XML Security Views

Queries, Updates, and Schema

Benoît Groz

University of Lille, Mostrare INRIA

PhD defense, October 2012

Talk Outline

Outline

Context: Protecting data

- March 2011: an attack retrieved huge mailing lists from Epsilon, a leading online marketing company.
- April 2011: Sony's PlayStation network : 100 million customer accounts compromised including street numbers, email, and passwords.
- June 2011: CitiBank communicated a breach into 1% of its credit card accounts (200.000 customers).
- March 2012: 1.500.000 card numbers compromised as a result of unauthorized access into GlobalPayment processing system.

Context: XML constellation

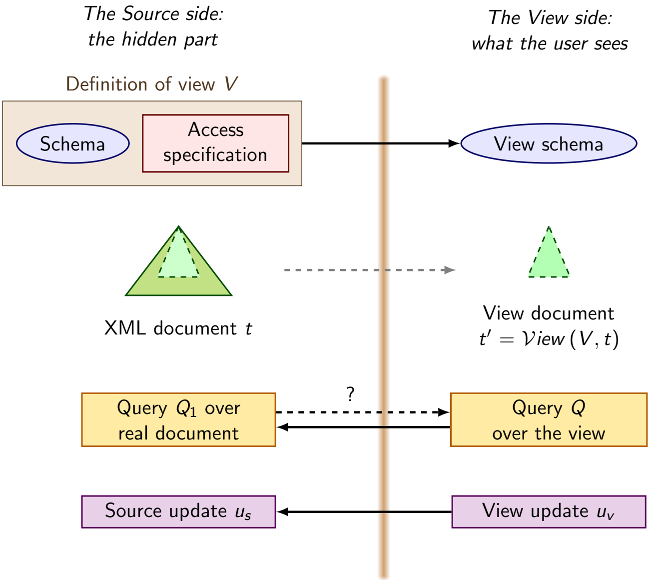
Purpose: large-scale electronic publishing

- usability over the Internet
- compatibility with SGML
- facilitating automatic processing of the documents

Features:

- document model: a document = a tree
- Languages to manipulate the document: Query and Transformation languages: XPath, XQuery, XQUF, XSLT
- Schema languages: DTD, RelaxNG, XML Schema, Schematron

Our project



The Source side:
the hidden part
The View side:
what the user sees
Definition of view V
Schema
Access specification
View schema
XML document t
View document
$t' = \mathcal{V}iew\,(V, t)$
?
Query Q_1 over real document
Query Q over the view
Source update u_s
View update u_v

Our project

Project: Develop techniques for XML security views.

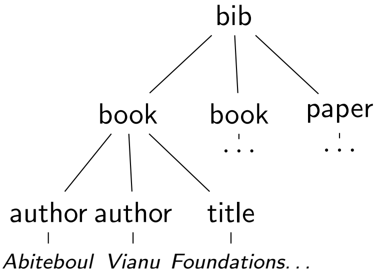
Originally: techniques to reason about XML security views.
... but the problem addressed are general database problems: can find application in any system using views, and more...

XML document

XMLDocument

```
<bib>
  <book>
    <author> Abiteboul </author>
    <author> Vianu </author>
    <title> Foundations... </title>
  </book>
  <book>
    ⋮
  </book>
  <paper>
    ⋮
  </paper>
</bib>
```

Tree representation



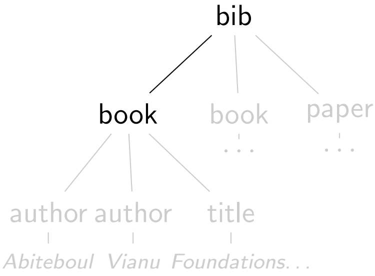
labeled ordered unranked trees

XML document

XMLDocument

```
<bib>
  <book>
    <author> Abiteboul </author>
    <author> Vianu </author>
    <title> Foundations... </title>
  </book>
  <book>
    ⋮
  </book>
  <paper>
    ⋮
  </paper>
</bib>
```

Tree representation



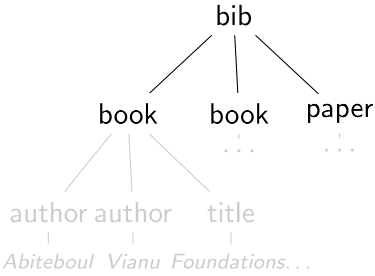
labeled ordered unranked trees

XML document

XMLDocument

```
<bib>
  <book>
    <author> Abiteboul </author>
    <author> Vianu </author>
    <title> Foundations... </title>
  </book>
  <book>
    ⋮
  </book>
  <paper>
    ⋮
  </paper>
</bib>
```

Tree representation



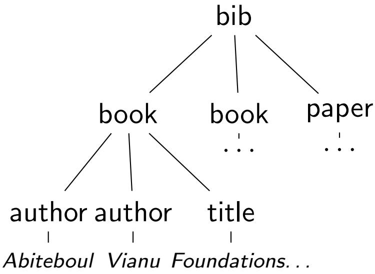
labeled ordered unranked trees

XML document

XMLDocument

```
<bib>
  <book>
    <author> Abiteboul </author>
    <author> Vianu </author>
    <title> Foundations... </title>
  </book>
  <book>
    ⋮
  </book>
  <paper>
    ⋮
  </paper>
</bib>
```

Tree representation



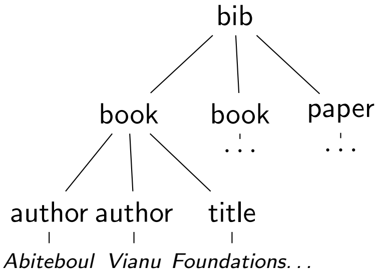
labeled ordered unranked trees

DTD

DTD D

$\texttt{bib} \rightarrow (\texttt{book} + \texttt{paper})^*$
$\texttt{book} \rightarrow \texttt{author}^*, \texttt{title}$
$\texttt{author} \rightarrow \#\texttt{PCDATA}$
$\texttt{title} \rightarrow \#\texttt{PCDATA}$

tree t satisfying D



XPath

Definition

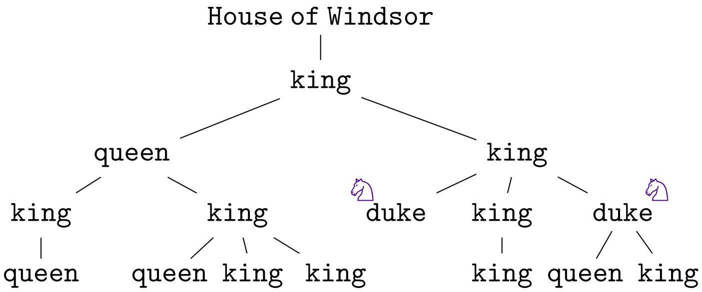
Query: function $t \mapsto Q(t) \subseteq Nodes(t)$

Several XPath languages: XPath 1.0, XPath 2.0, XPath 3.0 ...

