Automata and Logic on Trees

Some XML-related Applications

Wim Martens[1] Stijn Vansummeren[2]

[1]University of Dortmund, Germany
[2]Hasselt University, Belgium

Our story so far

- **Tree automata**
  deterministic, bottum-up, top-down, constructions, ...
  ranked and unranked
- **Decision problems for tree automata**
  Equivalence, universality, emptiness, intersection emptiness, ...,
- **MSO on trees**
  equivalent to tree automata

Our story so far

- **Tree automata**
  deterministic, bottum-up, top-down, constructions, ...
  ranked and unranked
- **Decision problems for tree automata**
  Equivalence, universality, emptiness, intersection emptiness, ...,
- **MSO on trees**
  equivalent to tree automata

Nice theory, what's the killer application?

And the winner is: XML!

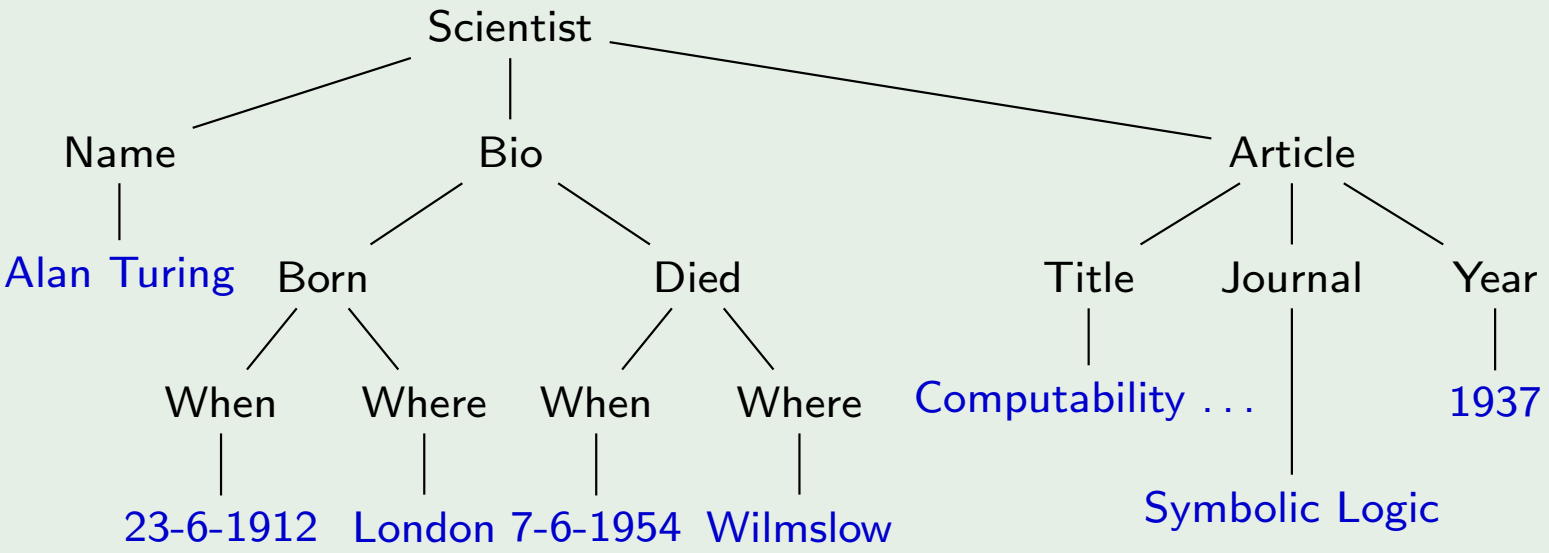
XML is the lingua franca of data on the Web

Consider this:

```
<Scientist>
  <Name>Alan Turing</Name>
  <Bio>
    <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
    <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
  </Bio>
  <Article>
    <Title> Computability and lambda-Definability </Title>
    <Journal> J. Of Symbolic Logic </Journal>
    <Year> 1937 </Year>
    ...
  </Article>
    ...
</Scientist>
...
```

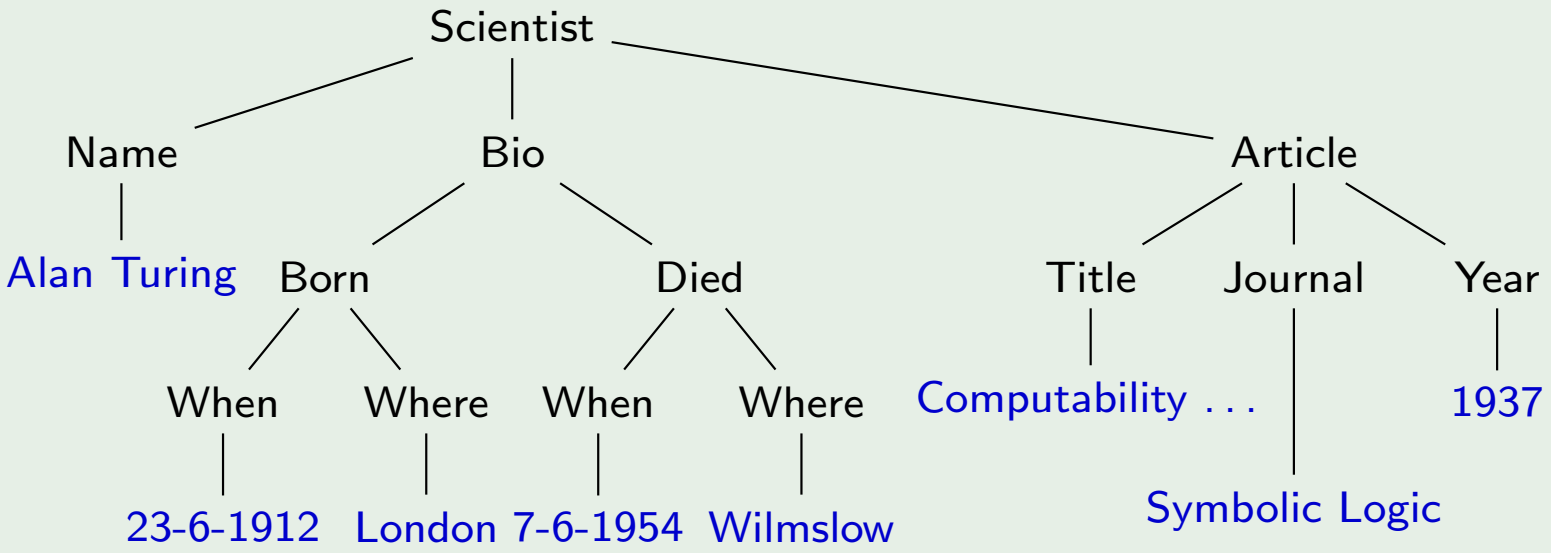
And the winner is: XML!

