

the rest of this course

Volume

size does matter
(thousands of TBs of data)



Veracity

data is often
incomplete/inconsistent



Variety

many data formats
(structured, semi-structured, etc.)

Velocity

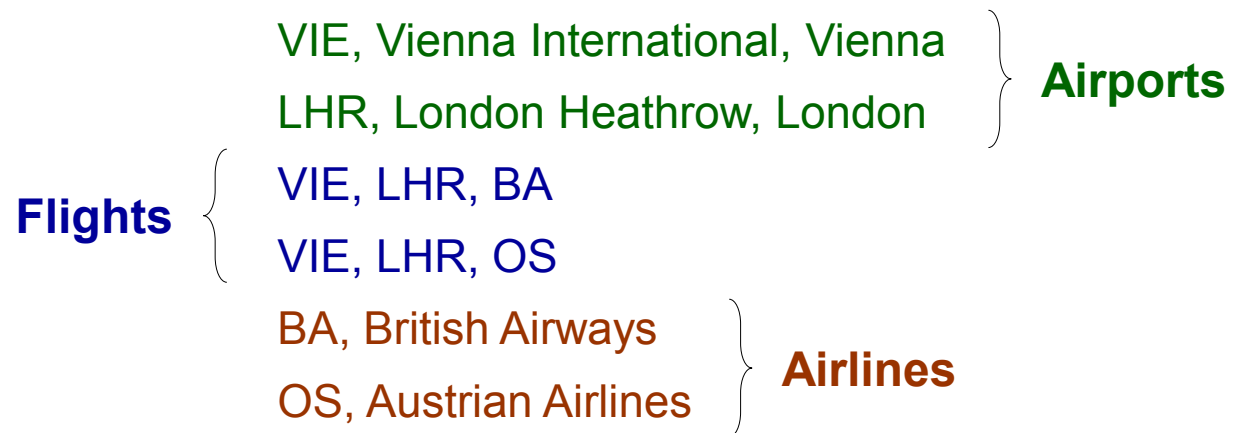
data often arrives at fast speed
(updates are frequent)

Structured Data

- Data is structured in semantic chunks - **entities**

VIE, Vienna International, Vienna
LHR, London Heathrow, London
VIE, LHR, BA
VIE, LHR, OS
BA, British Airways
OS, Austrian Airlines

- Similar entities are grouped together - **classes**



Structured Data

- Entities in the same class have the same descriptions - **attributes**

Airports

(VIE, Vienna International, Vienna)
(LHR, London Heathrow, London)

(Airport_Code, Name, City)

Airlines

(BA, British Airways)
(OS, Austrian Airlines)

(Airline_Code, Name)

Flights

(VIE, LHR, BA)
(VIE, LHR, OS)

(Origin, Destination, Airline)

