Linear Time Membership in a Class of Regular Expressions with Counting, Interleaving and Unordered Concatenation

Dario Colazzo

joint work with Giorgio Ghelli and Carlo Sartiani

appeared in ACM TODS

April 18, 2019 - JDL

Regular expressions

- Ubiquitous in database and programming languages
  - data extraction from text
  - path expressions in query languages over tree- and graph-shaped data

Regular expressions

- Ubiquitous in database and programming languages
  - data extraction from text
  - path expressions in query languages over tree- and graph-shaped data
- Standard REs:

$$T ::= \quad \epsilon \mid a \mid T + T \mid T \cdot T \mid T^*$$

- Example
  - $T = a \cdot (c + d)^* \cdot (f + \epsilon)$
  - $L(T) = [\![T]\!] = \{a,\ af,\ acf,\ acb,\ acbf,\ acbb,\ acbbf,\ \ldots\}$

Regular expressions

- Ubiquitous in database and programming languages
  - data extraction from text
  - path expressions in query languages over tree- and graph-shaped data
- Standard REs:

$$T ::= \quad \epsilon \mid a \mid T + T \mid T \cdot T \mid T^*$$

- Example
  - $T = a \cdot (c + d)^* \cdot (f + \epsilon)$
  - $L(T) = [\![T]\!] = \{a,\ af,\ acf,\ acb,\ acbf,\ acbb,\ acbbf,\ \ldots\}$
- Membership checking: does $w \in L(T)$ hold?

Regular expressions

- Ubiquitous in database and programming languages
  - data extraction from text
  - path expressions in query languages over tree- and graph-shaped data
- Standard REs:

$$T ::= \epsilon \mid a \mid T + T \mid T \cdot T \mid T^*$$

- Example
  - $T = a \cdot (c + d)^* \cdot (f + \epsilon)$
  - $L(T) = [\![T]\!] = \{a,\ af,\ acf,\ acb,\ acbf,\ acbb,\ acbbf,\ \ldots\}$
- Membership checking: does $w \in L(T)$ hold?
- Polynomial for standard REs [J.E. Hopcroft and J.D. Ullman1979]

Adding counting # and shuffle &

$$T ::= \quad \epsilon \;\mid\; a \;\mid\; T + T \;\mid\; T \cdot T \;\mid\; T\,[n..m] \;\mid\; T \& T \;\mid$$

- Attracted research interest in recent years, useful for XML/JSON schema languages

Adding counting # and shuffle &

$$T ::= \quad \epsilon \;\mid\; a \;\mid\; T + T \;\mid\; T \cdot T \;\mid\; T\,[n..m] \;\mid\; T \& T \;\mid$$

- Attracted research interest in recent years, useful for XML/JSON schema languages
- Counting
  - T= $a \cdot (b + c)\,[2..3]$
  - $[\![T]\!] = \{a, acc, abc, abb, abc, accc, abcc, abbc, abcc, accb, abcb, abbb, abcb\}$

Adding counting # and shuffle &

$$T ::= \quad \epsilon \;\mid\; a \;\mid\; T + T \;\mid\; T \cdot T \;\mid\; T\,[n..m] \;\mid\; T \& T \;\mid$$

- Attracted research interest in recent years, useful for XML/JSON schema languages
- Counting
  - T= $a \cdot (b + c)\,[2..3]$
  - $[\![T]\!] = \{a, acc, abc, abb, abc, accc, abcc, abbc, abcc, accb, abcb, abbb, abcb\}$
- Shuffle
  - $T = (a \cdot b) \& (X \cdot Y)$
  - $[\![T]\!] = \{abXY, aXbY, aXYb, XabY, XaYb, XYab\}$

Adding counting # and shuffle &

$$T ::= \quad \epsilon \;\mid\; a \;\mid\; T + T \;\mid\; T \cdot T \;\mid\; T\,[n..m] \;\mid\; T \& T \;\mid$$

- Attracted research interest in recent years, useful for XML/JSON schema languages
- Counting
  - T= $a \cdot (b + c)\,[2..3]$
  - $[\![T]\!] = \{a, acc, abc, abb, abc, accc, abcc, abbc, abcc, accb, abcb, abbb, abcb\}$
- Shuffle
  - $T = (a \cdot b)\&(X \cdot Y)$
  - $[\![T]\!] = \{abXY, aXbY, aXYb, XabY, XaYb, XYab\}$
- Membership is NP-hard for RE(&) [Mayer and Stockmeyer1994]

Adding counting # and shuffle &

$$T ::= \epsilon \mid a \mid T + T \mid T \cdot T \mid T\,[n..m] \mid T \& T \mid$$

- Attracted research interest in recent years, useful for XML/JSON schema languages
- Counting
  - T$= a \cdot (b + c)\,[2..3]$
  - $[\![T]\!] = \{a, acc, abc, abb, abc, accc, abcc, abbc, abcc, accb, abcb, abbb, abcb\}$
- Shuffle
  - $T = (a \cdot b)\&(X \cdot Y)$
  - $[\![T]\!] = \{abXY, aXbY, aXYb, XabY, XaYb, XYab\}$
- Membership is NP-hard for RE(&) [Mayer and Stockmeyer1994]
- But is polynomial for RE(#) [Kilpeläinen and Tuhkanen2003]

Our results

- Polynomial membership checking for a class of REs including & and #, obtained by means of mild restrictions
- More precisely
  - Quadratic membership-checking algorithm based on *constraints* for REs
  - Characterisation of a stability property, and it use for the design of linear membership algorithms
  - Extension of n-ary REs enriched with unordered concatenation, application to XML schema
  - Extensive experiments

The conflict-free restriction

- Conflict-free REs
  - *counting restricted to symbols*:

$$T ::= \quad \epsilon \;\mid\; a\,[m..n] \;\mid\; T+T \;\mid\; T\cdot T \;\mid\; T\&T$$

  - *single occurrence:* for any $a\,[m..n]$ and $a'\,[m'..n']$ in T, a is different from a'.