Now consider this:



And the winner is: XML!

Now consider this:



A natural correspondence

- Trees reflect the hierarchical structure of XML
- The data model underlying XML is tree-based

XML Processing

Important kinds of XML processing

- Validation
  Check whether an XML document is of given type
- Querying
  Extract information from an XML document
- Transformation
  Construct a new XML document from a given one

XML Processing

Important kinds of XML processing

- Validation → DTD, XML Schema
  Check whether an XML document is of given type
- Querying → XPath, XQuery
  Extract information from an XML document
- Transformation → XSLT, XDuce, $\mathbb{C}$Duce
  Construct a new XML document from a given one

Outline

Document Type Definitions (DTDs)

DTDs describe types of XML documents

Example document

```
<Scientist>
  <Name>Alan Turing</Name>
  <Bio>
    <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
    <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
  </Bio>
  <Article>
    <Title> Computability and lambda-Definability </Title>
    <Journal> J. Of Symbolic Logic </Journal>
    <Year> 1937 </Year>
    ...
  </Article>
    ...
</Scientist>
```

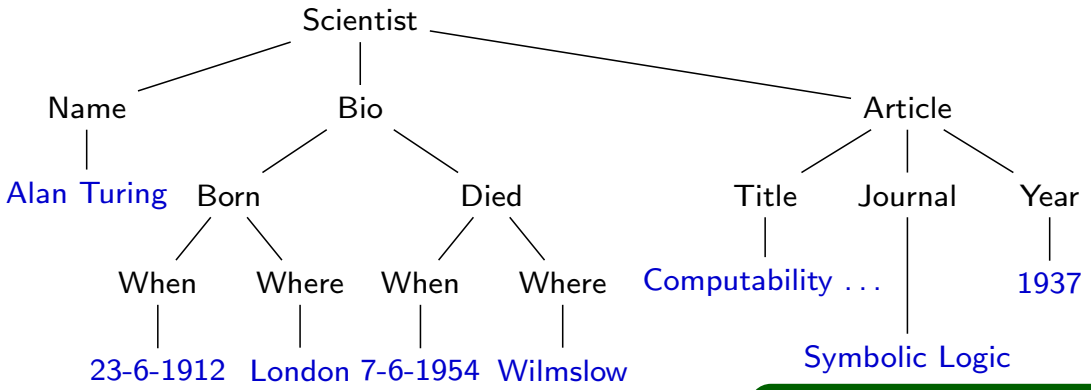
Example DTD

```
<!DOCTYPE Scientist [
    <!ELEMENT Scientist (Name, Bio, Article*)>
    <!ELEMENT Bio (Born, Died?)>
    <!ELEMENT Born (When, Where)>
    <!ELEMENT Died (When, Where)>
    <!ELEMENT Article (Title, Journal, Year)>
]>
```

Document Type Definitions (DTDs)

Validation algorithm:

- For each node: check that the children are ok w.r.t. parent's rule
- But ignore data values (Alan Turing, 23-6-1912, . . .)



Example DTD

```
<!DOCTYPE Scientist [
    <!ELEMENT Scientist (Name, Bio, Article*)>
    <!ELEMENT Bio (Born, Died?)>
    <!ELEMENT Born (When, Where)>
    <!ELEMENT Died (When, Where)>
    <!ELEMENT Article (Title, Journal, Year)>
]>
```

Hmmm . . . that looks familiar!

Example DTD

```
<!DOCTYPE Scientist [
    <!ELEMENT Scientist (Name, Bio, Article*)>
    <!ELEMENT Bio (Born, Died?)>
    <!ELEMENT Born (When, Where)>
    <!ELEMENT Died (When, Where)>
    <!ELEMENT Article (Title, Journal, Year)>
]>
```

Corresponding Tree Automaton

- $\text{Alphabet}(A) = \{\text{Scientist}, \text{Name}, \text{Bio}, \ldots, \text{When}, \text{Where}\}$
- $\text{States}(A) = \{\text{Scientist}, \text{Name}, \text{Bio}, \ldots, \text{When}, \text{Where}\}$
- $\text{Final}(A) = \{\text{Scientist}\}$
- $\text{Name}, \text{Bio}, \text{Article}^* \xrightarrow{\text{Scientist}} \text{Scientist}$
- . . .

Actually . . .

Fact

The XML standard requires all regular expressions occurring in a DTD to be deterministic

Example

Intuitively, an expression is deterministic if it is always determined which expression symbol will match the next input symbol of an input string

- Not deterministic: $a(bc + bb)$
- Deterministic: $ab(c + b)$

Not all regular expressions can be written as deterministic regular expressions [Brüggemann-Klein,Wood 1998]

Actually . . .

Fact

Deterministic regular expressions can be translated into deterministic string automata in linear time

Immediate corollary

- Every DTD can be translated into an equivalent stepwise deterministic unranked tree automaton in linear time
- Such an automaton gives a validation algorithm!
- Moreover, since containment of deterministic TA is in **ptime**, we can also check that every XML document valid w.r.t a DTD D_1 is also valid w.r.t. a DTD D_2 (useful in schema evolution, data exchange,. . .).

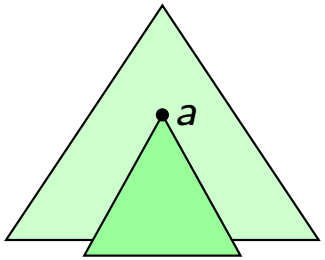
Expressive power?

