9-Month Progress Report

Meshal Binnasban

King's College London — Department of Informatics

October 30, 2025

Overview

- Introduction
  - Brzozowski's Derivatives
  - Size Explosion
- Marked Approach
  - Fischer et al.'s Marked Approach
  - Limitations & Our Direction
- Our Work
  - String-Carrying Marks
  - *shifts* Definition

Brzozowski's Derivatives

- The notion of derivatives for regular expression matching was introduced by Brzozowski (1964).
- Regained interest in the last decade.[1]
- Compute derivatives successively with respect to each input character.
- Acceptance for a word $w = a_1 \cdots a_n$ is tested by: $\varepsilon \in L(der_{a_n}(\cdots(der_{a_1}(r))))$.
- Elegant in design; can be easily implemented in functional programming languages and reasoned about in theorem provers.
- **Challenge — Size Explosion:** successive derivatives can rapidly increase the expression size, especially in sequences and Kleene stars.

[1] See Owens [4] and Might [3].

Brzozowski's Derivative

$$
\begin{array}{lcl}
der_a(\mathbf{0}) & \Rightarrow & \mathbf{0} \\
der_a(\mathbf{1}) & \Rightarrow & \mathbf{0} \\
der_a(c) & \Rightarrow & \begin{cases} \mathbf{1} & \text{if } a = c, \\ \mathbf{0} & \text{otherwise.} \end{cases} \\
der_a(r_1 + r_2) & \Rightarrow & der_a(r_1) + der_a(r_2) \\
der_a(r_1 \cdot r_2) & \Rightarrow & der_a(r_1) \cdot r_2 + \begin{cases} der_a(r_2) & \text{if } nullable(r_1), \\ \mathbf{0} & \text{otherwise.} \end{cases} \\
der_a(r^*) & \Rightarrow & der_a(r) \cdot r^*
\end{array}
$$

Brzozowski's derivatives

Matching Example

Regular expression $(ab + ba)$ **String** ba

$$
\begin{aligned}
der_b\, r &= der_b\,(ab + ba) \\
&= der_b\,(ab) + der_b\,(ba) \\
&= (der_b\, a) \cdot b + (der_b\, b) \cdot a \\
&= \mathbf{0} \cdot b + \mathbf{1} \cdot a
\end{aligned}
$$

$$
\begin{aligned}
der_a\,(der_b\, r) &= der_a\,(\mathbf{0} \cdot b + \mathbf{1} \cdot a) \\
&= der_a\,(\mathbf{0} \cdot b) + der_a\,(\mathbf{1} \cdot a) \\
&= der_a\,(\mathbf{0}) \cdot b + (der_a\,(\mathbf{1}) \cdot a + der_a\, a) \\
&= \mathbf{0} \cdot b + (\mathbf{0} \cdot a + \mathbf{1})
\end{aligned}
$$

Size Explosion

- Expression size can increase as new subexpressions are introduced, particularly in sequences and Kleene stars.
- This repeated expansion may lead to **size explosion**, where the number of derivative expressions grows substantially with input length.
- Simplifications can reduce redundancy but cannot fully prevent size explosion in the worst case.
- Subexpressions cannot always be simplified, such as when identical terms are separated by other expressions or occur at different nesting levels.

Regular expression $((a)^* + (aa)^* + (aaa)^* + (aaaa)^* + (aaaaa)^*)^*$ **String** $\underbrace{a \ldots a}_{n}$

$$
\begin{array}{rcl}
der_a\, r & = & ((a)^* + (a \cdot (aa)^*) + (aa \cdot (aaa)^*) + \ldots) \cdot r^* \\
der_a(der_a\, r) & = & ((a)^* + (aa)^* + (a \cdot (aaa)^*) + \ldots) \cdot r^* \; + \; ((a)^* + (a \cdot (aa)^*) + \ldots) \cdot r^* \\
 & \vdots &
\end{array}
$$

