REDoS Detection in "Domino" Regular Expressions by Ambiguity Analysis

Nepeivoda A., Belikova Yu., Shevchenko K., Teriykha M.,
Knyazikhin D., Delman A., Terentyeva A.
PSI + BMSTU

SYRCoSE 2023

Regular Expressions

(...known to be easier to write than to read)

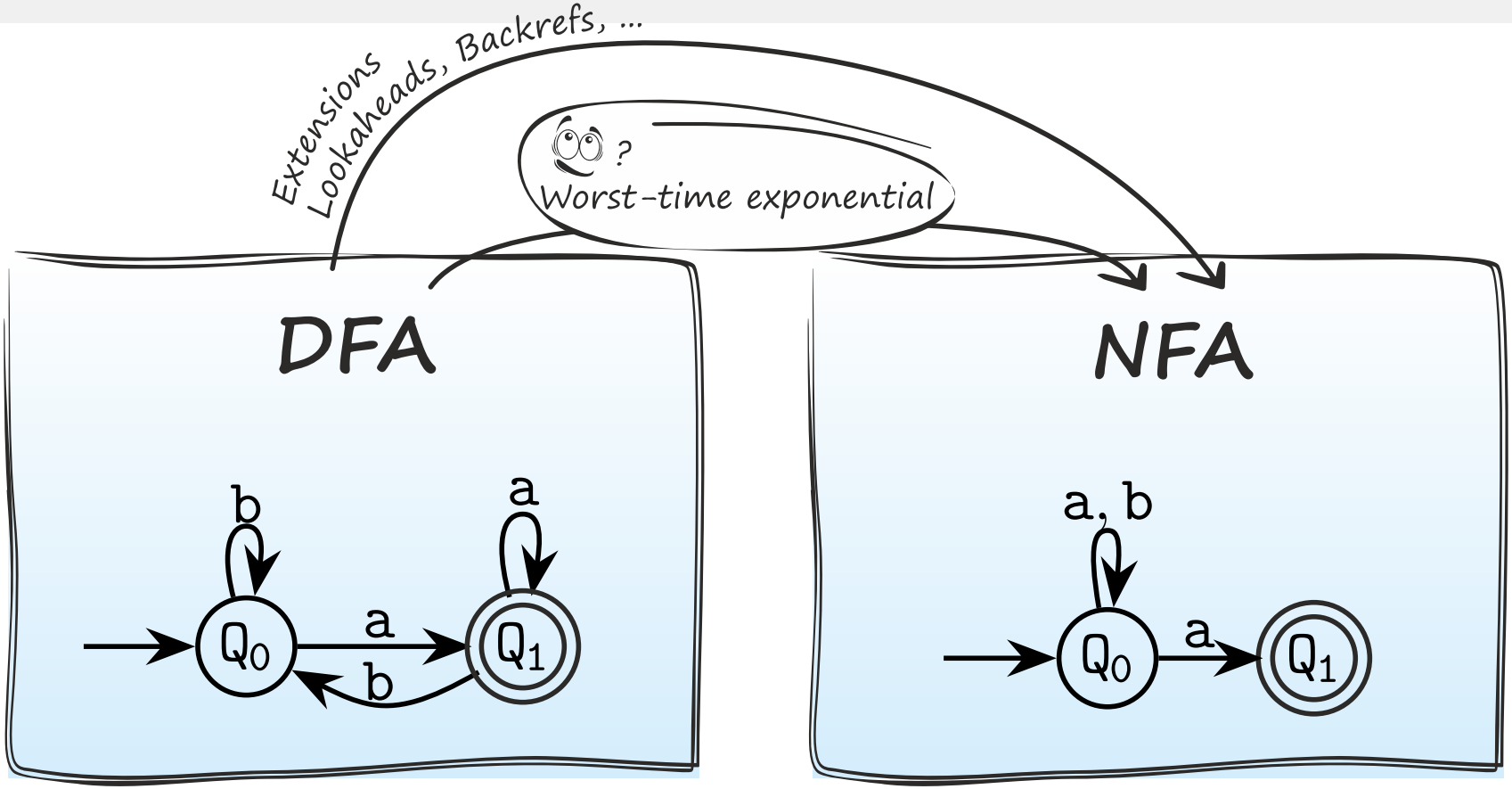
Academic Regular Expressions ($\mathcal{R}$) is a set of expressions over alphabet Σ closed *w.r.t.*:

- concatenation,
- union,
- iteration (Kleene star).

