Program Semantics, Specification, and Verification

Innopolis, November 3th–4th, 2023

Disambiguation of Regular Expressions with Backreferences via Term Rewriting

Daria Ismagilova
armi.din1902@gmail.com

Antonina Nepeivoda
a_nevod@mail.ru

Terminological Clash

Academic regexes

- $|, \cdot, ^*$ (sometimes $^+$, ?) operations;
- define regular languages;
- studied in university courses (compilers & formal languages)

REGEX (extended regexes)

- lookaheads, backreferences, etc;
- define non-context-free languages;
- used in practice (PCRE2 standart).

- Almost identical names are used for completely different (although related) notions.

Regex Processing Problem

Even for "academic" case...

Given a regex ρ, whether it is «bad» or «good» for matching?

- For Thompson NFA-based matching algorithm, $(([\mathtt{A}-\mathtt{Z}]^*) \mid ([\mathtt{1}-\mathtt{9}]^*))^*$ is bad.
- For classical regex⇒DFA conversion, $.^*\mathtt{a}(.\{\mathtt{n}\})$ is bad.
- For RE2 library, both $.^*\mathtt{a}(.\{\mathtt{n}\})$ and $(([\mathtt{A}-\mathtt{Z}]^*) \mid ([\mathtt{1}-\mathtt{9}]^*))^*$ are good.

Uses mixed NFA+DFA approach, based on regex structure and its algebraic properties

Constructing “the best” regex?

Regex

Worst-case exponential in DFA size

State elimination

Determinization

DFA

Worst-case exponential in regex size

- Regex heavily depends on state order in elimination procedure

(Conway & Krob)

The results of state elimination method are equivalent modulo:

- $x(yx)^* = (xy)^* x$ (sliding rule)
- $x^*(yx^*)^* = (x \mid y)^*$ (denesting rule)
- $\varepsilon \mid xx^* = x^*, \varepsilon \mid x^*x = x^*$ (star folding rule)
- $x(y \mid z) = xy \mid xz, (y \mid z)x = yx \mid zx$ (distributivity)

Other Rewriting Techniques

- Subexpression simplification wrt equivalence rules
  - Star-Normal Form (A. Bruggemann)
  - Harmful Patterns Elimination and heuristic-based rewriting for reducing matching complexity

- Utilizing full power of Kleene algebra for regex shortening

Kleene Algebra

A semiring $\langle \Sigma, (+, \varepsilon), (\cdot, \varnothing), ^* \rangle$, idempotent wrt $+$ and satisfying the following star axioms:

- $\varepsilon + x \cdot x^* = \varepsilon + x^* x = x^*$ (unfolding rule)
- $a \cdot x + b + x = x \Rightarrow a^* b + x = x$
- $x \cdot a + b + x = x \Rightarrow b \cdot a^* + x = x$

(Kozen's axioms)

What is a “regex” Nowadays?

Extensions

Regular

Lookaheads & Lookbehinds
Negative Lookaheads
(not in PCRE, but...)
Async. Composition

Non-Regular

Backreferences

Brute force solutions EXP-harder (but polynomial if a “natural” NFA model is deterministic)

Even wider language class with lookaheads or negations

Glushkov Automaton

1960s–1980s

Introduced by V.M. Glushkov in 1961.
Till 1990s, mainly of theoretical interest.

1990s

Formalisation of SGML unambiguity notion
in terms of Glushkov NFA (Wood&Bruggemann).

00s

Glushkov NFA is proved to generate (or be generated by)
well-known NFA models by equivalence or simulation relations.
Breaking fast implementation of Glushkov-NFA-based approach
in RE2 library (still $20\times$ faster than DFA-based Go regex library)

Glushkov Automaton

1990s

Formalisation of SGML unambiguity notion
in terms of Glushkov NFA (Wood&Bruggemann).

00s

Glushkov NFA is proved to generate (or be generated by)
well-known NFA models by equivalence or simulation relations.
Breaking fast implementation of Glushkov-NFA-based approach
in RE2 library (still $20\times$ faster than DFA-based Go regex library)

10-20s

"The Mother of All Automata" (Broda, 2017)
Development of Glushkov models for extended set of regex operations.