Structured Data

- Entities in the same class have the same descriptions - **attributes**

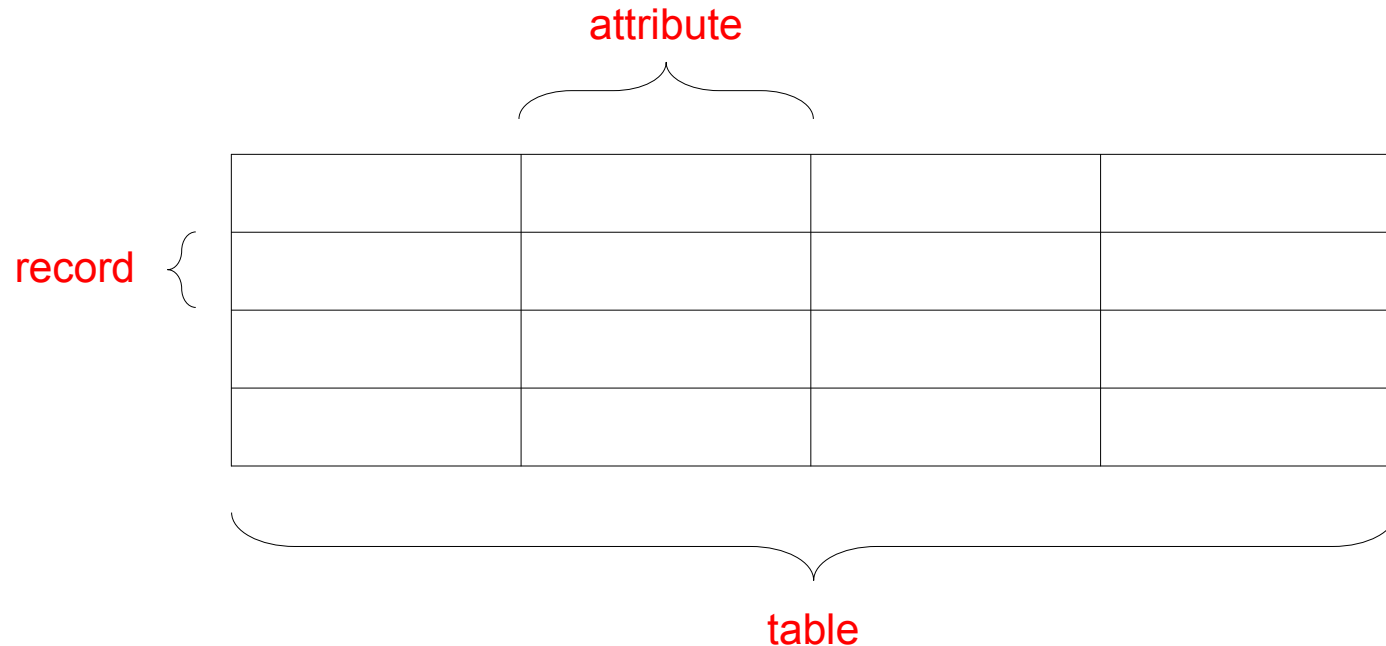


- Attributes in similar entities {
 - same format (string, integer, date, etc.)
 - predefined length
 - all present
 - same order

... strict structure forced by a schema!!!

Structured Data - Relational Model

- Database model for structured data: entities → records (or tuples)
classes → tables (or relations)
- Records grouped in tables



“Persons” Example

Gerti Kappel, 180870, 188096, gerti@big.tuwien.ac.at

Andreas, Pieris, apieris@inf.ed.ac.uk, 740072, 184943

Wolfgang Fischl, wfischl@dbai.tuwien.ac.at, 740050

Bill, Robert, 188316, bill@big.tuwien.ac.at

Semi-structured Data (SSD)

- Data is structured in semantic entities
- Similar entities are grouped in classes
- Entities in the same class may not have the same attributes
- Attributes of similar entities
 - may have different format
 - may have different length
 - not all required
 - may have different order

**there is
structure**

**but not
too much
structure**

Semi-structured Data: “Persons” Example

Gerti Kappel, 180870, 188096, gerti@big.tuwien.ac.at

Andreas, Pieris, apieris@inf.ed.ac.uk, 740072, 184943

Wolfgang Fischl, wfischl@dbai.tuwien.ac.at, 740050

Bill, Robert, 188316, bill@big.tuwien.ac.at

- **There is structure**
 - Each row is a semantic entity - **person**
 - All entities are grouped in a class - **persons**
- **But not too much structure**
 - Entities have no regular structure
 - Structure of future entities is unpredictable

Data Model

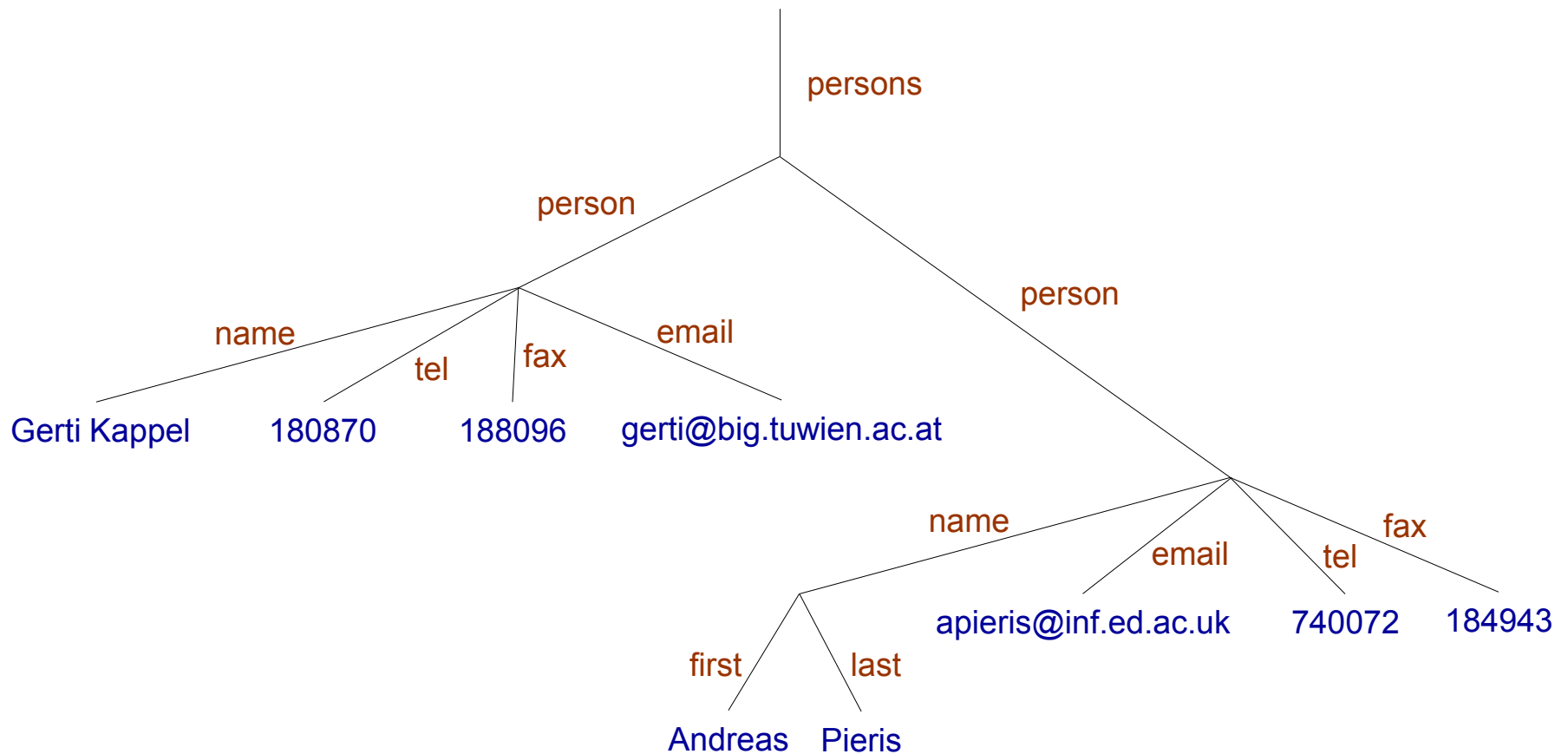
- We need an effective way **to represent semi-structured data**
- Like the relational model for structured data