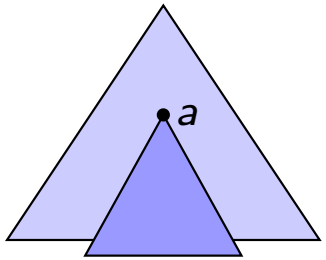
Can DTDs specify all regular tree languages?

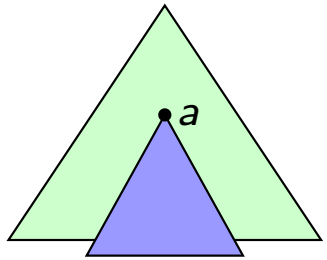
- (a) No, because they can be translated in dop-down deterministic unranked tree automata
- (b) No, because they cannot define the boolean circuits that evaluate to true
- (c) No, because the labels are the same as the states
- (d) Yes, but you have to extend them a little

DTD's are quite limited

Observation: There is only one rule for every label in a DTD D

Hence if 

$\in D$ and 

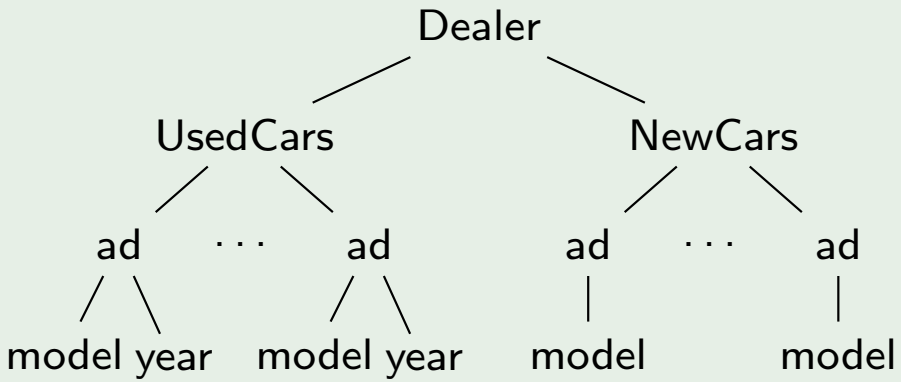
$\in D$ then 

$\in D$

We can use this to show that a tree language is not expressible as a DTD

DTD's are quite limited

Example: there is no DTD recognizing only

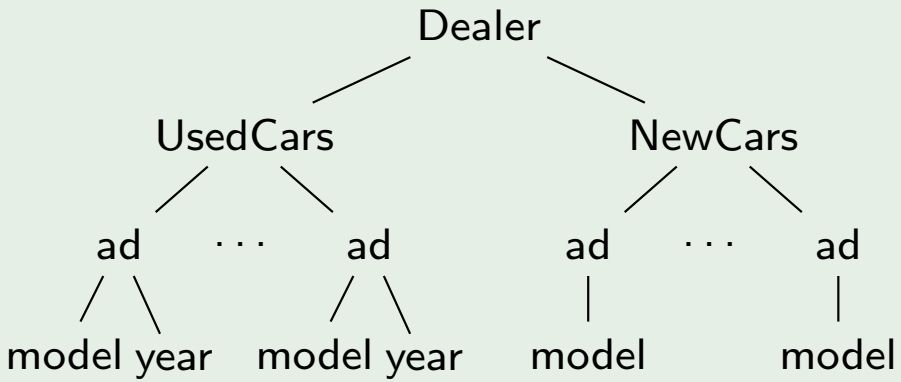


Obviously incorrect:

```
<!DOCTYPE Dealer [
    <!ELEMENT Dealer (UsedCars, NewCars)>
    <!ELEMENT UsedCars (ad*)>
    <!ELEMENT NewCars (ad*)>
    <!ELEMENT ad ((model, year) + model)>
]>
```

XML Schema to the rescue

Example: there is an XML Schema recognizing only



XML Schema (using abstract syntax):

Dealer $\mapsto$ (UsedCars, NewCars)
UsedCars $\mapsto$ (ad^1*)
NewCars $\mapsto$ (ad^2*)
ad^1 $\mapsto$ (model, year)
ad^2 $\mapsto$ (model)

Hmm . . . this looks familiar

Corresponding Tree Automaton

- $\text{Alphabet}(A) = \{\text{Dealer}, \text{UsedCars}, \text{Newcars}, \text{ad}, \text{model}, \text{year}\}$
- $\text{States}(A) = \{\text{Dealer}, \text{UsedCars}, \text{Newcars}, \text{ad}^1, \text{ad}^2, \text{model}, \text{year}\}$
- $\text{UsedCars}, \text{NewCars} \xrightarrow{\text{Dealer}} \text{Dealer}$
- $\text{ad}^1* \xrightarrow{\text{UsedCars}} \text{UsedCars}$
- $\text{model}, \text{year} \xrightarrow{\text{ad}} \text{ad}^1$
- $\text{model} \xrightarrow{\text{ad}} \text{ad}^2$
- . . .

XML Schema (using abstract syntax):

Dealer $\mapsto$ (UsedCars, NewCars)
UsedCars $\mapsto$ (ad^1*)
NewCars $\mapsto$ (ad^2*)
ad^1 $\mapsto$ (model, year)
ad^2 $\mapsto$ (model)

Actually . . .

Fact

- The XML Schema standard forbids rules like

$$\text{FunkyCars} \mapsto (\text{ad}^1*, \text{sec}, \text{ad}^2*)$$

  in which the same label occurs with two different types
- When ignoring types, the regular expressions must again be deterministic

And again things transfer nicely

Facts:

- Every XML Schema can be translated into an equivalent deterministic unranked tree automaton in linear time
- Such an automaton gives a validation algorithm!
- Moreover, since containment of deterministic TA is in **ptime**, we can also check that every XML document valid w.r.t an XML Schema D_1 is also valid w.r.t. an XML Schema D_2 (useful in schema evolution, data exchange,...).
- Moreover, we can minimize XML Schema's in **ptime**

Expressive power?

Can XML Schema's specify all regular tree languages?

