POSIX Regular Expression Parsing with Derivatives

Martin Sulzmann[1] and Kenny Zhuo Ming Lu [2]

Hochschule Karlsruhe - Technik und Wirtschaft
martin.sulzmann@hs-karlsruhe.de

Nanyang Polytechnic
luzhuomi@gmail.com

June 5, 2014

Regular Expression Matching is Ambiguous

- Regular Expression matching is ambiguous
- For example, there are two ways of matching “ab” with

$$(a + (b + ab))^*$$

Regular Expression Matching is Ambiguous

- Regular Expression matching is ambiguous
- For example, there are two ways of matching "ab" with

$$(a + (b + ab))^*$$

Regular Expression Matching is Ambiguous

- Regular Expression matching is ambiguous
- For example, there are two ways of matching "ab" with

$$(a + (b + ab))^*$$

Regular Expression Matching is Ambiguous

- Regular Expression matching is ambiguous
- Several disambiguation strategies exist
  - POSIX $\surd$
  - PCRE
  - ...

"Subpatterns should match the longest possible substrings, where sub- patterns that start earlier (to the left) in the regular expression take priority over ones starting later. Hence, higher-level subpatterns take priority over their lower-level component subpatterns. Matching an empty string is considered longer than no match at all."

$$(a + (b + ab))^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
    - consider matching "abcd" with

$$(a + b + c + d + ab + bc + cd + abc + bcd + abcd)^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
    - consider matching "abcd" with

      $$(a + b + c + d + ab + bc + cd + abc + bcd + abcd)^*$$

    - re2 yields

      $$(\color{red}{a} + \color{blue}{b} + \color{magenta}{c} + \color{green}{d} + ab + bc + cd + abc + bcd + abcd)^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
    - consider matching "abcd" with

$$(a + b + c + d + ab + bc + cd + abc + bcd + abcd)^*$$

    - re2 yields

$$({\color{red}a} + {\color{blue}b} + {\color{magenta}c} + {\color{green}d} + ab + bc + cd + abc + bcd + abcd)^*$$

    - POSIX matching should yield

$$(a + b + c + d + ab + bc + cd + abc + bcd + {\color{red}abcd})^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
  - The default C POSIX shipped with Linux (GNU) and Mac OS (BSD)
    - consider matching "abcdefg" with

$$(a + bcdef + g + ab + c + d + e + efg + fg)^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
  - The default C POSIX shipped with Linux (GNU) and Mac OS (BSD)
    - consider matching "abcdefg" with

$$(a + bcdef + g + ab + c + d + e + efg + fg)^*$$

    - default C POSIX yields

$$({\color{red}a} + {\color{blue}bcdef} + {\color{magenta}g} + ab + c + d + e + efg + fg)^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
  - The default C POSIX shipped with Linux (GNU) and Mac OS (BSD)
    - consider matching "abcdefg" with

$$(a + bcdef + g + ab + c + d + e + efg + fg)^*$$

    - default C POSIX yields

$$(\color{red}{a} + \color{blue}{bcdef} + \color{magenta}{g} + ab + c + d + e + efg + fg)^*$$

    - POSIX matching should yield

$$(a + bcdef + g + \color{red}{ab} + \color{blue}{c} + \color{magenta}{d} + e + \color{green}{efg} + fg)^*$$

POSIX Implementation is Tricky

- We found bugs in existing NFA approaches
  - Google's re2
  - The default C POSIX shipped with Linux (GNU) and Mac OS (BSD)
  - Many others
- Kuklewicz's TDFA addresses the POSIX matching

  http://hackage.haskell.org/package/regex-tdfa
- We consider POSIX parsing using derivative (a DFA approach)

Parsing as Type Inhabitance Check

- Parsing "ab" with

$$(a + (b + ab))^*$$

produces

$$[Right\ Right\ (a, b)]$$

- Can be viewed as a type inhabitance check

$$\vdash [Right\ Right\ (a, b)] : (a + (b + ab))^*$$

i.e. Parse trees are proof terms of the type inhabitance check. Detailed rules and definitions can be found in the paper.

Regex Parsing is also ambiguous

$$
\dfrac{\dfrac{\dfrac{\vdash\ a : a \qquad \vdash\ b : b}{\vdash\ (a, b) : ab}}{\vdash\ \mathit{Right}\ (a, b) : (a + ab)} \qquad \dfrac{\vdash\ () : \epsilon}{\vdash\ \mathit{Right}\ () : (b + \epsilon)}}{\vdash\ (\mathit{Right}\ (a, b), \mathit{Right}()) : (a + ab)(b + \epsilon)}
$$

versus

$$
\dfrac{\dfrac{\vdash\ a : a}{\vdash\ \mathit{Left}\ a : a + ab} \qquad \dfrac{\vdash\ b : b}{\vdash\ \mathit{Left}\ b : (b + \epsilon)}}{\vdash\ (\mathit{Left}\ a, \mathit{Left}\ b) : (a + ab)(b + \epsilon)}
$$

