Complexity of Earliest Query Answering with Streaming Tree Automata

Olivier Gauwin Anne-Cécile Caron Joachim Niehren Sophie Tison

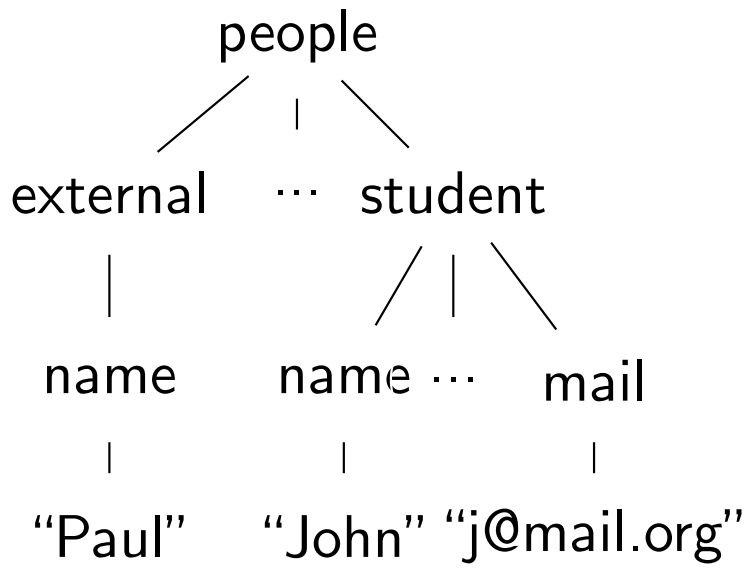
INRIA Lille, Mostrare

9th January 2008

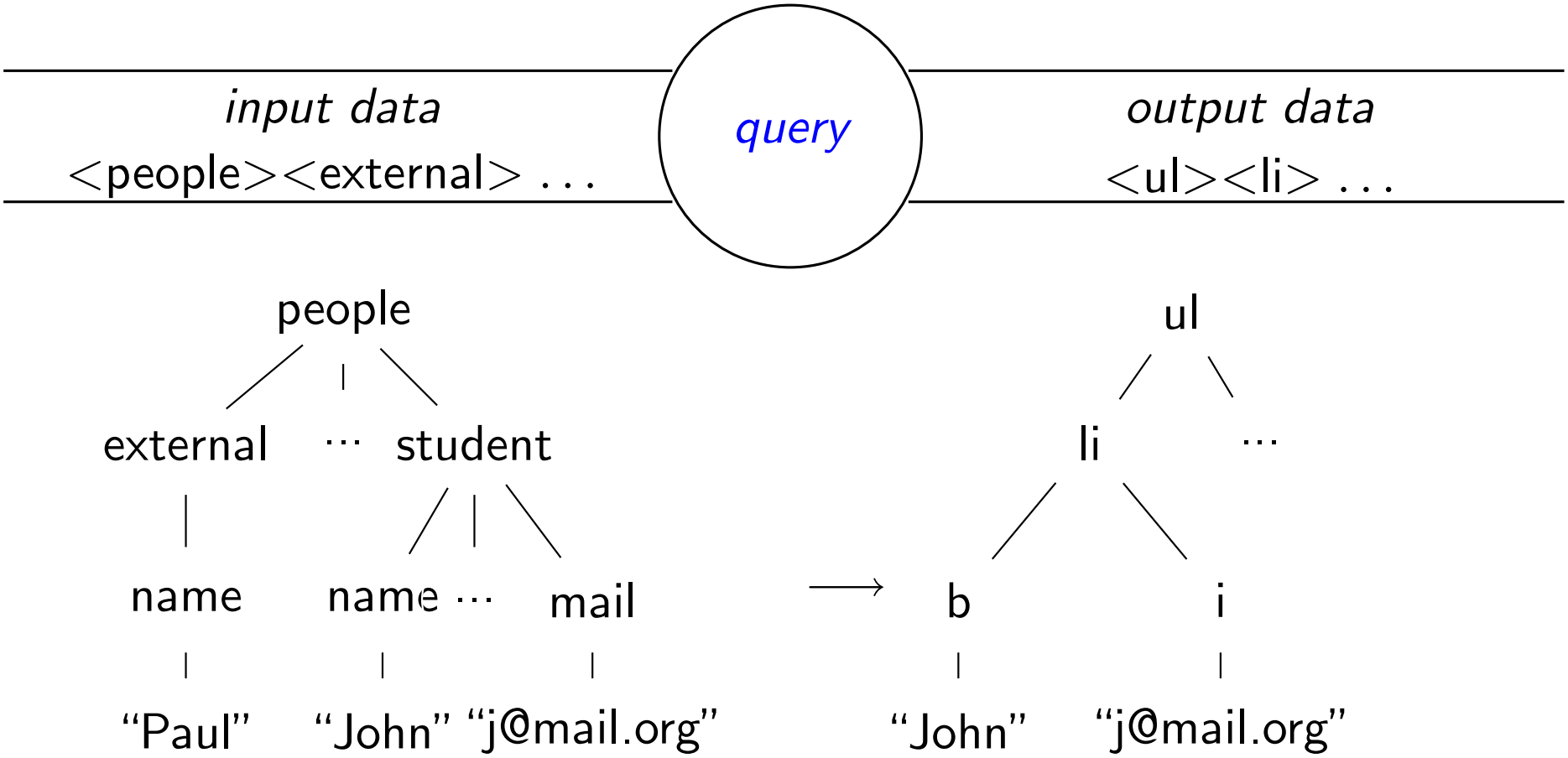
Streaming for XML

Stream: <people><external> . . .

Tree:



Query Answering in Streaming

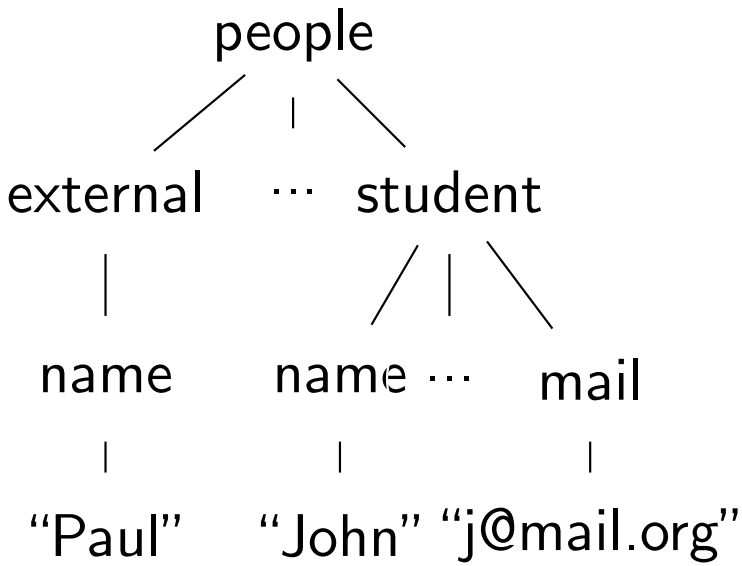


Related work on streaming

- XQuery:
  - Schmidt, Scherzinger & Koch (07)
  - Stark, Fernandez, Michiels & Siméon (07)
- Transducers:
  - Frisch & Nakano (07)
  - Nakano (04)
- XPath:
  - Gupta & Suciu (03)
  - Green, Miklau & Onizuka (03)
  - Bar-Yossef, Fontoura & Josifovski (04)
  - Grohe, Koch & Schweikardt (05)

Earliest Query Answering

XPath query: /people/∗[mail]



