Deterministic Regular Expressions in Linear Time

reasoning about deterministic regular expressions
without building the automata

Benoit Groz¹,³, Sebastian Maneth²,³, Slawek Staworko¹,³

¹University of Lille, Mostrare INRIA
²UNSW, NICTA
³Équipe associée Transduce

PODS, May, 2012
Context

DTD and XML Schema use regular expressions to specify which sequence of children may appear below a node.

```xml
<!ELEMENT book (author, chapter*, index?)>
```

Constraint: regular expression must be deterministic.

We provide new algorithms to:

- Check if a regular expression is deterministic.
- Decide the membership problem for deterministic regular expressions.
Outline

1. Glushkov relations: First, Last, Follow ... and determinism
2. Problem statement
3. Structure of the expression
4. Algorithms to test membership
Outline

1. Glushkov relations: First, Last, Follow ... and determinism
2. Problem statement
3. Structure of the expression
4. Algorithms to test membership
Structure of regular expressions

\[ ab^* b \quad \text{and} \quad a b b^* \]
Structure of regular expressions

Expression is non-deterministic if:

\[ a_1 b_2^* b_3 \]

\[ a_1 b_2 b_3^* \]

\[ b_3 \text{ follows } a_1, \ b_2 \text{ follows } a_1 \ldots \]
Structure of regular expressions

**Expression** is non-deterministic if:

\[ a_i b_j^* a_k \ (j \neq k) \]

Example:

- \( a_1 b_2^* b_3 \)

- \( a_1 b_2 b_3^* \)
Structure of regular expressions

An expression is non-deterministic if:

\[ a_1 b_2^* b_3 \]

\[ \# a_1 b_2^* b_3 \$

\[ \# a_2 b_3^* b_3 \$

\[ \# a_1 b_2 b_3^* \$

\[ \# a_1 b_2 b_3 \$
Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is *non deterministic* if:

\[ b_i \rightarrow a_j \rightarrow a_k \ (j \neq k) \]

\ [#a_1 b_2^* b_3$] \Rightarrow \textit{non deterministic}  \\
\ [#a_1 b_2 b_3^*$] \Rightarrow \textit{deterministic}
Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is *non deterministic* if:

\[ b_i \rightarrow a_j \rightarrow a_k \quad (j \neq k) \]

\( #a_1 b_2^* b_3 $ \)  \[ \Rightarrow \text{non deterministic} \]

\( #a_1 b_2 b_3^* $ \)  \[ \Rightarrow \text{deterministic} \]
Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is *non deterministic* if:

\[ b_i \rightarrow a_j \rightarrow a_k \quad (j \neq k) \]

\[ #a_1 b_2^* b_3$ \]

⇒ *non deterministic*

\[ #a_1 b_2 b_3^*$ \]

⇒ *deterministic*
Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is *non deterministic* if:

\[ b_i \rightarrow a_j \rightarrow a_k \quad (j \neq k) \]

\[ \#a_1b_2^*b_3$ \] \[ \Rightarrow \text{non deterministic} \]

\[ \#a_1b_2b_3^*$ \] \[ \Rightarrow \text{deterministic} \]

Ambiguity parsing \( w = ab \)
Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is *non deterministic* if:

\[ b_i \xrightarrow{a_j} a_k \quad (j \neq k) \]

\#a_1b_2^*b_3$ \quad \Rightarrow \text{non deterministic} \\
\#a_1b_2b_3^*$ \quad \Rightarrow \text{deterministic}

\[ e = (a + b)b?(ab)^* \quad ? \]
\[ e = (ab+ba?)^* \quad ? \]
Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is \textit{non deterministic} if:

\[ b_i \, a_j \, a_k \quad (j \neq k) \]

\[ #a_1b_2^*b_3$ \quad \Rightarrow \text{non deterministic} \]

\[ #a_1b_2b_3^*\$ \quad \Rightarrow \text{deterministic} \]

\[ e = (a + b)b?(ab)^* \quad \Rightarrow \text{deterministic} \]

\[ e = (ab + ba?)^* \quad \Rightarrow \text{non deterministic: } w = ba \]
Outline

1. Glushkov relations: First, Last, Follow ... and determinism
2. Problem statement
3. Structure of the expression
4. Algorithms to test membership
Problems of interest

Testing determinism:
Input: expression \(e\),
Question: is \(e\) deterministic?

Membership:
Input: word \(w\), deterministic expression \(e\),
Question: \(w \in L(e)\)?

Scenario: big expression, big alphabet.

Remark:
size of \(e\) \(\simeq\) number of nodes in the parse tree
\(\simeq\) number of positions.
Problems of interest

Testing determinism:
Input: expression $e$,
Question: is $e$ deterministic?

Membership:
Input: word $w$, deterministic expression $e$,
Question: $w \in L(e)$?