The conflict-free restriction

- Conflict-free REs
  - *counting restricted to symbols*:

$$T ::= \quad \epsilon \;\mid\; a\,[m..n] \;\mid\; T + T \;\mid\; T \cdot T \;\mid\; T \& T$$

  - *single occurrence:* for any $a\,[m..n]$ and $a'\,[m'..n']$ in T, a is different from a'.
- Used very often in practice, especially in the setting of XML schemas [Barbosa et al.2006, Choi2002].
- Example:

$$T = (a\,[1..3] \cdot b\,[2..2]) + c\,[1..*]$$

- Counterexamples

$$\begin{aligned} T' &= (a\,[1..3] \cdot b\,[2..2])\,[3..4] + c\,[1..*] \\ T'' &= (a\,[1..3] \cdot b\,[2..2]) + a\,[5..*] \end{aligned}$$

The conflict-free restriction

- Conflict-free REs
  - *counting restricted to symbols*:

$$T ::= \quad \epsilon \mid a\,[m..n] \mid T + T \mid T \cdot T \mid T \& T$$

  - *single occurrence:* for any $a\,[m..n]$ and $a'\,[m'..n']$ in T, a is different from a'.
- Used very often in practice, especially in the setting of XML schemas [Barbosa et al.2006, Choi2002].
- Example:

$$T = (a\,[1..3] \cdot b\,[2..2]) + c\,[1..*]$$

- Counterexamples

$$\begin{aligned} T' &= (a\,[1..3] \cdot b\,[2..2])\,[3..4] + c\,[1..*] \\ T'' &= (a\,[1..3] \cdot b\,[2..2]) + a\,[5..*] \end{aligned}$$

- Benefits:
  - inclusion $T \leq U$ over CF-REs can be decided in polynomial time [Colazzo et al.2009b]
  - even in the mixed case $T \leq U$ where only U is CF [Colazzo et al.2009a, Colazzo et al.2013b, Colazzo et al.2013a].
  - Quasi linear membership [Ghelli et al.2008] - extended abstract of this paper.

The conflict-free restriction

- Conflict-free REs
  - *counting restricted to symbols*:

$$T ::= \quad \epsilon \mid a\,[m..n] \mid T + T \mid T \cdot T \mid T \& T$$

  - *single occurrence:* for any $a\,[m..n]$ and $a'\,[m'..n']$ in T, a is different from a'.
- Used very often in practice, especially in the setting of XML schemas [Barbosa et al.2006, Choi2002].
- Example:

$$T = (a\,[1..3] \cdot b\,[2..2]) + c\,[1..*]$$

- Counterexamples

$$\begin{array}{rcl} T' & = & (a\,[1..3] \cdot b\,[2..2])\,[3..4] + c\,[1..*] \\ T'' & = & (a\,[1..3] \cdot b\,[2..2]) + a\,[5..*] \end{array}$$

- Benefits:
  - inclusion $T \leq U$ over CF-REs can be decided in polynomial time [Colazzo et al.2009b]
  - even in the mixed case $T \leq U$ where only U is CF [Colazzo et al.2009a, Colazzo et al.2013b, Colazzo et al.2013a].
  - Quasi linear membership [Ghelli et al.2008] - extended abstract of this paper.
- Main ingredient of our approaches: constraint-based characterisation of CF expressions [Colazzo et al.2009b].

The constraints

- To illustrate the intuition behind our constraints, consider again

$$T = ((a\,[1..3] \cdot b\,[2..2]) + c\,[1..*])$$

- It can be fully represented by the *conjunction* of the following constraints:
- Flat constraints $\mathcal{FC}(T)$:
  - Lower-bound: abc^+
  - Upper-bound: $\mathrm{upper}(abc)$
  - Cardinality: $a?[1..3] \wedge b?[2..2] \wedge c?[1..*]$
- Nested constraints $\mathcal{NC}(T)$:
  - Co-occurrence : $a^+ \Rrightarrow b^+ \wedge b^+ \Rrightarrow a^+$
  - Order: $a \prec b$
  - Mutual exclusion: $(a \prec c \wedge c \prec a) \wedge (b \prec c \wedge c \prec b)$

Syntax and semantics

$$F \ ::= \quad A^+ \ \mid \ A^+ \Rrightarrow B^+ \ \mid \ a?[m..n] \ \mid \ \text{upper}(A) \ \mid \ a \prec b \ \mid \ F \wedge F' \ \mid \ \mathbf{true}$$

$$\begin{array}{rcl}
w \models A^+ & \Leftrightarrow & (S(w) \cap A) \neq \emptyset, \text{ i.e., some } a \in A \text{ appears in } w \\
w \models A^+ \Rrightarrow B^+ & \Leftrightarrow & w \not\models A^+ \text{ or } w \models B^+ \\
w \models a?[m..*] & \Leftrightarrow & \text{if } a \text{ appears in } w\text{, then it appears at least } m \text{ times} \\
w \models a?[m..n] & \Leftrightarrow & \text{if } a \text{ appears in } w\text{, then it appears at least } m \text{ times} \\
(n \neq *) & & \text{and at most } n \text{ times} \\
w \models a \prec b & \Leftrightarrow & \text{there is no occurrence of } a \text{ in } w \text{ that follows an occurrence} \\
& & \text{of } b \text{ in } w \text{ (hence, both } a \text{ and } b \text{ may be missing)} \\
w \models \text{upper}(A) & \Leftrightarrow & S(w) \subseteq A \\
w \models F_1 \wedge F_2 & \Leftrightarrow & w \models F_1 \text{ and } w \models F_2 \\
w \models \mathbf{true} & \Leftrightarrow & \text{always}
\end{array}$$