Earliest selection:
when <mail> is read

Earliest failure:
when </external> is read

Related work on Earliest Query Answering

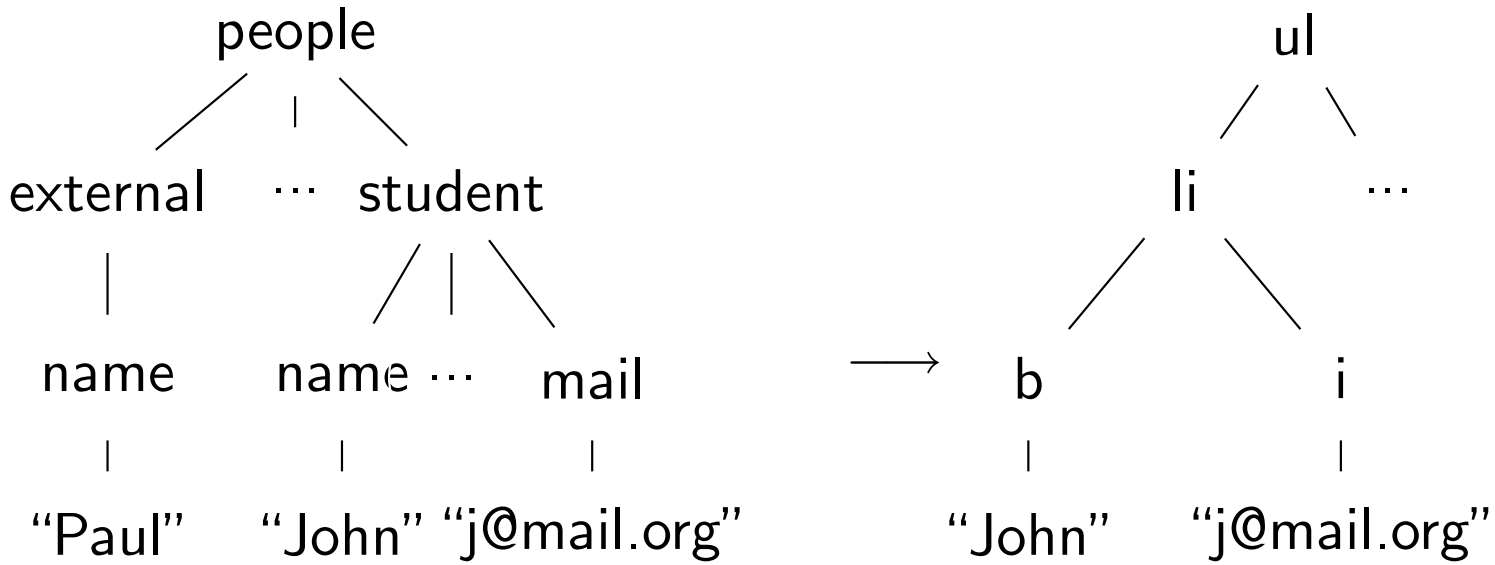
- Automata:
  - Berlea (06, 07): queries defined Pre-Order Automata
- XPath:
  - Olteanu (07): Forward XPath
  - Benedikt & Jeffrey (07): XPath filters with zero lookahead

What is missing?

n-ary queries ($n \geq 0$)

- collect *n*-tuples of nodes, like in XQuery
- $q(t) \subseteq \mathtt{nodes}(t)^n$

Select pairs (name,mail)

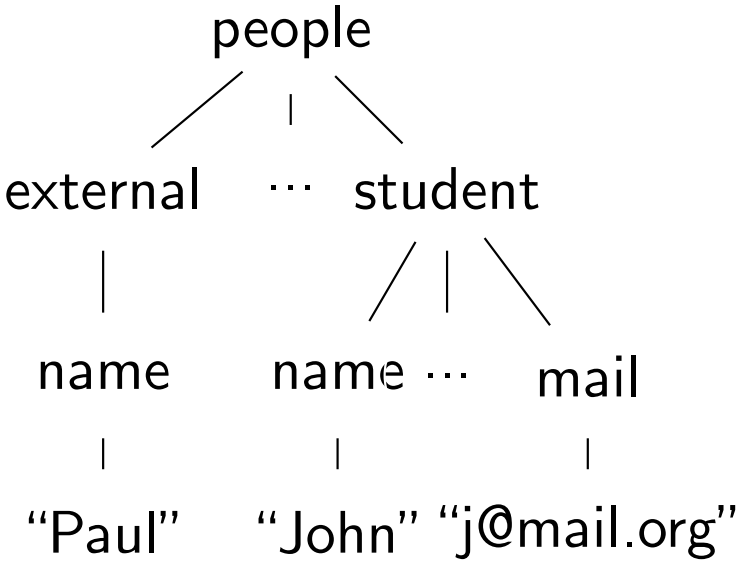


What is missing?

schema: safe for *valid* continuations

- example:

XPath query: $/people/*[mail]$



$$\text{DTD} \begin{cases} \text{external} \longrightarrow \text{name} \\ \text{student} \longrightarrow \text{name} \ \ldots \ \text{mail} \\ \ldots \end{cases}$$

Earliest selection:
when <student> is read

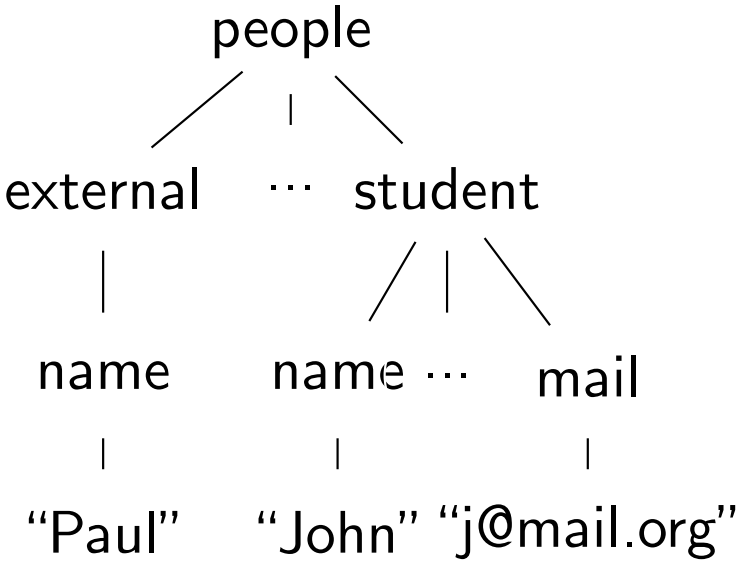
Earliest failure:
when <external> is read

What is missing?

schema: safe for *valid* continuations

- example:

XPath query: $/people/*[mail]$



$$\text{DTD} \begin{cases} \text{external} \longrightarrow \text{name} \\ \text{student} \longrightarrow \text{name} \ \ldots \ \text{mail} \\ \ldots \end{cases}$$

Earliest selection:
when <student> is read

Earliest failure:
when <external> is read

Consequence: MSO queries defined by automata

Fondamental questions

- how to define *earliest selection* and *earliest failure*?

Fondamental questions

- how to define *earliest selection* and *earliest failure*?
- is there an algorithm to compute these earliest positions?

Fondamental questions

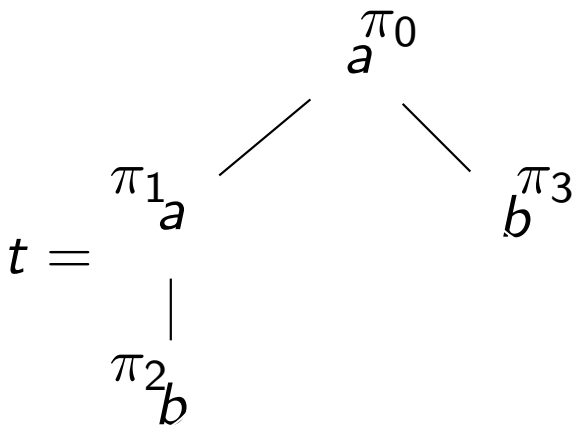
- how to define *earliest selection* and *earliest failure*?
- is there an algorithm to compute these earliest positions?
- how complex is it?