Straightforward solution through Glushkov automaton.
Build Glushkov in $O(|\Sigma| \times |e|)$ [Brüggeman-Klein TCS’93].
\[ \Rightarrow \text{(quadratic in } |e|) \]

Number of transitions of Glushkov can be quadratic:
\[
\begin{align*}
  e &= (a + b + c \ldots)(a + b + c \ldots), \\
  e' &= (a + b + c \ldots)^*, \\
  e'' &= (a?b?c?\ldots)
\end{align*}
\]
Problems of interest

Testing determinism:
Input: expression $e$,
Question: is $e$ deterministic?

Membership:
Input: word $w$, deterministic expression $e$,
Question: $w \in L(e)$?

Straightforward solution through Glushkov automaton.
Build Glushkov in $O(|\Sigma| \times |e|)$ [Brüggeman-Klein TCS’93].
$\Rightarrow$ (quadratic in $|e|$)

Number of transitions of Glushkov can be quadratic:

$$e = (a + b + c \ldots)(a + b + c \ldots),$$
$$e' = (a + b + c \ldots)^*,$$
$$e'' = (a?b?c?\ldots)$$
Problems of interest

Testing determinism:
Input: expression \( e \),
Question: is \( e \) deterministic?

Membership:
Input: word \( w \), deterministic expression \( e \),
Question: \( w \in L(e) \) ?

Straightforward solution through Glushkov automaton.
Build Glushkov in \( O(|\Sigma| \times |e|) \) [Brüggeman-Klein TCS’93].
\( \Rightarrow (\text{quadratic in } |e|) \)

Can we do better?
### Summary

<table>
<thead>
<tr>
<th></th>
<th>Glushkov</th>
<th>Our results</th>
</tr>
</thead>
<tbody>
<tr>
<td>testing determinism</td>
<td>$O(</td>
<td>\Sigma</td>
</tr>
<tr>
<td>membership</td>
<td>$O(</td>
<td>\Sigma</td>
</tr>
<tr>
<td>★ $k$-occurrence</td>
<td></td>
<td>$O(k \times</td>
</tr>
<tr>
<td>★ restrictions on +</td>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td>★ star free</td>
<td></td>
<td>$O(</td>
</tr>
</tbody>
</table>
Preprocessing

We do not construct automaton. Instead we work on parse tree which we preprocess in linear time to build some pointers + datastructures.

● **Testing determinism.**
  We search a witness for non-determinism in $e$: pair of two positions with same label that follow a common position.

  $$b_i \xrightarrow{a_j} a_k \quad (j \neq k)$$

  *We limit the number of pairs examined to $O(|e|)$ using skeleta from [Bojańczyk and Parys JACM’11].*

● **Testing membership.**
  We simulate transitions on-the-fly using the pointers from preprocessing.
Roadmap: transition simulation

We want a procedure for transition simulation:

**Input:**
- a position $a_i$ in the expression,
- a letter $b$ (the next letter of the word)

**Output:**
- the unique $b$-labeled position that follows $a_i$
Outline

1. Glushkov relations: First, Last, Follow ... and determinism
2. Problem statement
3. Structure of the expression
4. Algorithms to test membership
Some tools

LCA preprocessing [Harel&Tarjan, SICOMP’84]

One can preprocess a tree in linear time, to answer lowest common ancestor (LCA) queries and ancestor queries in constant time.

(Input for preprocessing: \( t \))

**LCA query:**
- Input: two nodes \( n_1, n_2 \) in \( t \)
- Output: the lowest common ancestor (LCA) of \( n_1 \) and \( n_2 \)

**Ancestor query:**
- Input: two nodes \( n_1, n_2 \) in \( t \)
- Question: is \( n_1 \) an ancestor of \( n_2 \)?
Work on parse tree

Goal: Given two positions $a_i$ and $b_j$, test if $b_j$ follows $a_i$ in constant time.
Work on parse tree: Follow

When does $b_j$ follow $a_i$?

$LCA(a_1, b_3)$

$\{a_1, c_2\}$

$b_3$

$a_4$

$a_1$

$c_2$

$LCA(b_7, a_5)$

$\{b_7, b_6\}$

$\{a_5, b_7\}$

$\star$

$LCA(b_7, a_5)$

$\{b_7, b_6\}$

$\{a_5, b_7\}$

$\star$

$\{\ldots\}$ : First set

$\{\ldots\}$ : Last set

Case 1 : ⬤

Case 2 : *

Benoit Groz (Mostrare)

Deterministic regular expressions

PODS, May, 2012 15 / 25
Assuming that after some preprocessing we can test $First$ and $Last$ in constant time,

**Theorem**

We can test if $b_j$ follows $a_i$ in constant time.
Work on parse tree: Follow (2)

Assuming that after some preprocessing we can test \textit{First} and \textit{Last} in constant time,

\begin{quote}
\textbf{Theorem}

We can test if \( b_j \) follows \( a_i \) in constant time.
\end{quote}

Preprocessing:

pointer to lowest \( * \) ancestor of each node.

build \textit{LCA}, \textit{First} and \textit{Last} structures.

