Bounded Delay and Concurrency for Earliest Query Answering

Olivier Gauwin Joachim Niehren Sophie Tison

INRIA Lille, Mostrare

LATA 2009

Streaming for XML

XML stream

<people>

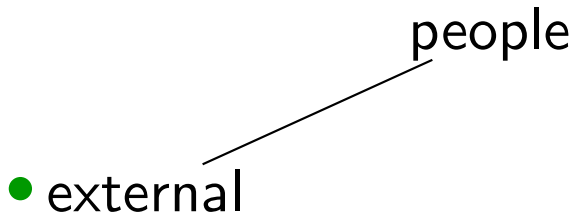
corresponding tree

Streaming for XML

XML stream

<people>
<external>

corresponding tree

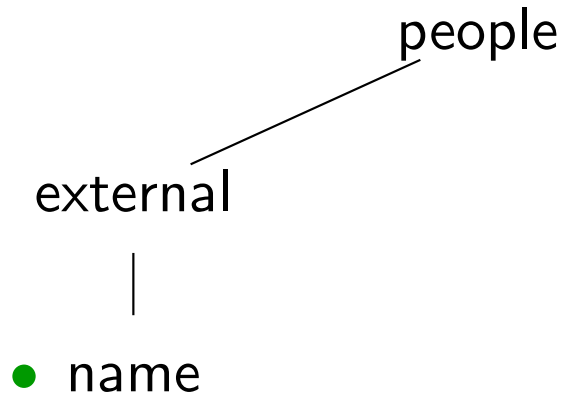


Streaming for XML

XML stream

<people>
<external>
<name>

corresponding tree

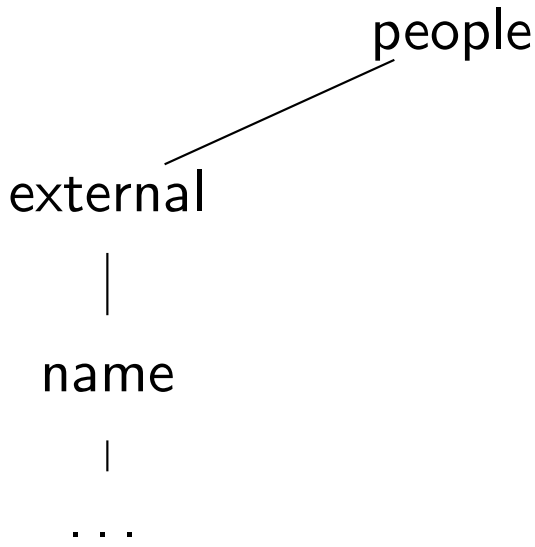


Streaming for XML

XML stream

<people>
<external>
<name>
...