Researchers very often focus on the navigational core.

Core XPath 1.0 $\subset$ Conditional XPath $\subset$ Regular XPath [Marx EDBT'04] .

XPath



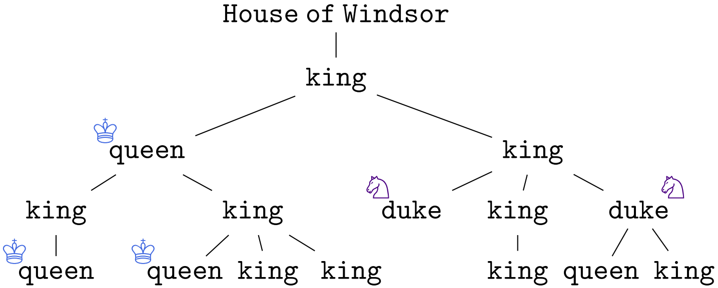
Regular XPath: path expressions with transitive closure and filters

♘ $\Downarrow^{*}$::duke

♔ $(\Downarrow$::king$/\Downarrow$::queen$)^{*}$

♕ $(\Downarrow$::king$/\Downarrow$::queen$)^{*}$/self::[$\Rightarrow$::king$/\Rightarrow$::king]

XPath



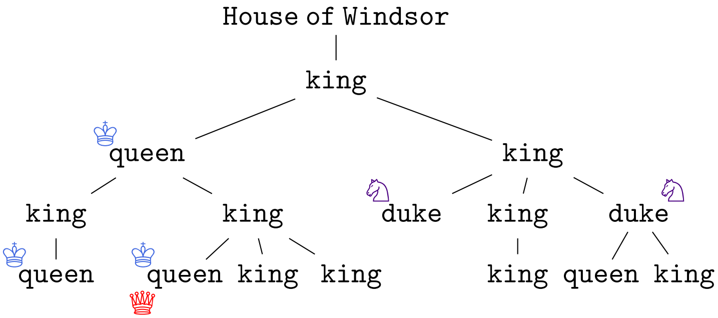
Regular XPath: path expressions with transitive closure and filters

♘ $\Downarrow^*::\mathtt{duke}$

♔ $(\Downarrow::\mathtt{king}/\Downarrow::\mathtt{queen})^*$

♕ $(\Downarrow::\mathtt{king}/\Downarrow::\mathtt{queen})^*/\mathrm{self}::[\Rightarrow::\mathtt{king}/\Rightarrow::\mathtt{king}]$

XPath



Regular XPath: path expressions with transitive closure and filters

♘ $\Downarrow^*$::duke

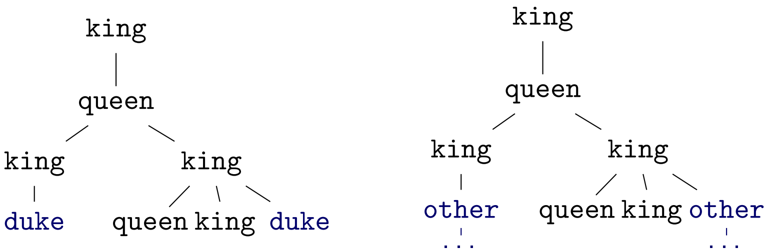
♔ $(\Downarrow$::king/$\Downarrow$::queen$)^*$

♕ $(\Downarrow$::king/$\Downarrow$::queen$)^*$/self::[$\Rightarrow$::king/$\Rightarrow$::king]

XQUF

Update language based on XQuery (thereby on XPath)

```
for $x in ⇓*::duke return
  delete node $x ,
  insert node <other>...</other> before $x
```



(Security) views

Security views are simple views defined in [Fan et al.'04 and '07].
Operations: hide or rename nodes.

Example

Storing successive versions of papers, hiding old versions
DTD D_0:

$\texttt{docs} \to \texttt{paper}^*$
$\texttt{paper} \to \texttt{name}, \texttt{version}$
$\texttt{version} \to \texttt{number}, \texttt{files}, \texttt{prev}$
$\quad \texttt{prev} \to \texttt{version} \mid \varepsilon$

$Q_0 = \Downarrow::\texttt{paper}/(\text{self} \cup \Downarrow::\texttt{name} \cup \Downarrow::\texttt{version}/\Downarrow::\texttt{files})$

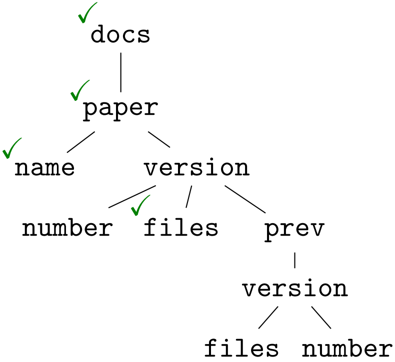
Here, security view = pair (D_0, Q_0)
Nodes selected by Q_0 (plus root) are visible, others are hidden.

(Security) views

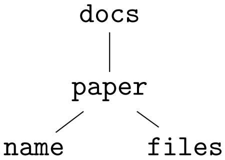
What happens when the parent of a visible node n is hidden?

Two approaches:

- forbid this (upward-closed queries) $\Longrightarrow$ makes things simpler
- or n gets adopted by its closest visible ancestor $\Longrightarrow$ more expressive



A document $t \vDash D_0$



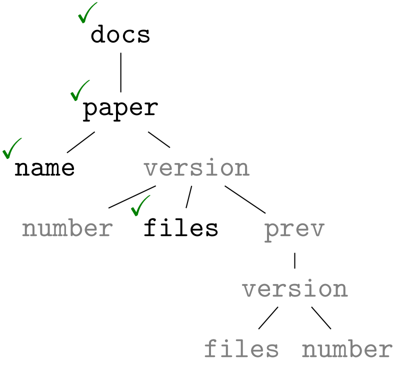
$\mathcal{V}iew\,(Q_0, t)$

(Security) views

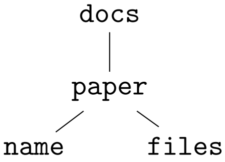
What happens when the parent of a visible node n is hidden?