The Key Idea

- We use a DFA construction via derivatives
- Brzozowski's Regular expression derivatives

$$
\begin{array}{lcl}
\phi \backslash l & = & \phi \\
\epsilon \backslash l & = & \phi \\
l_1 \backslash l_2 & = & \left\{ \begin{array}{ll} \epsilon & \text{if } l_1 == l_2 \\ \phi & \text{otherwise} \end{array} \right. \\
(r_1 + r_2) \backslash l & = & r_1 \backslash l + r_2 \backslash l \\
(r_1 r_2) \backslash l & = & \left\{ \begin{array}{ll} (r_1 \backslash l) r_2 + r_2 \backslash l & \text{if } \epsilon \in L(r_1) \\ (r_1 \backslash l) r_2 & \text{otherwise} \end{array} \right. \\
r^* \backslash l & = & (r \backslash l) r^*
\end{array}
$$

The Key Idea

1. Apply Brzozowski's Derivative to perform the matching by extraction pass

   $$\text{Matching by extraction:} \qquad r_0 \xrightarrow{l_1} r_1 \xrightarrow{l_2} \ldots \xrightarrow{l_n} r_n$$

   $r \xrightarrow{l} r'$ is s short hand for $r \backslash l = r'$.

2. Build the empty proof term v_n of the final expression r_n
3. Reversing the previous pass to construct the parse trees by injection.

   $$\text{Parse trees by injection:} \qquad v_0 \xleftarrow{l_1} v_1 \xleftarrow{l_2} \ldots \xleftarrow{l_n} v_n$$

   $v \xleftarrow{l} v'$ is short-hand for $v = inj_{r\backslash l} v'$ where $\vdash v : r$ and $\vdash v' : r\backslash l$

A POSIX Parsing Example

$(a + ab)(b + \epsilon)$

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \quad \xrightarrow{a} \qquad (\epsilon + \epsilon b)(b + \epsilon)$$

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \quad \xrightarrow{a} \quad (\epsilon + \epsilon b)(b + \epsilon) \quad \xrightarrow{b} \quad (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi)$$

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \xrightarrow{a} (\epsilon + \epsilon b)(b + \epsilon) \xrightarrow{b} (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi)$$

$$mkEps_{(\phi+(\phi b+\epsilon))(b+\epsilon)+(\epsilon+\phi)}$$

Building the Final Proof - Success Case

$$
\begin{array}{l}
mkEps_{r^*} \ = [] \\
mkEps_{\,r_1\ r_2} \ = (mkEps_{r_1},\ mkEps_{r_2}) \\
mkEps_{\,r_1+r_2} \\
\quad \mid \epsilon \in L(r_1) \ = \ Left \ \ mkEps_{r_1} \\
\quad \mid \epsilon \in L(r_2) \ = \ Right \ \ mkEps_{r_2} \\
mkEps_{\epsilon} \ = \ ()
\end{array}
$$

$$mkEps_{(\phi+(\phi b+\epsilon))(b+\epsilon)+(\epsilon+\phi)} = Left\ (Right\ (Right\ ()), Right\ ())$$

- We have that $\vdash mkEps_r : r$ and $|mkEps_r| = \epsilon$ if $\epsilon \in L(r)$.
- Yields the empty POSIX parse tree.

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \xrightarrow{a} (\epsilon + \epsilon b)(b + \epsilon) \xrightarrow{b} (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi)$$

$$\mathit{Left}\ (\mathit{Right}\ (\mathit{Right}\ ()), \mathit{Right}\ ())$$

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \;\xrightarrow{a}\; (\epsilon + \epsilon b)(b + \epsilon) \;\xrightarrow{b}\; (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi)$$

$$inj_{((\epsilon+\epsilon b)(b+\epsilon))\backslash b}\ v_3 \;\xleftarrow{b}\; Left\ (Right\ (Right\ ()),\ Right\ ())$$

$$||$$

$$v_3$$

Derivative-Guided Proof Transformation

Brzozowski's Derivatives:

$$(r_1 r_2)\backslash l \quad = \quad \begin{cases} (r_1\backslash l r_2) + r_2\backslash l & \text{if } \epsilon \in L(r_1) \\ (r_1\backslash l r_2) & \text{otherwise} \end{cases}$$

Derivative-Guided Proof Transformation

Brzozowski's Derivatives:

$$(r_1 r_2)\backslash l \quad = \quad \begin{cases} (r_1\backslash l r_2) + r_2\backslash l & \text{if } \epsilon \in L(r_1) \\ (r_1\backslash l r_2) & \text{otherwise} \end{cases}$$

Proof Transformation:

$$\begin{array}{l} inj_{(r_1 r_2)\backslash l}\ v \\ \quad \begin{array}{ll} |\epsilon \in L(r_1) = & \text{case } v \text{ of} \\ & \quad Left\ (v_1, v_2) \rightarrow (inj_{r_1\backslash l}\ v_1, v_2) \\ & \quad Right\ \ v_2 \rightarrow (mkEps_{r_1}, inj_{r_2\backslash l}\ v_2) \\ |\text{otherwise} = & \text{case } v \text{ of} \\ & \quad (v_1, v_2) \rightarrow (inj_{r_1\backslash l}\ v_1, v_2) \end{array} \end{array}$$