corresponding tree

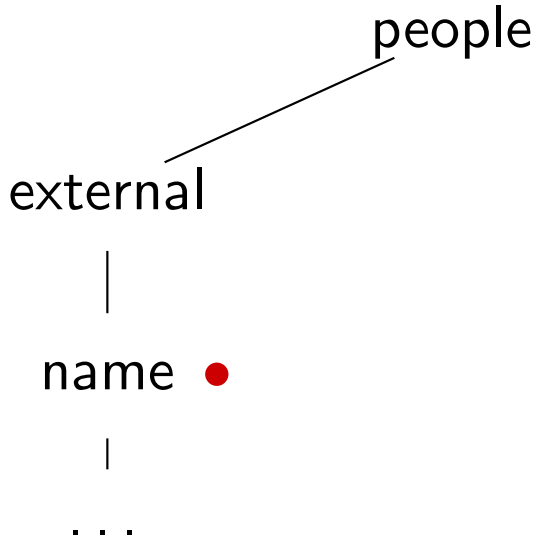


Streaming for XML

XML stream

<people>
<external>
<name>
...
</name>

corresponding tree

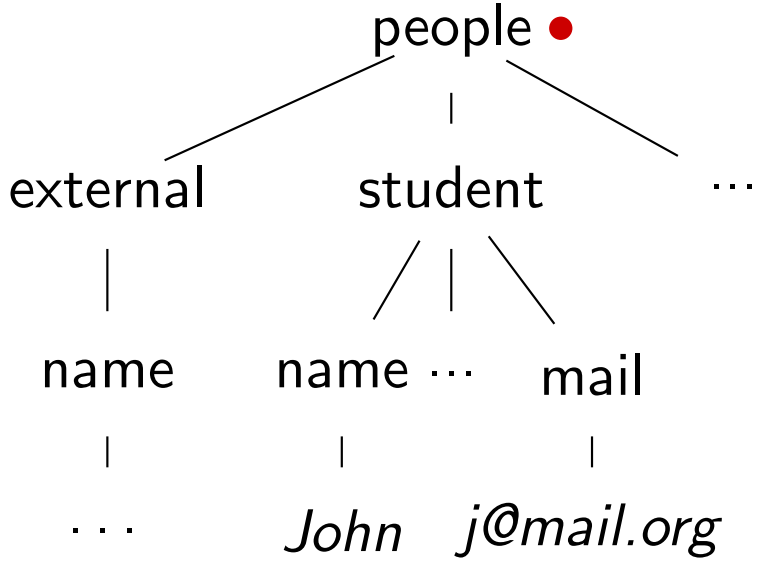


Streaming for XML

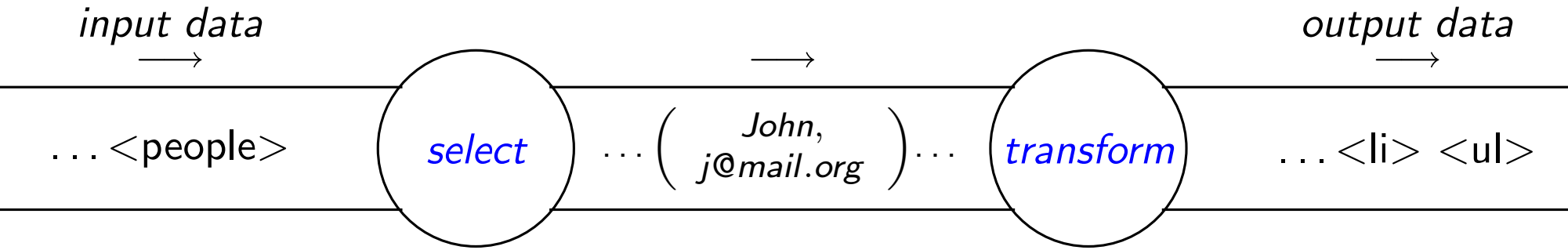
XML stream

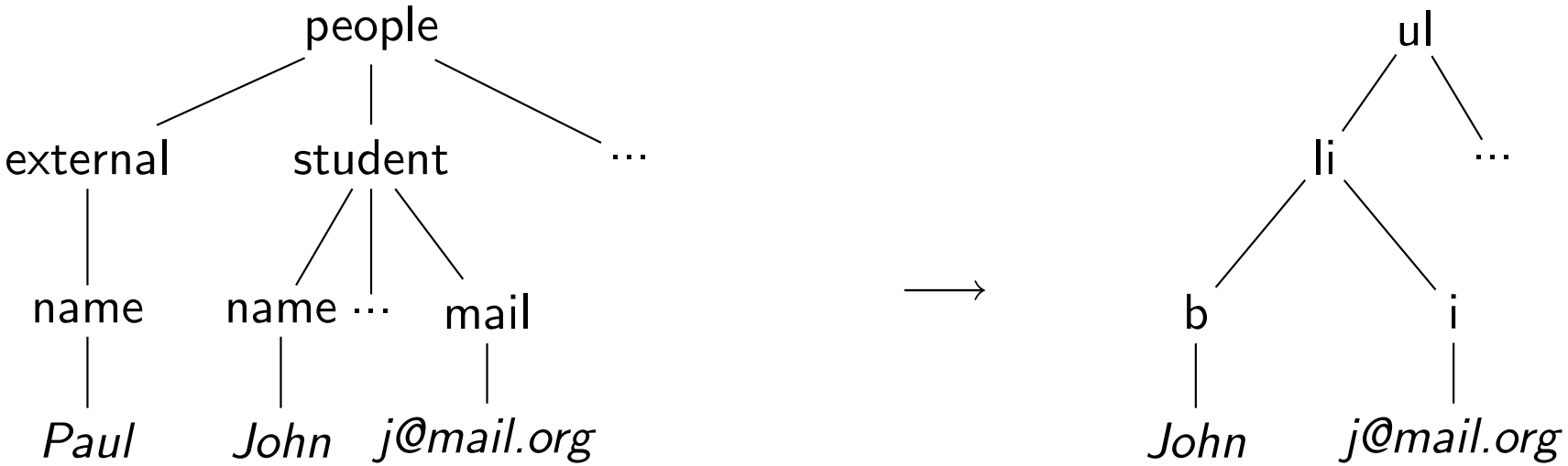
```
<people>
  <external>
    <name>
      ...
    </name>
  </external>
  <student>
    ⋮
  </student>
</people>
```

corresponding tree



Query Answering in Streaming





"Streamability"

Queries

- q1: select all nodes labeled by *a*
  - nothing to store
  - almost nothing to compute (just a test)

"Streamability"

Queries

- q1: select all nodes labeled by *a*
  - nothing to store
  - almost nothing to compute (just a test)
- q2: select all nodes if the last child of the root is labeled by *b*
  - store the whole stream
  - for these candidates, test whether they can be output

"Streamability"

Queries

- q1: select all nodes labeled by *a*
  - nothing to store
  - almost nothing to compute (just a test)
- q2: select all nodes if the last child of the root is labeled by *b*
  - store the whole stream
  - for these candidates, test whether they can be output

Schemas

Schema information might improve the streamability.
In q2, a schema could say that all last siblings are labeled by *b*.

"Streamability"

Queries

- q1: select all nodes labeled by *a*
  - nothing to store
  - almost nothing to compute (just a test)
- q2: select all nodes if the last child of the root is labeled by *b*
  - store the whole stream
  - for these candidates, test whether they can be output

Schemas

Schema information might improve the streamability.
In q2, a schema could say that all last siblings are labeled by *b*.

$\longrightarrow$ Streamable classes of query+schema?

Streamable classes of query+schema

We propose 2 streamable classes:

1. query+schema with bounded delay
2. query+schema with bounded concurrency

Delay

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \ldots & \ldots & \ldots \end{cases}$

...	$<student>$	$<name>$	John	$</name>$

Delay

Example

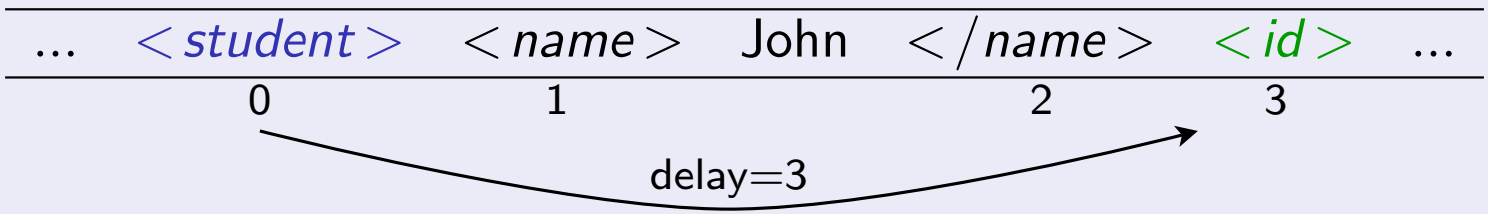
- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \ldots & \ldots & \ldots \end{cases}$

...	$<student>$	$<name>$	John	$</name>$	$<id>$	...

Delay

Example

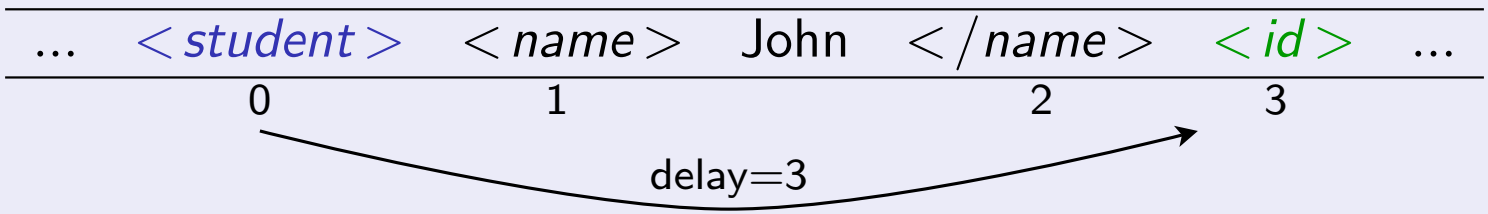
- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \ldots & \ldots & \ldots \end{cases}$



Delay

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \ldots & \ldots & \ldots \end{cases}$



Delay = number of tags between

1. the tag where the candidate becomes complete
2. the first tag where the candidate can be selected

Bounded Delay

A query+schema (q, S) has a bounded delay iff $\exists k \geq 0$ s.t.

$\forall$ XML documents t,

$\forall$ candidates $\tau \in \mathtt{nodes}(t)^n$,

$\tau \in q(t)$ can be decided with delay $\leq k$.

Concurrency

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{address}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \text{address} & \rightarrow & \text{line+} \\ \ldots & \ldots & \ldots \end{cases}$

Concurrency

Example

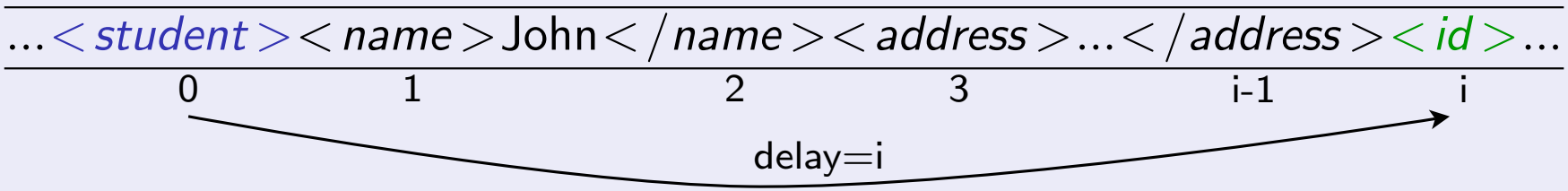
- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{address}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \text{address} & \rightarrow & \text{line+} \\ \ldots & \ldots & \ldots \end{cases}$

$\ldots <\mathit{student}><\mathit{name}>\text{John}</\mathit{name}><\mathit{address}>\ldots</\mathit{address}><\mathit{id}>\ldots$

Concurrency

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{address}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \text{address} & \rightarrow & \text{line}+ \\ \ldots & \ldots & \ldots \end{cases}$



Concurrency

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{address}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \text{address} & \rightarrow & \text{line+} \\ \ldots & \ldots & \ldots \end{cases}$

$\ldots <\textit{student}><\textit{name}>\text{John}</\textit{name}><\textit{address}>\ldots</\textit{address}><\textit{id}>\ldots$