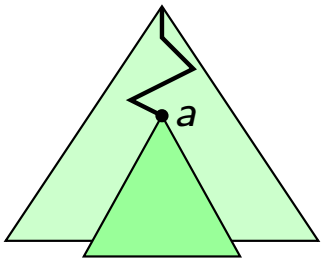
- (a) No, because they can be translated in dop-down deterministic unranked tree automata
- (b) No, because they cannot define the boolean circuits that evaluate to true
- (c) Yes, but you have to extend them a little

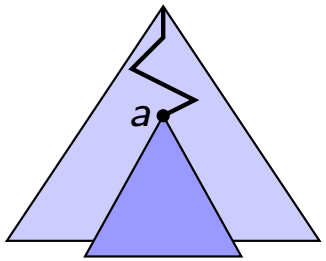
XML Schema's are also limited

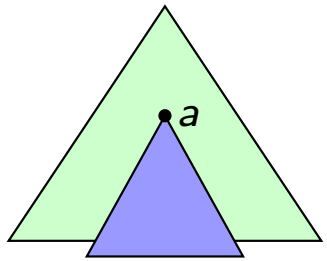
Observation: Since rules like

$$\text{FunkyCars} \mapsto (\text{ad}^1*, \text{sec}, \text{ad}^2*)$$

are forbidden, the "type" of a node is determined by the string of labels encountered on the path from the root to that node.

Hence if 

$\in S$ and 

$\in S$ then 

$\in S$

We can use this to show that a tree language is not definable in XML Schema

XML Schema's are also limited

Exercise: Show that boolean circuit evaluation is hence not definable by an XML Schema

XML Schema's are also limited

Exercise: Show that boolean circuit evaluation is hence not definable by an XML Schema

Fact

By allowing rules like

$$\text{FunkyCars} \mapsto (\text{ad}^1*, \text{sec}, \text{ad}^2*)$$

in which the same label occurs with two different types and by allowing all regular expressions we reach the full regular languages

In summary

Tree automata

- Form a general framework for schema languages
- Provide an execution environment for linear time validation
- Also serve as a basis for restricted classes with better algorithmic properties w.r.t. static analysis

Outline

XPath

XPath expressions select sets of nodes of XML documents by specifying navigational patterns

Example document

```
<Scientist>
  <Name>Alan Turing</Name>
  <Bio>
    <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
    <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
  </Bio>
  <Article>
    <Title> Computability and lambda-Definability </Title>
    <Journal> J. Of Symbolic Logic </Journal>
    <Year> 1937 </Year>
    ...
  </Article>
    ...
</Scientist>
```

Example query

```
//Bio/Died/*
```

XPath

XPath expressions select sets of nodes of XML documents by specifying navigational patterns

Example document

```
<Scientist>
  <Name>Alan Turing</Name>
  <Bio>
    <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
    <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
  </Bio>
  <Article>
    <Title> Computability and lambda-Definability </Title>
    <Journal> J. Of Symbolic Logic </Journal>
    <Year> 1937 </Year>
    ...
  </Article>
    ...
</Scientist>
```

Example query

```
//Bio/Died/*
```

Node-Selecting Queries

Observation: Such queries can also be expressed by MSO formulas with one free variable

Example document

```
<Scientist>
  <Name>Alan Turing</Name>
  <Bio>
    <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
    <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
  </Bio>
  <Article>
    <Title> Computability and lambda-Definability </Title>
    <Journal> J. Of Symbolic Logic </Journal>
    <Year> 1937 </Year>
    ...
  </Article>
    ...
</Scientist>
```

Example query

$$\phi(x) := \exists y\, E(y,x) \wedge L_{\text{Died}}(y)$$

Node-Selecting Queries

Terminology

- A (node-selecting) query is a function q: tree $\longrightarrow$ nodes
- A query is MSO-definable if there exists a MSO formula $\phi(x)$ such that $n \in q(t)$ iff $t \models \phi(n)$, for all trees t and all nodes n

Node-Selecting Queries

Terminology

- A (node-selecting) query is a function q: tree $\longrightarrow$ nodes
- A query is MSO-definable if there exists a MSO formula $\phi(x)$ such that $n \in q(t)$ iff $t \models \phi(n)$, for all trees t and all nodes n

Theorem

- Every XPath query is MSO-definable
- But XPath cannot express every MSO-definable query

Node-Selecting Queries

Terminology

- A (node-selecting) query is a function q: tree $\longrightarrow$ nodes
- A query is MSO-definable if there exists a MSO formula $\phi(x)$ such that $n \in q(t)$ iff $t \models \phi(n)$, for all trees t and all nodes n

Theorem

- Every XPath query is MSO-definable
- But XPath cannot express every MSO-definable query

In fact

- Every XPath query is FO-definable when FO is endowed with the descendant and sibling relations (as opposed to parent and brother)
- But XPath cannot express every FO-definable query [Marx, 2004]

Node-Selecting Queries

In conclusion

- MSO provides a general framework for node-selecting queries
- although practical languages are often less expressive

Automaton Model?

Question: What is the corresponding automaton model?

Motivation for this question:

- A formula $\phi(x)$ gives a declarative specification for a query
- An automaton gives an algorithm for computing the query

Automaton Model?

Question: What is the corresponding automaton model?

Let's try this:

A query automaton Q consists of a non-deterministic bottom-up automaton A plus a select function

$$s\colon \mathrm{States}(A) \times \mathrm{Alphabet}(A) \to \{0,1\}$$

Node n is in the result for tree t if there is an accepting computation on t in which n gets a state q such that $s(q,a) = 1$, where a is the label of n

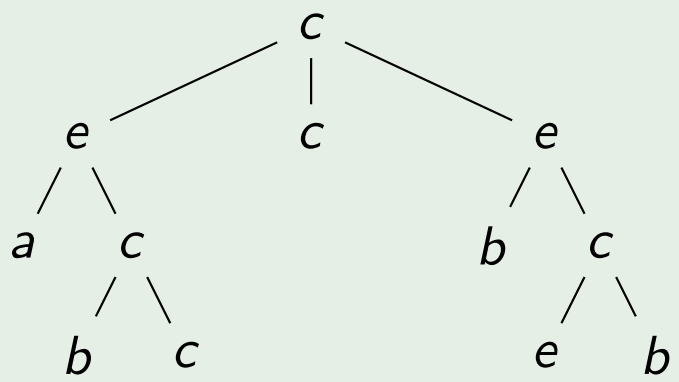
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 1



