

The Practical Efficiency of Regular Expression Membership Algorithms

Justin Gray
(Supervised by Dr. Konstantinidis)



Formal Languages

- Alphabet set Σ of symbols
 $\{0, 1\}$ or $\{a, b, c, d, \dots, z\}$ or ...
- Word string made from these symbols
0101011, jbc, ...
 ϵ for the empty word “”
- Language set of words
 $\{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$
- “Regular” languages can be described by regular expressions

Applications of Regular Expressions

Searching



The screenshot shows a Wikipedia search for "Stephen Cole Kleene". The search bar at the top contains "19[0-9][0-9]" and "1 of 78" results. The article title "Stephen Cole Kleene" is highlighted. Below the title, a brief biography is provided: "Stephen Cole Kleene (/ˈkleɪni/ KLAY-nee;[a] January 5, 1909 – January 25, 1994) was an American mathematician. One of the students of Alonzo Church, Kleene, along with Rózsa Péter, Alan Turing, Emil Post, and others, is best known as a founder of the branch of mathematical logic known as recursion theory, which subsequently helped to provide the foundations of theoretical computer science. Kleene's work grounds the study of computable functions. A number of mathematical concepts are named after him: Kleene hierarchy, Kleene algebra, the Kleene star (Kleene closure), Kleene's recursion theorem and the Kleene fixed-point theorem. He also invented regular expressions in 1951 to describe McCulloch-Pitts neural networks, and made significant contributions to the foundations of mathematical intuitionism." A table of contents is visible on the left, listing Biography, Legacy, Selected publications, See also, Notes, References, and External links. The right side of the page shows a portrait of Stephen Kleene and a table of his biographical details, including Born (January 5, 1909, Hartford, Connecticut, U.S.), Died (January 25, 1994, aged 85, Madison, Wisconsin, U.S.), Nationality (American), Alma mater (Amherst College, Princeton University), Known for (show), Awards (Leroy P. Steele Prize (1983), National Medal of Science (1990)), Scientific career (Fields: Mathematics, Institutions: University of Wisconsin–Madison, Doctoral advisor: Alonzo Church).

Validating

Enter your SMU email address:

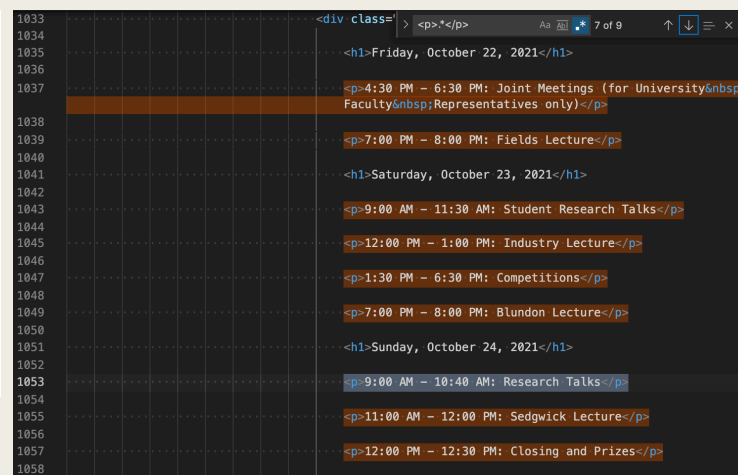
justin@notsmu.ca

Submit



Please match the requested format.

Extracting



The screenshot shows a shell output with line numbers 1033 to 1058. The output displays HTML content extracted from a document using a regular expression. The extracted content includes dates and times for various events, such as "Friday, October 22, 2021" and "Saturday, October 23, 2021", and specific times for lectures and meetings, such as "4:30 PM - 6:30 PM: Joint Meetings (for University Faculty Representatives only)" and "7:00 PM - 8:00 PM: Fields Lecture".

- DNA searching
 - Online quiz fill-in-the-blank correctness
 - Filtering shell output (grep)
- AND... Anywhere things are encoded as text.

Mathematical Regular Expressions

- Concatenation

$$L(ab) = \{ab\}$$

“a” followed by a “b”

- Disjunction

$$L(a + b) = \{a, b\}$$

“a” or “b”