1. Earliest Query Answering
2. Streaming Tree Automata
3. Streaming Algorithm

Notations

Trees and events

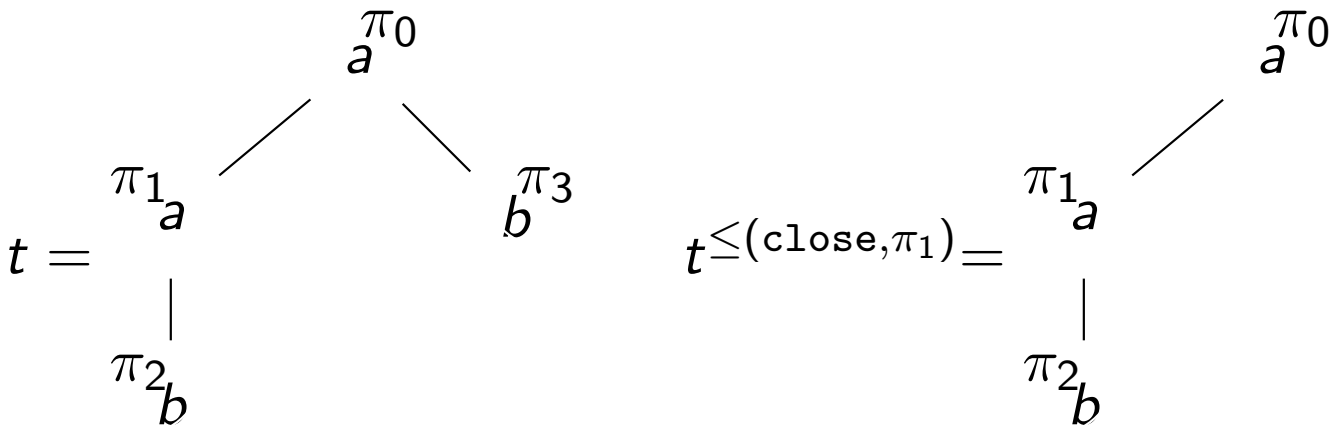
- $\texttt{events}(t) = \{\texttt{start}\} \cup (\{\texttt{open}, \texttt{close}\} \times \texttt{nodes}(t))$



Notations

Trees and events

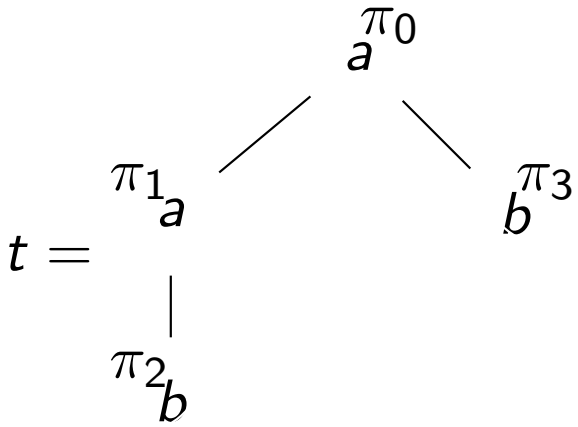
- $\texttt{events}(t) = \{\texttt{start}\} \cup (\{\texttt{open}, \texttt{close}\} \times \texttt{nodes}(t))$
- $t^{\leq e}$: the tree which contains all nodes of t opened before e

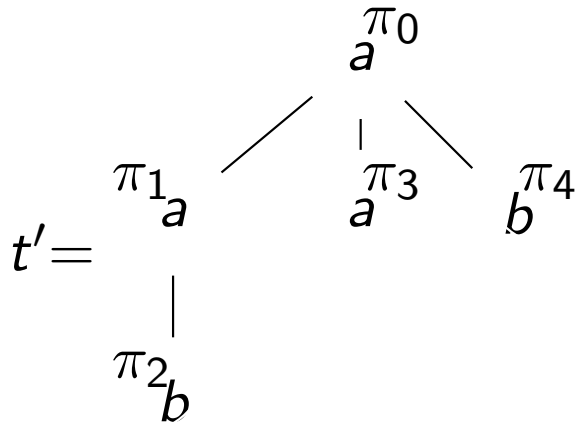


Notations

Trees and events

- $\texttt{events}(t) = \{\texttt{start}\} \cup (\{\texttt{open}, \texttt{close}\} \times \texttt{nodes}(t))$
- $t^{\leq e}$: the tree which contains all nodes of t opened before e
- $\text{equal}_e(t, t') \Leftrightarrow \begin{cases} e \in \texttt{events}(t) \cap \texttt{events}(t') \\ \text{and} \\ t^{\leq e} = t'^{\leq e} \end{cases}$





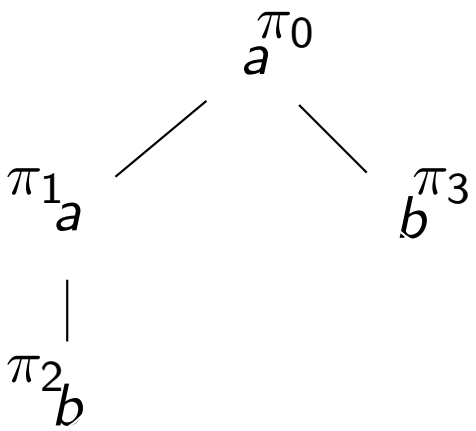
$\text{equal}_{(\texttt{close},\pi_1)}(t, t')$
$\neg\text{equal}_{(\texttt{open},\pi_3)}(t, t')$

Sufficient events for selection

$$(\tau, e) \in \mathtt{sel}_q^S(t) \Leftrightarrow \begin{cases} \tau \in \mathtt{nodes}(t^{\leq e})^n \wedge \\ \forall t' \in S.\ \mathrm{equal}_e(t, t') \Rightarrow \tau \in q(t') \end{cases}$$

Example

q_0 ="select nodes that don't have a next sibling"



$\mathtt{sel}_{q_0}(t)$ contains:

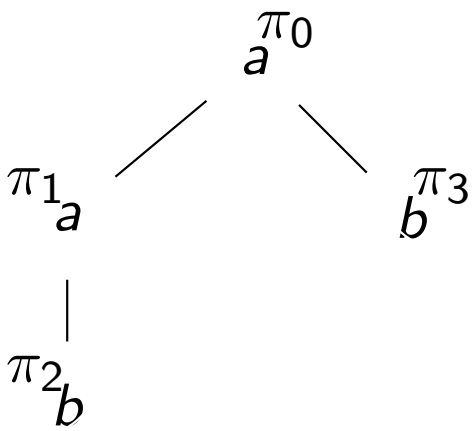
- (π_0, e) for events e following $(\mathtt{open}, \pi_0)$
- (π_2, e) for events e following $(\mathtt{close}, \pi_1)$
- $(\pi_3, (\mathtt{close}, \pi_0))$

Sufficient events for selection

$$(\tau, e) \in \mathtt{sel}_q^S(t) \Leftrightarrow \begin{cases} \tau \in \mathtt{nodes}(t^{\leq e})^n \wedge \\ \forall t' \in S.\ \mathrm{equal}_e(t, t') \Rightarrow \tau \in q(t') \end{cases}$$