Abbreviations

$$
\begin{aligned}
A^{+} \Leftrightarrow B^{+} \quad &=_{def} \quad A^{+} \Rrightarrow B^{+} \wedge B^{+} \Rrightarrow A^{+} \\
a \prec\succ b \quad &=_{def} \quad (a \prec b) \wedge (b \prec a) \\
A \prec B \quad &=_{def} \quad \bigwedge_{a \in A, b \in B} a \prec b \\
A \prec\succ B \quad &=_{def} \quad \bigwedge_{a \in A, b \in B} a \prec\succ b \\
\mathbf{false} \quad &=_{def} \quad \emptyset^{+} \\
A^{-} \quad &=_{def} \quad A^{+} \Rrightarrow \emptyset^{+}
\end{aligned}
$$

Constraint extraction

- Quadratic time extraction, soundness and completeness [Colazzo et al.2009b]:

$$w \in [\![T]\!] \quad \Leftrightarrow \quad w \models \mathcal{FC}(T) \wedge \mathcal{NC}(T)$$

- An expression T is nullable, noted as $\mathrm{N}(T)$, iff $\epsilon \in [\![T]\!]$
- To illustrate, consider $T = ((a + \epsilon)\&b\,[1..5]) \cdot (c + d\,[1..*])$, note that $\mathrm{N}(a + \epsilon)$
  - CC: $a^+ \Mapsto b^+ \;\wedge\; ab^+ \Leftrightarrow cd^+$
  - OC: $c \prec\succ d \;\wedge\; ab \prec cd$
  - Flat:

$$\begin{array}{ll} abcd^+ \wedge & \text{Lower-bound} \\ a?[1..1] \wedge b?[1..5] \wedge c?[1..1] \wedge d?[1..*] \wedge & \text{Cardinality} \\ \mathrm{upper}(abcd) & \text{Upper-bound} \end{array}$$

Constraint residuation

- $F \xrightarrow{a} F'$ means that F is transformed (or residuated) into F' by parsing the symbol a
- Main cases:

Computing the residual of a nested co-occurrence constraint.

Condition	$a \in A$	$a \in B$	$a \in A$	$a \in B$	$a \in A$
F	$A^+ \Mapsto B^+$	$A^+ \Mapsto B^+$	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	A^+
F'	B^+	**true**	B^+	A^+	**true**

Computing the residual of a nested order constraint.

Condition	$a \in A$	$a \in B$	$a \in A$	$a \in B$	$a \in A$
F	$A \prec B$	$A \prec B$	$A \prec\succ B$	$A \prec\succ B$	A^-
F'	$A \prec B$	A^-	B^-	A^-	**false**

Word membership checking

- Residuation is shifted to words $F \xrightarrow{w}{}^{*} F'$ in the obvious way:

$$
\begin{aligned}
&F \xrightarrow{\epsilon}{}^{*} F \\
&F \xrightarrow{aw}{}^{*} F'' \quad \Leftrightarrow_{\text{def}} \quad F \xrightarrow{a} F' \ \wedge \ F' \xrightarrow{w}{}^{*} F''
\end{aligned}
$$

- Corresponds to constraint semantics

$$w \models F \Leftrightarrow F \downarrow^{w} \textbf{true}$$

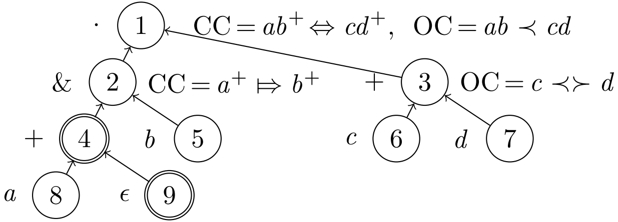
- Example of residuation:

$$ab^{+} \Leftrightarrow cd^{+} \xrightarrow{b} cd^{+} \xrightarrow{a} cd^{+} \xrightarrow{c} \textbf{true}$$

- In order to check $w \in [\![T]\!]$ we can residuate $\mathcal{NC}(T)$
- Complexity: $O(|w| * |\mathcal{NC}(T)|)$:
  - given w, for each constraint F in $\mathcal{NC}(T)$, we need to eventually parse the whole w in order to evaluate $F \downarrow^{w} G$.

We can do much better

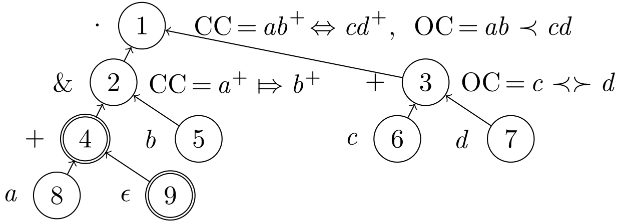
- Keep constraints/residuals implicit in a tree-shaped data structure with size $O(|T|)$
- For $T = ((a+\epsilon)\& b\,[1..5]) \cdot (c + d^+)$ we have:



- The two nullable nodes have double line in the picture.

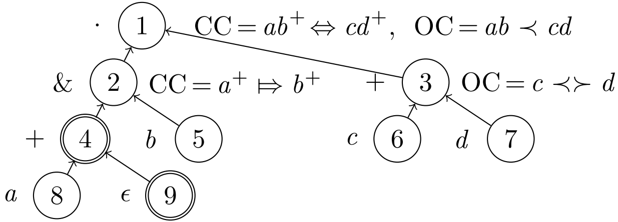
Residuation algorithm

- For each character a from the input word w:
  - it scans the ancestors of $a\,[m..n]$ in the constraint tree,
  - residuates all the constraints in this branch, and keeps track of all the resulting A^+ constraints.