Syntactic sugar:

- positive iteration,
- option,
- intervals,
- negation

Regular Expressions

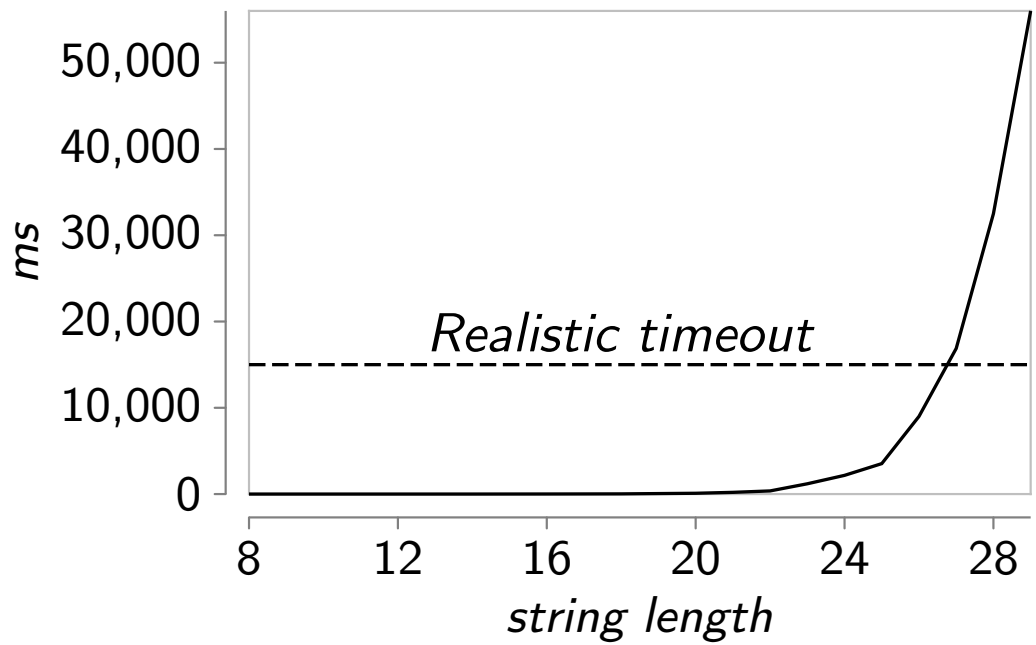


- Go
- Rust
- RE2

- Python
- JavaScript
- Java
- PCRE

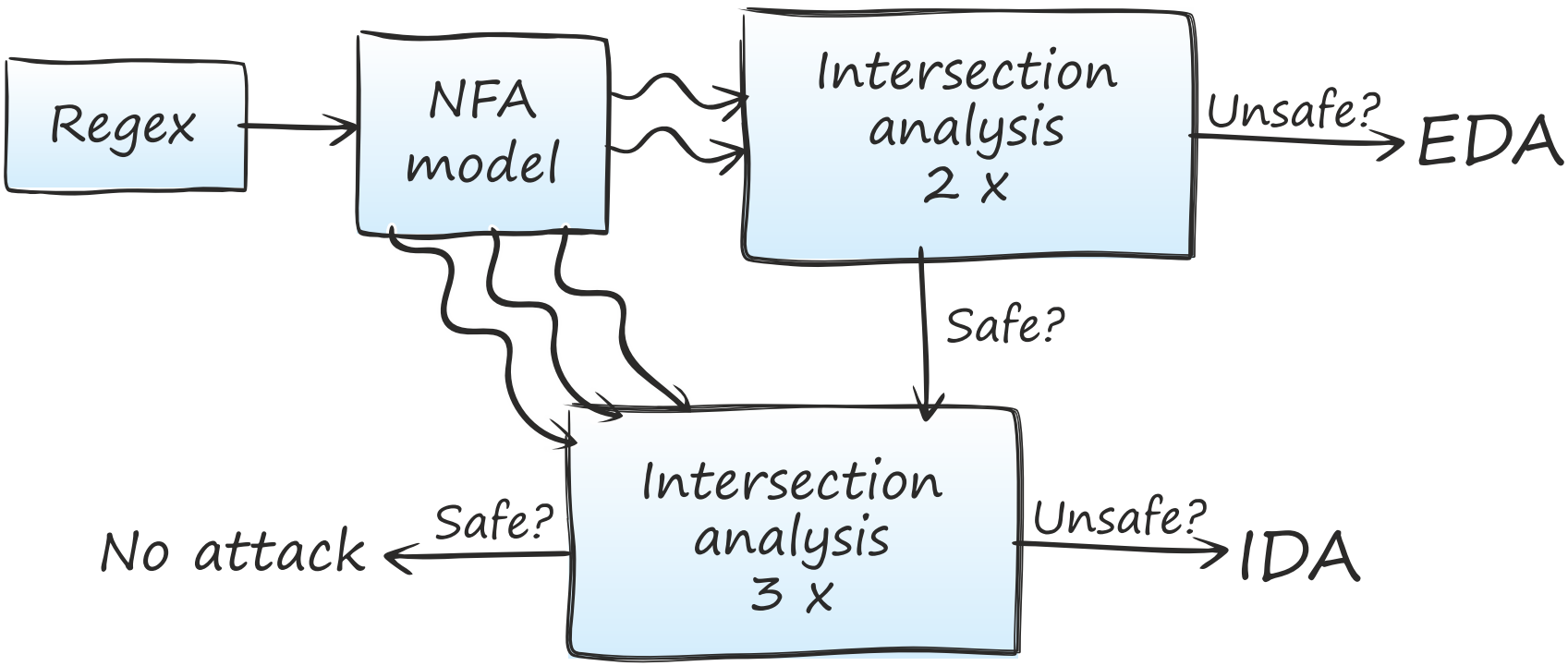
Catastrophic Backtracking

- Python regex engine
- Regular expression: `(a*b*)*`
- String: $\texttt{a}^{\texttt{n}}\texttt{c}$