Glushkov Automaton

10-20s

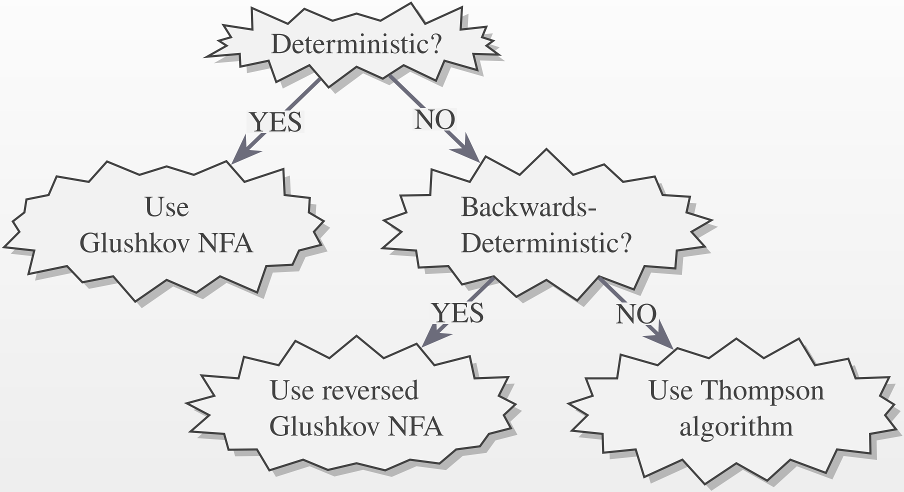
"The Mother of All Automata" (Broda, 2017)
Development of Glushkov models for extended set of regex operations.

Construction for Academic Regexes

- Linearize regex: $\tau = (.^{+})^{*}.$ becomes $\tau' = (._1{}^{+})^{*}._2$;
- Compute $\mathrm{First}(\tau')$, $\mathrm{Follow}(\tau')$, $\mathrm{Last}(\tau')$:
  - $s_i \in \mathrm{First}(\tau') \Leftrightarrow s_i$ can begin a string recognized by τ';
  - $s_i \in \mathrm{Last}(\tau') \Leftrightarrow s_i$ can end a string recognized by τ';
  - $\langle s_i, s_j \rangle \in \mathrm{Follow}(\tau') \Leftrightarrow s_i s_j$ can occur in $w \in \mathscr{L}(\tau')$.
- States of Glushkov NFA — letters of τ'. Transitions are determined by $\mathrm{First}(\tau')$ and $\mathrm{Follow}(\tau')$; final states — by $\mathrm{Last}(\tau')$.

A regex is deterministic $\Leftrightarrow s_i, t_j \in \mathrm{First}(\tau') \Rightarrow s \neq t$
and $\langle r_k, s_i \rangle, \langle r_k, t_j \rangle \in \mathrm{Follow}(\tau') \Rightarrow s \neq t$

RE2 Matching Algorithm



RE2 library deals successfully with regexes like $(.^{+})^{*}.^{+}$.
How to deal with $(.^{+})^{*}\backslash\mathtt{1}$? (more determined from the end)
Maybe reverse it to $(.^{+}?)\backslash\mathtt{1}.^{*}$ and parse backwards?

Without formalization,
we cannot search for equivalences.

Research Questions

- Is it possible to partly simulate RE2 implementing clever regex structure analysis and rewriting rules in presence of backreferences?
- At least, find deterministic and backwards-deterministic cases leading to “natural” linear-time matching?

Backreferences Formalisations

- Appeared much later than implementations of backref-regexes.
- Some almost repeated PCRE (Perl) backref-regexes.

Campeanu–Salomaa-Yu (CSY) formalisation:

$$\begin{cases} (\tau) & \text{(anonymous capturing)} \\ \backslash k & \text{(reading memory cell from k-th group)} \end{cases}$$

Example: $(\mathtt{a}^+)(\backslash\mathtt{1})^+$ defines $\{a^n \mid \text{n is not prime}\}$

- Any capture group is initialized exactly once.
- Any reference must be preceded by the capture group textually.

Gains

Define practical languages
No cyclic memory problem

Problems

Algebraic properties are ruined:
$x(yz)$ initializes \1 by yz,
while $(xy)z$ — by xy

Backreferences Formalisations

- Second try: regularly-bounded patterns.
- Regular restrictions (possibly w/some other variables) are imposed on variables.
- Example: $\begin{cases} \mathtt{(XaX)}^* \\ \mathtt{X} \in \mathtt{b}^* \end{cases}$ defines $\{b^{k_1}ab^{k_1}b^{k_2}ab^{k_2}\dots b^{k_m}ab^{k_m}\}$.