Two approaches:

- forbid this (upward-closed queries) $\Longrightarrow$ makes things simpler
- or n gets adopted by its closest visible ancestor $\Longrightarrow$ more expressive



A document $t \models D_0$



$\mathcal{V}iew\,(Q_0, t)$

3 selected pieces

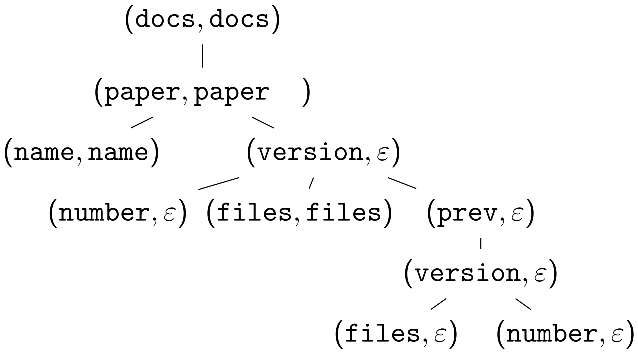
- PB 1 (Queries): Determinacy and Query rewriting
- PB 2 (Updates): The view update problem
- PB 3 (Schema): check if a schema is “correct” w.r.t. W3C specifications

Outline

Queries, Views, and Updates as Alignment languages

Representing a query with alignments

$Q_0 = \Downarrow::\texttt{paper}/(\text{self} \cup \Downarrow::\texttt{name} \cup \Downarrow::\texttt{version}/\Downarrow::\texttt{files})$



One alignment in Q_0

Queries only select: alphabet$=\{(a, \beta) \mid a \in \Sigma, \beta = a \text{ or } \beta = \varepsilon\}$

Views select or rename: alphabet$=\{(a, \beta) \mid a \in \Sigma, \beta \in \Sigma \cup \{\varepsilon\}\}$

Queries, Views, and Updates as Alignment languages

Representing a view with alignments

$Q_0 = \Downarrow::\mathtt{paper}/(\mathsf{self} \cup \Downarrow::\mathtt{name} \cup \Downarrow::\mathtt{version}/\Downarrow::\mathtt{files})$

- $(\mathtt{docs}, \mathtt{docs})$
  - $(\mathtt{paper}, \mathtt{article})$
    - $(\mathtt{name}, \mathtt{id}\ \ )$
    - $(\mathtt{version}, \varepsilon)$
      - $(\mathtt{number}, \varepsilon)$
      - $(\mathtt{files}, \mathtt{files})$
      - $(\mathtt{prev}, \varepsilon)$
        - $(\mathtt{version}, \varepsilon)$
          - $(\mathtt{files}, \varepsilon)$
          - $(\mathtt{number}, \varepsilon)$

One alignment in Q_0

Queries only select: alphabet$=\{(a, \beta) \mid a \in \Sigma, \beta = a \text{ or } \beta = \varepsilon\}$
Views select or rename: alphabet$=\{(a, \beta) \mid a \in \Sigma, \beta \in \Sigma \cup \{\varepsilon\}\}$

Queries, Views, and Updates as Alignment languages

Representing an update with upward-closed alignments

```
f: for $x in ⇓*::paper return (rename node $x into article
     delete nodes $x/⇓::version/⇓* ,
     insert node <author>...</author> as first into $x)
```

- (docs, docs)
  - (paper, article)
    - (ε, author)
    - (name, name)
    - (version, ε)
      - (number, ε)
      - (files, ε)
      - (prev, ε)
        - (version, ε)
          - (files, ε)
          - (number, ε)

One alignment of update function f

Automata

VPA

Visibly Pushdown Automata (VPA) [Alur&Madhusudan'04]
2 main applications: Verification and XML processing.

Characteristics: Work on linearization of the trees: read one element after another, and update the state accordingly.

Uses a stack, but stack operation determined by the element read.

Output = no iff

- cannot process the document until its end

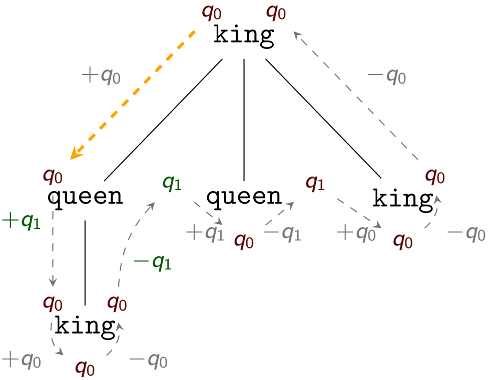
or

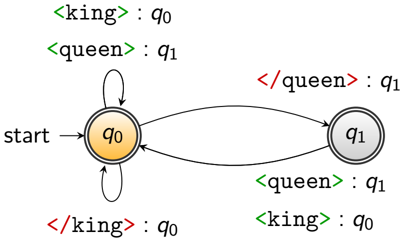
- state at the end not accepting

VPA: run

```
<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>
```

stack: q_0

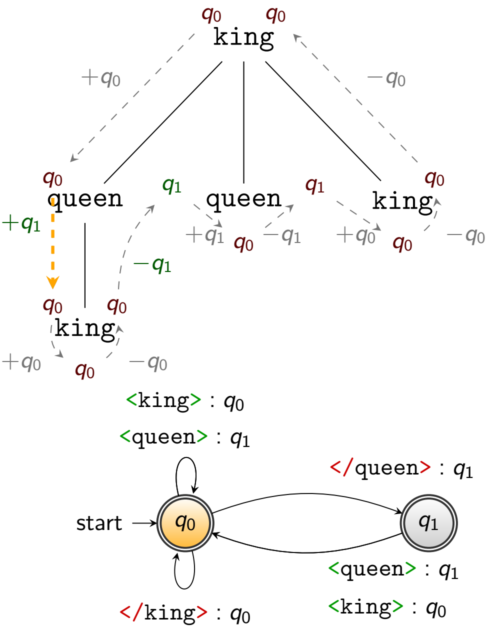




VPA: run

```
<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>
```

stack: q_0 q_1

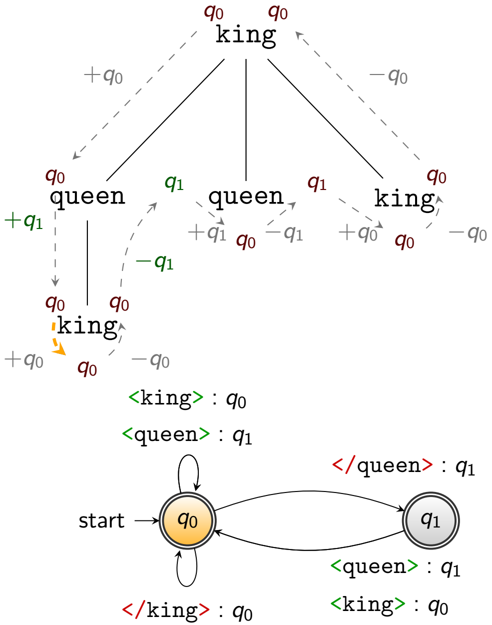


VPA: run

```
<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>
```

stack:

q_0
q_1
q_0

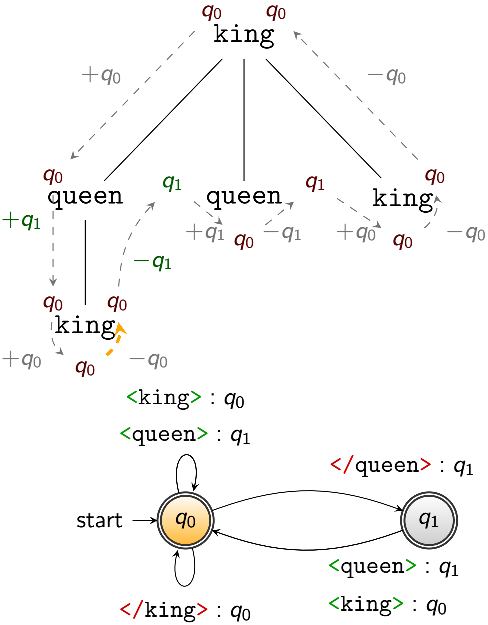


VPA: run

```
<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>
```

stack:

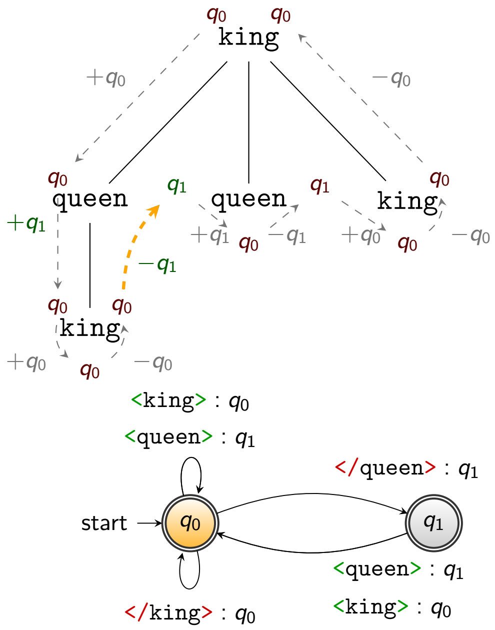
q_0
q_1
q_0



VPA: run

```
<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>
```

stack: q_0 q_1

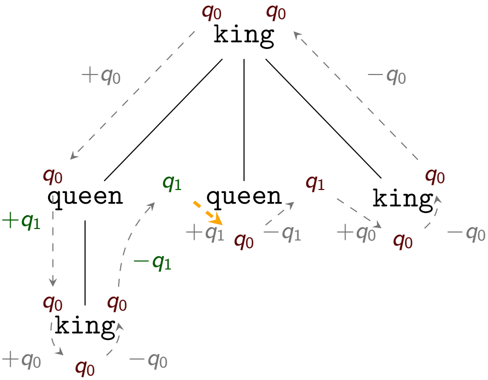


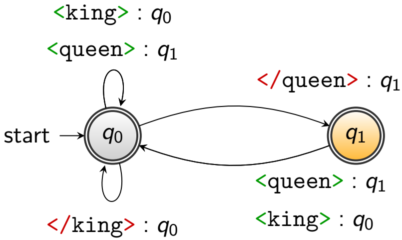
VPA: run

<king>
<queen>
<king>
</king>
</queen>
<queen>
</queen>
<king>
</king>
</king>

stack:

q_0
q_1

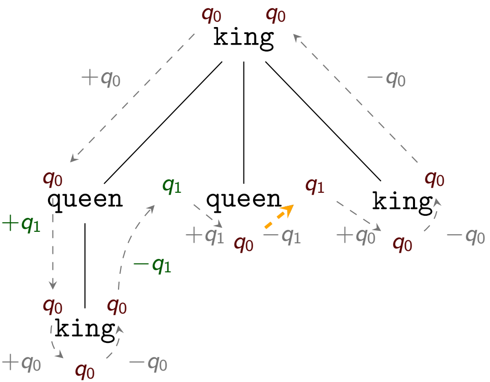


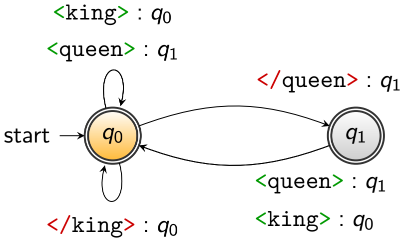


VPA: run

<king>
<queen>
<king>
</king>
</queen>
<queen>
</queen>
<king>
</king>
</king>

stack: q_0 q_1

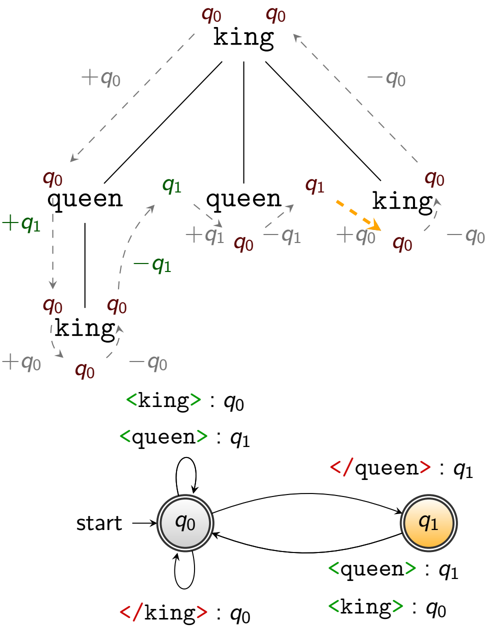




VPA: run

<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>

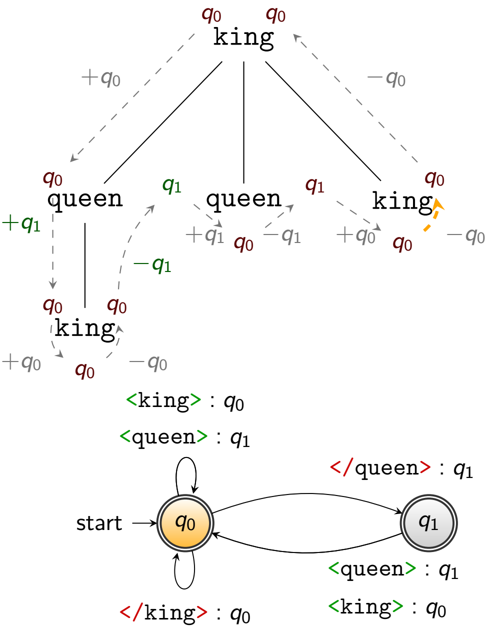
stack: q_0 q_0



VPA: run

<king>
<queen>
<king>
</king>
</queen>
<queen>
</queen>
<king>
</king>
</king>

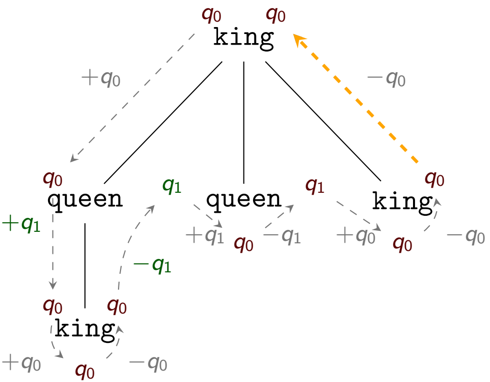
stack: q_0 q_0

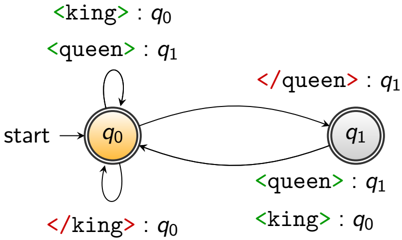


VPA: run

<king>
<queen>
<king>
</king>
</queen>
<queen>
</queen>
<king>
</king>
</king>

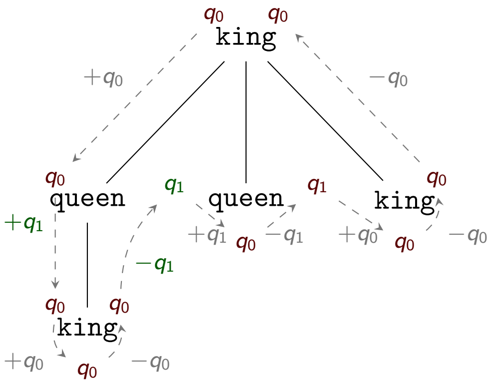
stack: q_0

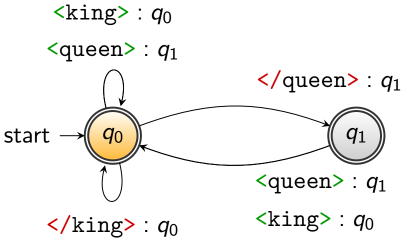




VPA: run

```
<king>
  <queen>
    <king>
    </king>
  </queen>
  <queen>
  </queen>
  <king>
  </king>
</king>
```





Language $L(\mathcal{A})$ = hedges in which all rightmost children are labeled king.

Along the path: detailed bounds for VPAs

Theorem (VPA emptiness)

One can decide emptiness of $L(\mathcal{A})$ in $O(|\Delta| \times |Q| + |Q|^3)$.

Theorem (VPA evaluation (depending on strategy))

- $O(|\mathcal{A}|^2 \times 2^{2Q^2} + |t|)$,
- $O((|\Delta| \times |Q| + |Q|^3) \times |t|)$,

Tight bounds for the pumping lemma

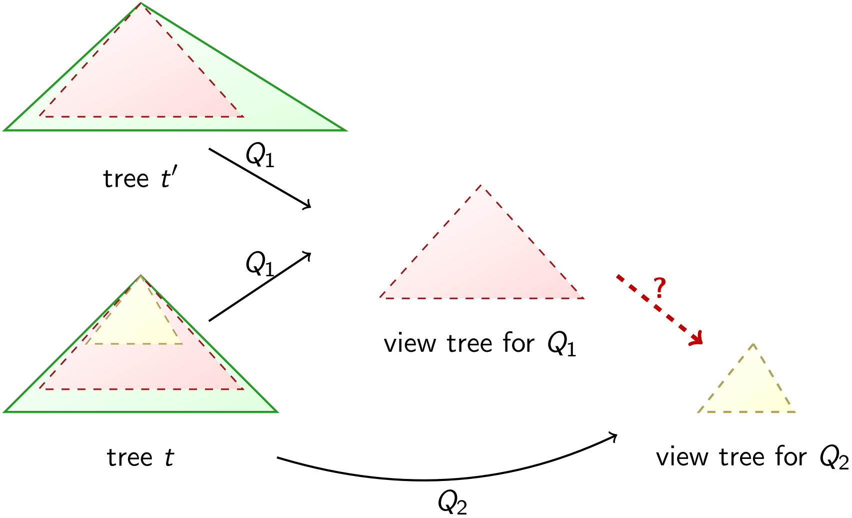
Theorem

There is a family of VPAs $\mathcal{A}_n$ with n states and stack symbols such that the smallest tree in $L(\mathcal{A}_n)$ has size $2^{\Omega(n^2)}$.

Outline

Problem(s) statement

Q_1 determines Q_2 iff $\forall t, t'$ $Q_1(t) = Q_1(t')$ implies $Q_2(t) = Q_2(t')$?