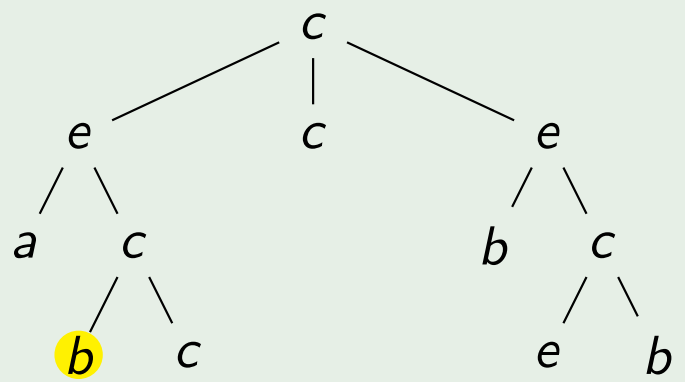
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 1



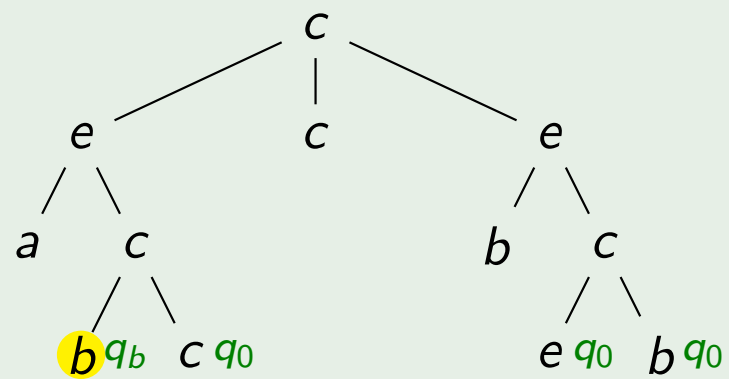
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 1



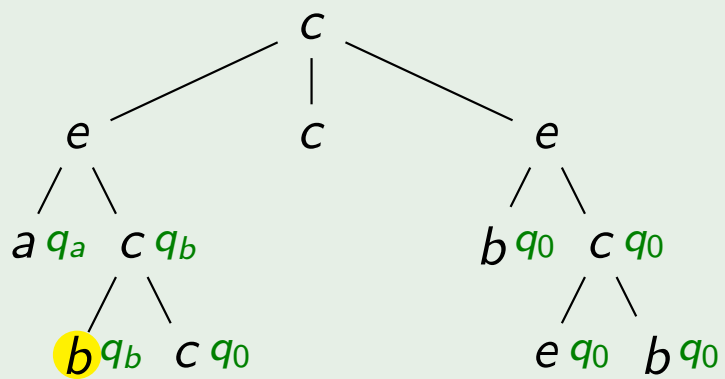
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 1



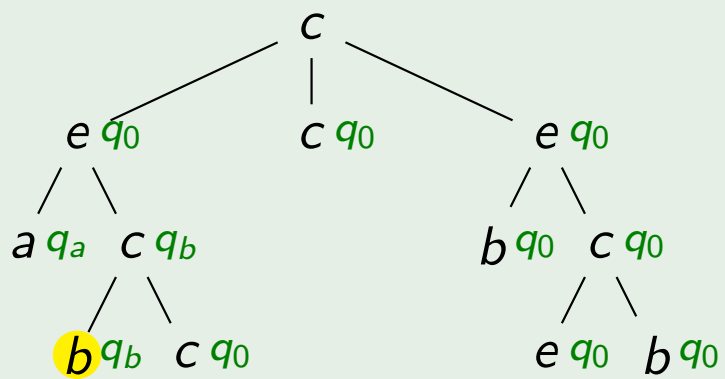
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 1



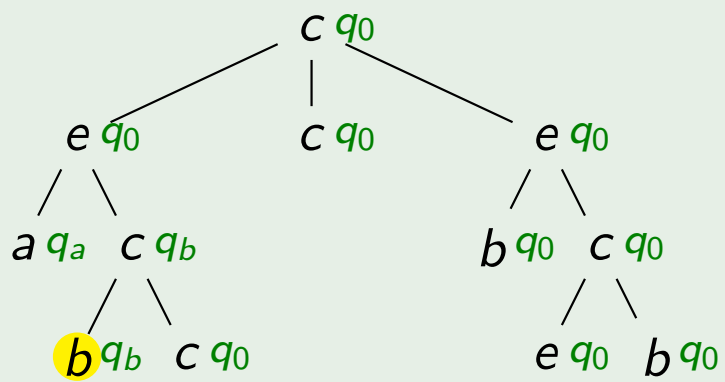
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 1



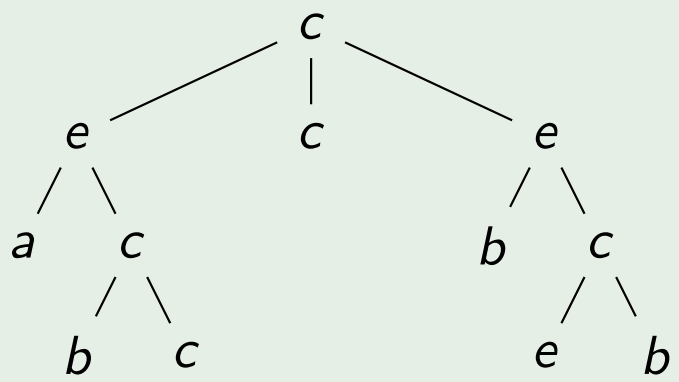
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



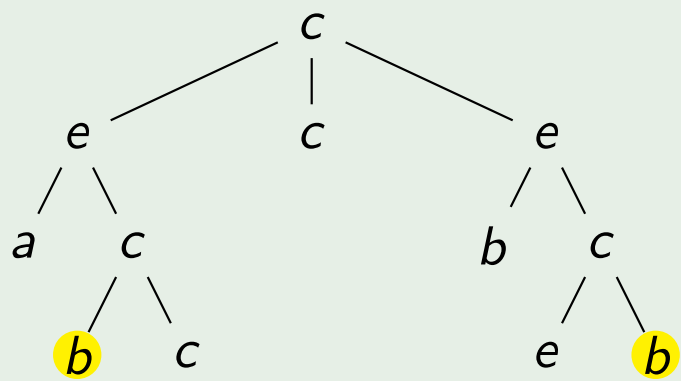
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