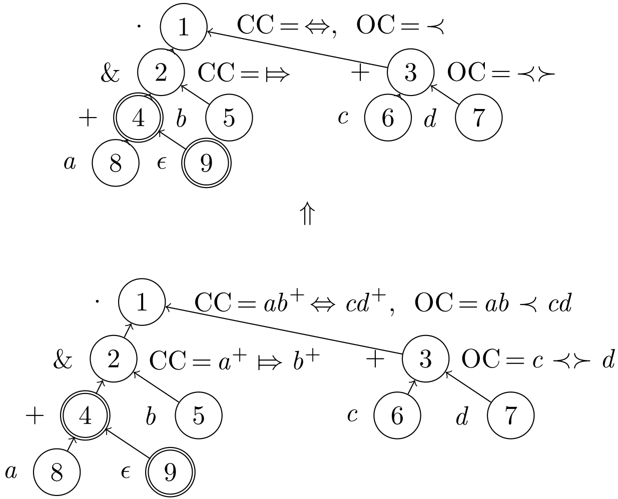


Residuation algorithm

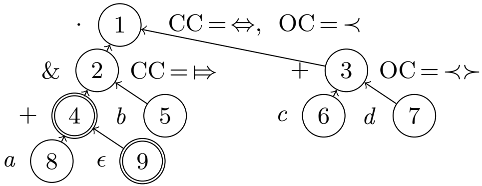
- For each character a from the input word w:
  - it scans the ancestors of $a\,[m..n]$ in the constraint tree,
  - residuates all the constraints in this branch, and keeps track of all the resulting A^+ constraints.
- At the end of w, it checks that all the A^+ constraints have been further residuated into **true** – the generation of a **false** causes an immediate failure.
- Flat constrains can be trivially checked in constant time for each symbol.



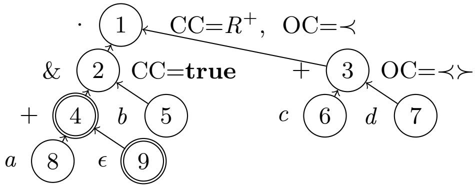
Optimising tree representation



Evolution of the tree while parsing $bbac$ for $T = ((a + \epsilon)\&b\,[1..5])\cdot(c + d^+)$



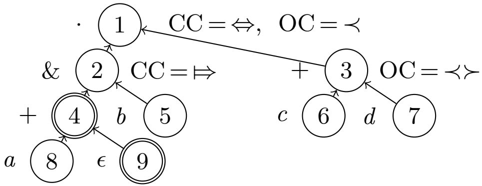
Initial tree



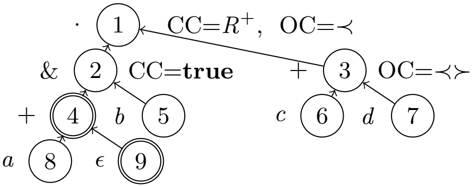
after $\underline{b}$ and after $b\underline{b}$

$a^+ \Rrightarrow b^+ \xrightarrow{b} \mathbf{true} \xrightarrow{b} \mathbf{true} \quad ab^+ \Leftrightarrow cd^+ \xrightarrow{b} cd^+ \xrightarrow{b} cd^+$

Evolution of the tree while parsing *bbac* for $T = ((a+\epsilon)\&b\,[1..5])\cdot(c+d^+)$

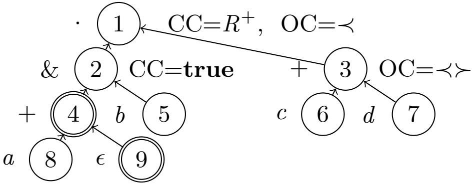


Initial tree



after $\underline{b}$ and after $b\underline{b}$

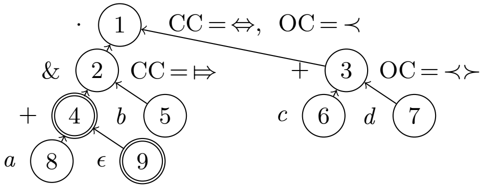
$a^+ \Mapsto b^+ \xrightarrow{b} \mathbf{true} \xrightarrow{b} \mathbf{true} \quad ab^+ \Leftrightarrow cd^+ \xrightarrow{b} cd^+ \xrightarrow{b} cd^+$



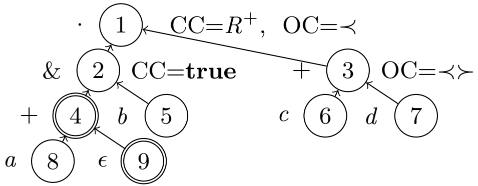
after $bb\underline{a}$
constraints unchanged

The evolution of the tree for $T = ((a+\epsilon)\&b\,[1..5])\cdot(c+d^+)$ while parsing bb

Evolution of the tree while parsing *bbac* for $T = ((a+\epsilon)\&b\,[1..5])\cdot(c+d^{+})$

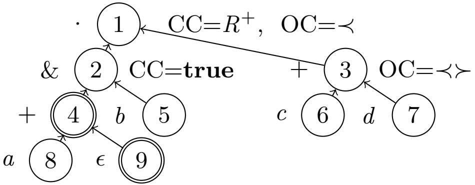


Initial tree



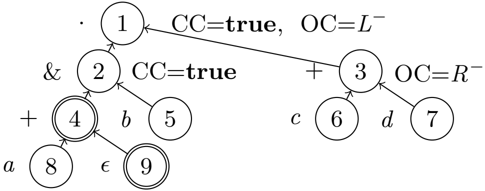
after $\underline{b}$ and after $b\underline{b}$

$a^{+} \Rrightarrow b^{+} \xrightarrow{b}$ **true** $\xrightarrow{b}$ **true** $\quad ab^{+} \Leftrightarrow cd^{+} \xrightarrow{b} cd^{+} \xrightarrow{b} cd^{+}$



after $bb\underline{a}$

constraints unchanged



after $bba\underline{c}$

$cd^{+} \xrightarrow{c}$ **true** $\quad ab \prec cd \xrightarrow{c} ab^{-}$

Soundness and completeness, complexity

- *Member(w,T)* yields *true* iff $w \in [\![T]\!]$
  - Follows from the fact the algorithms checks both nested and flat constraints of T.
- *Member(w,T)* runs in time $O(|T| + |w| * \mathit{depth}(T))$
  - The constraint tree can be built in time $O(|T|)$.
  - For every symbol a in w, only visit the path from the leaf node a to the root.
  - For each node in this path, a constant number of unit time operations is executed.
  - The final check on CC constraints scans at most $|T|$ nodes, while flat contraints can be checked in linear time.