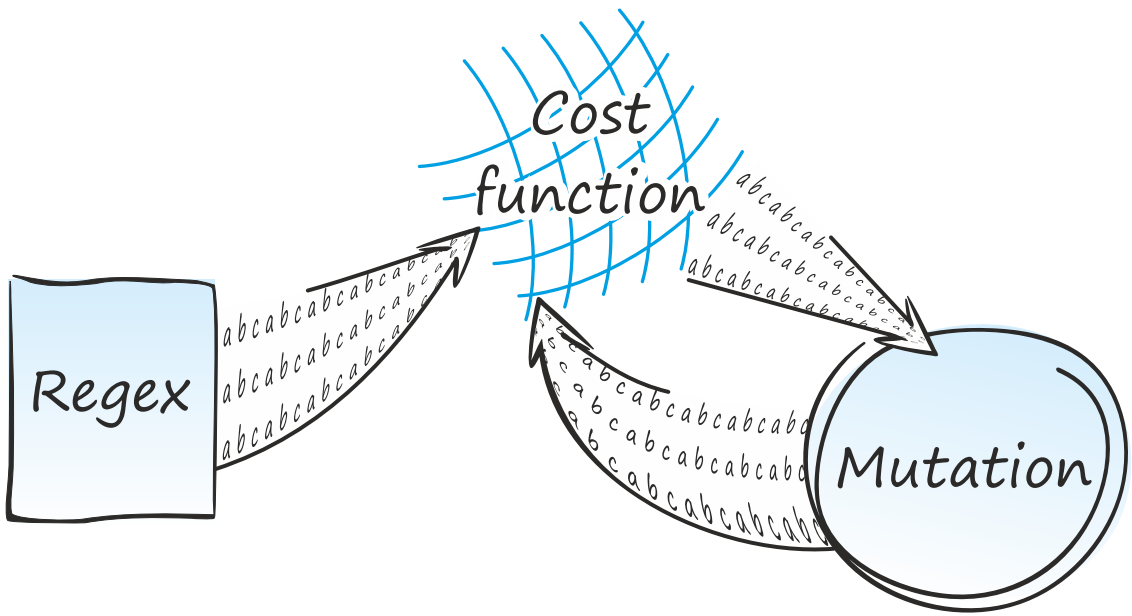
REDoS Detection

- Static analysis
- Fuzz
- Combined + pattern-based methods



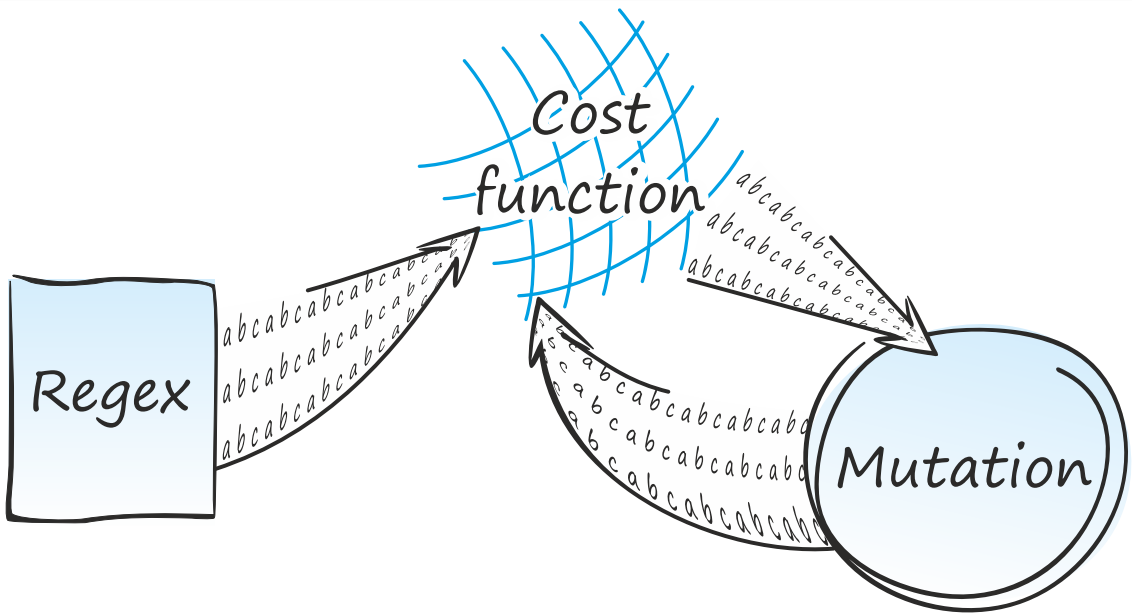
REDoS Detection

- Static analysis
- Fuzz
- Combined + pattern-based methods



REDoS Detection

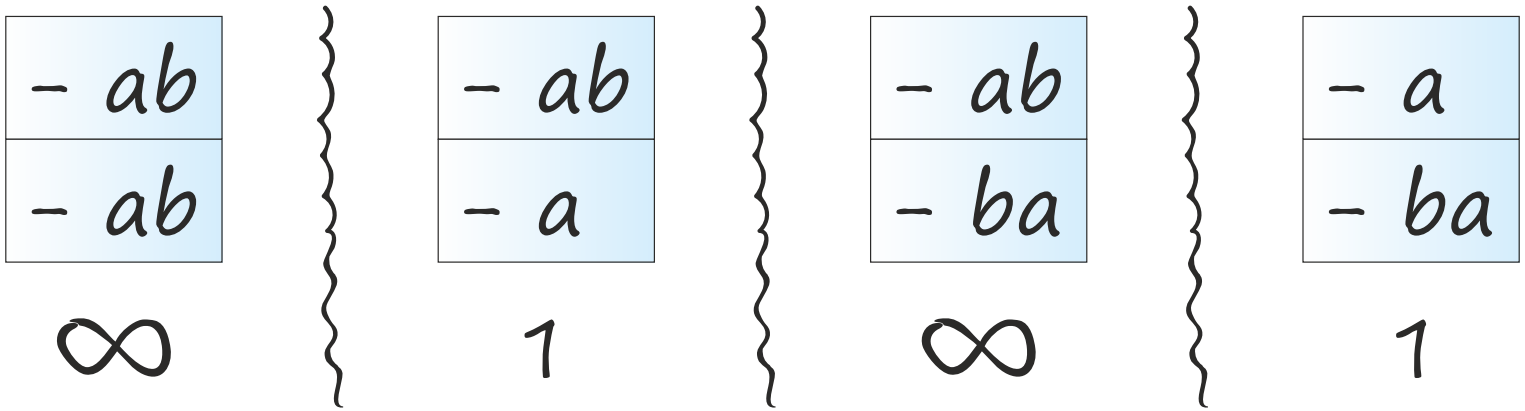
- Static analysis
- Fuzz
- Combined + pattern-based methods



- pattern-based ambiguity situations detection (subexpression $\mathtt{r}_1^*\mathtt{r}_2^*$, where $\mathscr{L}(\mathtt{r}_1) \cap \mathscr{L}(\mathtt{r}_2) \neq \varnothing$, is ambiguous);
- using mutation heuristics, based on the NFA structure.