Algo. to test if \( b_j \) follows \( a_i \):

compute \( \text{LCA}(b_j, a_i) \)

follow the \( * \) pointer (for case 2: \( * \))

test that \( a_i, b_j \) in \textit{First} and \textit{Last} of appropriate nodes

\textit{New objective: test if } \( a_i \in \text{First}(n) \) \textit{ in constant time}
Work on parse tree: First and Last

```
# n₁
  a₁ * n₂
    n₃
      ? n₄
        ? ? n₅
        a₂ c₃ b₄ a₅

§
```

Figure: Expression $e₀ = (a(a(c(ba))?)?)a$.
Work on parse tree: First and Last

\[
\{a_2, \ldots \} \text{ : First set}
\]
Work on parse tree: First and Last

Figure: Expression $e_0 = (a( (a ? c ? (ba)?)?) \ast .

\{a_2, \ldots\} : First set
Work on parse tree: First and Last

\[ \text{First set} \]

\[ \{ a_2, \ldots \} : \text{First set} \]
Work on parse tree: First and Last

\[
\{a_2, \ldots \} : \text{First set}
\]
Work on parse tree: First and Last

Figure: Expression $e_0 = (a (a (c) (b))?)^*$. 

Deterministic regular expressions
Work on parse tree: First and Last

Figure: Expression $e_0 = (a (a c) b)^*$. 

{ $a_2, \ldots$ } : First set
First set: \( a \) SupFirst pointer

\( \{ a_2, \ldots \} : \) First set

\( \longrightarrow \) : a SupFirst pointer

Figure: Expression

\( e_0 = (a(a(c(a(ba)))))^* \).
We compute $SupFirst$ pointers in simple traversal.

$b_4 \in First(n_3)$ because $n_2$ ancestor of $n_3$ and $n_3$ ancestor of $b_4$.

$a_5 \notin First(n_3)$
We compute $SupFirst$ pointers in simple traversal.

$b_4 \in First(n_3)$ because $n_2$ ancestor of $n_3$ and $n_3$ ancestor of $b_4$.

$a_5 \notin First(n_3)$

✓ We can test if $a_i \in First(n)$ in constant time.

Symmetrically, we can test if $a_i \in Last(n)$ in constant time.
Reminder

**Theorem**

We can test if $b_j$ follows $a_i$ in constant time.
Outline

1. Glushkov relations: First, Last, Follow . . . and determinism
2. Problem statement
3. Structure of the expression
4. Algorithms to test membership
Simple case: $k$-occurrence expression

**Theorem**

For deterministic $k$-occurrence expression, the membership problem can be solved in $O(k|w|)$ after $O(|e|)$ preprocessing.

each transition simulated in $O(k)$:

$$e = (ba)^* c acabcc?b$$
General case
We put color $a$ in $\text{parent}(\text{SupFirst}(a_i))$ and store $a_i$ as the witness for color $a$ in that node.

Observation: positions followed by $a_5$ are below $n_5$, those followed by $a_2$ or $b_4$ are below $n_1$...

\[ a \mapsto a_5 \] (color $a$, witness $a_5$)
Finding the right ancestor

We can use “nearest colored ancestor queries”.

**Nearest colored ancestor [Muthukrishnan et al. 96]**

We can preprocess a tree \( t \) in expected linear time \( O(|t|) \) to answer nearest colored ancestor queries in \( O(\log \log |t|) \).

Expected time because of hashmaps, but becomes worst-case linear using lazy arrays.

**Evaluation algorithm**

Repeatedly jump to the nearest ancestor with color \( a \), and test if its witness follows.

**Why is it linear?**

Use amortization argument
## Summary

<table>
<thead>
<tr>
<th></th>
<th>Glushkov</th>
<th>Our results</th>
</tr>
</thead>
<tbody>
<tr>
<td>testing determinism</td>
<td>$O(</td>
<td>\Sigma</td>
</tr>
<tr>
<td>membership</td>
<td>$O(</td>
<td>\Sigma</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(k \times</td>
</tr>
<tr>
<td>$k$-occurrence ($k$-ORE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restrictions on $+$</td>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td>star free</td>
<td></td>
<td>$O(</td>
</tr>
</tbody>
</table>
Future work

- Close the log log gap for membership.
- Searching regular pattern instead of matching (KMP...)
For a few dollars more...

Questions are most welcome!

...but there is no guarantee for the answer

Questions?