- Star

$$L(a^*) = \{\epsilon, a, aa, aaa, \dots\}$$

0 or more “a”s

Some Compatible UNIX Extensions

- Wild dot matches any single symbol

$$L(a.a) = \{a\sigma a : \text{for any } \sigma \in \Sigma\}$$

$$\{aaa, a a, aba, a0a\} \subset L(a.a)$$

- Character Classes

$$L([abc0-9]) = \{a, b, c, 0, 1, 2, 3, \dots, 9\}$$

$$L([^\wedge a]) = \{\sigma \in \Sigma \setminus \{a\}\}$$

- Optional

$$L(a?) = \{\epsilon, a\}$$

Regular Expression Trees

Consider the expression

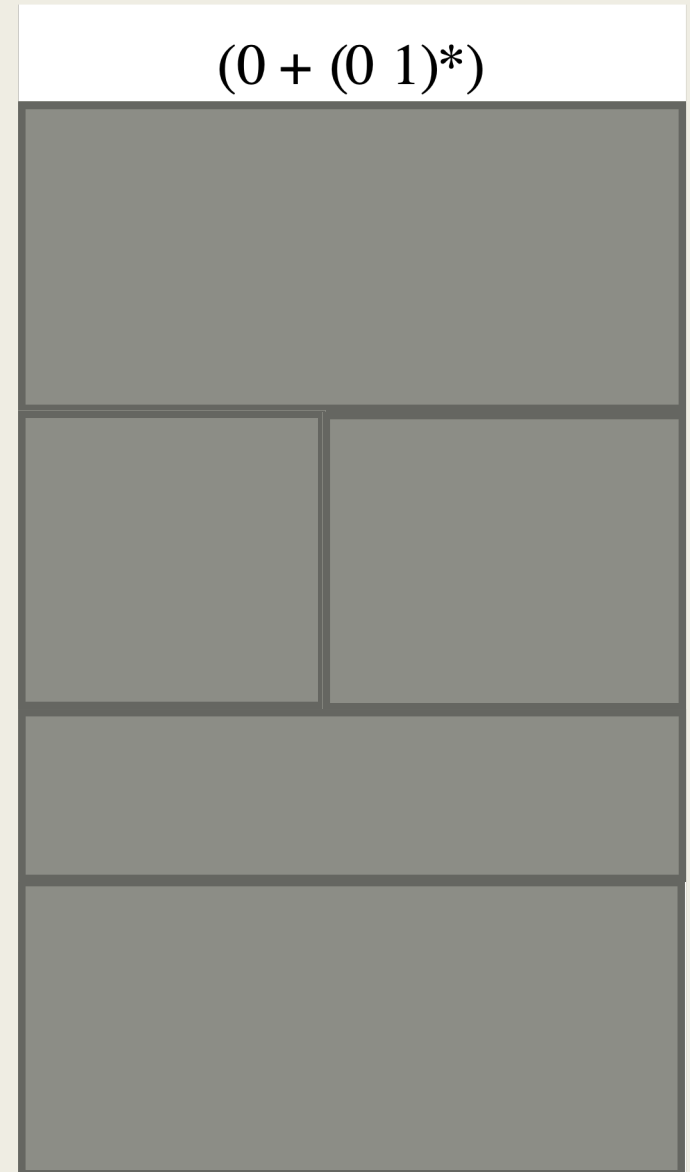
$(0 + (0\ 1)^*)$

- Disjunction:
- “0” is a symbol
- Star:
- Concatenation:
- “0” is a symbol
- “1” is a symbol

$(0 + (0\ 1)^*)$

$(0\ 1)^*$

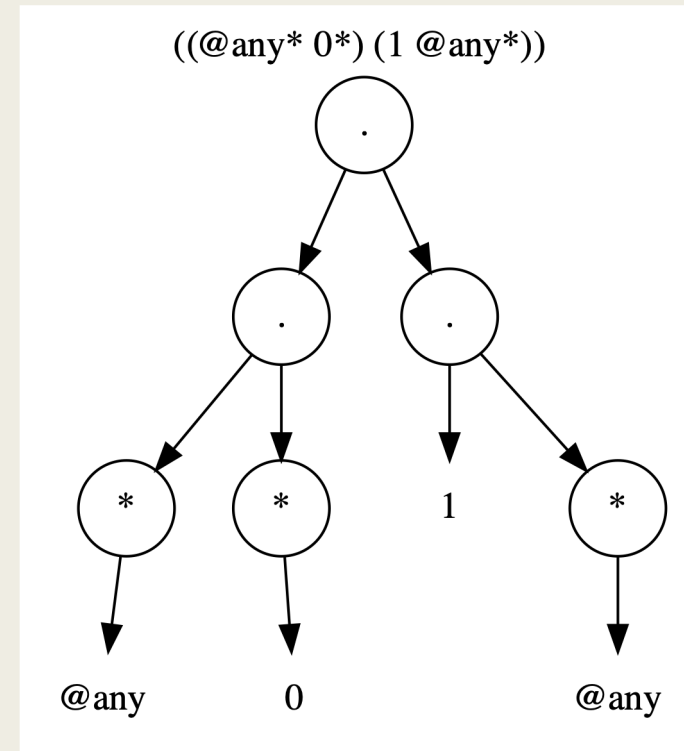
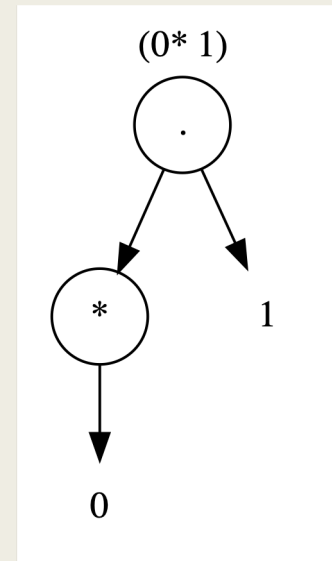
$(0\ 1)$



Partial Matching

- UNIX regular expressions generally match words if any substring of that word is accepted. (I.e., α will accept xwy if $w \in L(\alpha)$)
- Mathematical regular expressions require full matches

- We can make our regular expressions partially match by adding “.” appropriately
(this generally means at the beginning and end of leaves)