Complex Ambiguities

(ab)*a(ba)*

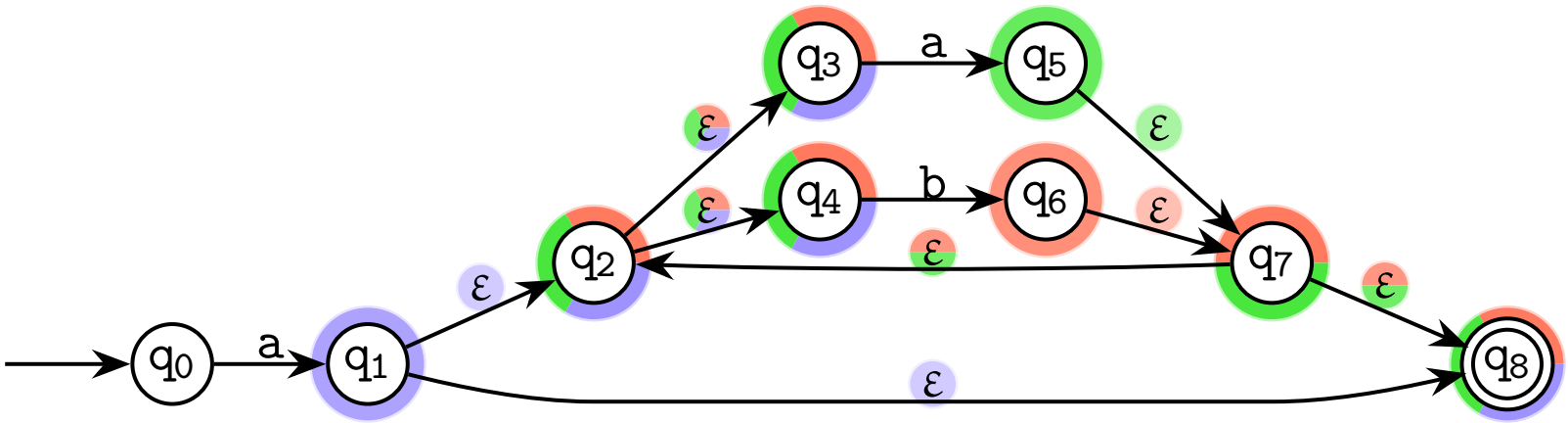


ababa

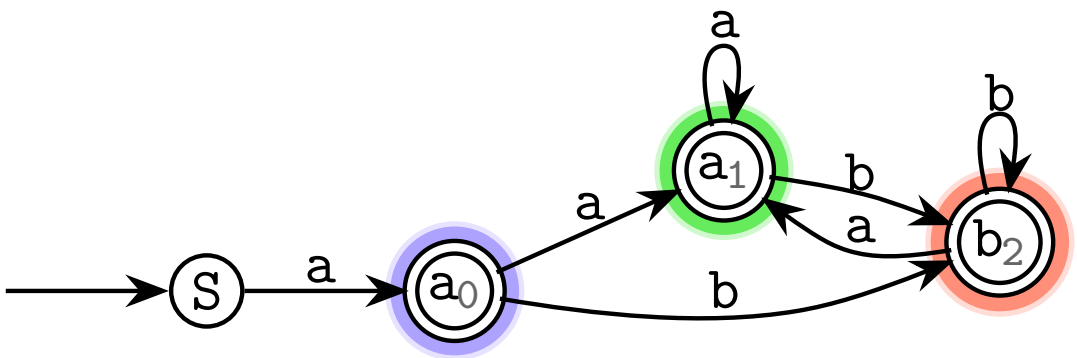
Research Questions

- Are the "domino" regexes really dangerous?
  - what sorts of ambiguities can they contain?
  - what is the complexity of their matching in real regex engines?
- If *yes*, how to deal with them?
  - use another sort of static analysis?
  - or use heuristic-based approaches?

NFA Models



(a) `Thompson(a(a | b)*)` with colored ε-closures

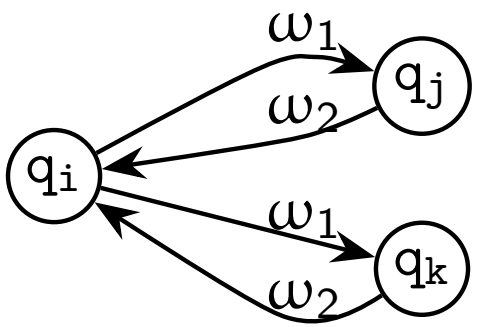


(b) `Glushkov(a(a | b)*)`

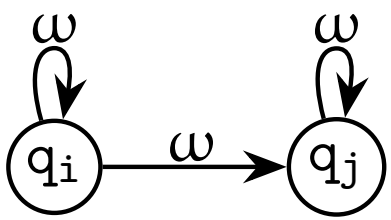
Degree of Ambiguity

Definition

A degree of ambiguity of an NFA $\mathscr{A}$ is a worst-case bound on the number of paths recognizing an input string (in a length of the string).