$\rightarrow$ not bounded delay

Concurrency

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{address}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \text{address} & \rightarrow & \text{line+} \\ \ldots & \ldots & \ldots \end{cases}$

$\ldots <student><name>\text{John}</name><address>\ldots</address><id>\ldots$

$\rightarrow$ not bounded delay

$\rightarrow$ but bounded concurrency: only 1 student to be stored at a time

Concurrency

Example

- query = select students that have an id
- schema = $\begin{cases} \text{student} & \rightarrow & (\text{name}, \text{address}, \text{id?}) \\ \text{name} & \rightarrow & \#\text{PCDATA} \\ \text{address} & \rightarrow & \text{line}+ \\ \ldots & \ldots & \ldots \end{cases}$

$\ldots <student> <name> \text{John} </name> <address> \ldots </address> <id> \ldots$

$\rightarrow$ not bounded delay

$\rightarrow$ but bounded concurrency: only 1 student to be stored at a time

Concurrency

= maximal number of candidates to be stored simultaneously

= maximal number of simultaneous candidates s.t.
selection or failure cannot be decided

Bounded Concurrency

A query+schema (q, S) has a bounded concurrency iff $\exists k \geq 0$ s.t.

$\forall$ valid XML documents t,

$\forall$ tags e of t,

$|\{\tau \in \mathtt{nodes}(t)^n \mid \tau \in q(t) \text{ cannot be decided at } e\}| \leq k$.

Bounded Delay vs Bounded Concurrency

- Bounded Concurrency $\not\Rightarrow$ Bounded Delay
  - cf example with: address $\rightarrow$ line+

Bounded Delay vs Bounded Concurrency

- Bounded Concurrency $\not\Rightarrow$ Bounded Delay
  - cf example with: address $\rightarrow$ line$+$
- Bounded Delay $\not\Rightarrow$ Bounded Concurrency
  - because delay does not count partial tuples

Bounded Delay vs Bounded Concurrency

- Bounded Concurrency $\not\Rightarrow$ Bounded Delay
  - cf example with: address $\longrightarrow$ line+
- Bounded Delay $\not\Rightarrow$ Bounded Concurrency
  - because delay does not count partial tuples

In terms of space complexity:

- k-Bounded Delay: full candidates can be removed after k tags
- k-Bounded Concurrency: at most k candidates at a time

Our goal

Decide bounded delay and concurrency in P-time for queries defined by deterministic automata.

Our goal

Decide bounded delay and concurrency in P-time for queries defined by deterministic automata.

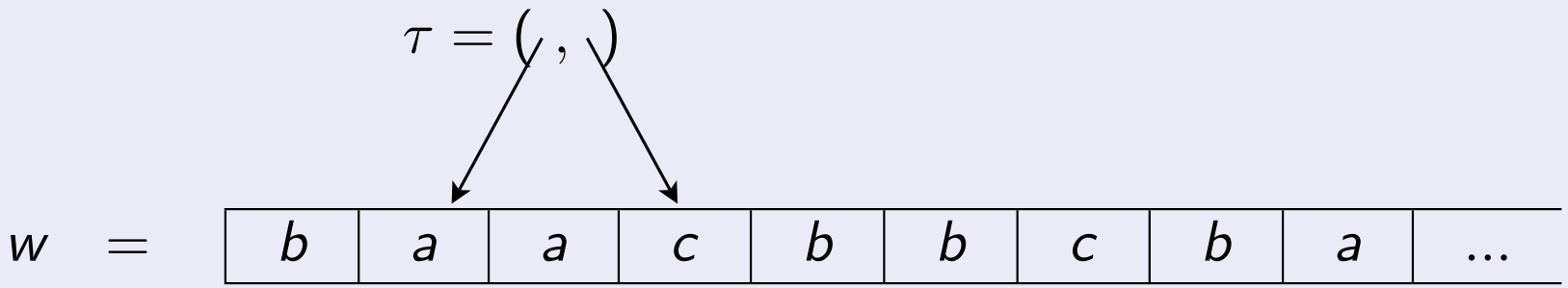
Outline

1. on words
2. on trees

1. Words
   - Bounded Delay
   - Bounded Concurrency
2. Trees
   - Tree Automata
   - Recognizable Relations

Queries defined by Automata

Canonical words

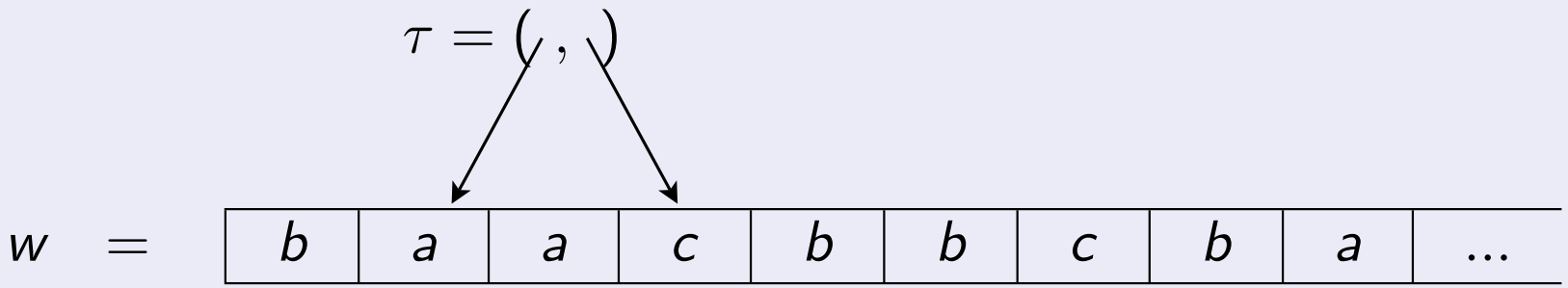


we define $w * \tau \in (\Sigma \times \mathbb{B}^n)^*$:

$$w * \tau = \boxed{b_{00}}\boxed{a_{10}}\boxed{a_{00}}\boxed{c_{01}}\boxed{b_{00}}\boxed{b_{00}}\boxed{c_{00}}\boxed{b_{00}}\boxed{a_{00}}\boxed{\ldots}$$

Queries defined by Automata

Canonical words



we define $w * \tau \in (\Sigma \times \mathbb{B}^n)^*$:

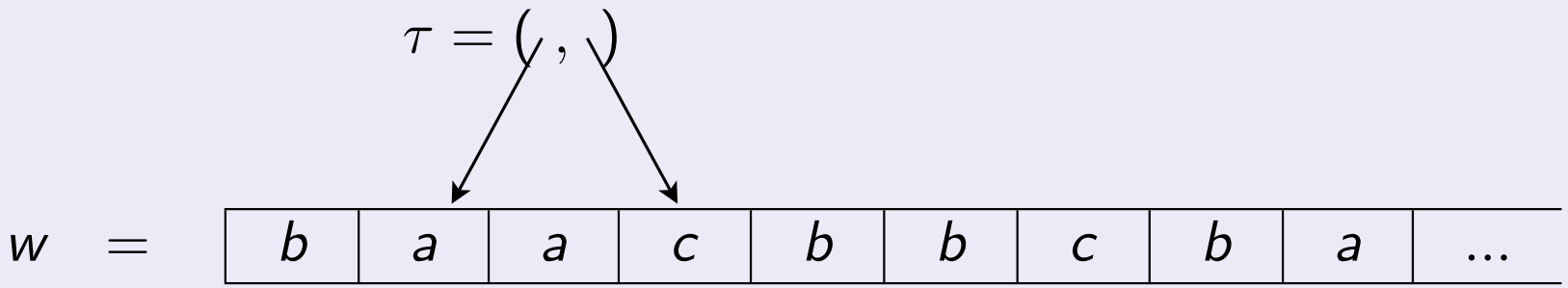
$$w * \tau = \boxed{b_{00}}\boxed{a_{10}}\boxed{a_{00}}\boxed{c_{01}}\boxed{b_{00}}\boxed{b_{00}}\boxed{c_{00}}\boxed{b_{00}}\boxed{a_{00}}\boxed{\ldots}$$