Determinacy by example

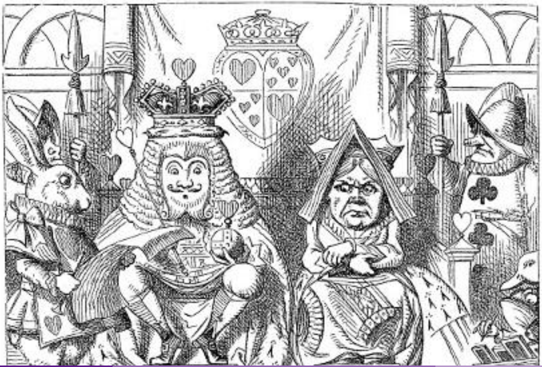
$\Downarrow^*$::king $\cup$ $\Downarrow^*$::king/$\Downarrow^*$::duke determines $\Downarrow^*$::king/$\Downarrow$::duke
easy: simply select $\Downarrow^*$::king/$\Downarrow$::duke

$\Downarrow^*$::king/$\Downarrow^*$::queen $\cup$ $\Downarrow^*$::queen/$\Downarrow^*$::king $\cup$ $\Downarrow^*$::duke determines
$\Downarrow^*$::duke[$\Uparrow^*$::queen and $\Uparrow^*$::king]:
select $\Downarrow^*$::duke[$\Uparrow^*$::queen] $\cup$ $\Downarrow^*$::duke[$\Uparrow^*$::king].

Determinacy by example

$\Downarrow^*$::king $\cup \Downarrow^*$::king/$\Downarrow^*$::duke determines $\Downarrow^*$::king/$\Downarrow$::duke
easy: simply select $\Downarrow^*$::king/$\Downarrow$::duke

$\Downarrow^*$::king/$\Downarrow^*$::queen $\cup \Downarrow^*$::queen/$\Downarrow^*$::king $\cup \Downarrow^*$::duke determines
$\Downarrow^*$::duke[$\Uparrow^*$::queen and $\Uparrow^*$::king]:
select $\Downarrow^*$::duke[$\Uparrow^*$::queen] $\cup \Downarrow^*$::duke[$\Uparrow^*$::king].



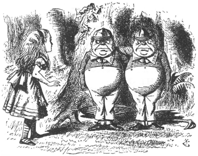
Determinacy by example

$\Downarrow^*$::king $\cup \Downarrow^*$::king/$\Downarrow^*$::duke determines $\Downarrow^*$::king/$\Downarrow$::duke
easy: simply select $\Downarrow^*$::king/$\Downarrow$::duke

$\Downarrow^*$::king/$\Downarrow^*$::queen $\cup \Downarrow^*$::queen/$\Downarrow^*$::king $\cup \Downarrow^*$::duke determines
$\Downarrow^*$::duke[$\Uparrow^*$::queen and $\Uparrow^*$::king]:
select $\Downarrow^*$::duke[$\Uparrow^*$::queen] $\cup \Downarrow^*$::duke[$\Uparrow^*$::king].

$\Downarrow^*$::king[$\Downarrow^*$::queen] does not determine $\Downarrow^*$::king (not even contained)

$\Downarrow^*$::king does not determine $\Downarrow^*$::king[$\Downarrow^*$::queen].



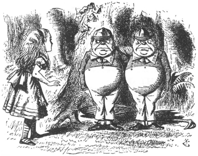
Determinacy by example

$\Downarrow^*::\mathtt{king} \cup \Downarrow^*::\mathtt{king}/\Downarrow^*::\mathtt{duke}$ determines $\Downarrow^*::\mathtt{king}/\Downarrow::\mathtt{duke}$
easy: simply select $\Downarrow^*::\mathtt{king}/\Downarrow::\mathtt{duke}$

$\Downarrow^*::\mathtt{king}/\Downarrow^*::\mathtt{queen} \cup \Downarrow^*::\mathtt{queen}/\Downarrow^*::\mathtt{king} \cup \Downarrow^*::\mathtt{duke}$ determines $\Downarrow^*::\mathtt{duke}[\Uparrow^*::\mathtt{queen}\ \mathtt{and}\ \Uparrow^*::\mathtt{king}]$:
select $\Downarrow^*::\mathtt{duke}[\Uparrow^*::\mathtt{queen}] \cup \Downarrow^*::\mathtt{duke}[\Uparrow^*::\mathtt{king}]$.

$\Downarrow^*::\mathtt{king}[\Downarrow^*::\mathtt{queen}]$ does not determine $\Downarrow^*::\mathtt{king}$ (not even contained)

$\Downarrow^*::\mathtt{king}$ does not determine $\Downarrow^*::\mathtt{king}[\Downarrow^*::\mathtt{queen}]$.



Deciding determinacy: undecidability in general

Theorem

In general determinacy is undecidable.

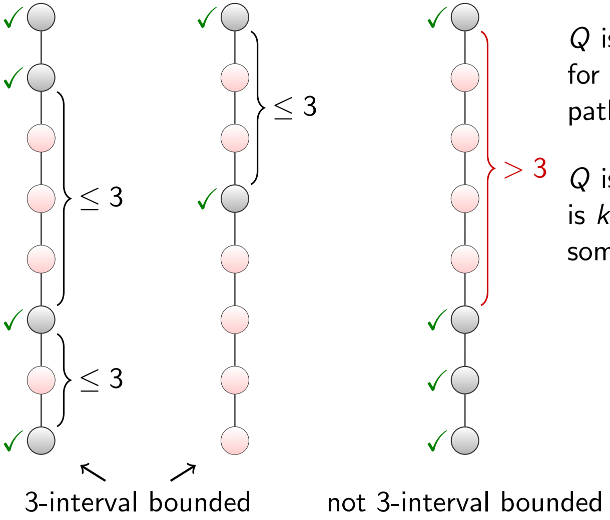
Proof.

Reduction from the emptiness of intersection of two CFG. □

For VPAs and Regular XPath, determinacy is harder than containment:

Tractable restrictions?

(Deciding determinacy) Restriction: IB queries



Q is *k-interval bounded* if for every tree, along every path to the root...

Q is *interval bounded* if it is k-interval bounded for some k.

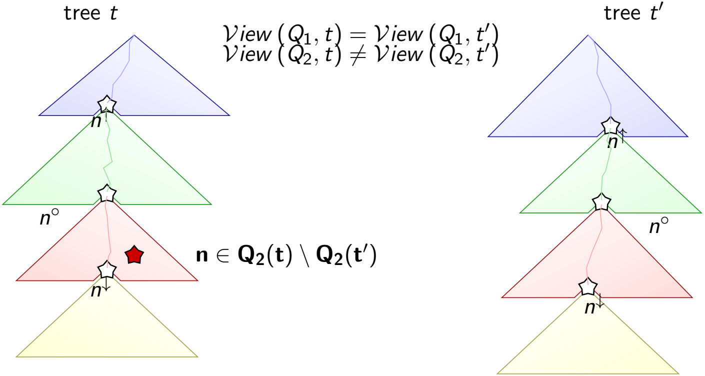
generalizes 1) bounded depth of trees 2) upward-closed queries

Determinacy for interval bounded queries

Can we find two trees t, t' such that $Q_1(t) = Q_1(t')$ but $Q_2(t) \neq Q_2(t')$?