... any ideas?

Trees as Data Model

Gerti Kappel, 180870, 188096, gerti@big.tuwien.ac.at

Andreas, Pieris, apieris@inf.ed.ac.uk, 740072, 184943



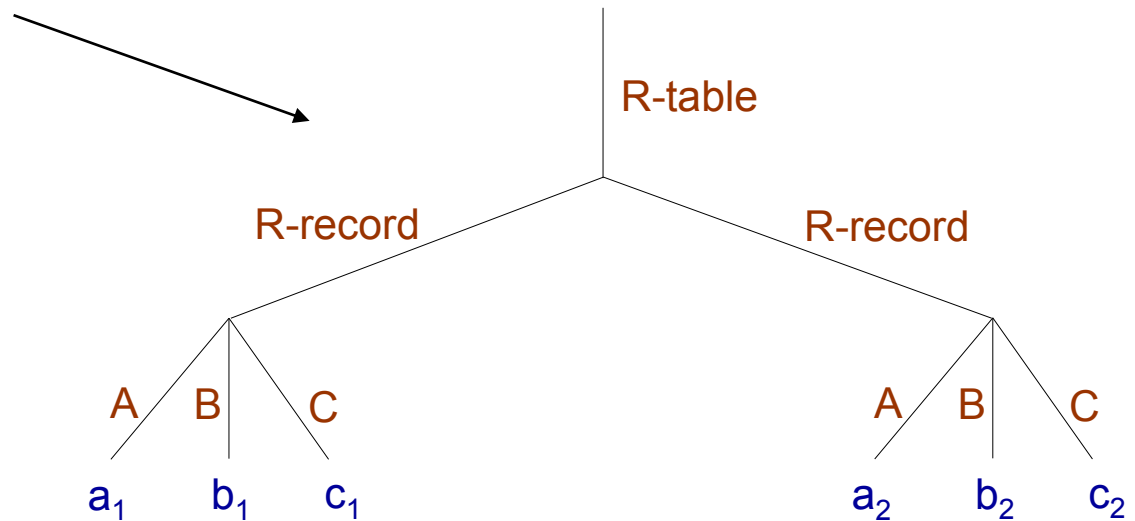
SSD: Representing Relational Data

Structured data is a **special case** of semi-structured data



relational data can be represented as a tree (with an overhead)