Example

q_0 ="select nodes that don't have a next sibling"



$\mathtt{sel}_{q_0}(t)$ contains:

- (π_0, e) for events e following $(\mathtt{open}, \pi_0)$
- (π_2, e) for events e following $(\mathtt{close}, \pi_1)$
- $(\pi_3, (\mathtt{close}, \pi_0))$

consider the DTD: $S_0 = \{a \rightarrow a^* b,\ \ b \rightarrow \epsilon\}$

$\mathtt{sel}_{q_0}^{S_0}(t)$ contains:

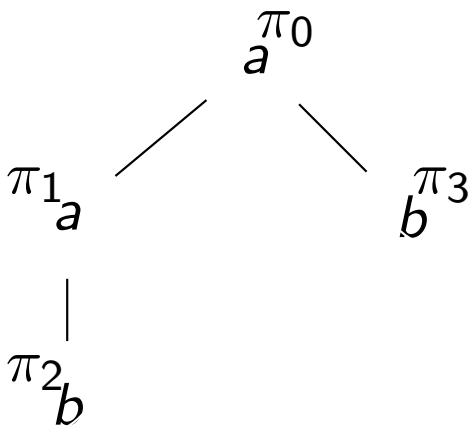
- (π_0, e) for events e following $(\mathtt{open}, \pi_0)$
- (π_2, e) for events e following $(\mathtt{open}, \pi_2)$
- (π_3, e) for events e following $(\mathtt{open}, \pi_3)$

Optimal events for selection

$$(\tau, e) \in \texttt{opt\_sel}_q^S(t) \Leftrightarrow e = \min_{\preceq^t}\{e' \mid (\tau, e') \in \texttt{sel}_q^S(t)\}$$

Example

q_0 ="select nodes that don't have a next sibling"



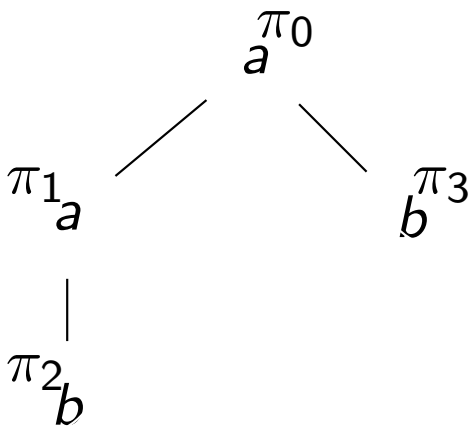
$$\texttt{opt\_sel}_{q_0}(t) = \begin{cases} (\pi_0, (\texttt{open}, \pi_0)), \\ (\pi_2, (\texttt{close}, \pi_1)), \\ (\pi_3, (\texttt{close}, \pi_0)) \end{cases}$$

Optimal events for selection

$$(\tau, e) \in \texttt{opt\_sel}_q^S(t) \Leftrightarrow e = \min_{\preceq^t}\{e' \mid (\tau, e') \in \texttt{sel}_q^S(t)\}$$

Example

q_0 ="select nodes that don't have a next sibling"



$$\texttt{opt\_sel}_{q_0}(t) = \begin{cases} (\pi_0, (\texttt{open}, \pi_0)), \\ (\pi_2, (\texttt{close}, \pi_1)), \\ (\pi_3, (\texttt{close}, \pi_0)) \end{cases}$$

consider the DTD: $S_0 = \{a \to a^* b,\ \ b \to \epsilon\}$

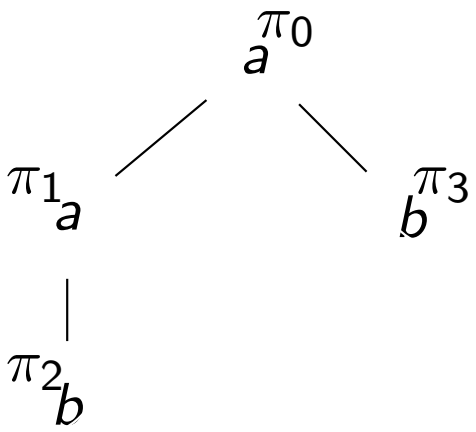
$$\texttt{opt\_sel}_{q_0}^{S_0}(t) = \begin{cases} (\pi_0, (\texttt{open}, \pi_0)), \\ (\pi_2, (\texttt{open}, \pi_2)), \\ (\pi_3, (\texttt{open}, \pi_3)) \end{cases}$$

Optimal events for selection

$$(\tau, e) \in \texttt{opt\_sel}_q^S(t) \Leftrightarrow e = \min_{\preceq^t}\{e' \mid (\tau, e') \in \texttt{sel}_q^S(t)\}$$

Example

q_0 ="select nodes that don't have a next sibling"



$$\texttt{opt\_sel}_{q_0}(t) = \begin{cases} (\pi_0, (\texttt{open}, \pi_0)), \\ (\pi_2, (\texttt{close}, \pi_1)), \\ (\pi_3, (\texttt{close}, \pi_0)) \end{cases}$$

consider the DTD: $S_0 = \{a \to a^* b,\ \ b \to \epsilon\}$

$$\texttt{opt\_sel}_{q_0}^{S_0}(t) = \begin{cases} (\pi_0, (\texttt{open}, \pi_0)), \\ (\pi_2, (\texttt{open}, \pi_2)), \\ (\pi_3, (\texttt{open}, \pi_3)) \end{cases}$$

Similarly, we define sufficient and optimal events for failure.

Complexity of Sufficiency: for MSO queries

Sufficiency problem: Decide whether $(\tau, e) \in \mathtt{sel}_q^S(t)$

Queries defined by MSO formula

The sufficiency problem for MSO-defined queries is decidable but non-elementary.

Queries defined by tree automata

Sufficiency and optimality for queries defined by non-deterministic tree automata are DEXPTIME-hard.

Complexity of Sufficiency: for MSO queries

Sufficiency problem: Decide whether $(\tau, e) \in \mathtt{sel}_q^S(t)$

Queries defined by MSO formula

The sufficiency problem for MSO-defined queries is decidable but non-elementary.

Queries defined by tree automata

Sufficiency and optimality for queries defined by non-deterministic tree automata are DEXPTIME-hard.

- Sufficiency for failure: same complexity (not obvious).

Complexity of Sufficiency: for MSO queries

Sufficiency problem: Decide whether $(\tau, e) \in \mathtt{sel}_q^S(t)$

Queries defined by MSO formula

The sufficiency problem for MSO-defined queries is decidable but non-elementary.

Queries defined by tree automata

Sufficiency and optimality for queries defined by non-deterministic tree automata are DEXPTIME-hard.

- Sufficiency for failure: same complexity (not obvious).
- We proved hardness results for other classes of queries.