Linear membership via stability

- The MEMBER algorithm visits all ancestors of an a-leaf every time a is found in w, which is redundant.
- We can avoid redundant operations, by relying on the fact that some constraints become stable after residuation (already seen in the example).

Linear membership via stability

- The Member algorithm visits all ancestors of an a-leaf every time a is found in w, which is redundant.
- We can avoid redundant operations, by relying on the fact that some constraints become stable after residuation (already seen in the example).
- Crucial observation: whenever a node constraint has been residuated because a symbol of A_L has been met, there is *almost* no reason to visit the node again because of a symbol from A_L (and the same holds for A_R)

Linear membership via stability

- More precisely, given the first $a \in A_1$
  - A_1^+ becomes **true**
  - ${A_1}^+ \Mapsto {A_2}^+$ or ${A_1}^+ \Leftrightarrow {A_2}^+$ becomes A_2^+.
  - $A_1 \prec\succ A_2$ becomes ${A_2}^-$.

Linear membership via stability

- More precisely, given the first $a \in A_1$
  - A_1^+ becomes **true**
  - ${A_1}^+ \Rrightarrow {A_2}^+$ or ${A_1}^+ \Leftrightarrow {A_2}^+$ becomes A_2^+.
  - $A_1 \prec\succ A_2$ becomes ${A_2}^-$.
- None of the obtained constraints can be affected by a second symbol from A_1.
- There is only one exception, for the constraint $A_1 \prec A_2$

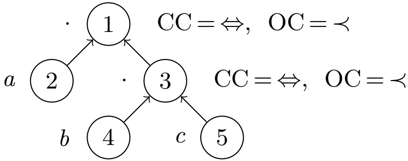
Linear membership via stability

- More precisely, given the first $a \in A_1$
  - A_1^+ becomes **true**
  - ${A_1}^+ \mapsto {A_2}^+$ or ${A_1}^+ \Leftrightarrow {A_2}^+$ becomes A_2^+.
  - $A_1 \prec\succ A_2$ becomes ${A_2}^-$.
- None of the obtained constraints can be affected by a second symbol from A_1.
- There is only one exception, for the constraint $A_1 \prec A_2$
  - the first $a \in A_1$ does not residuate it
  - but a subsequent symbol in A_2 residuates it into ${A_1}^-$
  - so another $a' \in A_1$ cannot be ignored, as it will cause the algorithm to *end* and yield "false".

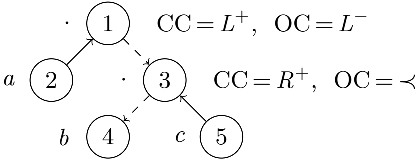
Linear mebership

- Our algorithm MEMBERSTAB exploits stability and has $O(|T| + |w|)$ complexity.
- In a nutshell, it 'deactivates' the edges along the paths that are visited, since they do not need to be visited again, unless an $A_1 \prec A_2$ node is residuated to ${A_1}^-$
- In this last case the paths below A_1 are eventually re-activated, but this is done at most once: as soon as one of the re-activated paths is traversed the algorithm stops, and the checking fails.
- So, in MEMBERSTAB each edge is traversed at most three times for any word w, and this yields linearity.
- Let's see un example.

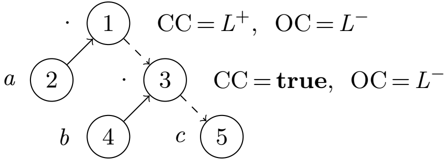
MemberStab example, $T = a \cdot (b \cdot c)$ and $w = bbcb$



Initial tree



After $\underline{b}$: residuation and edge discarding
After $b\underline{b}$: *no action*



After $bb\underline{c}$: the (4,3) edge is restored
After $bbc\underline{b}$: return **false**

Also in the paper

- Extension with unordered concatenation
- Extension to n-ary types (still linear)
- Extensive experiments.

Thank you !

Adding unordered concatenation

$$w \in [\![T_1 \% T_2 \% \ldots \% T_n]\!] \iff \exists \pi \in S_n.\ w \in [\![T_{\pi(1)} \cdot T_{\pi(2)} \cdot \ldots \cdot T_{\pi(n)}]\!]$$

- Not associative
- So we have considered n-ary expressions of the form

  $$T_1 \otimes \ldots \otimes T_n$$

  with $\otimes$ ranging over $\{\cdot,\ \&,\ \%\}$
- Needed to switch to flat-constraints

  $$F \ ::= \ \ \%\{A_1, \ldots, A_n\} \ \mid A_1 \prec \ldots \prec A_n \mid A_0{}^+ \Mapsto \{A_1{}^+, \ldots, A_n{}^+\} \mid \ \ldots$$

- Residuation for % constraints generates hybrid constraints including $\prec$-constraints.
- To illustrate, let $a \in A_2$

  $$\%\{A_1, A_2, A_3\} \xrightarrow{a} A_2 \prec \%\{A_1, A_3\}$$

MemberFlat and MemberFlatStab

- MemberFlat is a generalisation of Member to n-ary expressions/constrains
- Complexity is $O(|T| + |w| * \mathit{flatdepth}(T))$ where $\mathit{flatdepth}(T)$ is the depth of the type after all operators have been flattened.
- Quasi linear as $\mathit{flatdepth}(T)$ has in practice $\mathit{flatdepth}(T) < 4$
- We have proved that stability transposes to the n-ary case, thus obtaining MemberFlatStab
- Linear complexity $O(|T| + |w|)$
- Also in the paper: multiword checking and application to XML schema membership