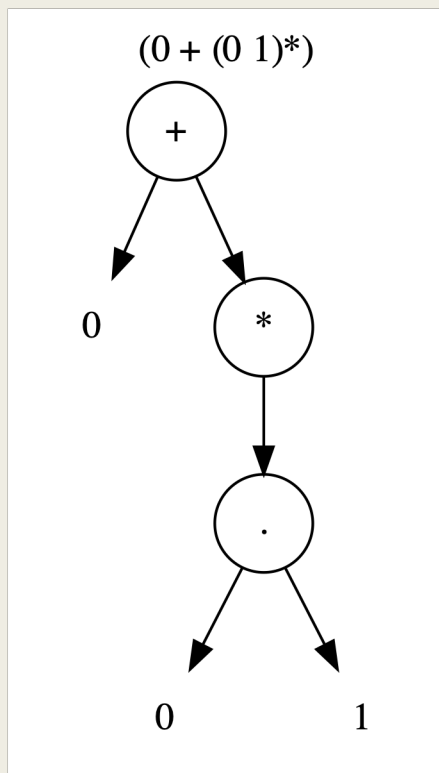


The Membership Problem

- Given a word w and a regular expression α , $w \in L(\alpha)$?
- 3 Approaches:
 - (Partial) Derivatives
 - Backtracking
 - Automata
- If you were designing a regular expression library today, which algorithm should you follow?

The Membership Problem: Partial Derivatives

- $\partial_\sigma(\sigma) = \{\epsilon\}$ for $\sigma \in \Sigma$
- $\partial_\sigma(b) = \partial_\sigma(\epsilon) = \{\}$ for $\sigma \neq b$
- $\partial_\sigma(\alpha + \beta) = \partial_\sigma(\alpha) \cup \partial_\sigma(\beta)$
- $\partial_\sigma(\alpha^*) = \partial_\sigma(\alpha)\alpha^*$
- $\partial_\sigma(\alpha\beta) = \begin{cases} \partial_\sigma(\alpha)\beta, & \text{if } \epsilon \notin L(\alpha) \\ \partial_\sigma(\alpha)\beta \cup \partial_\sigma(\beta), & \text{if } \epsilon \in L(\alpha) \end{cases}$



$0101 \in L(0+(01)^*)?$

Letters of 0101	Current PDs	Next PDs
0	$\{0+(01)^*\}$	$\partial_0(0+(01)^*) = \partial_0(0) \cup \partial_0((01)^*)$ $= \{\epsilon\} \cup \partial_0(01)(01)^*$ $= \{\epsilon\} \cup \{1\}(01)^*$ $= \{\epsilon, 1(01)^*\}$
1	$\{\epsilon, 1(01)^*\}$	$\partial_1(\epsilon) = \{\}$ $\partial_1(1(01)^*) = \{(01)^*\}$
0	$\{\} \cup \{(01)^*\}$ $= \{(01)^*\}$	$\partial_0((01)^*) = \partial_0(01)(01)^*$ $= \{1\}(01)^*$ $= \{1(01)^*\}$
1	$\{1(01)^*\}$	$\partial_1(1(01)^*) = \{(01)^*\}$
	$\{(01)^*\}$	

$0101 \in L(0+(01)^*) \Leftrightarrow \epsilon \in L((01)^*)$
 $\Leftrightarrow \text{Yes}$

The Membership Problem: Backtracking

- Implemented in almost every programming language
- Supports non-regular language operations (i.e., backreferences)
- Can work without constructing an automaton
- Usually very fast:

0101 $\in L(0+(01)^*)$

0101 X

0101

0101

0101

0101 ✓

- 0101 can be processed through $L(0+(01)^*)$ in “5 steps”

The Membership Problem: Backtracking

Let $\alpha = (a + a)^*$

aab $\in L(\alpha)$

aab
aab X
aab X
aab
aab X
aab X

Answer: No

aaab $\in L(\alpha)$

aaab
aaab
aaab X
aaab X
aaab
aaab X
aaab X
aaab
aaab
aaab X
aaab X
aaab
aaab X
aaab X
aaab
aaab X
aaab X

Answer: No

aaaab $\in L(\alpha)$

aaaab
aaaab
aaaab
aaaab X
aaaab X
aaaab
aaaab X
aaaab X
aaaab X
aaaab
aaaab
aaaab
aaaab X
aaaab X
aaaab
aaaab
aaaab
aaaab X
aaaab X
aaaab
aaaab
aaaab X
aaaab X
aaaab
aaaab X
aaaab X