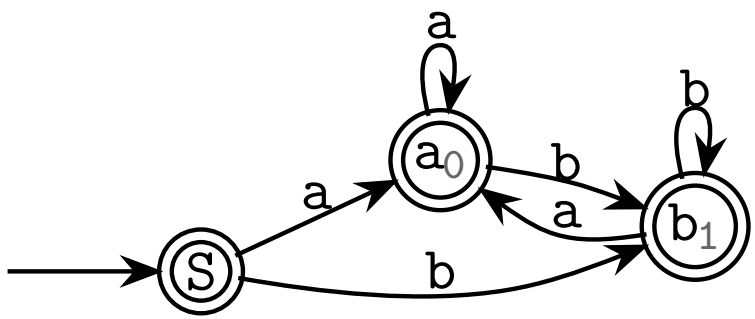
(a) Exponential ambiguity
(q_j or q_k may coincide with q_i)



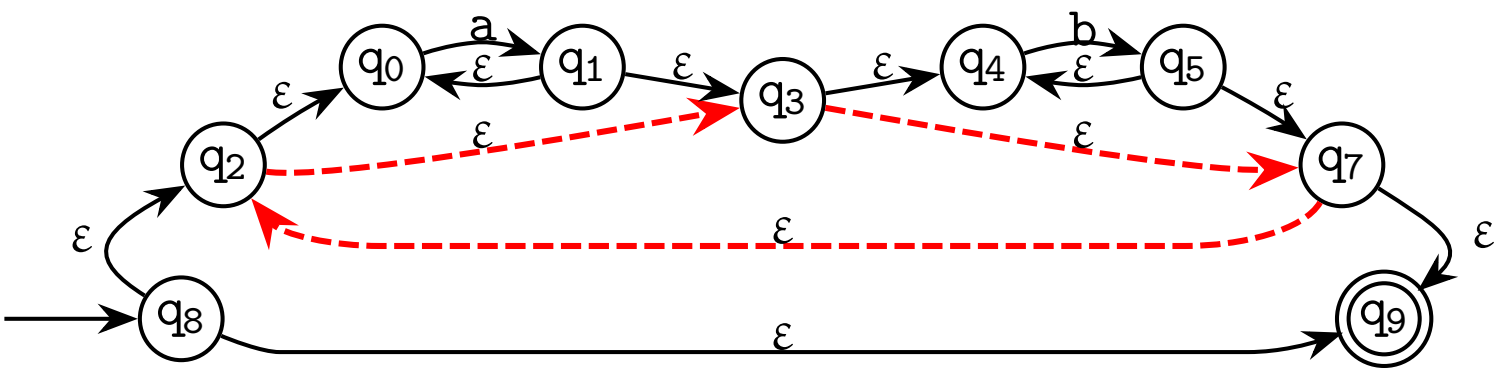
(b) Polynomial ambiguity

SSNF Property

Regex $(a^*b^*)^*$.
Glushkov construction is deterministic:



Thompson construction is highly ambiguous and leads to critical timeouts:



SSNF Property

Weideman in his PhD (2015) uses the similar example to prove that the Glushkov construction is inadequate for REDoS detection. But still, Glushkov automaton can be used to detect ambiguities, preceded with a simple linear-time test!

A regex $\mathtt{r}$ is in a (strong) star-normal form (SSNF) if it does not contain a subexpression $(\mathtt{r}_0)^*$ s.t. $\varepsilon \in \mathscr{L}(\mathtt{r}_0)$

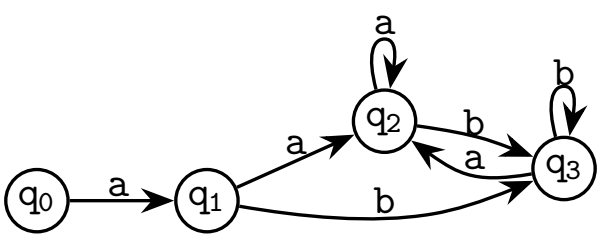
If $\mathtt{r}$ is in the SSNF, then
$\mathtt{Ambiguity}(\mathtt{Thompson}(\mathtt{r})) = \mathtt{Ambiguity}(\mathtt{Glushkov}(\mathtt{r}))$.

Transformation Monoid

- Usual approach to detect ambiguities in NFA: iterative self-intersection.
  - Considering the transition function as the state acting over the letter set: $F(q_i, \alpha) = F_{q_i}(\alpha) = \{q_{j_1}, \ldots, q_{j_k}\}$.
  - Costly, if the regex has a plenty of overlaps.
- Consider the transition function as the letter acting over the state set: $F(q_i, \alpha) = F_\alpha(q_i) = \{q_{j_1}, \ldots, q_{j_k}\}$, and then $F_\alpha(q_i) \circ F_\beta(q_i) = F_{q_i}(\beta\alpha) = F_{\beta\alpha}(q_i)$.
  - State set is finite $\Rightarrow$ set of the equivalence classes of the actions is finite.
  - The ambiguity criterion is then easily formulated, given the actions imposed by the members of the equivalence classes.