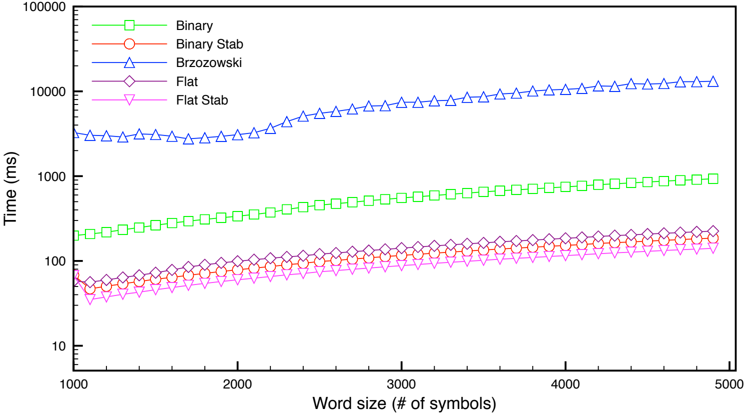
Performance and scalability

- We implemented our algorithms in Java
- To perform experimental analysis we performed comparisons with a baseline approach based on Brzozowski derivatives - as we did not found any competitor.
- Brzozowski derivatives are widely used to check membership, and to define finite state automata for REs.
- Examples:
  - $d_a(a\,[2..4]) = a\,[1..3]$
  - $d_a(d_a(a\,[2..4])) = a\,[0..2]$
  - $d_a(a\,[0..0]) = \emptyset$
  - $d_a(a\,[2..4] + b\,[1..3]) = d_a(a\,[2..4])$
  - $d_a((b\,[1..3] + \epsilon)\cdot\, a\,[2..4])) = d_a(a\,[2..4])$
- Complete formalisation in the paper.
- Soundness and completeness: $w \in [\![T]\!] \Leftrightarrow \epsilon \in [\![d_w(T)]\!]$
- Complexity of the baseline algorithm is $O(|w| * |T|^2)$.

Performance and scalability

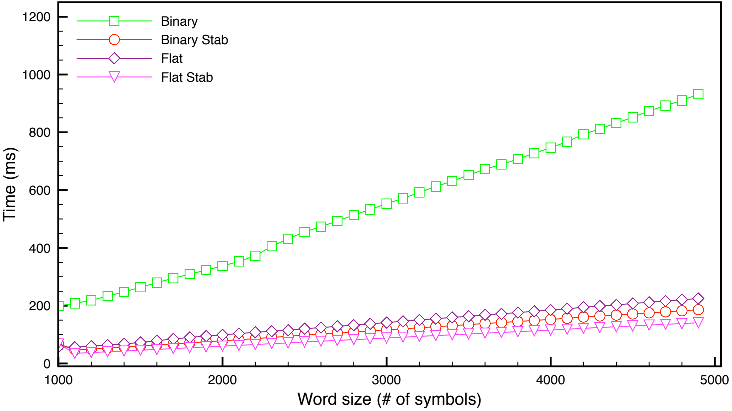
- We implemented a generator of synthetic expressions ensuring that operators have same probability of being generated
- For adopted types binary depth is 15 while flat depth is 4.
- We generated both positive and negative sets of words for each expression.
- The data generation algorithm guarantees that the expression is *equally covered* by the words.
- For each generated T we generated 30000 positive/negative words having from 1000 to 5000 symbols each

Positive experiments



Positive experiments for cf-RE(#, &): logarithmic scale.

Positive experiments



Positive experiments for cf-RE($\#, \&$): binary and flat algorithms.

Positive experiments

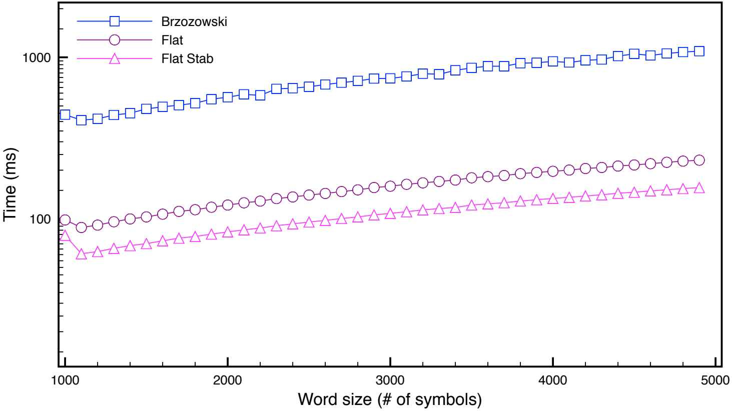


Figure: Positive experiments for cf-RE(#, &, %): logarithmic scale.

Negative experiments

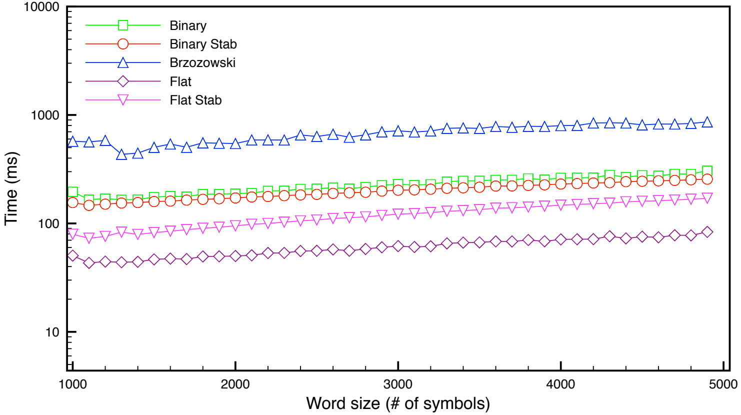


Figure: Negative experiments for cf-RE$(\#, \&)$: logarithmic scale on y-axis.

Negative experiments

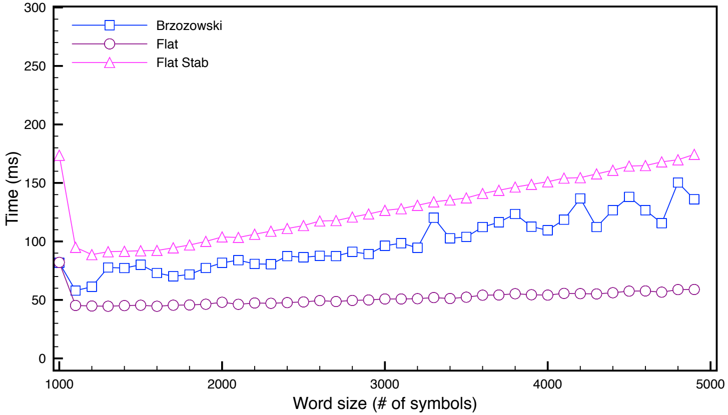


Figure: Negative experiments for cf-RE(#, &, %).