Answer: No

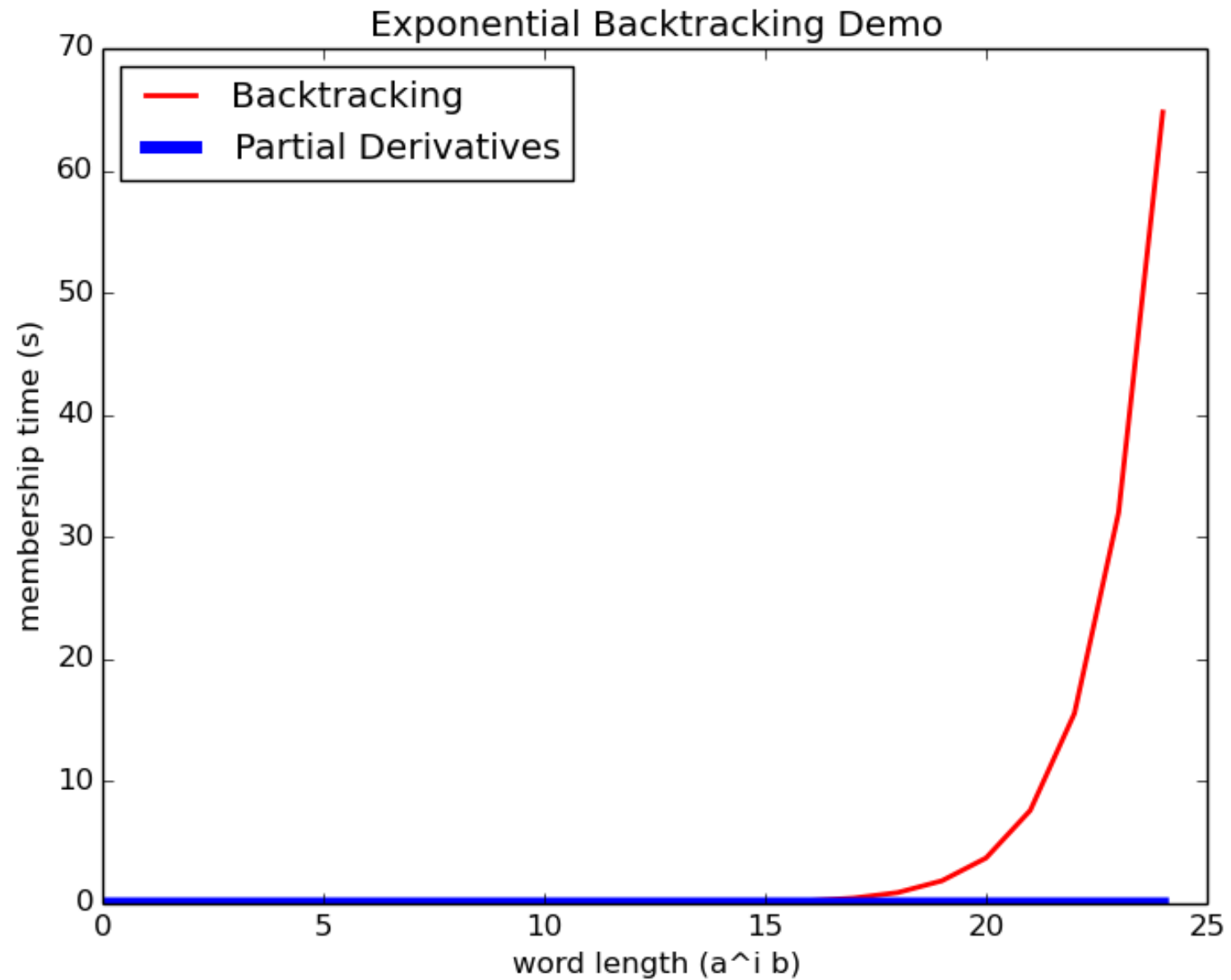


Figure 1: My implementation using the expression $(a + a)^*$

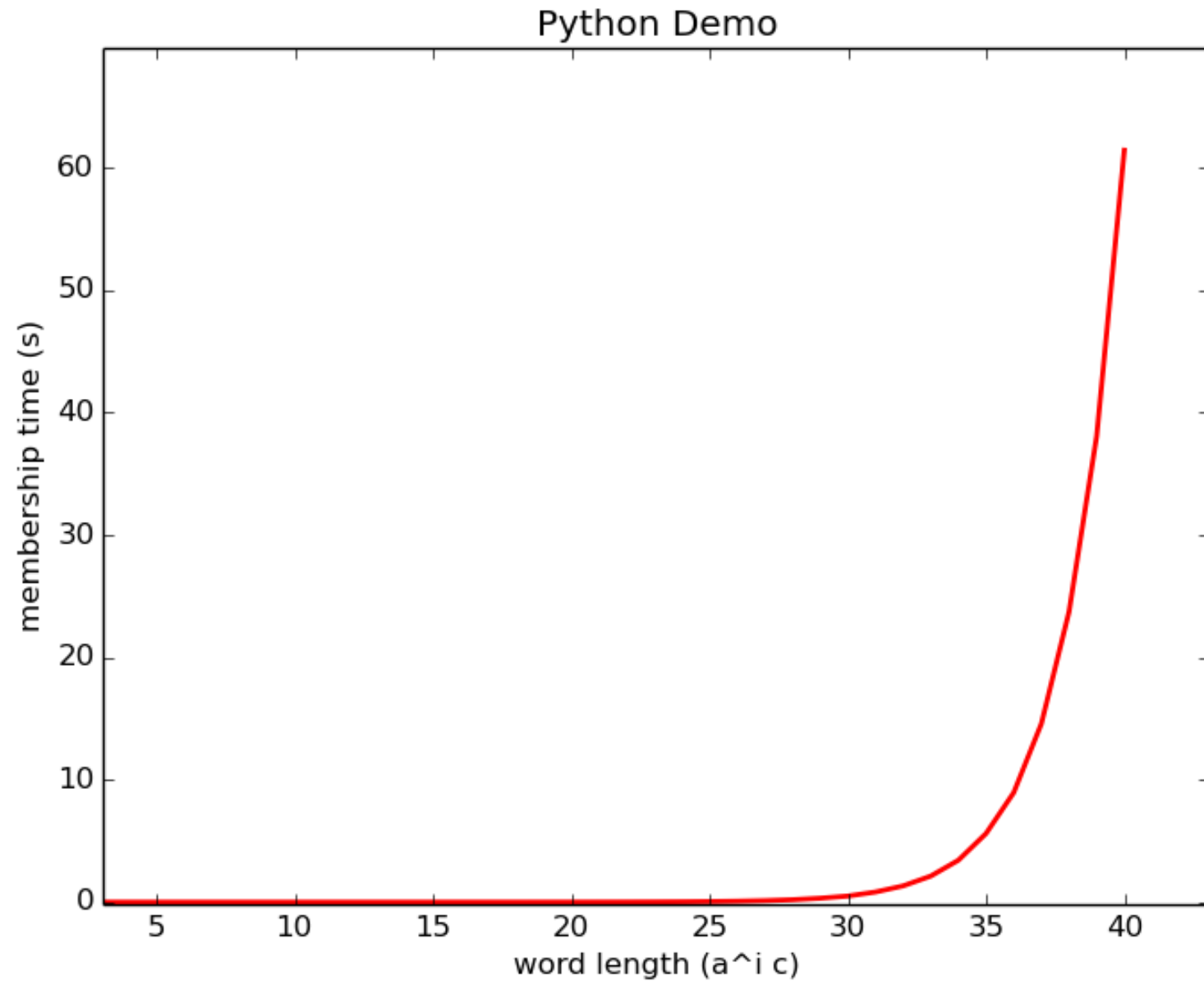
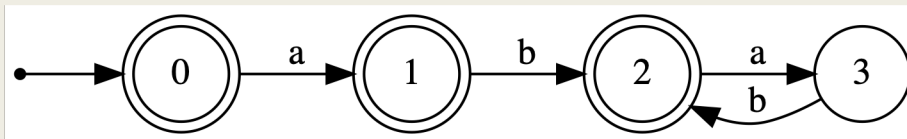