Gains

Define practical languages
No cyclic memory problem
Reversal is possible

Problems

Star unfolding requires introduction of fresh variables (unrestricted alphabet growth)

Backreferences Formalisations

- Last try: only named capture groups.

Backref-regex (ref-words, by Shmid) operations:

$$\begin{cases} [_k\tau]_k & \text{(named capturing)} \\ \&k & \text{(reading memory cell)} \end{cases}$$

Example: $[_1\mathtt{a}^*]_1\mathtt{a}^+\mathtt{b}\&1$ defines $\{a^m b a^n \mid m > n\}$

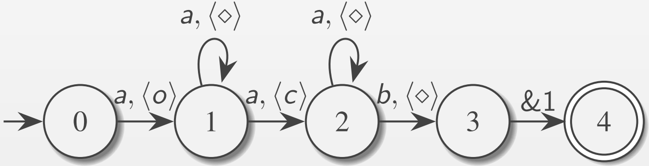
- ε-semantics (Shmid) — uninitialized reference recognizes $\{\varepsilon\}$;
- $\varnothing$-semantics (regex engines) — uninitialized reference recognizes $\varnothing$.

No impact on language properties.

Backreferences Formalisations

- Possibly unbalanced and nested (but not self-nested) capturing.
- References on k-th memory cell cannot occur inside a capturing group for k.
- Consequence: extended Glushkov automaton construction.

Memory Finite Automaton



Memory operations: $\begin{cases} o & \text{(open)} \\ \diamond & \text{(retain)} \\ c & \text{(close)} \end{cases}$

Bottleneck: Memory Loops

- Ref-words equivalence problem is harder, as compared to one for CSY regexes or reg-patterns (e.g., no pumping lemma is proven).

Issues with Cyclic Memories

- Loose extension of practical regexes, increasing expressibility of ref-words.

None of more than 3000 backref-regexes from StackOverflow contains recursive memoisation.

- Non-trivial to detect "by eye":
  - ref-word $([_1 \&2]_1 [_2 \mathtt{a}\&1]_2 [_1 \mathtt{a}^*]_1)^*$ is cyclic,
  - ref-word $([_1 \&2]_1 [_1 \mathtt{a}^*]_1 [_2 \mathtt{a}\&1]_2)^*$ is not.

Semantics-Preserving Renaming

Initial regex	$[_1 \mathtt{a}^*]_1 \mapsto [_3 \mathtt{a}^*]_3$	$[_1 \&2]_1 \mapsto [_3 \&2]_3$	Cycles after subst
$([_1 \&2]_1 [_2 \mathtt{a}\&1]_2 [_1 \mathtt{a}^*]_1)^*$	✓	✗	✓
$([_1 \&2]_1 [_1 \mathtt{a}^*]_1 [_2 \mathtt{a}\&1]_2)^*$	✗	✓	✗

- Memory-safe regex can be consistently renamed to an acyclic regex.
- Memory-cyclic regex can only be consistently renamed to a memory-cyclic regex.

Dependent Memory Chains

- A dependent memory chain is $[_{i_1} \& i_2]_{i_1} [_{i_2} \& i_3]_{i_2} \dots [_{i_{n-1}} \& i_n]_{i_{n-1}}$.
- $\omega \in \Sigma_{\mathcal{M}}^*$ contains a dependent memory chain $C \Leftrightarrow C$ occurs in ω as a scattered subword, and no memory brackets $[_{i_j}]_{i_j}$ are used outside of the chain in ω preceding usage of $\& i_j$ in the chain after initialization of i_j.

$$[_1 \& 2]_1 \overbrace{\;[_2\;}^{\text{1-st cell dependency}} \& 1]_2 \overbrace{[_1 \mathtt{a}^*]_1 [_1}^{\text{2-nd cell dependency}} \& 2]_1$$

$[_1 \& 2]_1 [_2 \& 1]_2 [_1 \& 2]_1$ is a memory chain

$$[_1 \& 2]_1 \overbrace{\color{red}{[_1 \mathtt{a}^*]_1 [_2}}^{\text{1-st cell dependency}} \& 1]_2 \overbrace{\;[_1\;}^{\text{2-nd cell dependency}} \& 2]_1$$

$[_1 \& 2]_1 [_2 \& 1]_2 [_1 \& 2]_1$ is not a memory chain

Dependent Memory Chains

$$[_1 \&2]_1 \overbrace{[_2}^{\text{1-st cell dependency}} \&1]_2 \overbrace{[_1 \mathtt{a}^*]_1 [_1}^{\text{2-nd cell dependency}} \&2]_1 \qquad [_1 \&2]_1 \overbrace{\color{red}{[_1 \mathtt{a}^*]_1 [_2}}^{\text{1-st cell dependency}} \&1]_2 \overbrace{[_1}^{\text{2-nd cell dependency}} \&2]_1$$