Canonical language of a query q

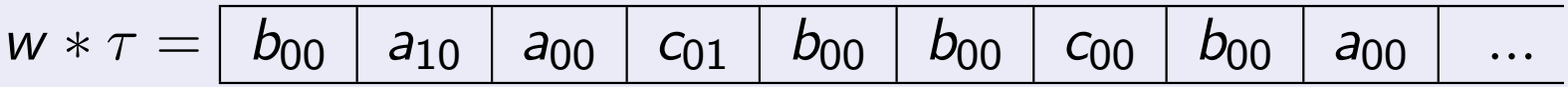
$$\mathtt{Can}_q = \{w * \tau \mid \tau \in q(w)\}$$

Queries defined by Automata

Canonical words



we define $w * \tau \in (\Sigma \times \mathbb{B}^n)^*$:



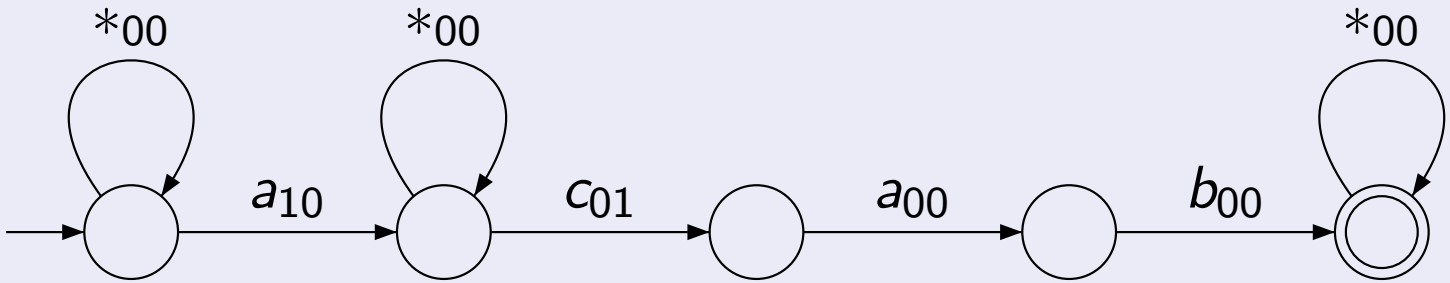
Canonical language of a query q

$$\mathtt{Can}_q = \{w * \tau \mid \tau \in q(w)\}$$

$\longrightarrow$ we define q by the deterministic automaton recognizing $\mathtt{Can}_q$.

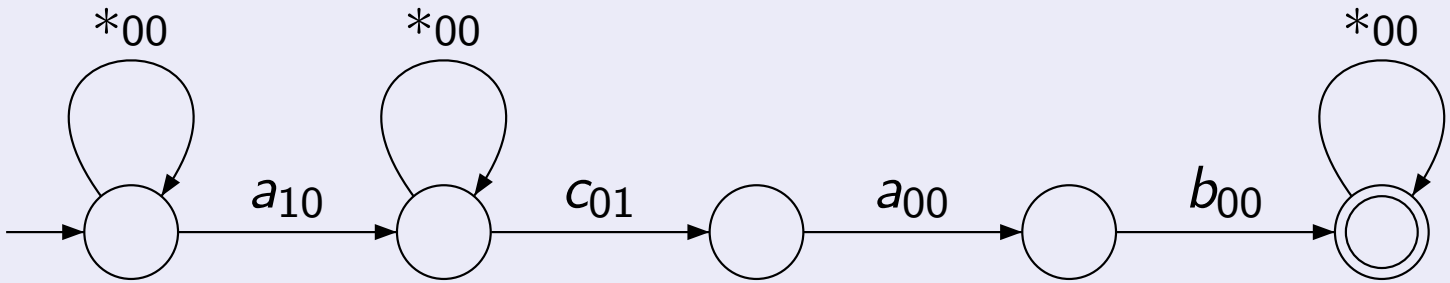
Queries defined by Automata: Example

q = select (a, c) if c is after a and immediately followed by ab



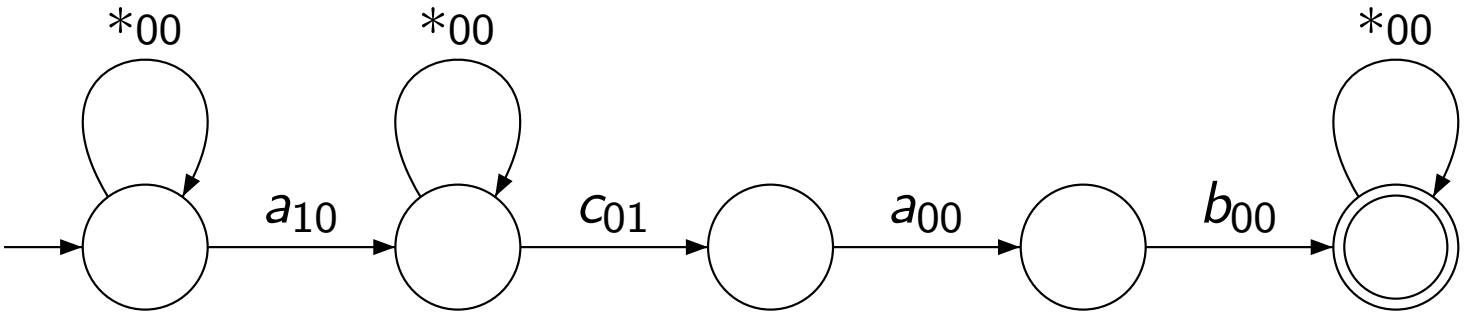
Queries defined by Automata: Example

q = select (a, c) if c is after a and immediately followed by ab

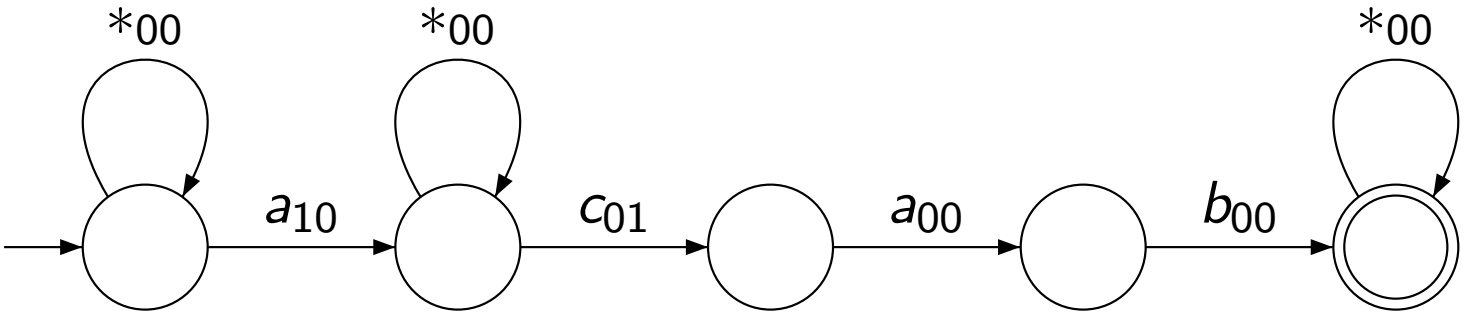


For clarity, we omit the schema in the following.