Figure 2: Python's re module and the expression $((a + aa)^* b)$ [1]

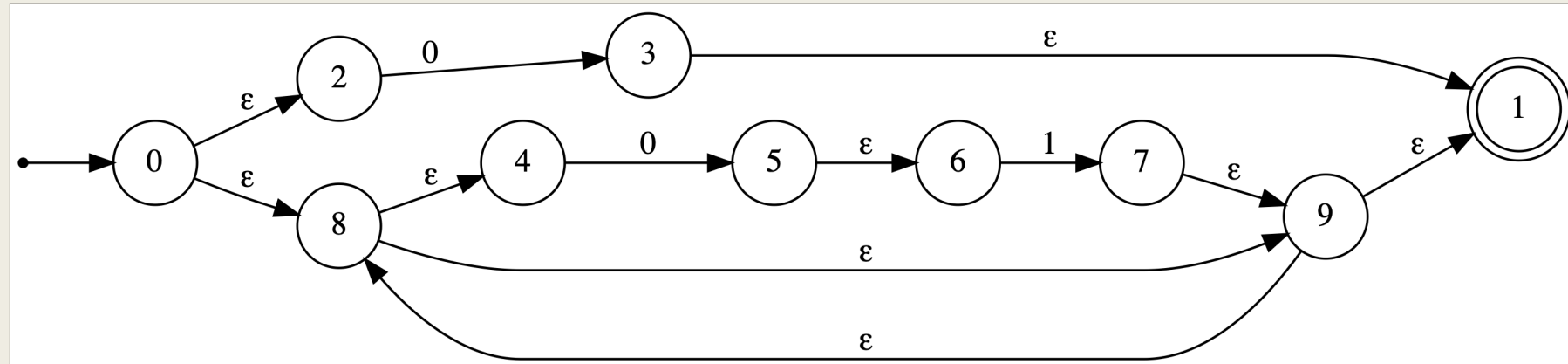
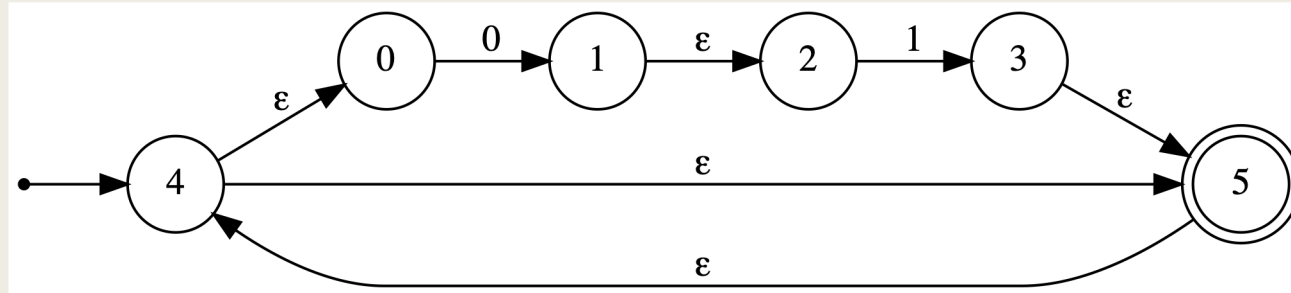
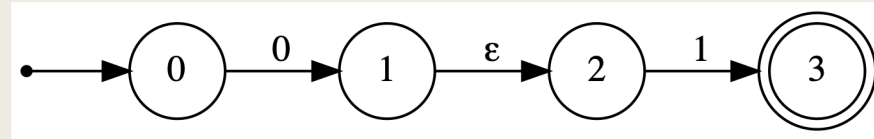
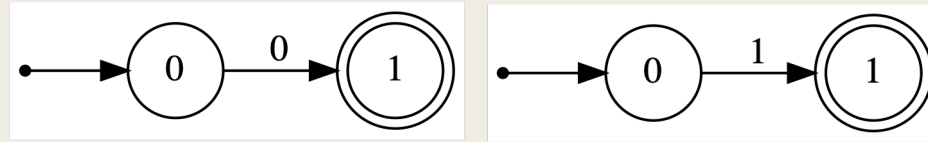
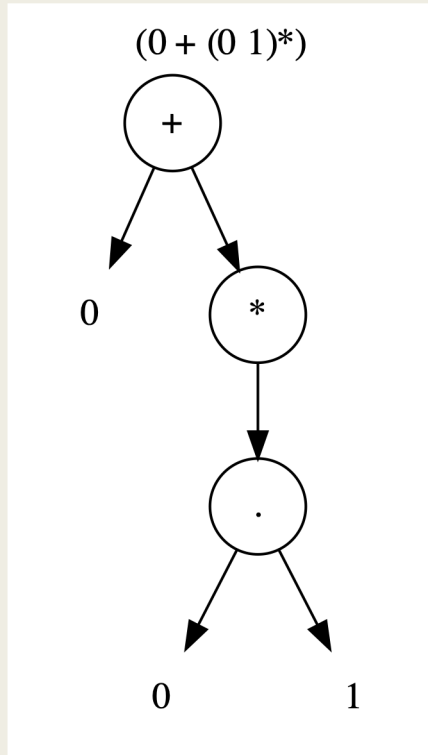
The Membership Problem: Automata