Negative experiments

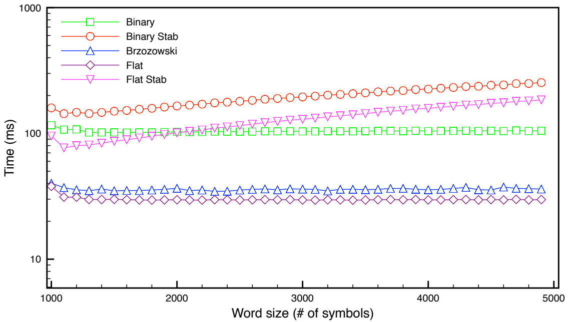


Figure: Negative experiments for cf-RE(#, &): random words.

Negative experiments

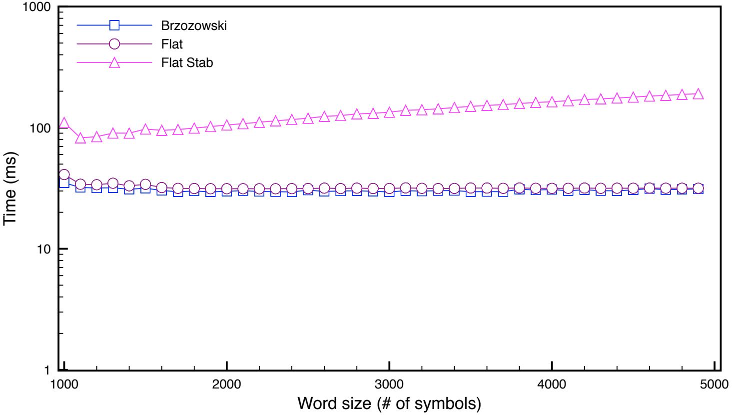


Figure: Negative experiments for cf-RE($\#, \&, \%$): random words.

Related works and conclusion

- Regular expressions have been extensively studied in the past (see the paper for an overview).
- Past studies [Gelade et al.2009, Hovland2012] have focused on extended REs, but no linear algorithm has been provided for a significative restriction.
- We have shown how restrictions and constraints can help in separating main properties that have to be checked during membership checking.
- This has enabled us to devise optimisations for specific aspects like order co-occurrence and order constraints.
- Experiments have validated complexity analysis on positive data sets.
- Surprisingly enough, experiments on negative data sets revealed that in this setting even a baseline algorithm can be competitive.

Carlos Buil Aranda, Marcelo Arenas, Óscar Corcho, and Axel Polleres. 2013.
Federating queries in SPARQL 1.1: Syntax, semantics and evaluation.
J. Web Sem. 18, 1 (2013), 1–17.

Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. 2004.
Incremental validation of XML documents.
ACM Trans. Database Syst. 29, 4 (2004), 710–751.

Denilson Barbosa, Gregory Leighton, and Andrew Smith. 2006.
Efficient Incremental Validation of XML Documents After Composite Updates. In *XSym (LNCS)*, Vol. 4156. Springer, 107–121.

Denilson Barbosa, Alberto O. Mendelzon, Leonid Libkin, Laurent Mignet, and Marcelo Arenas. 2004.
Efficient Incremental Validation of XML Documents.. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004*. IEEE Computer Society, 671–682.

Geert Jan Bex, Frank Neven, and Jan Van den Bussche. 2004.

DTDs versus XML Schema: A Practical Study. In *Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004, June 17-18, 2004, Colocated with ACM SIGMOD/PODS 2004*. 79–84.

Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. 2010.
Inference of concise regular expressions and DTDs.
ACM Trans. Database Syst. 35, 2 (2010), 11:1–11:47.

Geert Jan Bex, Frank Neven, and Stijn Vansummeren. 2007.
Inferring XML Schema Definitions from XML Data. In *VLDB*. 998–1009.

Henrik Björklund, Wim Martens, and Thomas Timm. 2015.
Efficient Incremental Evaluation of Succinct Regular Expressions. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM 2015*. ACM, 1541–1550.

Iovka Boneva, Radu Ciucanu, and Slawek Staworko. 2013.
Simple Schemas for Unordered XML. In *Proceedings of the 16th International Workshop on the Web and Databases 2013, WebDB 2013*. 13–18.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. 2006.
Extensible Markup Language (XML) 1.1 (Second Edition).
Technical Report. World Wide Web Consortium.
W3C Recommendation.

Anne Brüggemann-Klein. 1993.
Unambiguity of Extended Regular Expressions in SGML Document Grammars. In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings (Lecture Notes in Computer Science)*, Vol. 726. Springer, 73–84.

Anne Brüggemann-Klein and Derick Wood. 1992.
Deterministic Regular Languages. In *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings (Lecture Notes in Computer Science)*, Vol. 577. Springer, 173–184.

Anne Brüggemann-Klein and Derick Wood. 1998.
One-Unambiguous Regular Languages.
Inf. Comput. 142, 2 (1998), 182–206.

Janusz A. Brzozowski. 1964.
Derivatives of Regular Expressions.
J. ACM 11, 4 (1964), 481–494.

Byron Choi. 2002.
What are real DTDs like?. In *WebDB*. 43–48.

Dario Colazzo, Giorgio Ghelli, Luca Pardini, and Carlo Sartiani. 2009c.
Linear inclusion for XML regular expression types. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*. ACM, 137–146.

Dario Colazzo, Giorgio Ghelli, Luca Pardini, and Carlo Sartiani. 2013a.
Almost-linear inclusion for XML regular expression types.
ACM Trans. Database Syst. 38, 3 (2013), 15.