Query Automaton

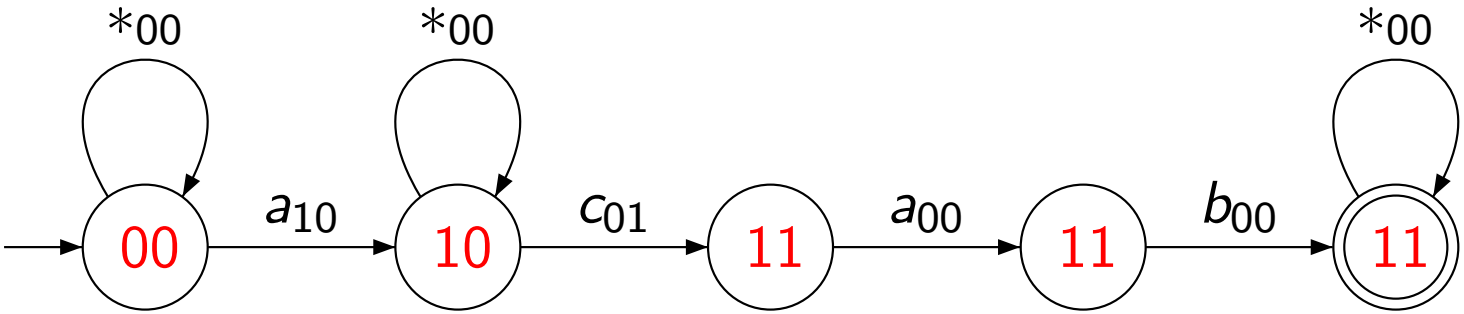


Query Automaton



Make all states of the (det.) automaton accessible + co-accessible.

Query Automaton

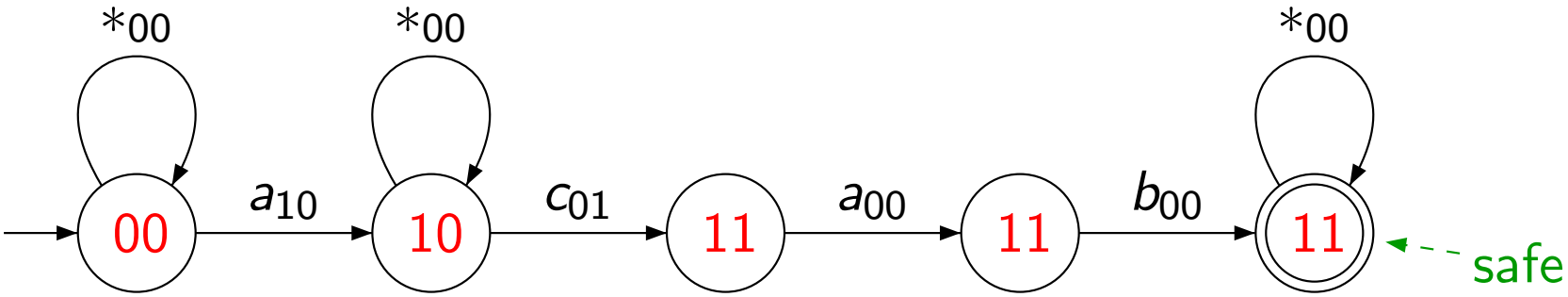


Make all states of the (det.) automaton accessible + co-accessible.

All states have a type

= the bitvector indicating on which components selection has been done.

Query Automaton



Make all states of the (det.) automaton accessible + co-accessible.

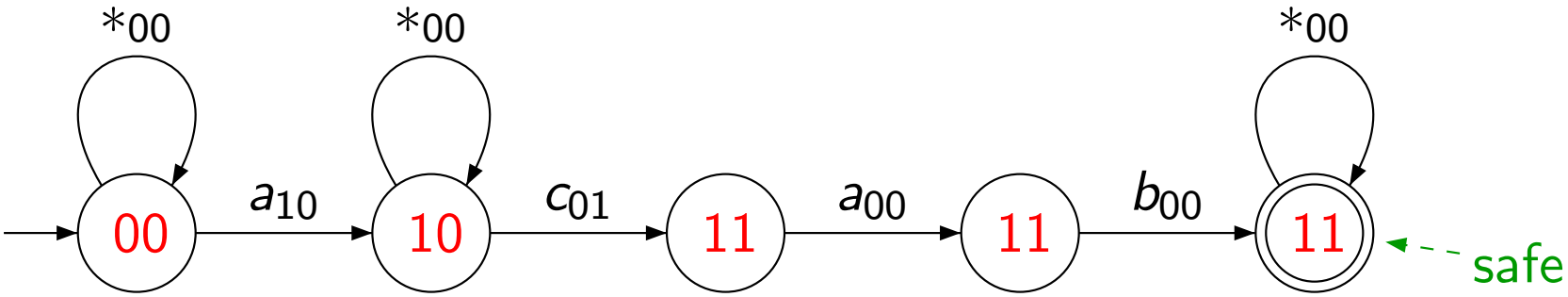
All states have a type

= the bitvector indicating on which components selection has been done.

Safe states p

$type(p) = 1^n \quad \wedge \quad L(A[\texttt{init} = \{p\}]) = (\Sigma \times \{0\}^n)^*$

Query Automaton



Make all states of the (det.) automaton accessible + co-accessible.

All states have a type

= the bitvector indicating on which components selection has been done.

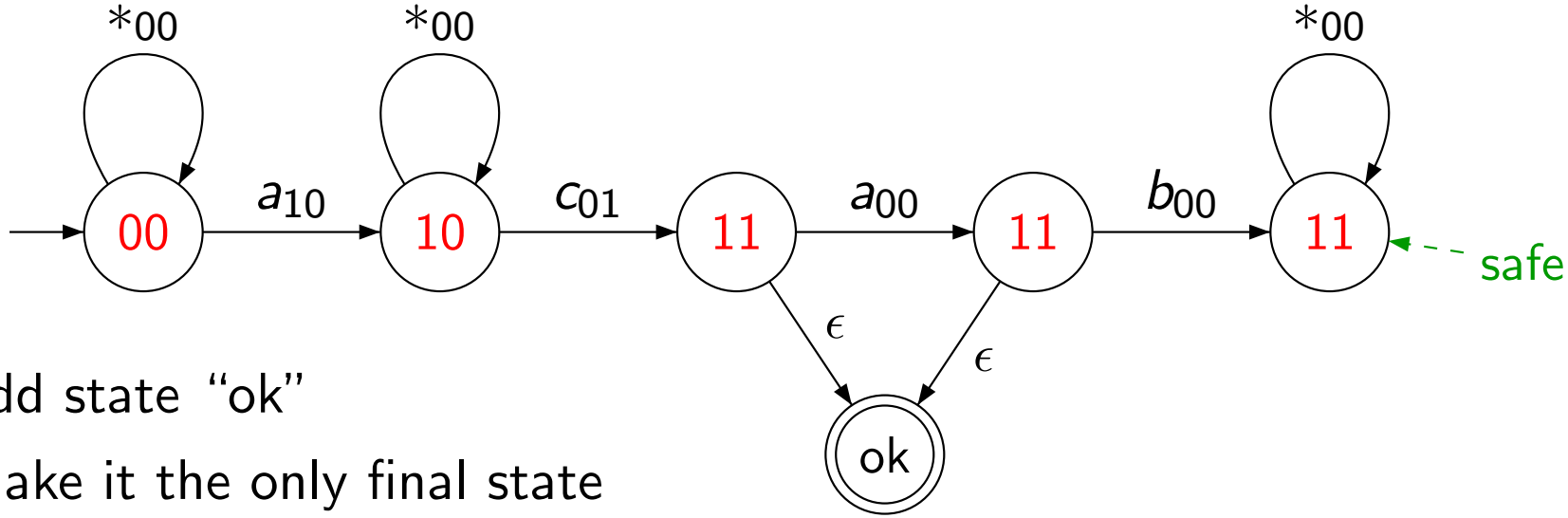
Safe states p

$type(p) = 1^n \quad \wedge \quad L(A[\texttt{init} = \{p\}]) = (\Sigma \times \{0\}^n)^*$