- Automata are state machines
- Input: word
- Output: YES if the entire word can be processed leading to any final state, NO otherwise
- Final states are double circled
- Symbols of the input word are consumed at each transition



$(\{0\}, abab)$
 $\vdash (\{1\}, bab)$
 $\vdash (\{2\}, ab)$
 $\vdash (\{3\}, b)$
 $\vdash (\{2\}, \epsilon)$
2 is final - accept

The Membership Problem: Automata



Collecting Practical Regular Expressions

<https://grep.app> indexes GitHub source code

Extracted 31,887 lines of code containing partial regular expressions from public GitHub repositories

12,023 of these lines have regular expressions satisfying:

Completeness

No errors

No backreferences,
lookaheads, boundary
assertions

Distinct when formatted as
equivalent unambiguous
mathematical expressions

Test Words

Accepting Words

- Pairwise-inspired generation [2]
- Enumeration
- Words are inserted into source code to simulate partial matching

Rejecting Words

- Distorting accepted words (adding, swapping, and removing characters)
- Comparing against a static list of word inputs

The average expression is tested against 1,980 accepting words and 4,468 rejecting words.

Only 726/12,023 regular expressions have been fully tested. Each regular expression is tested multiple times to ensure accuracy.

Results

- Regular expressions are grouped by string length
 - This is how they are most commonly defined
- The time measurement consists of:
 - Time to construct the appropriate object (parsing from a string into the tree, plus any automaton construction time)
 - Plus the time to evaluate the average test word