Apply a pumping lemma for VPAs: if there exist two such trees then there exist two "small" such trees (polynomial depth, exponential size).

Double pumping argument in order to preserve the difference for Q_2.



Determinacy for interval bounded queries

Can we find two trees t, t' such that $Q_1(t) = Q_1(t')$ but $Q_2(t) \neq Q_2(t')$?

Apply a pumping lemma for VPAs: if there exist two such trees then there exist two "small" such trees (polynomial depth, exponential size).

Theorem

Determinacy is $\textsc{Pspace}$*-complete for interval bounded VPAs*

Proof.

Upper-bound via pumping: guess the trees step by step, check in $\textsc{Pspace}$.

Lower bound: compressed membership for regular expressions with squares is $\textsc{Pspace}$-hard [Lohrey IJFCS'10]. $\square$

Summary of our results on determinacy

	VPA			$\mathcal{X}Reg$		
Schema	non-rec	IB	gen	non-rec	IB	gen
containmt.	PTIME	PTIME	PTIME	PSPACE-c	EXPTIME-c	EXPTIME-c
determ.	PSPACE-c [1]	PSPACE-c [2]	undec	PSPACE-c	EXPTIME-c	undec

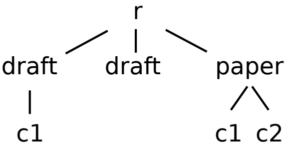
[1] polynomial when the depth of the DTD is bounded by a *fixed* integer k.
[2] polynomial when the constant for interval boundedness is a *fixed* integer k.

Figure: Containment and Determinacy in a nutshell.

- ⋆ *Translating Regular XPath to Automata [Calvanese et al. DBPL'09]*
- ⋆ *Pumping Lemma on VPAs*
- ⋆ *Transducers functionality [Gurari Ibarra JCSS'81, MST'83]*
- ⋆ *Language Theory (hardness results on CFG) [Szymanski Williams FOCS'73, Lohrey IJFCS'10...]*

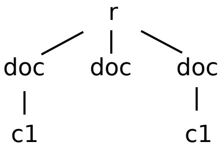
Outline

Problem 2: the view update problem



View

"hide all c2,
rename draft and papers into docs"



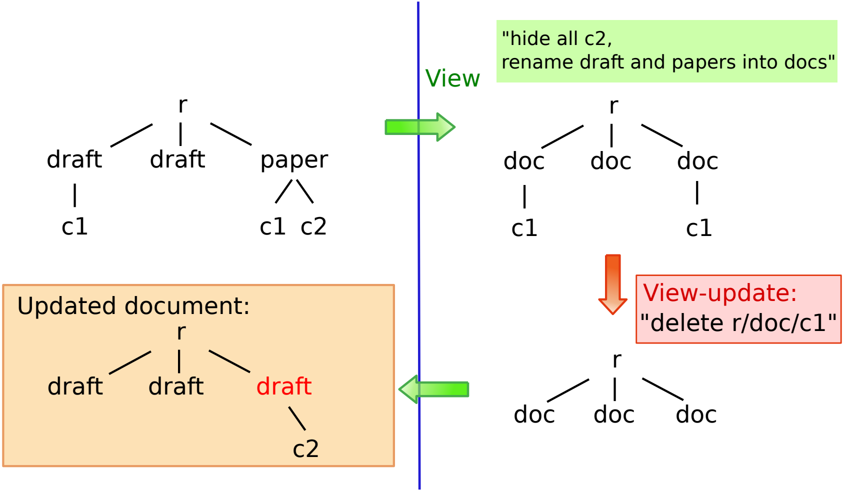
Schema:

r	⟶	(draft | paper)*
draft	⟶	c1? | c2?
paper	⟶	c1,c2

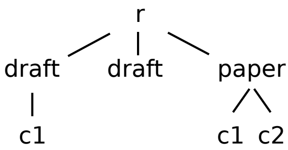
View-Schema:

r	⟶	doc*
doc	⟶	c1?

Problem 2: the view update problem



Problem 2: the view update problem



View

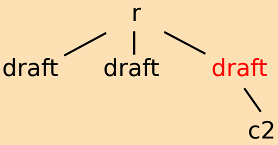
"hide all c2,
rename draft, papers into docs"

Translation of the view update

"delete r/draft/c1,
delete r/paper/c1,
for $p in r/paper return rename node $p as draft"

View-update:
"delete r/doc/c1"

Updated document:



The *View-Update* pb

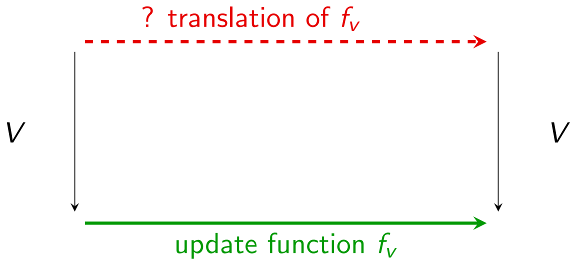


Figure: View update propagation: a synopsis.

The *View-Update* pb with set of authorized updates U_s

For instance $U_s =$ all updates that do not modify `file` nodes

$$\begin{tikzcd} {} \arrow[r, dashed, "?\ \text{translation of } f_v \subseteq U_s"] \arrow[d, "V"'] & {} \arrow[d, "V"] \\ {} \arrow[r, "\text{update function } f_v"'] & {} \end{tikzcd}$$

Figure: View update propagation: a synopsis.

Contributions

- A notion of equivalence for alignments
- Properties of alignment languages w.r.t. composition and equivalence
- Study of the view update problem for update functions, for two settings:
  1. when all updates (respecting the schema) are authorized
  2. when there are constraints on document updates

Contributions: results

We can in PTIME:

✓ test if a set of updates is a function

✓ test if two functions are equivalent

✓ compute the translation of a view update (without constraints)

With constraints, one cannot decide if an update function can be translated, but we identified a very large 'tractable' fragment for which this problem is EXPTIME-complete.