Determinism

Wanted: deterministic automata when evaluated in streaming order

Determinism

Wanted: deterministic automata when evaluated in streaming order

- bottom-up / top-down automata: does not fit

Determinism

Wanted: deterministic automata when evaluated in streaming order

- bottom-up / top-down automata: does not fit
- document order: Nested Word Automata
  - problem: queries on events, not nodes

Determinism

Wanted: deterministic automata when evaluated in streaming order

- bottom-up / top-down automata: does not fit
- document order: Nested Word Automata
  - problem: queries on events, not nodes
- we define Streaming Tree Automata

1 Earliest Query Answering

2 Streaming Tree Automata

3 Streaming Algorithm

Streaming Tree Automata

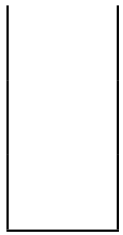
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

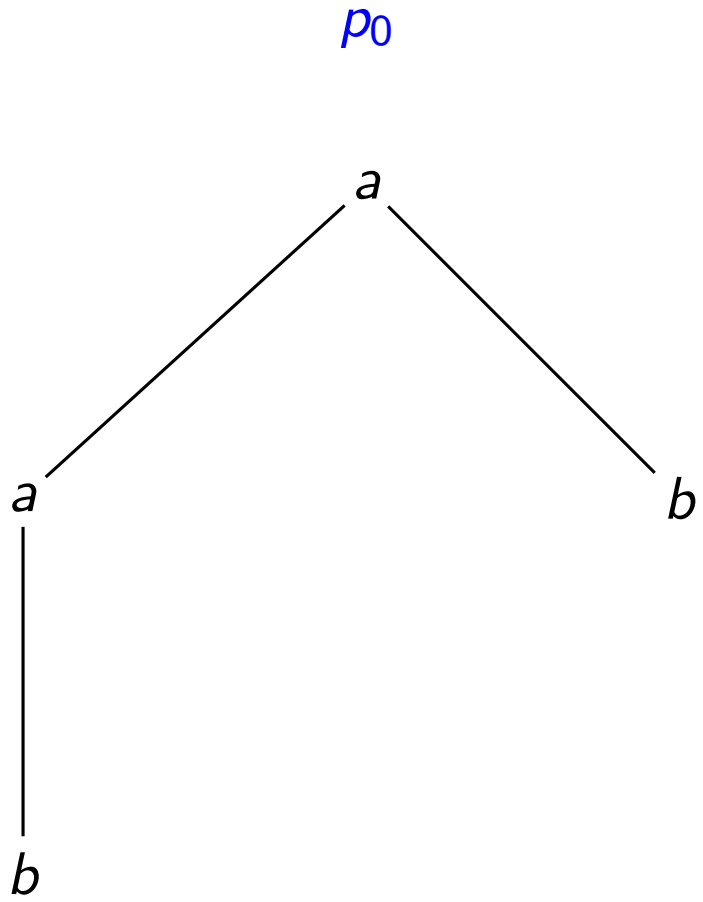
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:





Streaming Tree Automata

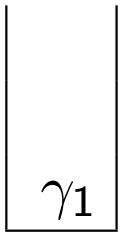
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

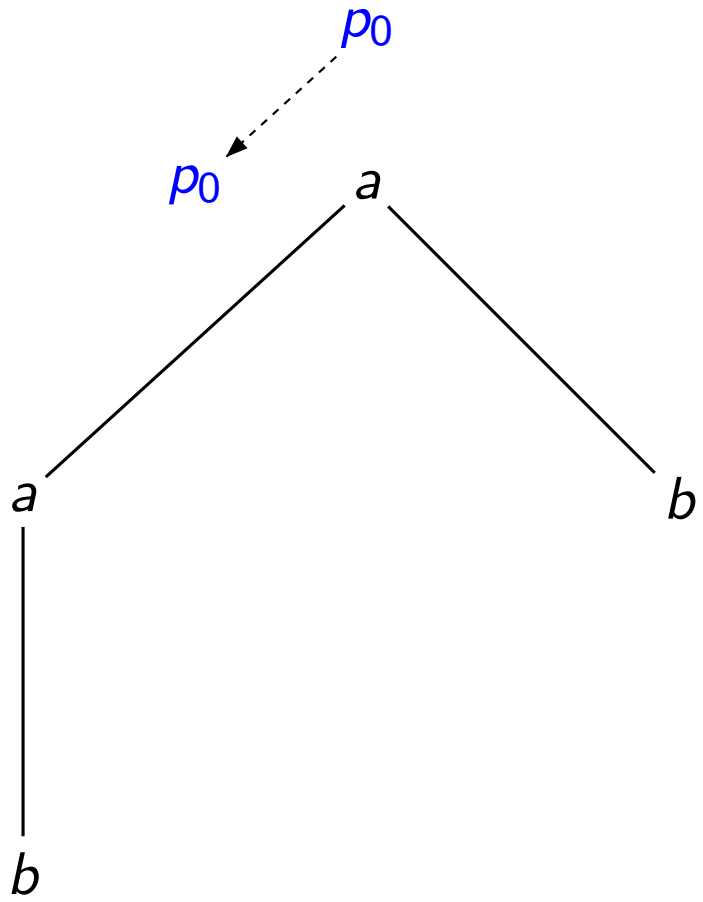
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:





Streaming Tree Automata

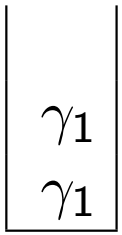
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

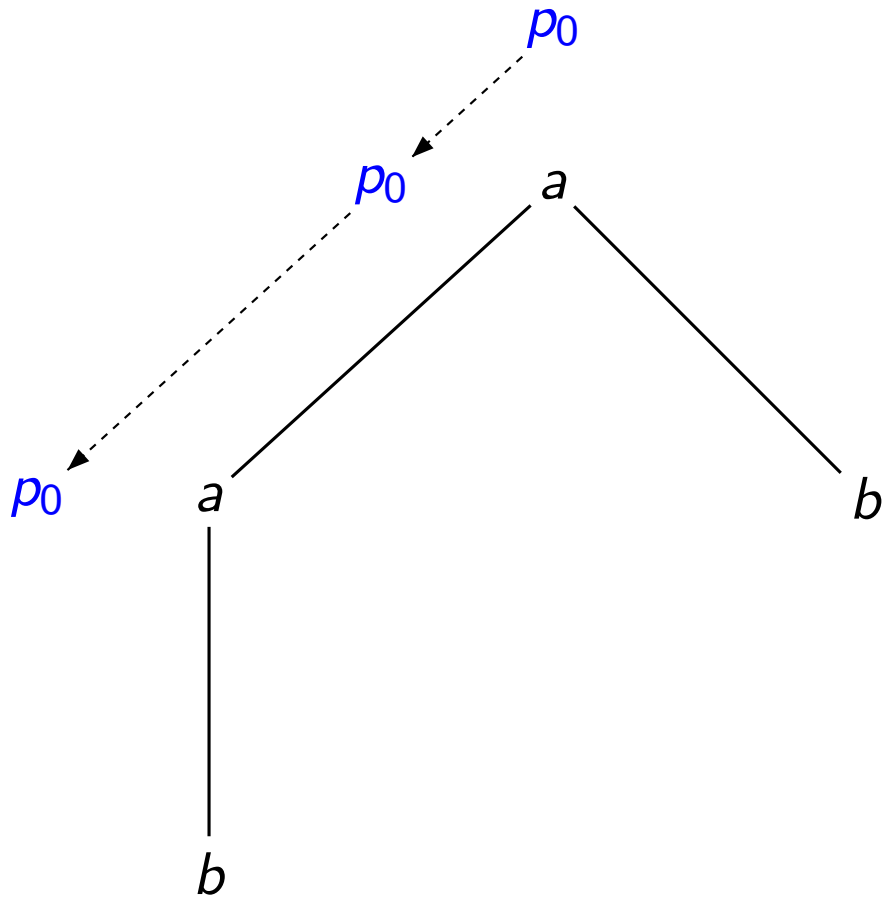
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:





Streaming Tree Automata

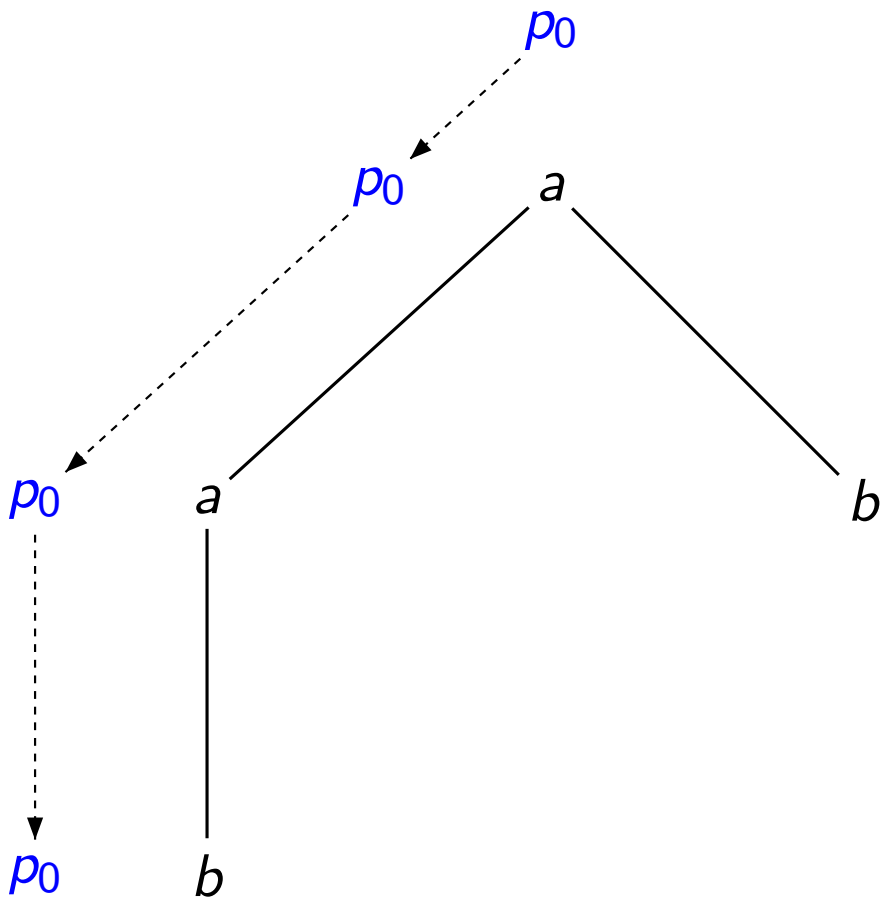
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack: $\begin{vmatrix} \gamma_2 \\ \gamma_1 \\ \gamma_1 \end{vmatrix}$



Streaming Tree Automata

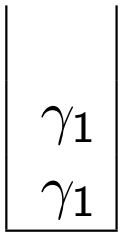
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

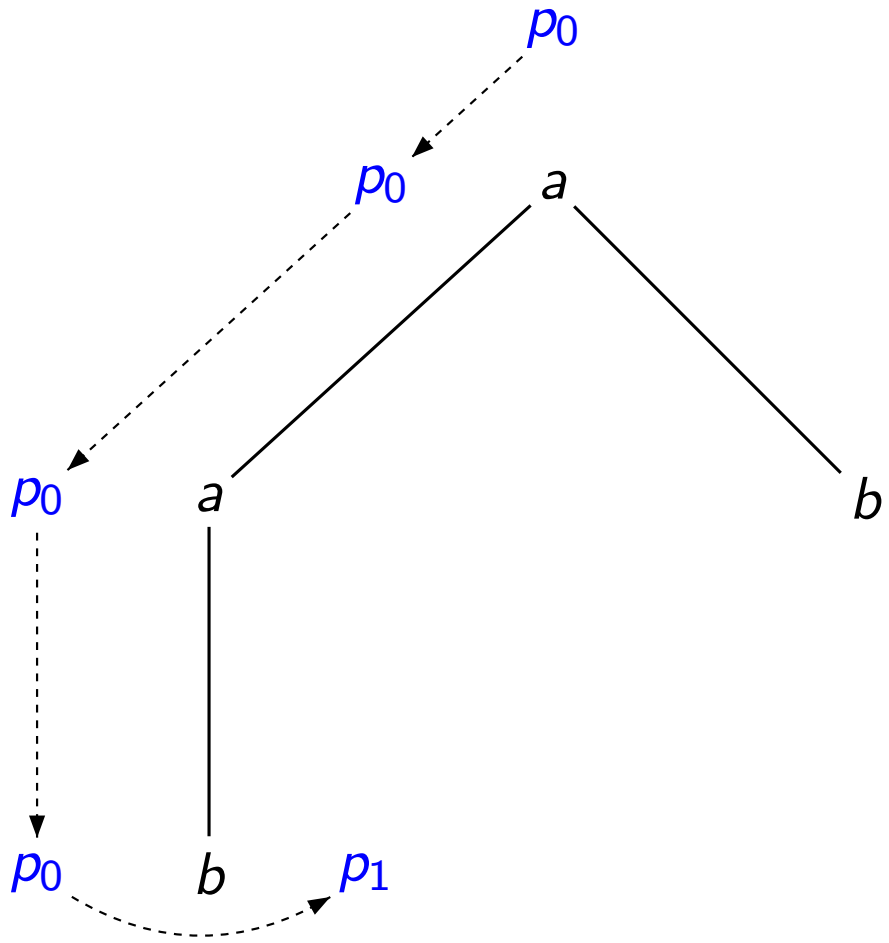
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \to p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \to p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \to p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \to p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \to p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \to p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \to p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \to p_2\}$

Stack:





Streaming Tree Automata

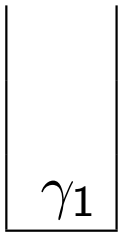
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

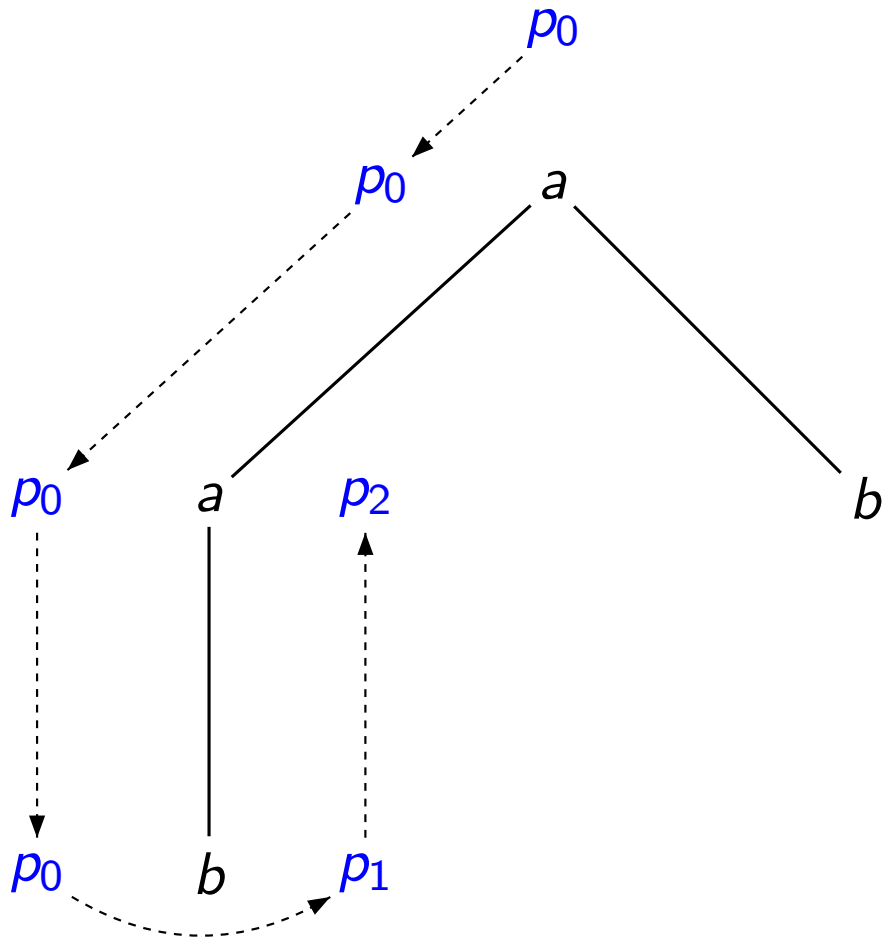
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:





Streaming Tree Automata

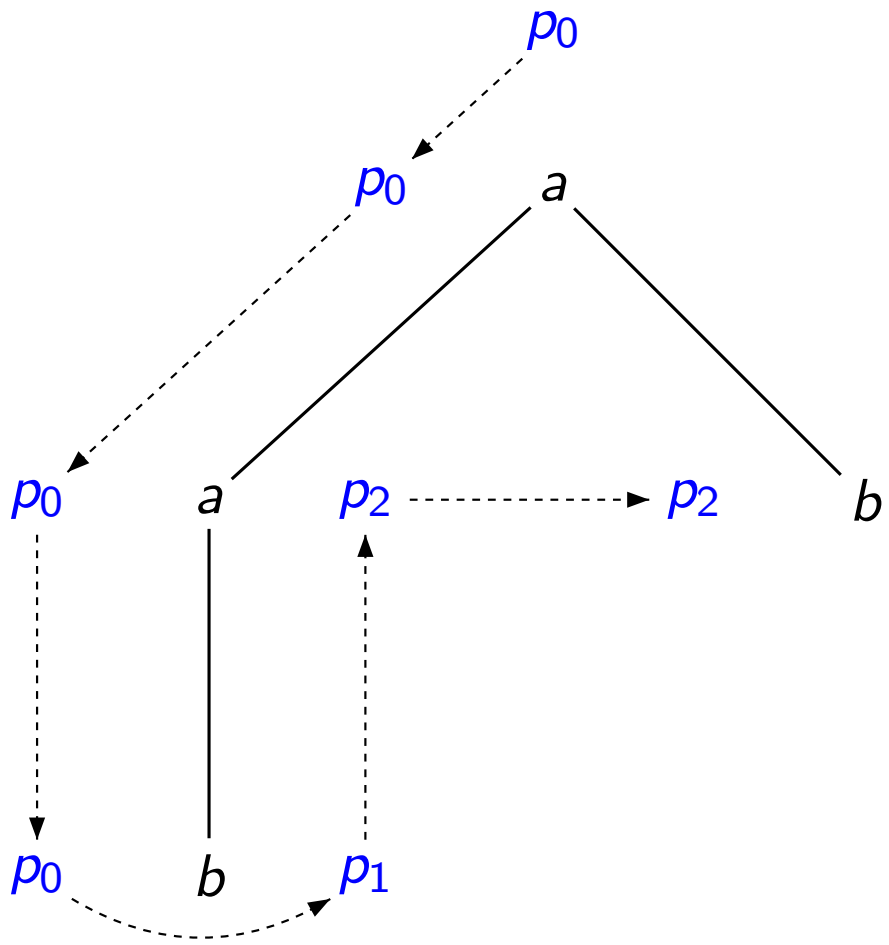
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:



Streaming Tree Automata

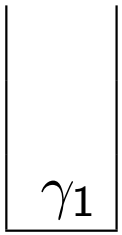
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

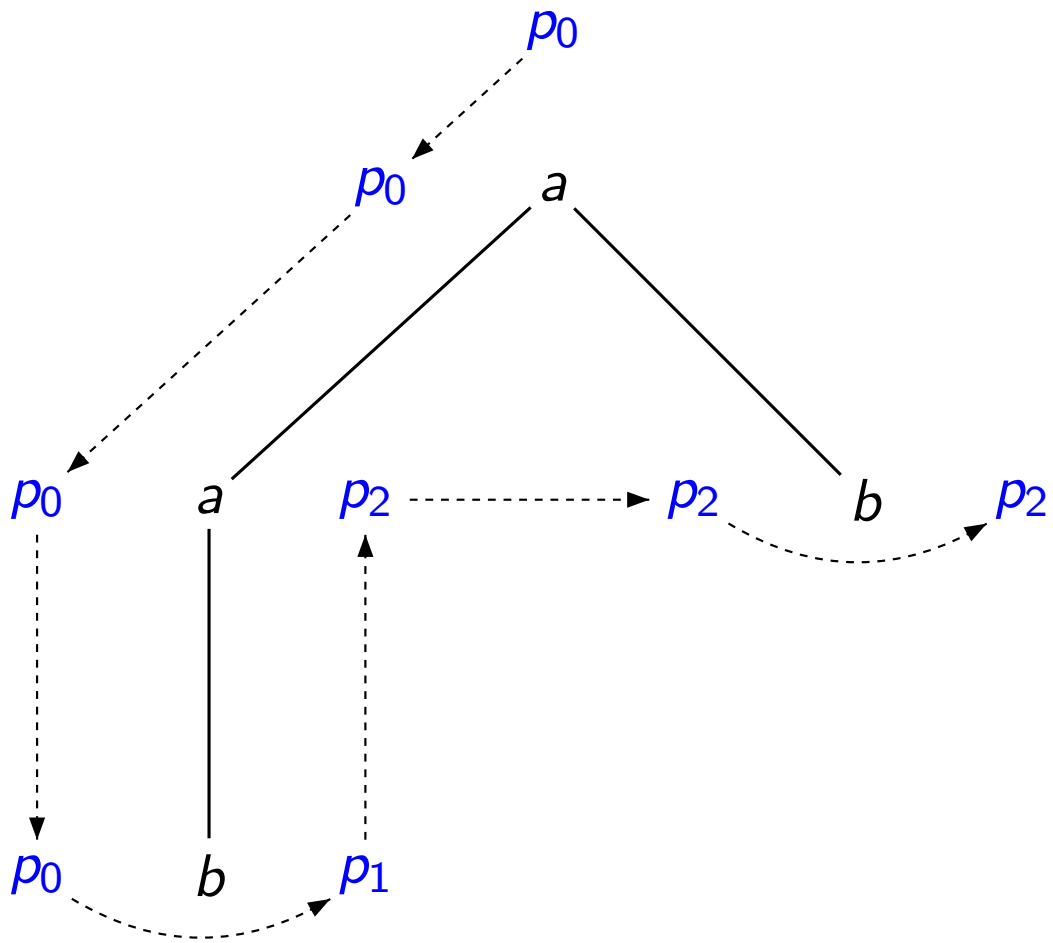
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:





Streaming Tree Automata

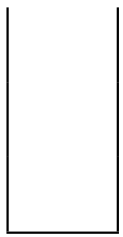
A: STA on T_Σ with $\Sigma = \{a, b\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$