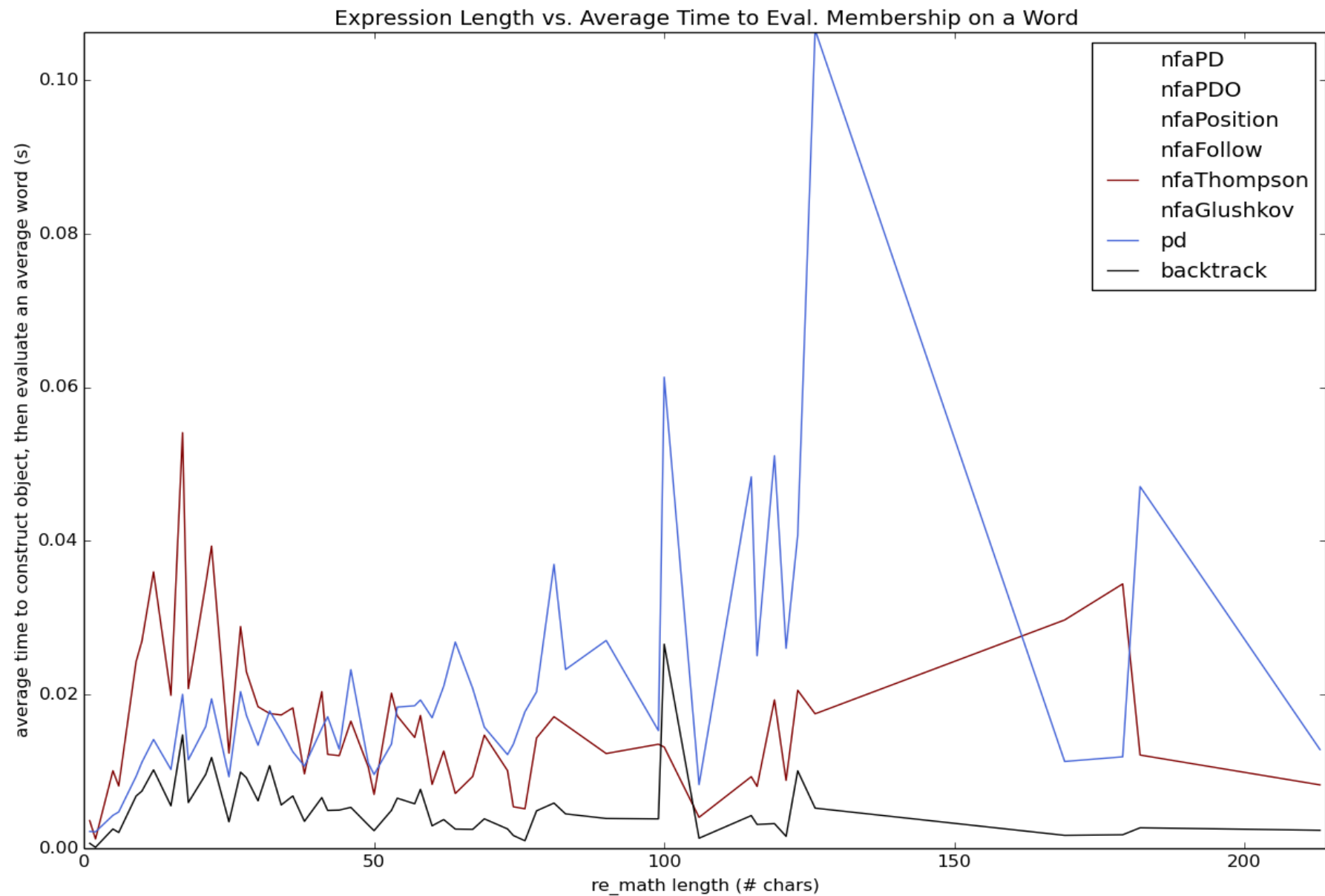


Figure 3: partial derivatives, vs. Thompson, vs. backtrack

$$|\text{pd}(\alpha)| \leq |\alpha|_{\Sigma}$$

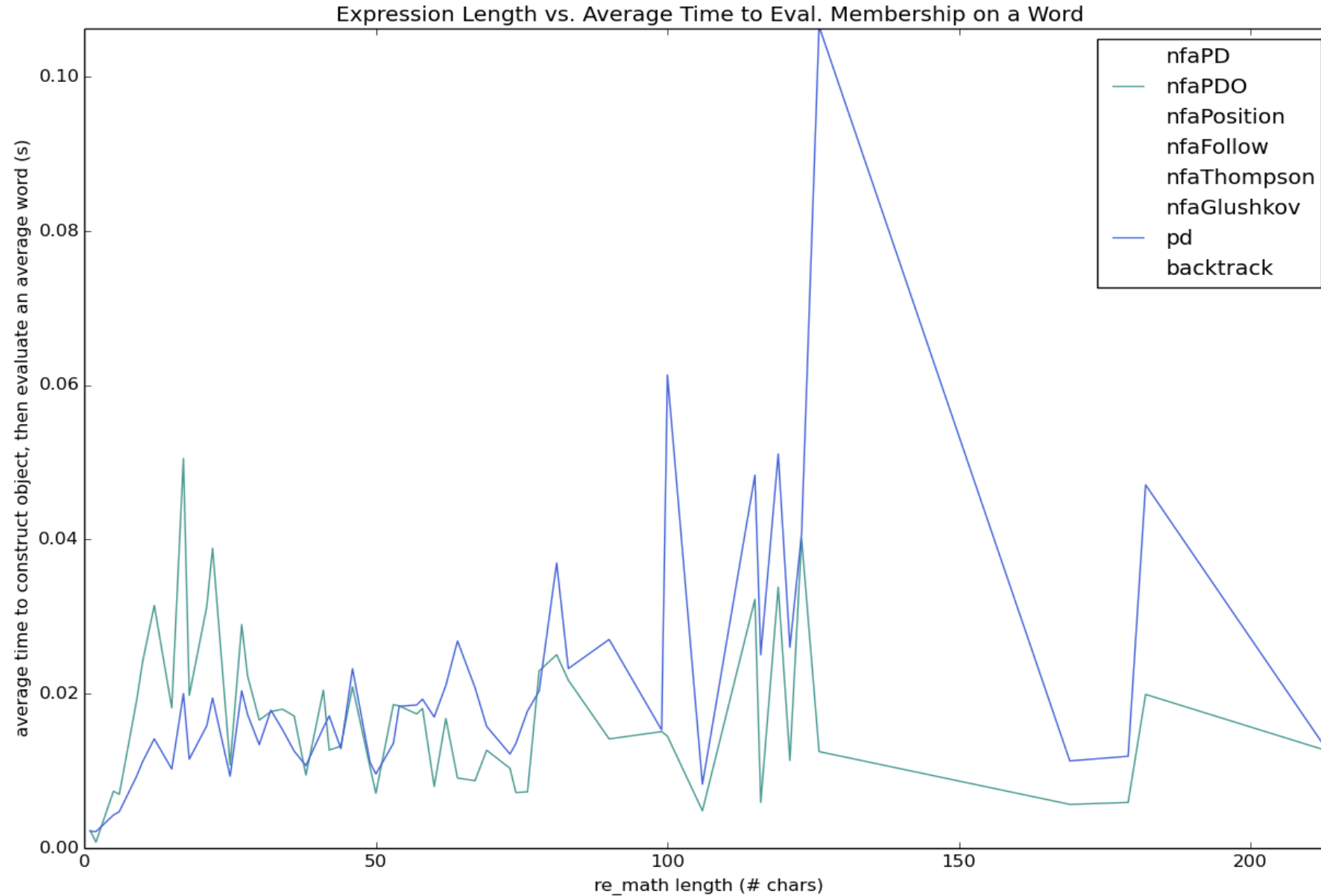


Figure 4: partial derivatives vs. partial derivative automaton

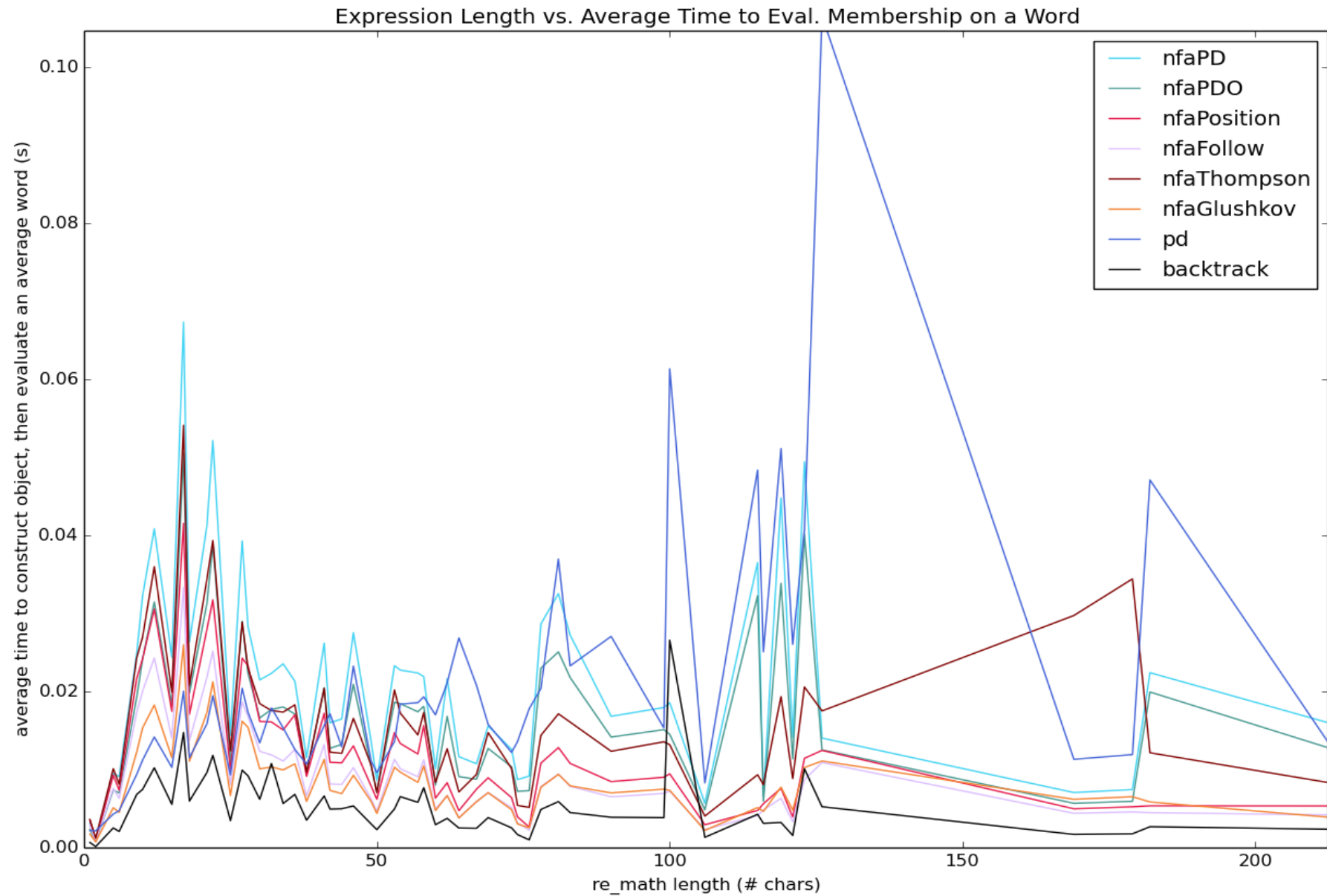


Figure 5: all algorithms

Further Work

- We hope to test these algorithms on randomly generated regular expressions (this has been done, but not with these UNIX extensions)
- Partial derivative optimizations when evaluating equality of large regular expression trees

Thank you!

1. To Dr. Konstantinidis for supervising this research and making his expertise available
2. For your attention
3. Questions?

The source code of this project is made publicly available at
<https://github.com/just1ngray/SMUHon-Practical-RE-Membership-Algs>

Reference List

1. R. Sedgewick, K. Wayne, *Algorithms*, 4th ed. Available: <https://algs4.cs.princeton.edu/54regex/>
2. L. Zheng, S. Ma, Y. Wang, and G. Lin, “String Generation for Testing Regular Expressions,” *The Computer Journal*, vol. 63, no. 1, Jan., pp. 41-65, 2020.