Idea: delay = longest path in unsafe states of type 1^n

Automaton capturing the Delay

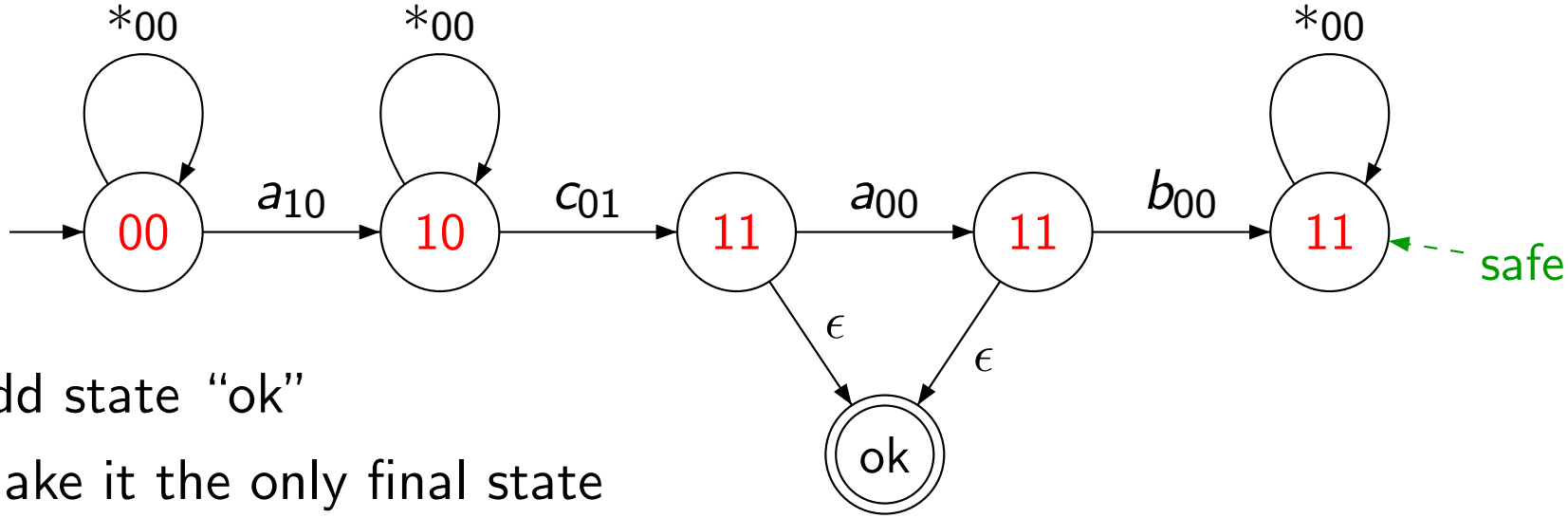
From the automaton A recognizing $\texttt{Can}_q$, we build $D(A)$:



1. add state "ok"
2. make it the only final state
3. add transitions to it, from unsafe states of type 1^n

Automaton capturing the Delay

From the automaton A recognizing $\texttt{Can}_q$, we build $D(A)$:

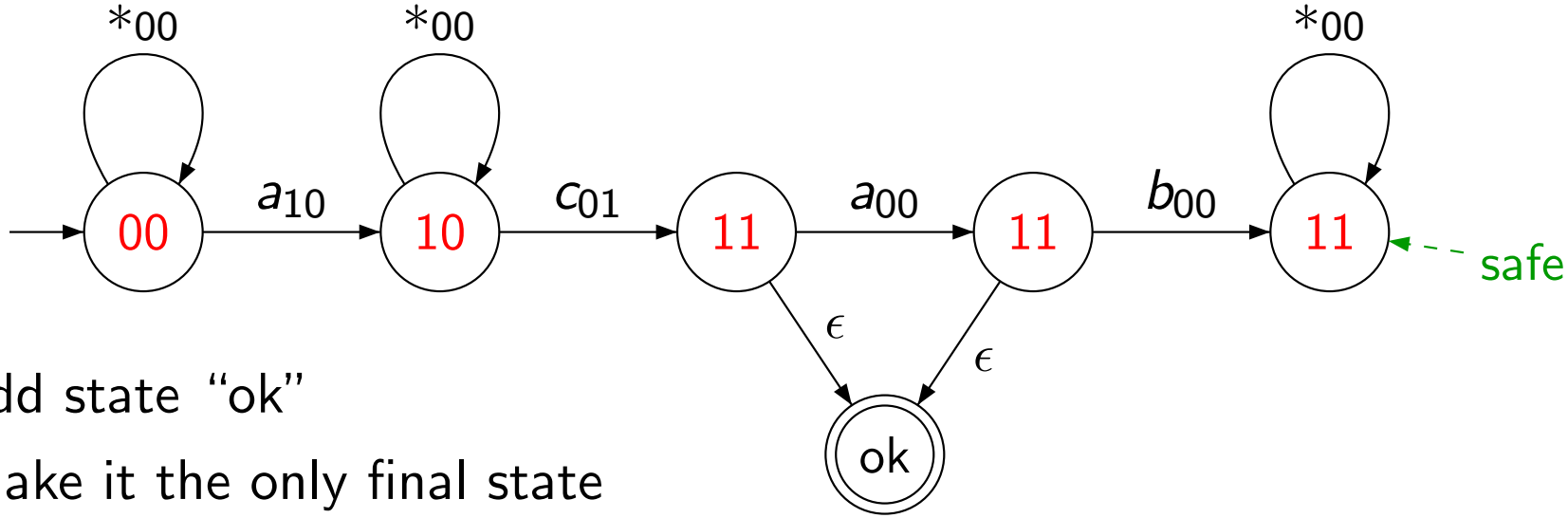


1. add state "ok"
2. make it the only final state
3. add transitions to it, from unsafe states of type 1^n

$$\text{delay}_q(w, \tau) = amb_{D(A)}(w * \tau)$$

Automaton capturing the Delay

From the automaton A recognizing $\texttt{Can}_q$, we build $D(A)$:



1. add state "ok"
2. make it the only final state
3. add transitions to it, from unsafe states of type 1^n

$$\text{delay}_q(w, \tau) = amb_{D(A)}(w * \tau)$$

Theorem

Bounded delay for queries q and schemas S defined by dFAs A, B can be decided in time $O(|A| \cdot |B|)$, and k-bounded delay in P-time.

1. Words
   - Bounded Delay
   - Bounded Concurrency
2. Trees
   - Tree Automata
   - Recognizable Relations

Comparison with Bounded Delay

Delay	Concurrency
complete candidates	complete + partial candidates

Comparison with Bounded Delay

Delay	Concurrency
complete candidates	complete + partial candidates
decide at position e whether $\tau \in q(w)$	decide at position e whether $\tau \in q(w)$ or $\tau \notin q(w)$

Comparison with Bounded Delay

Delay	Concurrency
complete candidates	complete + partial candidates
decide at position e whether $\tau \in q(w)$	decide at position e whether $\tau \in q(w)$ or $\tau \notin q(w)$

but we use the same technique, *i.e.*, we build $C(A)$ s.t.

$$\text{concur}_q(w, e) = amb_{C(A)}(w * e)$$

Theorem

Bounded and k-bounded concurrency for queries and schemas defined by canonical dFAs can be decided in P-time for any fixed $k \geq 0$.