Dario Colazzo, Giorgio Ghelli, Luca Pardini, and Carlo Sartiani. 2013b.
Efficient asymmetric inclusion of regular expressions with interleaving and counting for XML type-checking.
Theor. Comput. Sci. 492 (2013), 88–116.

Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2009a.
Efficient asymmetric inclusion between regular expression types. In *ICDT (ACM International Conference Proceeding Series)*, Ronald Fagin (Ed.), Vol. 361. ACM, 174–182.

Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2009b.
Efficient inclusion for a class of XML types with interleaving and counting.
Inf. Syst. 34, 7 (2009), 643–656.

Silvano Dal-Zilio and Denis Lugiez. 2003.
XML Schema, Tree Logic and Sheaves Automata. In *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings (Lecture Notes in Computer Science)*, Vol. 2706. Springer, 246–263.

David C. Fallside and Priscilla Walmsley. 2004.
XML Schema Part 0: Primer – Second Edition.
(Oct 2004).
W3C Recommendation.

Wouter Gelade, Marc Gyssens, and Wim Martens. 2012.

Regular Expressions with Counting: Weak versus Strong Determinism.
SIAM J. Comput. 41, 1 (2012), 160–190.

Wouter Gelade, Wim Martens, and Frank Neven. 2009.
Optimizing Schema Languages for XML: Numerical Constraints and Interleaving.
SIAM J. Comput. 38, 5 (2009), 2021–2043.

Giorgio Ghelli, Dario Colazzo, and Carlo Sartiani. 2007.
Efficient Inclusion for a Class of XML Types with Interleaving and Counting. In *Database Programming Languages, 11th International Symposium, DBPL 2007, Vienna, Austria, September 23-24, 2007, Revised Selected Papers (Lecture Notes in Computer Science)*, Marcelo Arenas and Michael I. Schwartzbach (Eds.), Vol. 4797. Springer, 231–245.

Giorgio Ghelli, Dario Colazzo, and Carlo Sartiani. 2008.
Linear time membership in a class of regular expressions with interleaving and counting. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*. ACM, 389–398.

V M Glushkov. 1961.

The Abstract Theory of Automata.
Russian Mathematical Surveys 16, 5 (1961), 1.

Charles F. Goldfarb. 1990.
SGML handbook.
Clarendon Press.

Steve Harris and Andy Seaborne. 2013.
SPARQL 1.1 Query Language.
Technical Report. World Wide Web Consortium.
W3C Recommendation.

Dag Hovland. 2012.
The Membership Problem for Regular Expressions with Unordered Concatenation and Numerical Constraints. In *Language and Automata Theory and Applications - 6th International Conference, LATA 2012, A Coruña, Spain, March 5-9, 2012. Proceedings (Lecture Notes in Computer Science)*, Adrian Horia Dediu and Carlos Martín-Vide (Eds.), Vol. 7183. Springer, 313–324.

J.E. Hopcroft and J.D. Ullman. 1979.
Introduction to Automata Theory, Languages and Computation.

Addison-Wesley.

Pekka Kilpeläinen and Rauno Tuhkanen. 2003.
Regular Expressions with Numerical Occurrence Indicators - preliminary results. In *Proceedings of the Eighth Symposium on Programming Languages and Software Tools, SPLST'03, Kuopio, Finland, June 17-18, 2003*, Pekka Kilpeläinen and Niina Päivinen (Eds.). University of Kuopio, Department of Computer Science, 163–173.

Pekka Kilpeläinen and Rauno Tuhkanen. 2004.
Towards efficient implementation of XML schema content models. In *Proceedings of the 2004 ACM Symposium on Document Engineering, Milwaukee, Wisconsin, USA, October 28-30, 2004*. ACM, 239–241.

Leonid Libkin, Wim Martens, and Domagoj Vrgoc. 2016.
Querying Graphs with Data.
J. ACM 63, 2 (2016), 14.

Anthony Mansfield. 1983.
On the computational complexity of a merge recognition problem.
Discrete Applied Mathematics 5, 1 (1983), 119 – 122.

Alain J. Mayer and Larry J. Stockmeyer. 1994.
Word Problems — This Time with Interleaving.
Inf. Comput. 115, 2 (1994), 293–311.

Anders Møller. 2010.
dk.brics.automaton – Finite-State Automata and Regular Expressions for Java.
(2010).
`http://www.brics.dk/automaton/`.

Manizheh Montazerian, Peter T. Wood, and Seyed R. Mousavi. 2007.
XPath Query Satisfiability is in PTIME for Real-World DTDs. In *XSym (LNCS)*, Vol. 4704. Springer, 17–30.

Sushant Patnaik and Neil Immerman. 1997.
Dyn-FO: A Parallel, Dynamic Complexity Class.
J. Comput. Syst. Sci. 55, 2 (1997), 199–209.

Randy Smith, Cristian Estan, Somesh Jha, and Shijin Kong. 2008.
Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications,*

Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008. ACM, 207–218.

C. M. Sperberg-McQueen. 2004.
Notes on finite state automata with counters.
Technical Report.
Available at `http://www.w3.org/XML/2004/05/msm-cfa.html`.

C. M. Sperberg-McQueen. 2005.
Applications of Brzozowski derivatives to XML Schema processing. In *Proceedings of the Extreme Markup Languages® 2005 Conference.*

Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. 2004.
XML Schema Part 1: Structures Second Edition.
Technical Report. World Wide Web Consortium.
W3C Recommendation.

Manfred K. Warmuth and David Haussler. 1984.
On the Complexity of Iterated Shuffle.
J. Comput. Syst. Sci. 28, 3 (1984), 345–358.

Peter T. Wood. 2003.
Containment for XPath Fragments under DTD Constraints. In *Proceedings of the 9th International Conference on Database Theory - ICDT 2003, Siena, Italy, January 8-10, 2003 (Lecture Notes in Computer Science)*, Vol. 2572. Springer, 297–311.