$[_1 \&2]_1 [_2 \&1]_2 [_1 \&2]_1$ is a memory chain $\quad$ $[_1 \&2]_1 [_2 \&1]_2 [_1 \&2]_1$ is not a memory chain

Checking dependencies

Dependency finite automaton (DepFA) for a ref-word ρ is an NFA for the linearized academic regex $h(\rho)$, where $h(\gamma) = \varepsilon$ for all γ in input alphabet Σ.

Observing dependent memory chains in $\mathscr{L}(\text{DepFA}) \Rightarrow$ more semantically-based definition of acyclic memory use.

ACREG formalization

Acyclic regexes

A ref-word ρ is acyclic if language of its DepFA does not contain a dependent memory chain starting and ending with the same indexed subexpression $[_{i,t_1} \& j_{t_2}]_{i,t_3}$.

- Reinitialization is finitely bounded
- Capture groups — memory operators
- Closed under semantics-preserving renaming

ACREG is an idempotent semiring satisfying sliding, denesting and star folding rules

Last initializations

- For binding, only the last memory initialization on the path makes sense.
- Less possible bindings $\Rightarrow$ more memory determinism (easier to analyse).

> If $\tau \in \mathit{ACREG}$, then every reference in τ can have only a finitely presented set of last-initializations.

- Still, the representation is context-sensitive.

Example

Let $\tau = [_1\mathtt{ba}^*]_1[_2\mathtt{ca}^*]_2([_1\&2\mathtt{ab}\&2]_1 \mid [_2\mathtt{bb}^*]_2)^*$.

- $\text{last}_{2:\text{init}}r = \{\mathtt{ca}^*, \mathtt{bb}^*\}$.
- $\text{last}_{1:\text{init}}r = \{\mathtt{ba}^*, [_2\mathtt{ca}^*]_2\mathtt{ab}\&2, [_2\mathtt{bb}^*]_2\mathtt{ab}\&2\}$.

Backref-Normal Form and Conway–Krob transformations

- $x^* \longrightarrow \varepsilon \mid xx^*$ (U)
- $(x \mid y)^* \longrightarrow x^*(yx^*)^*$ (N)
- $x(y \mid z) \longrightarrow xy \mid xz, (y \mid z)x \longrightarrow yx \mid zx$ (D)
- τ is deterministic wrt the last initializations $\Rightarrow \tau'$ resulting from UND is also deterministic.
- Cardinality of the set of last-initializations wrt k-th memory cell can be stepwise reduced by UND compositions.

semi-BNF

If for every subexpression ρ ending with $\&i$, $|\mathrm{last}_{\mathrm{i:init}}\rho| = 1$.

BNF

Last usage of a reference before its re-initialization or the end of the regex is explicit

Reversing Ref-Words

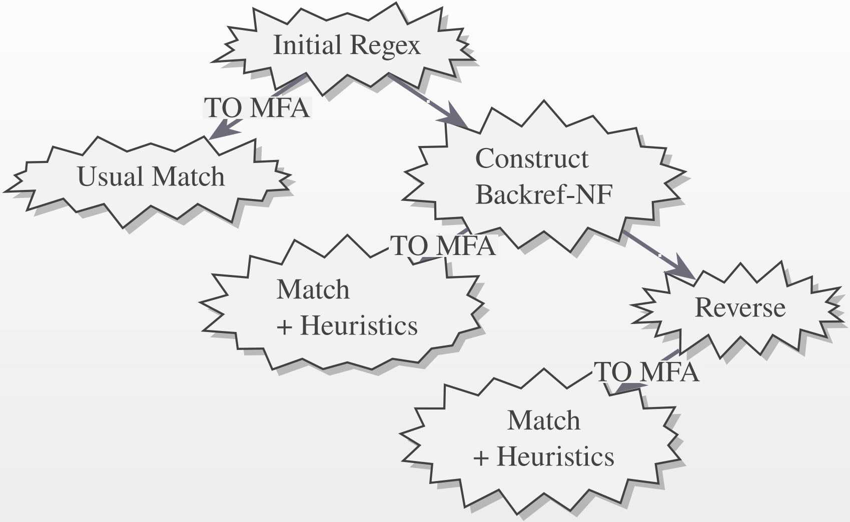
- Backreferences are 'binded' to the initializations, thus the initializations and references can be swapped.