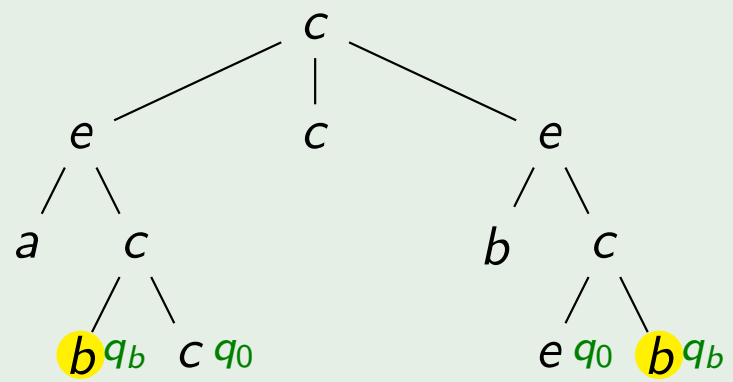
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



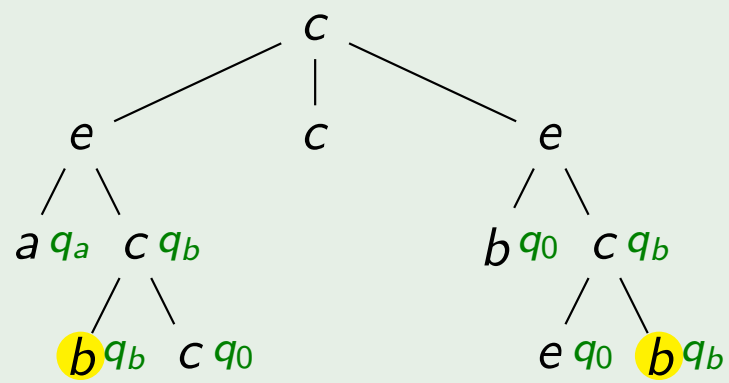
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



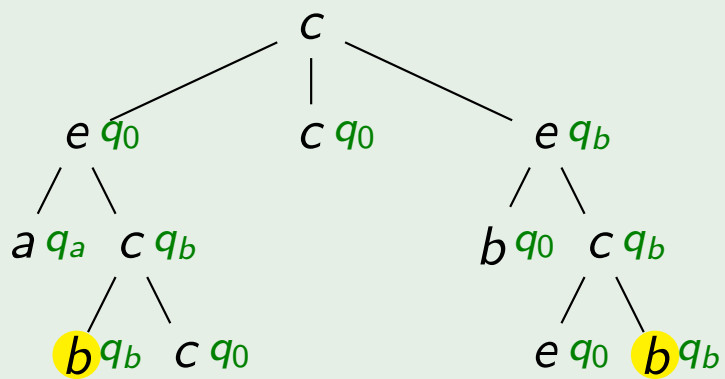
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



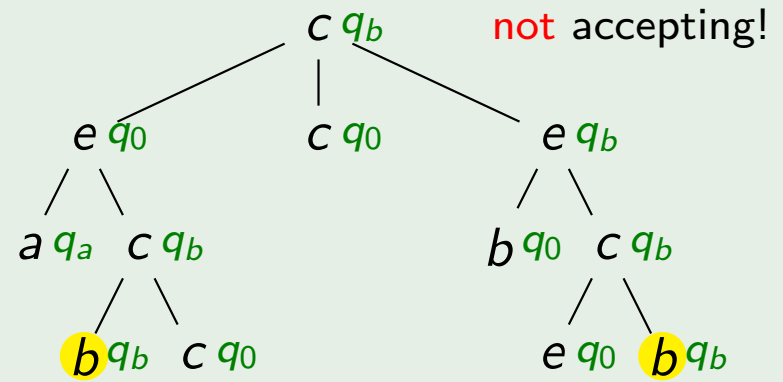
Example of a Query Automaton

Select all b-labeled nodes for which there is an ancestor with an a-labeled child

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2

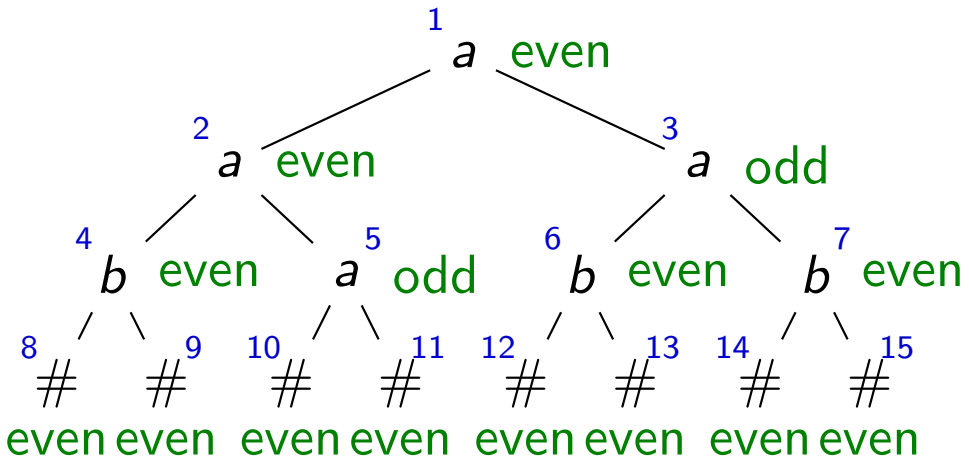


Automaton Model?

Theorem

Every query expressible by a query automaton (A, s) is MSO-definable.

Recall: If States$(A) = \{q_1, \ldots, q_n\}$ then every run of A on a tree t can be represented by sets of nodes $Q_1, \ldots, Q_n$



$\text{EVEN} := \{1, 2, 4, 6, 7, \ldots\}$
$\text{ODD} := \{3, 5\}$

Automaton Model?

Theorem

Every query q expressible by a query automaton (A, s) is MSO-definable.

Also recall: We can guess such a run in MSO:

$$\exists Q_1 \ldots \exists Q_n \, \mathit{validrun}(Q_1, \ldots, Q_n)$$

Automaton Model?