Transformation Monoid

Given an ε-free automaton $\mathscr{A}$ over the alphabet Σ, its transformation monoid $\mathscr{TM} = \texttt{Transmonoid}(\mathscr{A})$ is the monoid of transformations imposed by elements of Σ^* on the states of $\mathscr{A}$.



(a) Labelled transition system of $\mathscr{A}$

	q_0	q_1	q_2	q_3
a	$\{q_1\}$	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$
b	$\{\}$	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$
aa	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$
ab	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$
ba	$\{\}$	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

(b) Equivalence classes

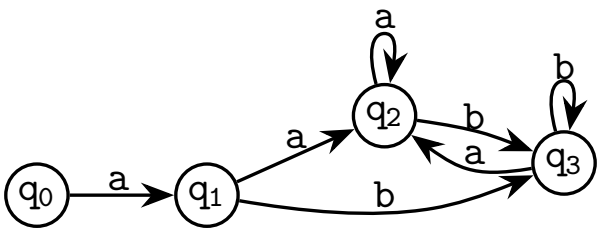
$\texttt{bb} \mapsto \texttt{b}$ $\quad$ $\texttt{aaa} \mapsto \texttt{aa}$

$\texttt{aab} \mapsto \texttt{ab}$ $\quad$ $\texttt{aba} \mapsto \texttt{aa}$

$\texttt{baa} \mapsto \texttt{ba}$ $\quad$ $\texttt{bab} \mapsto \texttt{bb}$

(c) Rewriting rules

Transformation Monoid



(a) Labelled transition system of $\mathscr{A}$

	q_0	q_1	q_2	q_3
a	$\{q_1\}$	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$
b	$\{\}$	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$
aa	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$
ab	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$
ba	$\{\}$	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

(b) Equivalence classes

$\mathtt{bb} \mapsto \mathtt{b}$ $\quad$ $\mathtt{aaa} \mapsto \mathtt{aa}$

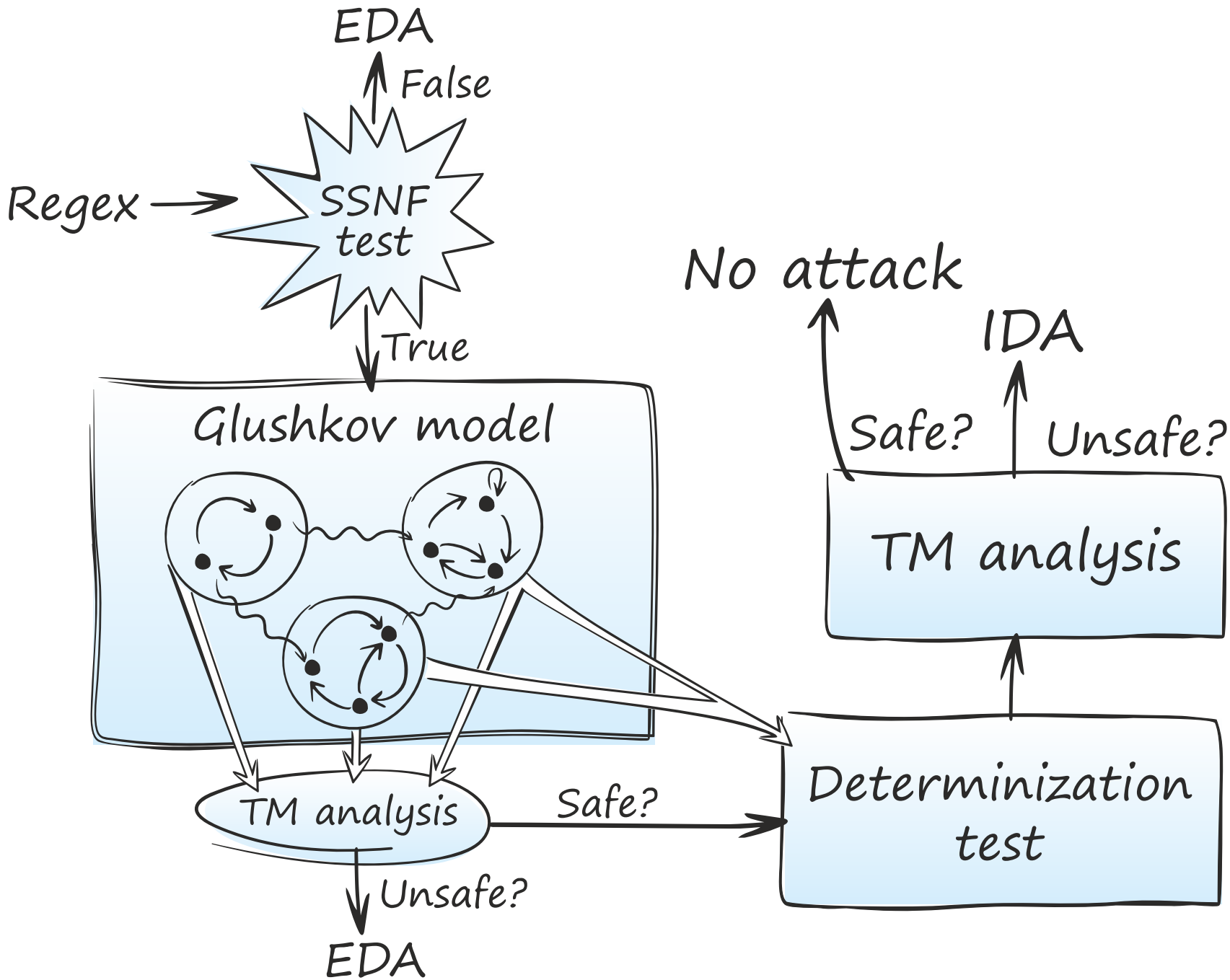
$\mathtt{aab} \mapsto \mathtt{ab}$ $\quad$ $\mathtt{aba} \mapsto \mathtt{aa}$