$$(\texttt{[}_1\texttt{a}^*\texttt{]}_1(\texttt{bb} \mid \texttt{\&1})\texttt{bb\&1} \mid \texttt{[}_1\texttt{b}^*\texttt{]}_1\texttt{a\&1})$$
$$\Rightarrow$$
$$(\texttt{[}_1\texttt{b}^*\texttt{]}_1\texttt{a\&1} \mid \texttt{[}_1\texttt{a}^*\texttt{]}_1\texttt{bb}(\texttt{\&1} \mid \texttt{bb})\texttt{\&1})$$

> Generic ref-words are not closed under reversal.
> Counterexample: $\texttt{[}_1\texttt{a]}_1\texttt{b}(\texttt{[}_2\texttt{\&1\&1]}_2\texttt{b[}_1\texttt{\&2\&2]}_1)^*$

- Complication: special case of dependent memory chains under an iteration, when a reference precedes the bounded initialization ($(\texttt{\&2[}_2\texttt{\&1b}^*\texttt{]}_2\texttt{[}_1\texttt{a}^*\texttt{]}_1)^*$). Requires introduction of additional memory cells (polynomial in the cardinality of memory).

Experimental Algorithm



Backref-Regexes “in the wild”

>3000 regexes collected from StackOverflow.

- No recursive memory dependencies;
- 60 non-trivial memory dependencies, but none of them — in RW-blocks (i.e. introduce new memory cells in reversed regex);
- Most (>80%) memory structures are patterns.

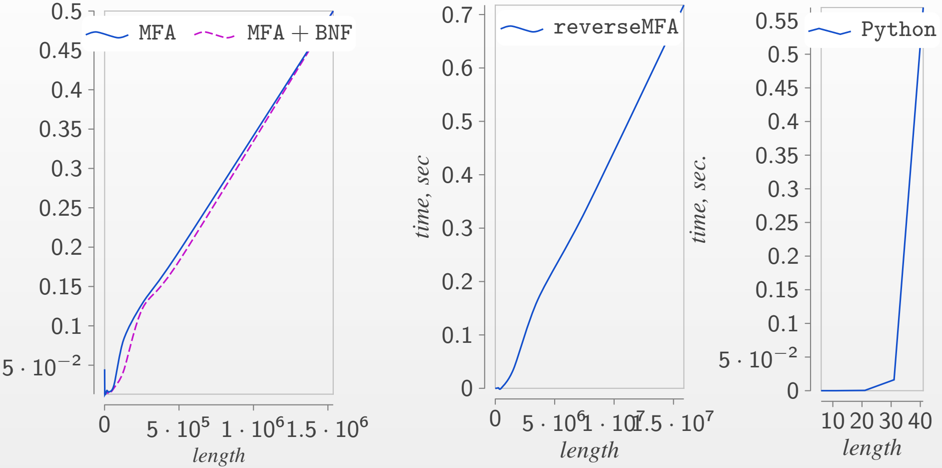
Similar observations: regexes from RegexLib (collection of “best practices”); but much smaller dataset.

Experimental Results

- Test dataset — 75 backref-regexes with various refs-iterations dependencies (most — imitating regexes from StackOverflow).
- Model input language (non-sugared academic regexes + memory operations).
- To estimate asymptotic growth: used series of words, following "pumping lemma" structure.

Comparable matching time	60% (45 cases)	50% — reversal+ 40% — init+ 10% — symmetrical
Reversed++	23% (17)	
Init++	17% (13)	

Experimental Results



Parse time of $([_1 \mathtt{a}^*]_1 \& 1 \mid (\mathtt{a}^* \mid \mathtt{b})\mathtt{a})^*$ on words $(\mathtt{baaa})^n\mathtt{b}$.

Future Work

- Optimised algorithm for dependent memory checking.
- Reversal preserving matching order (take in account greedy and lazy quantifiers).
- Implementing notion of deterministic ref-word and refining the experimental algorithm.

Discussion

Why the language is model?

- Real regex syntax is heavily sugared;
- To avoid redundant work: first — explore the properties, second — implement all the sugared constructions. We are still on the first step.

Certified implementation for a model?

- Interesting! But requires a lot of labour, doing all memory operations from scratch...
- Basic theory of backref-regexes is not implemented in libraries of theorem provers.