\text{If } s \in r \to v_1 \text{ and } s \in r \to v_2 \text{ then } v_1 = v_2$$

- Correctness

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

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)

Conclusions

- 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