$\mathtt{baa} \mapsto \mathtt{ba}$ $\quad$ $\mathtt{bab} \mapsto \mathtt{bb}$

(c) Rewriting rules

An ε-free $\mathscr{A}$ satisfies IDA $\Leftrightarrow$ its transformation monoid contains an equivalence class ω s.t. for some states q_i, q_j, $q_i \in F_\omega(q_i)$, $q_j \in F_\omega(q_j)$, and $q_j \in F_\omega(q_i)$.

- Costly, if the regex has few overlaps;
- Less costly, if applied to the strongly connected components (orbits) of the NFA;
- Since IDA (not EDA) can occur in the pair of the orbits, a preliminary test helps to accelerate the monoid criterion.


EDA
False
Regex
SSNF test
True
Glushkov model
No attack
IDA
Safe?
Unsafe?
TM analysis
TM analysis
Safe?
Determinization test
Unsafe?
EDA

Other Tools

(open-source tools, described in research papers, and with adequate specification)

- RegexStaticAnalyzer, intersection-based analysis of academic regexes;
- ReScue, fuzzing-based analysis of extended regexes;
- Revealer, combined analysis of extended regexes.

(did not succeed to find a working implementation)

- Regulator, fuzzing based on regex structure;
- RedosHunter, pattern-based approach, the patterns given in the paper do not detect dominoes;
- RedosScalpel, the same problem with the patterns.

Test Set

- length < 50 terms, alphabet < 6 letters (to construct non-trivial dominoes);
- alternation under an iteration, or several iterations (to have possible ambiguities).

Ambiguity Classes

- Purely polynomial — critical on long input strings
- Non-SSNF — critical on short strings
- Polynomial under iteration — another critical case, non-trivial to detect

Results

	Exponential		Polynomial		Safe		Unsafe		Timeouts
Tool	μ (s)	σ (s)	μ (s)	σ (s)	μ (s)	σ (s)	μ (s)	σ (s)	
RSA	1.895	2.614	3.480	3.748	0.836	0.341	2.578	3.221	13
ReScue	–	–	–	–	0.940	1.724	8.803	6.263	43
Revealer	0.410	0.035	0.402	0.021	0.320	0.065	0.409	0.033	0
Our method	0.846	1.059	1.178	1.259	0.484	0.400	1.014	1.169	0

Tool	F_1-score	Total error rate	Vuln. error rate
RSA	0.90	0.13	0.00
ReScue	0.39	–	–
Revealer	0.55	0.45	0.04
Our method	1.00	0.00	0.00

Conclusion

Despite the success on the domino test set, extension to the whole class of regexes requires further developments:

- to apply alphabet factorization,
- to make use of the given approach for more precise analysis of extended regexes with complex overlaps,
- to combine the monoid-based and intersection-based ambiguity detection algorithms, based on heuristics detecting possible overlaps.