1. Words
   - Bounded Delay
   - Bounded Concurrency

2. Trees
   - Tree Automata
   - Recognizable Relations

Can we do the same for unranked trees?

- ✓ define delay and concurrency for queries on trees
  - $\mathtt{pos}(w) \longrightarrow \mathtt{nodes}(t)$ and $\mathtt{tags}(t)$

Can we do the same for unranked trees?

- ✓ define delay and concurrency for queries on trees
  - $\mathtt{pos}(w) \longrightarrow \mathtt{nodes}(t)$ and $\mathtt{tags}(t)$
- ✓ define queries by tree automata A

Can we do the same for unranked trees?

- ✓ define delay and concurrency for queries on trees
  - $\mathtt{pos}(w) \longrightarrow \mathtt{nodes}(t)$ and $\mathtt{tags}(t)$
- ✓ define queries by tree automata A
- ✓ states of A have a unique type
  - and they can be computed in P-time

Can we do the same for unranked trees?

- ✓ define delay and concurrency for queries on trees
  - $\mathtt{pos}(w) \longrightarrow \mathtt{nodes}(t)$ and $\mathtt{tags}(t)$
- ✓ define queries by tree automata A
- ✓ states of A have a unique type
  - and they can be computed in P-time
- x A does not have safe states for selection/failure
  - we need $O(2^{|A|})$ to compute an equivalent automaton w/ safe states

Can we do the same for unranked trees?

- ✓ define delay and concurrency for queries on trees
  - $\mathtt{pos}(w) \longrightarrow \mathtt{nodes}(t)$ and $\mathtt{tags}(t)$
- ✓ define queries by tree automata A
- ✓ states of A have a unique type
  - and they can be computed in P-time
- x A does not have safe states for selection/failure
  - we need $O(2^{|A|})$ to compute an equivalent automaton w/ safe states

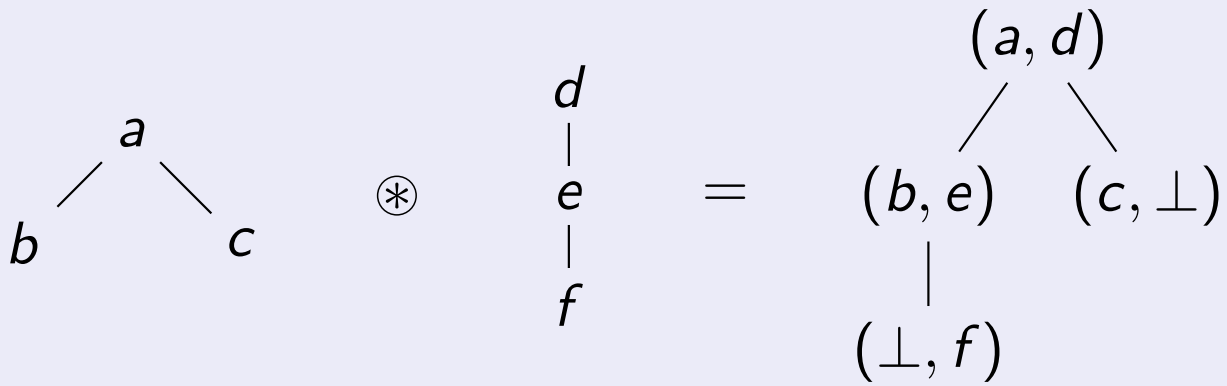
we use another technique: recognizable relations between trees.

words	delay concurrency	} = ambiguity of an automaton
trees	delay concurrency	} = valuedness of a rec. relation between trees

1. Words
   - Bounded Delay
   - Bounded Concurrency
2. Trees
   - Tree Automata
   - Recognizable Relations

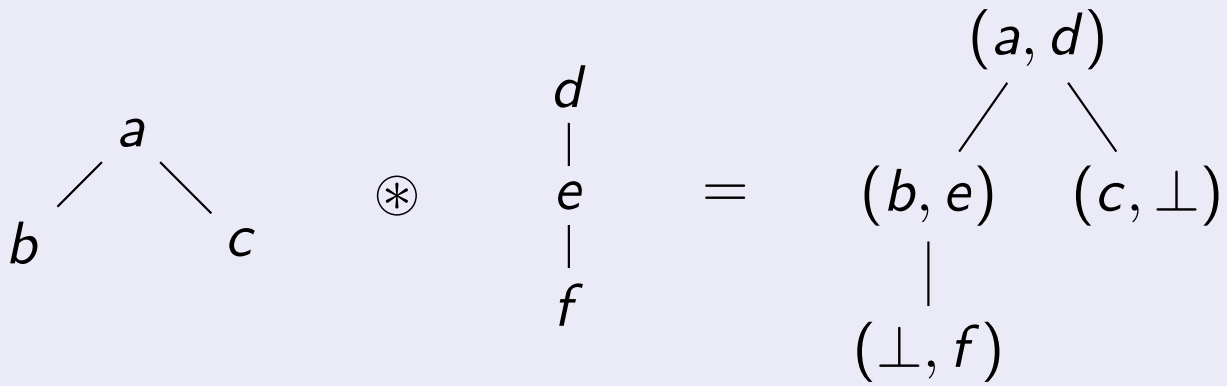
Overlays

Overlays of trees



Overlays

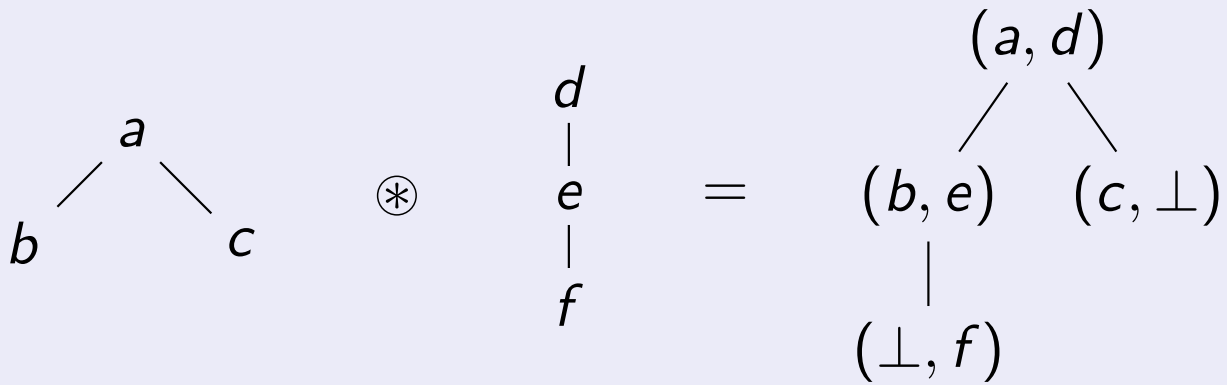
Overlays of trees



$\circledast$ can be extended to k trees.

Overlays

Overlays of trees



$\circledast$ can be extended to k trees.