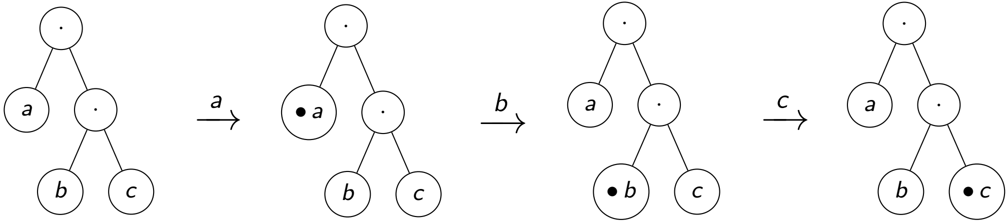
Fischer et al.'s Marked Approach

- Moves marks through the regular expression without modifying it.
- Marks indicate the progress of matching; acceptance occurs when a mark reaches a final position.
- The structure of the regular expression remains **unchanged** during matching.
- A promising alternative to derivative- and automaton-based approaches, as the regular expression structure remains fixed in size.
- Existing works focus on **matching only**, without value extraction.[2]

[2] See Fischer et al. [2] and Asperti et al. [1].

Shift Example

Regular expression $a \cdot (b \cdot c)$ **String** abc



Limitations & Our Direction

- Our main aim is to extend the marked approach (currently acceptance only) to support **POSIX value extraction** and to formally verify its correctness.
- One of our goals is also to handle additional constructors, such as bounded repetition and intersection.
- We extended marks from single-character matching to **string-carrying marks**, where each mark carries the remaining input suffix.
- The basic mark representation loses valuable information, such as ordering, and cannot directly handle constructors like intersection.
- Extending the mark structure allows this information to be retained, giving better control over ordering and disambiguation.
- Future direction: continue the lexing version and establish formal verification and equivalence with the derivative-based semantics.

String-Carrying Marks

- Each mark carries the **remaining input suffix**.
- Matching begins with an initial mark containing the full input string.
- At each shift, a matching character is stripped from the mark's string.
- A match occurs when an **empty mark** is produced, indicating full input consumption.

shifts Definition

$$
\begin{array}{rcl}
\mathit{shifts}(ms, 0) & \Rightarrow & [\,] \\
\mathit{shifts}(ms, 1) & \Rightarrow & [\,] \\
\mathit{shifts}(ms, d) & \Rightarrow & [\bullet s \mid \bullet d :: s \in ms\,] \\
\mathit{shifts}(ms, r_1 + r_2) & \Rightarrow & \mathit{shifts}(ms, r_1)\ @\ \mathit{shifts}(ms, r_2) \\
\mathit{shifts}(ms, r_1 \cdot r_2) & \Rightarrow & \text{let } ms' = \mathit{shifts}(ms, r_1) \text{ in}
\end{array}
$$

$$
\begin{cases}
\mathit{shifts}(ms' @ ms, r_2) @ ms' & \text{if } \mathit{nullable}(r_1) \wedge \mathit{nullable}(r_2), \\
\mathit{shifts}(ms' @ ms, r_2) & \text{if } \mathit{nullable}(r_1), \\
\mathit{shifts}(ms', r_2) @ ms' & \text{if } \mathit{nullable}(r_2), \\
\mathit{shifts}(ms', r_2) & \text{otherwise.}
\end{cases}
$$

$$
\begin{array}{rcl}
\mathit{shifts}(ms, r^*) & \Rightarrow & \text{let } ms' = \mathit{shifts}(ms, r) \text{ in} \\
 & & \text{if } ms' = [\,] \text{ then } [\,] \text{ else } \mathit{shifts}(ms', r^*) @ ms'
\end{array}
$$

Each mark carries a remaining input suffix; ms is a list of marks.

Matching Example

Regular expression $(a + (a \cdot c))$ **String** ac

1. $\big|_{[\bullet\, ac]}\ (a + (a \cdot c))$
2. $(\big|_{[\bullet\, ac]} a) \ + \ (\big|_{[\bullet\, ac]} (a \cdot c))$
3. $(a \ \big|_{[\bullet\, c]}) \ + \ ((a \cdot c) \ \big|_{[\bullet\, []]})$
4. $[_{\bullet}\, c,_{\bullet}\, []]$

A. Asperti, C. S. Coen, and E. Tassi.
Regular Expressions, au point.
arXiv, http://arxiv.org/abs/1010.2604, 2010.

S. Fischer, F. Huch, and T. Wilke.
A Play on Regular Expressions: Functional Pearl.
In *Proc. of the 15th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 357–368, 2010.

M. Might, D. Darais, and D. Spiewak.
Parsing with derivatives: A functional pearl.
In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 189–195. ACM, 2011.

S. Owens, J. Reppy, and A. Turon.
Regular-expression derivatives re-examined.
Journal of Functional Programming, 19(2):173–190, 2009.