Derivative-Guided Proof Transformation

Brzozowski's Derivatives:

$$(r_1 r_2)\backslash l \quad = \quad \begin{cases} (r_1\backslash l r_2) + r_2\backslash l & \text{if } \epsilon \in L(r_1) \\ (r_1\backslash l r_2) & \text{otherwise} \end{cases}$$

Proof Transformation:

$$\begin{array}{l} inj_{(r_1 r_2)\backslash l}\ v \\ \quad |\epsilon \in L(r_1) = \text{case } v \text{ of} \\ \qquad\qquad Left\ (v_1, v_2) \rightarrow (inj_{r_1\backslash l}\ v_1, v_2) \\ \qquad\qquad Right\ v_2 \rightarrow (mkEps_{r_1}, inj_{r_2\backslash l}\ v_2) \\ \quad |\text{otherwise} = \text{case } v \text{ of} \\ \qquad\qquad (v_1, v_2) \rightarrow (inj_{r_1\backslash l}\ v_1, v_2) \end{array}$$

Derivative-Guided Proof Transformation

- If $\vdash v : r\backslash l$ then $\vdash inj_{r\backslash l}\ v : r$.
- Injection preserves POSIX parse trees.
- Given

$$
\begin{array}{l}
inj_{r_1 r_2\backslash l}\ v \\
\quad |\epsilon \in L(r_1) = \text{case } v \text{ of} \\
\qquad\qquad Left\ (v_1, v_2) \rightarrow (inj_{r_1\backslash l}\ v_1, v_2) \\
\qquad\qquad Right\ v_2 \rightarrow (mkEps_{r_1}, inj_{r_2\backslash l}\ v_2) \\
\quad |\text{otherwise} \ = \text{case } v \text{ of} \\
\qquad\qquad (v_1, v_2) \rightarrow (inj_{r_1\backslash l}\ v_1, v_2) \\
inj_{r_1+r_2\backslash l}\ (Left\ v_1)|\epsilon \in L(r_1) = Left\ inj_{r_1\backslash l}\ v_1 \\
inj_{r_1+r_2\backslash l}\ (Right\ v_2)|\epsilon \in L(r_2) = Right\ inj_{r_2\backslash l}\ v_2
\end{array}
$$

we have

$$
\begin{array}{ll}
inj_{((\epsilon+\epsilon b)(b+\epsilon))\backslash b}\ (Left\ (Right\ (Right\ ()), Right\ ())) & \longrightarrow \\
(inj_{(\epsilon+\epsilon b)\backslash b}\ (Right\ (Right\ ()), Right\ ())) & \longrightarrow \\
(Right\ (inj_{\epsilon b\backslash b}\ (Right\ ())), Right\ ()) & \longrightarrow^* \\
(Right\ ((), b), Right\ ()) &
\end{array}
$$

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \xrightarrow{a} (\epsilon + \epsilon b)(b + \epsilon) \xrightarrow{b} (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi)$$

$$\underbrace{(\mathit{Right}\ ((), b), \mathit{Right}\ ())}_{v_2} \xleftarrow{b} \underbrace{\mathit{Left}\ (\mathit{Right}\ (\mathit{Right}\ ()), \mathit{Right}\ ())}_{v_3}$$

A POSIX Parsing Example

$$
\begin{array}{ccccc}
(a + ab)(b + \epsilon) & \xrightarrow{a} & (\epsilon + \epsilon b)(b + \epsilon) & \xrightarrow{b} & (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi) \\
\\
inj_{((a+ab)(b+\epsilon))\backslash a}\ v_2 & \xleftarrow{a} & (Right\ ((), b), Right\ ()) & \xleftarrow{a} & Left\ (Right\ (Right\ ()), Right\ ()) \\
 & & \| & & \| \\
 & & v_2 & & v_3
\end{array}
$$

A POSIX Parsing Example

$$(a + ab)(b + \epsilon) \xrightarrow{a} (\epsilon + \epsilon b)(b + \epsilon) \xrightarrow{b} (\phi + (\phi b + \epsilon))(b + \epsilon) + (\epsilon + \phi)$$

$$(\mathit{Right}\ (a, b), \mathit{Right}\ ()) \xleftarrow{a} \underbrace{(\mathit{Right}\ ((), b), \mathit{Right}\ ())}_{= v_2} \xleftarrow{b} \underbrace{\mathit{Left}\ (\mathit{Right}\ (\mathit{Right}\ ()), \mathit{Right}\ ())}_{= v_3}$$

Implementation

- Implemented in Haskell
- Apply simplification to ensure the finiteness of the derivatives
- BitCoding Forward Parsing for efficiency
- Compilation by constructing DFA from the derivatives
- Fine-tuned (sub-matching) version to keep the last match of Kleene's star
- Competitive performance

Our Contributions and Conclusion

- Formally define POSIX parsing by view regular expressions as types and parse tree as values.
- Relate parsing to the more specific submatching problem.
- Present a method to compute POSIX parse trees based on Brzozowski's derivatives and verify its correctness
- Built an optimized versions for parsing as well as submatching.

Thank you. Questions are welcome!