Theorem

Every query q expressible by a query automaton (A, s) is MSO-definable.

Hence: q is equivalently expressed by

$$\phi(x) := \exists Q_1 \ldots \exists Q_n \, \mathit{validrun}(Q_1, \ldots, Q_n) \wedge \bigvee_{\substack{q_i \in \mathrm{States}(A) \\ a \in \mathrm{Alphabet}(A) \\ s(q_i, a) = 1}} (Q_i(x) \wedge L_a(x))$$

Automaton Model?

Theorem

Every MSO-definable query $\phi(x)$ is expressible by a query automaton.

Automaton Model?

Theorem

Every MSO-definable query $\phi(x)$ is expressible by a query automaton.

Recall: $\phi(x)$ is equivalently expressed as a formula $\psi(X)$ such that

$$\underline{t} \models \phi(n) \Leftrightarrow \underline{t} \models \psi(\{n\})$$

where

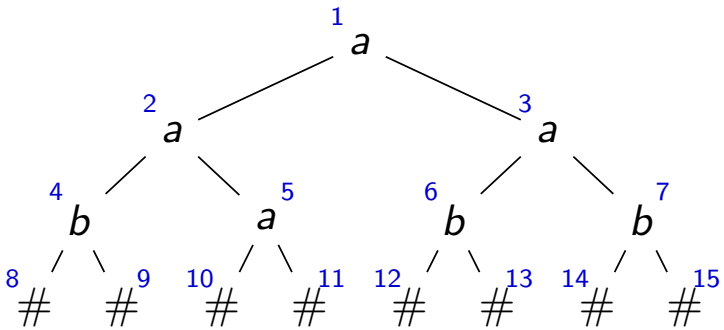
$$\begin{array}{rcl} \psi & := & X \subseteq Y \mid \mathrm{Sing}(X) \mid E(X,Y) \mid X < Y \mid X \subseteq L_a \mid \cdots \mid X \subseteq L_b \\ & \mid & \psi \wedge \psi \mid \neg\psi \mid \exists X\, \phi \end{array}$$

Automaton Model?

Theorem

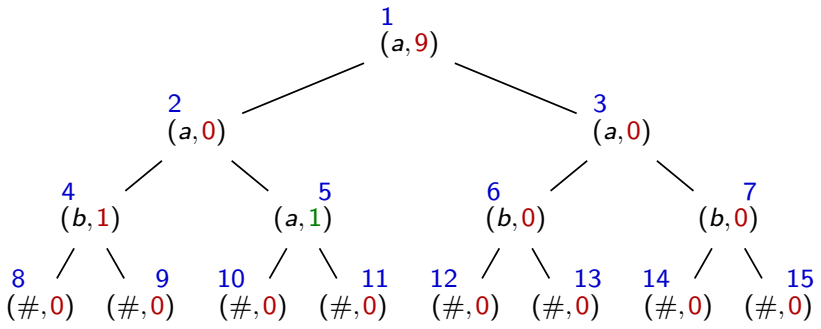
Every MSO-definable query $\phi(x)$ is expressible by a query automaton.

Also recall: We can view a formula $\psi(X)$ as defining a tree language over the extended alphabet $\Sigma \times \{0,1\}^n$. This language is recognizable by a tree automaton A.



ϕ selects node 5

$\underline{t} \models \phi(5) \Leftrightarrow \underline{t} \models \psi(\{5\})$



A accepts tree $t[\{5\}]$ over $\Sigma \times \{0,1\}^2$

Automaton Model?

Theorem

Every MSO-definable query $\phi(x)$ is expressible by a query automaton.

Hence: $\phi(x)$ is equivalently expressed by the query automaton (A', s) where

- A' is the automaton we obtain from A by replacing every rule

$$(q_1, \ldots, q_k) \xrightarrow{(a,b)} q \qquad \text{by} \qquad (q_1, \ldots, q_k) \xrightarrow{a} q$$

- s is the function such that $s(a, q) = 1$ if and only if there is a rule in A of the form

$$(q_1, \ldots, q_k) \xrightarrow{(a,1)} q$$

Query evaluation

The bad, the ugly, and the good:

- Unfortunately, the translation from formula $\phi(x)$ to automaton can be prohibitively expensive. The number of states is proportional to

$$\left.\begin{array}{c} 2^{2^{\cdot^{\cdot^{\cdot^{2^{\mathrm{size}(\psi)}}}}}} \\ \end{array}\right\} \mathrm{size}(\psi) \text{ times}$$

- Actually, unless $P = NP$ there is no elementary f such that MSO-formulas can be evaluated in time $f(\mathrm{size}(\phi)) \times p(\mathrm{size}(t))$ with p polynomial [Frick, Grohe 2002]
- This makes MSO useless as a query language. However Monadic Datalog [Gottlob, Koch 2002] can also express all MSO-definable queries and can be evaluated efficiently

Some questions about query automata

Question: does it matter that A is non-deterministic in a query automaton (A, s)?

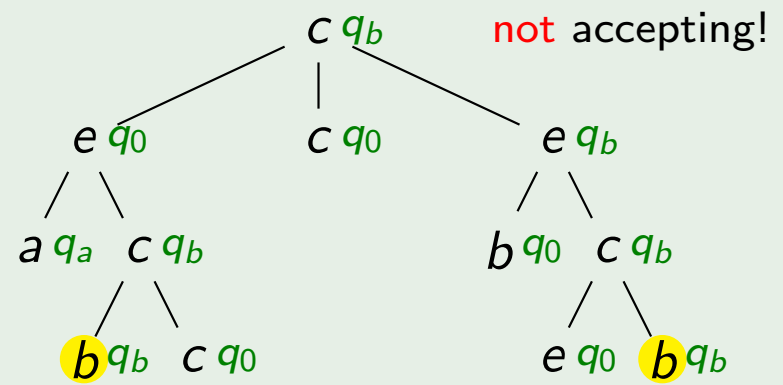
Some questions about query automata

Question: does it matter that A is non-deterministic in a query automaton (A, s)?