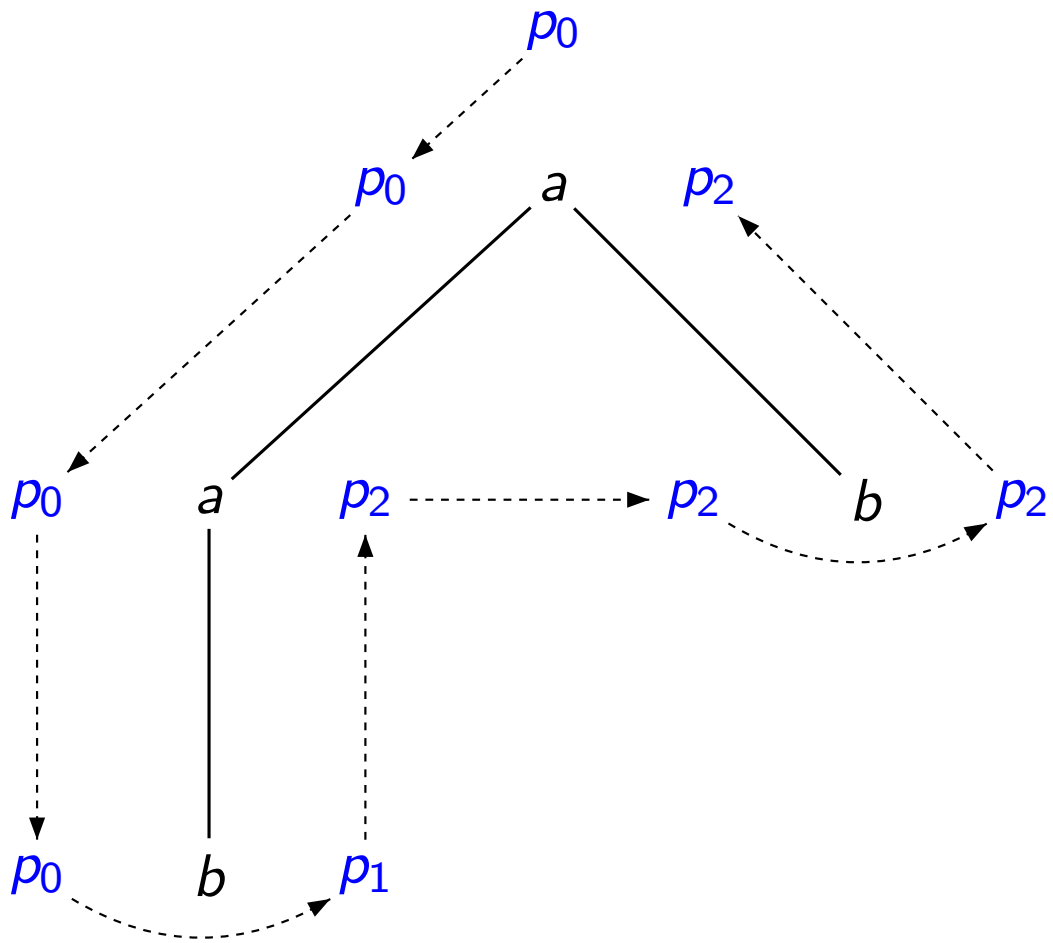
$\texttt{states} = \{p_0, p_1, p_2\}$ $\quad$ $\texttt{init} = \{p_0\}$ $\quad$ $\texttt{final} = \{p_1, p_2\}$

$\texttt{rules} = \{$

- $\texttt{open}\ a\ p_0 \longrightarrow p_0\ \gamma_1$
- $\texttt{open}\ b\ p_0 \longrightarrow p_0\ \gamma_2$
- $\texttt{open}\ a\ p_2 \longrightarrow p_2\ \gamma_1$
- $\texttt{open}\ b\ p_2 \longrightarrow p_2\ \gamma_2$
- $\texttt{close}\ a\ p_0\ \gamma_1 \longrightarrow p_0$
- $\texttt{close}\ b\ p_0\ \gamma_2 \longrightarrow p_1$
- $\texttt{close}\ a\ p_2\ \gamma_1 \longrightarrow p_2$
- $\texttt{close}\ b\ p_2\ \gamma_2 \longrightarrow p_2\}$

Stack:





Implications on Complexity

Deterministic STAs

The sufficiency problem of queries defined by deterministic STAs is in PTIME combined complexity.

Non-deterministic STAs

The sufficiency problem of queries represented by non-deterministic STAs is DEXPTIME-complete.

hardness is known, membership is obtained by determinization

1 Earliest Query Answering

2 Streaming Tree Automata

3 Streaming Algorithm

A basic algorithm

Signature

- Input:
  - a query q and a schema S
  - the stream of events of a tree $t \in T_\Sigma$
- Output:
  - all the tuples of $q(t)$ at their earliest event w.r.t. S

A basic algorithm

Signature

- Input:
  - a query q and a schema S
  - the stream of events of a tree $t \in T_\Sigma$
- Output:
  - all the tuples of $q(t)$ at their earliest event w.r.t. S

Computation

- generate all candidate tuples on the fly
- if a candidate is sufficient for selection, output it and remove it

This algorithm is clearly earliest.

Reducing space consumption

Concurrency

- τ is *alive* at event e if it is neither selected nor failed at e
- the *concurrency* is the maximal number of candidates that are alive at a same event

introduced by Bar-Yossef, Fontoura, Josifovski (without schema)

Complexity

- by testing for *failures*, we can discard some candidates
- we only keep alive candidates
- space complexity bounded by concurrency

An efficient algorithm for STAs

Main ideas

- input: deterministic STA A recognizing the canonical language of q

At each event and for each alive candidate:

- memoize the state reached in A
- compute safe states for selection and failure

2 steps

1. Logic: build the STA $\mathtt{E}(A)$ that recognizes sufficiency
2. Algorithm: compute safe states

$\mathtt{E}(A)$: an STA for detecting sufficiency

$\mathtt{E}(A) = A +$ additional information on sufficiency

- states of $\mathtt{E}(A)$ are of the form $(p, \mathcal{S}, \mathcal{F})$ where:
  - $p \in \mathtt{states}^A$
  - $\mathcal{S} \subseteq \mathtt{states}^A$ are safe states for selection
  - $\mathcal{F} \subseteq \mathtt{states}^A$ are safe states for failure

$\mathtt{E}(A)$: an STA for detecting sufficiency

$\mathtt{E}(A) = A +$ additional information on sufficiency

- states of $\mathtt{E}(A)$ are of the form $(p, \mathcal{S}, \mathcal{F})$ where:
  - $p \in \mathtt{states}^A$
  - $\mathcal{S} \subseteq \mathtt{states}^A$ are safe states for selection
  - $\mathcal{F} \subseteq \mathtt{states}^A$ are safe states for failure
- let $r(e) = (p, \mathcal{S}, \mathcal{F})$, r being a run on t for one candidate τ
  - e is sufficient for selection iff $p \in \mathcal{S}$
  - e is sufficient for failure iff $p \in \mathcal{F}$

$\mathtt{E}(A)$: an STA for detecting sufficiency

$\mathtt{E}(A) = A +$ additional information on sufficiency

- states of $\mathtt{E}(A)$ are of the form $(p, \mathcal{S}, \mathcal{F})$ where:
  - $p \in \mathtt{states}^A$
  - $\mathcal{S} \subseteq \mathtt{states}^A$ are safe states for selection
  - $\mathcal{F} \subseteq \mathtt{states}^A$ are safe states for failure
- let $r(e) = (p, \mathcal{S}, \mathcal{F})$, r being a run on t for one candidate τ
  - e is sufficient for selection iff $p \in \mathcal{S}$
  - e is sufficient for failure iff $p \in \mathcal{F}$

$\Rightarrow$ Sufficiency is MSO-definable!

Efficient algorithm

Overview

- A is deterministic $\Rightarrow \mathtt{E}(A)$ is deterministic
- we don't build $\mathtt{E}(A)$, but its runs on the fly
- to do so, we compute the safe states in 2 steps
  - precomputation of an accessibility relation (in PTIME)
  - at reception of an event, use this relation to compute the safe states (in PTIME)

Perspectives

- use this formalization on some decision problems
- earliest query answering problem of Forward XPath with schemas
- find other fragments suitable for streaming
- improve data structure for candidates (Meuss, Schultz and Bry, 2001)
- queries defined by selection automata

Thank you!