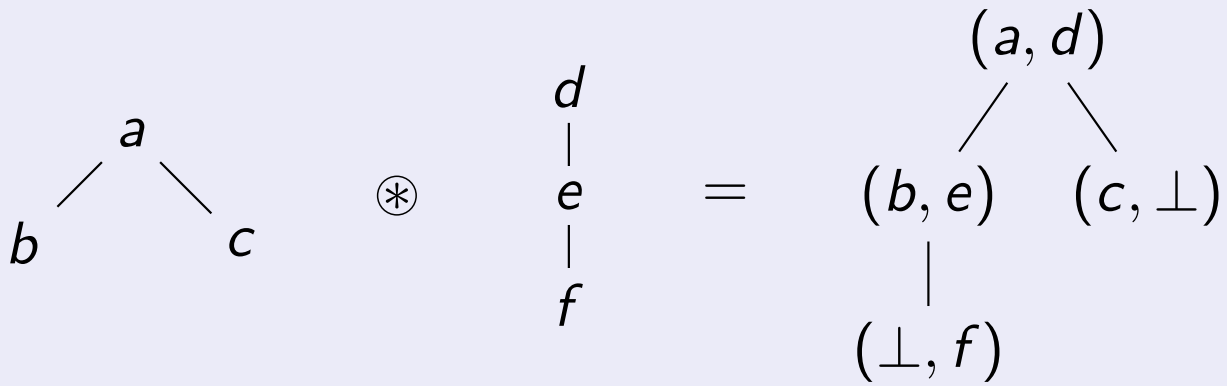
Overlay of a relation R

Let $R \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_k}$ be a relation over unranked trees.

$$\mathrm{ovl}(R) = \{t_1 \circledast \ldots \circledast t_k \mid (t_1, \ldots, t_k) \in R\}$$

Overlays

Overlays of trees



$\circledast$ can be extended to k trees.

Overlay of a relation R

Let $R \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_k}$ be a relation over unranked trees.

$$\mathrm{ovl}(R) = \{t_1 \circledast \ldots \circledast t_k \mid (t_1, \ldots, t_k) \in R\} \subseteq T_{\Sigma_1 \times \ldots \times \Sigma_k}$$

Recognizable Relations between Unranked Trees

$R \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_k}$

Definition

R is recognizable iff $\mathrm{ovl}(R)$ can be recognized by a tree automaton.

Which class of tree automata?

- tree automata on ranked trees
  - by encoding unranked $\longrightarrow$ ranked
- tree automata on unranked trees
  - Stepwise Tree Automata, Streaming Tree Automata...
- or any equivalent class.

Defining Relations by FO-formulas

Closure properties

- Recognizable relations are closed under Boolean operations, projection and cylindrification.
- All corresponding operations on tree automata can be performed in P-time and preserve determinism except for projection.

These properties were known in the ranked case.

$FO[\Re]$

$$\phi \ ::= \ R(X_1, \ldots, X_{\mathrm{ar}(R)}) \ \mid \ \phi \wedge \phi' \ \mid \ \neg\phi \ \mid \ \exists X \in T_\Sigma.\ \phi$$

where $R \in \Re$, $X_1, \ldots, X_{\mathrm{ar}(R)} \in V$, and $\Sigma \in \Omega$.
All formulas are supposed well-typed.

$\Rightarrow$ If $\Re$ is a set of rec. relations, then $R_{\phi(X_1,\ldots,X_m)}$ is also recognizable.

Valuedness of Binary Recognizable Relations

Definition

Let $R \subseteq T_{\Sigma_1} \times T_{\Sigma_2}$. Then $\quad \mathtt{val}(R) = \max_{t_1 \in T_{\Sigma_1}} |\{t_2 \mid (t_1, t_2) \in R\}|$

Valuedness of Binary Recognizable Relations

Definition

Let $R \subseteq T_{\Sigma_1} \times T_{\Sigma_2}$. Then $\mathtt{val}(R) = \max_{t_1 \in T_{\Sigma_1}} |\{t_2 \mid (t_1, t_2) \in R\}|$

Theorem

For every automaton A recognizing R (i.e., $L(A) = ovl(R)$):

1. $val(R) < \infty$ *can be decided in P-time in* $|A|$.
2. $val(R) < k$ *(for a fixed k) can be decided in NP-time in* $|A|$.

This was obtained using results on finite valuedness of tree transducers by Seidl.

Valuedness of Binary Recognizable Relations

Definition

Let $R \subseteq T_{\Sigma_1} \times T_{\Sigma_2}$. Then $\quad \mathtt{val}(R) = \max_{t_1 \in T_{\Sigma_1}} |\{t_2 \mid (t_1, t_2) \in R\}|$

Theorem

For every automaton A recognizing R (i.e., $L(A) = ovl(R)$):

1. $val(R) < \infty$ *can be decided in P-time in* $|A|$.
2. $val(R) < k$ *(for a fixed k) can be decided in NP-time in* $|A|$.

This was obtained using results on finite valuedness of tree transducers by Seidl.

Goal

Define the relation $Delay_q^S(X_{t*\tau}, X_e)$ such that

$$\mathrm{delay}_q^S = \mathtt{val}(Delay_q^S)$$

and $Delay_q^S$ is recognized by an automaton built in P-time.

FO restrictions to remain in P-time

$$\frac{\Re = \{R_1, ..., R_k\} \quad \rightarrow \quad R_\phi \text{ with } \phi \in FO[\Re]}{(A_{R_i})_{R_i \in \Re} \quad \overset{P-time?}{\rightarrow} \quad A_{R_\phi}}$$

Some automata operations are not in P-time (determinization...).

FO restrictions to remain in P-time

$$\frac{\Re = \{R_1, ..., R_k\} \quad \rightarrow \quad R_\phi \text{ with } \phi \in FO[\Re]}{(A_{R_i})_{R_i \in \Re} \quad \overset{P-time?}{\rightarrow} \quad A_{R_\phi}}$$

Some automata operations are not in P-time (determinization...).

$FO_\exists[\Re]$

= formulas of $FO[\Re]$ where all quantifiers are existential and outermost.

$$\frac{\Re = \{R_1, ..., R_k\} \quad \rightarrow \quad R_\phi \text{ with } \phi \in FO_\exists[\Re]}{(A_{R_i})_{R_i \in \Re} \quad \overset{P-time}{\rightarrow} \quad A_{R_\phi}}$$

Defining $Delay_q^S$ in $FO_\exists[\Re]$

$$Delay_q^S(X_t, X_\tau, X_e) \;=\; \exists X'_t \in T_\Sigma.\; S(X_t) \;\wedge\; Bef\&Can_q(X_t, X_\tau, X_e) \\ \wedge\; S(X'_t) \;\wedge\; \mathtt{Same}(X_t, X'_t, X_e) \;\wedge\; \neg Can_q(X'_t, X_\tau)$$

then we turn it into a binary relation $2Delay_q^S(X_{t*\tau}, X_e)$.

Defining $Delay_q^S$ in $FO_\exists[\Re]$

$$Delay_q^S(X_t, X_\tau, X_e) \;=\; \exists X'_t \in T_\Sigma.\; S(X_t) \;\wedge\; Bef\&Can_q(X_t, X_\tau, X_e) \\ \wedge\; S(X'_t) \;\wedge\; \mathtt{Same}(X_t, X'_t, X_e) \;\wedge\; \neg Can_q(X'_t, X_\tau)$$

then we turn it into a binary relation $2Delay_q^S(X_{t*\tau}, X_e)$.

Theorem (Main)

- *Bounded delay is decidable in P-time for n-ary queries and schemas in unranked trees defined by det. tree automata, where n may be variable. Bounded concurrency is decidable in P-time for fixed n.*
- *For fixed k and n, k-bounded delay and concurrency are decidable in NP-time.*

Conclusion

Applications

- for query design
  - progressively compute the delay
  - indicate whether the concurrency is bounded
- for query evaluation
  - adapt data structures and algorithm to known memory needs

Future work

- XPath queries
  - by translation to (det.) tree automata
  - by syntactic analysis