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



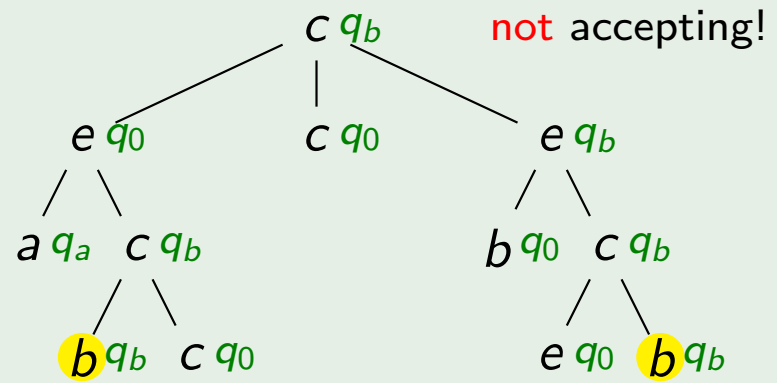
Some questions about query automata

Question: does it matter that A is non-deterministic in a query automaton (A, s)?

Example query automaton (A, s)

- $\text{States}(A) = \{q_0, q_a, q_b\}$
- $\text{Final}(A) = \{q_0\}$
- $\text{States}(A)^* \xrightarrow{a} q_a$
- $\text{States}(A)^* \xrightarrow{\sigma} q_b$
- $(\varepsilon + q_0^* + \text{States}(A)^* q_a \text{States}(A)^* \xrightarrow{\sigma} q_0$
- $s(q_b, b) = 1$
- all others: 0

Example tree - run 2



Quiz:

Can we select all b-labeled nodes for which there is an ancestor with an a-labeled child when A is deterministic?

Some questions about query automata

The bad, the ugly, and the good

It matters that A is non-deterministic in a query automaton (A, s)!

- Non-deterministic query automata cannot be implemented efficiently (need to check all possible runs)
- This renders them essentially useless as an model for specifying query algorithms
- But query automata can equivalently be defined as a triple (A_1, A_2, s) were A_1 is deterministic bottom-up, A_2 is deterministic top-down over $\text{States}(A_2)$, and s is a selection function

$$s\colon \text{States}(A_1) \times \text{States}(A_2) \times \text{Alphabet}(A_1) \to \{0, 1\}$$

See [Schwentick, Neven 2002]

Some questions about query automata (2)

Two possible semantics

- Existential semantics
  a node is in the result if there is an accepting run that selects it
- Universal semantics
  a node is in the result if every accepting run selects it.

Quiz:

Does it matter which semantics we take?

Some questions about query automata (2)

Two possible semantics

- Existential semantics
  a node is in the result if there is an accepting run that selects it
- Universal semantics
  a node is in the result if every accepting run selects it.

Quiz:

Does it matter which semantics we take?

No: Universal semantics can be stated in MSO:

$$\phi(x) := \forall Q_1 \ldots \forall Q_n \, validrun(Q_1, \ldots, Q_n) \rightarrow \bigvee_{\substack{q_i \in \text{States}(A) \\ a \in \text{Alphabet}(A) \\ s(q_i,a)=1}} (Q_i(x) \wedge L_a(x))$$

and hence translated back into a query automaton with existential semantics.

Some questions about query automata (2)

Two possible semantics

- Existential semantics
  a node is in the result if there is an accepting run that selects it
- Universal semantics
  a node is in the result if every accepting run selects it.

Quiz:

Does it matter which semantics we take?

No: Existential semantics can be transformed into a universal semantics by adapting A and s. Exercise

Outline

XSLT transforms documents by means of templates

Example input document

```
<Scientist>
  <Name>Alan Turing</Name>
  <Bio>
    <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
    <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
  </Bio>
  <Article>
    <Title> Computability and lambda-Definability </Title>
    <Journal> J. Of Symbolic Logic </Journal>
    <Year> 1937 </Year>
    ...
  </Article>
    ...
</Scientist>
```

Example XSLT Program

```
<xsl:template match="*">
    <Person>
      <xsl:copy-of select="Name"/>
      <xsl:copy-of select="Bio/Born"/>
      <xsl:copy-of select="Bio/Died"/>
    </Person>
<xsl:template>
```

XSLT transforms documents by means of templates

Example output

```
<Person>
  <Name>Alan Turing</Name>
  <Born> <When> June 23, 1912 </When> <Where> London </Where> </Born>
  <Died> <When> June 7, 1954 </When> <Where> Wilmslow </Where> </Died>
</Person>
```

Example XSLT Program

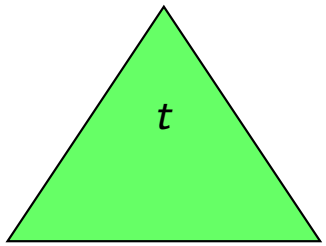
```
<xsl:template match="*">
    <Person>
      <xsl:copy-of select="Name"/>
      <xsl:copy-of select="Bio/Born"/>
      <xsl:copy-of select="Bio/Died"/>
    </Person>
<xsl:template>
```

XML Typechecking

The typechecking problem:

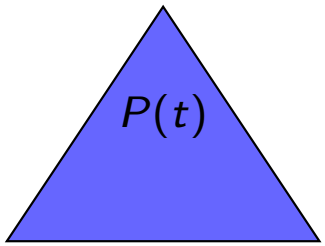
Given an XSLT program P, an XML Schema S and an XML Schema T, check that $P(t) \in T$ for every $t \in S$.

Motivation for this problem:



Microshaft has documents in S-form

$\overset{P}{\longmapsto}$



Macrosoft wants those documents in T-form

XML Typechecking

Theorem

The typechecking problem (without data values) is decidable!

Proof idea:

- It is possible to compute the inverse image of T under P:

$$P^{-1}(T) = \{ \text{ tree } t \mid P(t) \in T \}$$

- Moreover, this inverse image is regular
- Hence, it suffices to check that $S \subseteq P^{-1}(T)$ (why?)

Of course: the complexity of the problem varies widely if one takes e.g. restricted fragments of XSLT or DTDs instead of XML schemas, ...

Acknowledgement

The slides of this lecture are based on the PODS 2004 tutorial of Thomas Schwentick

http://ls1-www.cs.uni-dortmund.de/~tick/homepage.html