⋆ *Plandowski's algorithm for testing equivalence of two morphisms on a context-free language [Plandowski ESA'94]*

⋆ *Language theory to prove intractability under constraints (PCP, transducer functionality) [Griffith JACM'68]*

Outline

Motivations

DTDs and XML Schema use regular expressions to define the content of elements. In DTDs, we have standard regular expressions.
In XML Schema regular expressions can use numeric occurrences.

CONSTRAINT: those regular expressions must be *deterministic*.

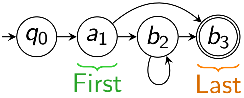
- How can we check if a regular expression is deterministic?
- How can we use determinism to speed up parsing ? (membership pb)

Structure of regular expressions

ab^*b abb^*

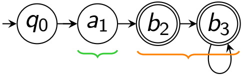
Structure of regular expressions

$a_1 b_2^* b_3$



b_3 *follows* a_1, b_2 *follows* $a_1 \ldots$

$a_1 b_2 b_3^*$

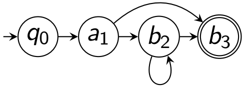


Deterministic regular expressions (a.k.a. one-unambiguous)

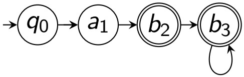
Expression is *non deterministic* if:

$$b_i \qquad a_j \qquad a_k \qquad (j \neq k)$$

$a_1 b_2^* b_3$
$\Rightarrow$ *non deterministic*



$a_1 b_2 b_3^*$
$\Rightarrow$ *deterministic*

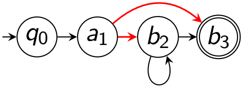


Deterministic regular expressions (a.k.a. one-unambiguous)

Expression is *non deterministic* if:

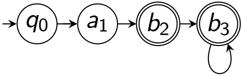
$$b_i \qquad a_j \qquad a_k \qquad (j \neq k)$$

$a_1 b_2^* b_3$
$\Rightarrow$ *non deterministic*



Ambiguity parsing $w = ab$

$a_1 b_2 b_3^*$
$\Rightarrow$ *deterministic*

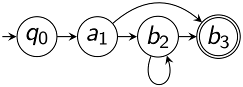


Deterministic regular expressions (a.k.a. one-unambiguous)

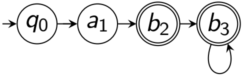
Expression is *non deterministic* if:

$$b_i \quad a_j \quad a_k \qquad (j \neq k)$$

$a_1 b_2^* b_3$
$\Rightarrow$ *non deterministic*



$a_1 b_2 b_3^*$
$\Rightarrow$ *deterministic*



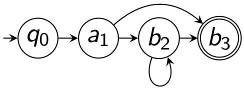
$$e = (a+b)b?(ab)^* \qquad ?$$
$$e' = (ab + ba?)^* \qquad ?$$

Deterministic regular expressions (a.k.a. one-unambiguous)

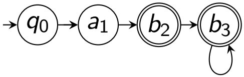
Expression is *non deterministic* if:

$$b_i \quad a_j \quad a_k \qquad (j \neq k)$$

$a_1 b_2^* b_3$
$\Rightarrow$ *non deterministic*



$a_1 b_2 b_3^*$
$\Rightarrow$ *deterministic*



$e = (a + b)b?(ab)^*$ $\quad\Rightarrow$ *deterministic*
$e' = (ab+ba?)^*$ $\quad\Rightarrow$ *non deterministic:* $w = ba$

Problem statement

Testing determinism:

Input: expression e,

Question: is e deterministic?

Scenario: big expression, big alphabet.

Remark:

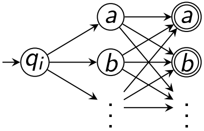
$$
\begin{aligned}
\text{size of } e &= \text{number of nodes in the parse tree} \\
&\simeq \text{number of positions.}
\end{aligned}
$$

Testing determinism

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

Number of transitions of Glushkov can be quadratic:

$e = (a + b + c \dots)(a + b + c \dots)$,
$e' = (a + b + c \dots)^*$,
$e'' = (a?b?c? \dots)$



Testing determinism

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

With numeric occurrences, same complexity $O(|\Sigma| \times |e|)$[Kilpelainen et al IC'07, Inf. Syst'11]

essentially build the Glushkov relations in $O(|\Sigma| \times |e|)$, but adapted with some tricky issues to handle numeric indicators

Testing determinism

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

With numeric occurrences, same complexity $O(|\Sigma| \times |e|)$[Kilpelainen et al IC'07, Inf. Syst'11]

Can we do better?

Testing determinism

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

With numeric occurrences, same complexity $O(|\Sigma| \times |e|)$[Kilpelainen et al IC'07, Inf. Syst'11]

Can we do better?

Theorem

Determinism can be tested in $O(|e|)$ even with numeric occurrences.

Testing Determinism

Do not build the automaton. Instead, work on parse tree and build some pointers+datastructures.

Then identify for each *a* the pairs of *a*-labeled positions which might follow a common position, and check if they do.

$\Longrightarrow$ we reduce the number of pairs to a linear number, and check each pair in constant time.

Testing Determinism

In order to reduce the number of pairs, we use

- ⋆ Several ideas from [Bojańczyk and Parys JACM'11] (data logic)
- ⋆ Glushkov relations [Bruggeman-Klein...] (automata)

Remark:

The structures built for testing determinism for the basis of new algorithms to decide membership in (almost) linear time, together with color ancestor queries and (further use of) *LCA*

- ⋆ LCA [Harel and Tarjan,SICOMP'84] (tree algorithms)
- ⋆ Nearest color ancestor [Muthukrishnan,96] (OO programming)

Conclusion

- PB 1 (Queries): Determinacy and Query rewriting
  ✓undecidable in general, exponential for interval bounded-fragment, polynomial for restricted cases
- PB 2 (Updates): The view update problem
  ✓ polynomial without constraints, undecidable with, but scarcely tractable except for simple cases
- PB 3 (Schema): check if a schema is "correct" w.r.t. W3C specifications
  ✓ linear algorithm

Along the way, we also developed new techniques and proved interesting results for word and tree automata.

Conclusion

Open Questions:

- *Is VPA evaluation quadratic?*
- *Is membership linear for deterministic regular expressions?*
- *Define and take into account quality of the translation for the view update problem.*

Automata theory provides a general framework to solve very diverse problems on XML databases...

...and database applications (esp. big data processing) also raises interesting challenges for automata theory