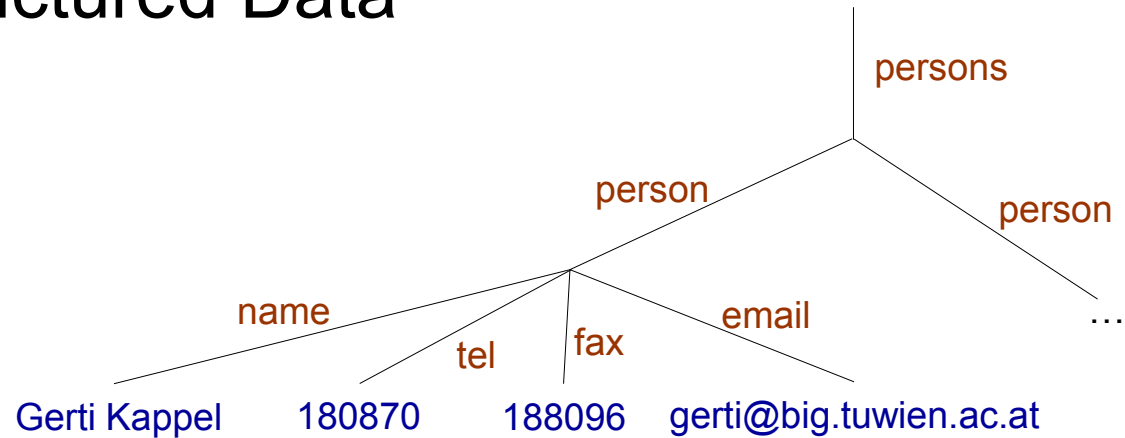
R	A	B	C
	a ₁	b ₁	c ₁
	a ₂	b ₂	c ₂



Store Semi-structured Data

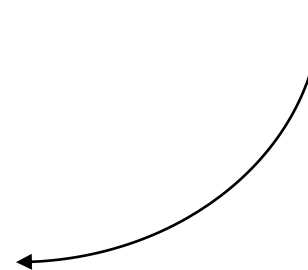
- There are **various formalisms** to store semi-structured data
 - Object Exchange Model (OEM)
 - JavaScript Object Notation (JSON)
 - eXtensible Markup Language (XML)

Store Semi-structured Data

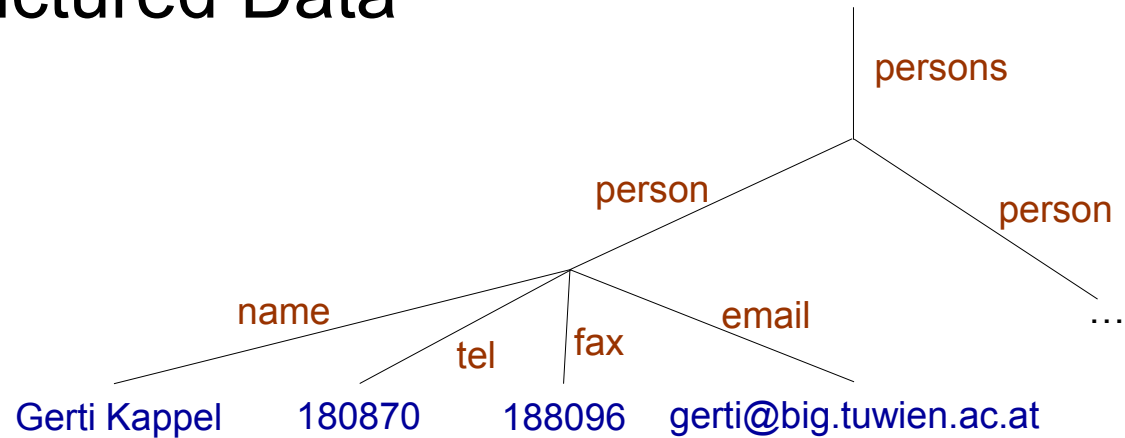


```
{persons:  
  {person:  
    {name: "Gerti Kappel"  
      tel: 180870  
      fax: 188096  
      email: "gerti@big.tuwien.ac.at"}  
  }  
  {person:  
    {name:  
      {first: "Andreas",  
        last: "Pieris"}  
      email: "apieris@inf.ed.ac.uk"  
      tel: 740072  
      fax: 184943}  
  }  
}
```

OEM Representation

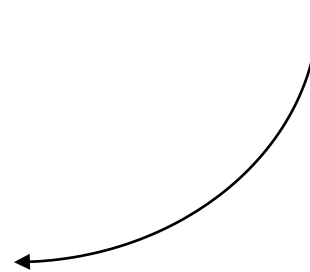


Store Semi-structured Data



```
<persons>
  <person>
    <name>Gerti Kappel</name>
    <tel>80870</tel>
    <fax>188096</fax>
    <email>gerti@big.tuwien.ac.at</email>
  </person>
  <person>
    <name>
      <first>Andreas</first>
      <last>Pieris</last>
    </name>
    <email>apieris@inf.ed.ac.uk</email>
    <tel>740072</tel>
    <fax>184943</fax>
  </person>
</persons>
```

XML Representation



Store Semi-structured Data

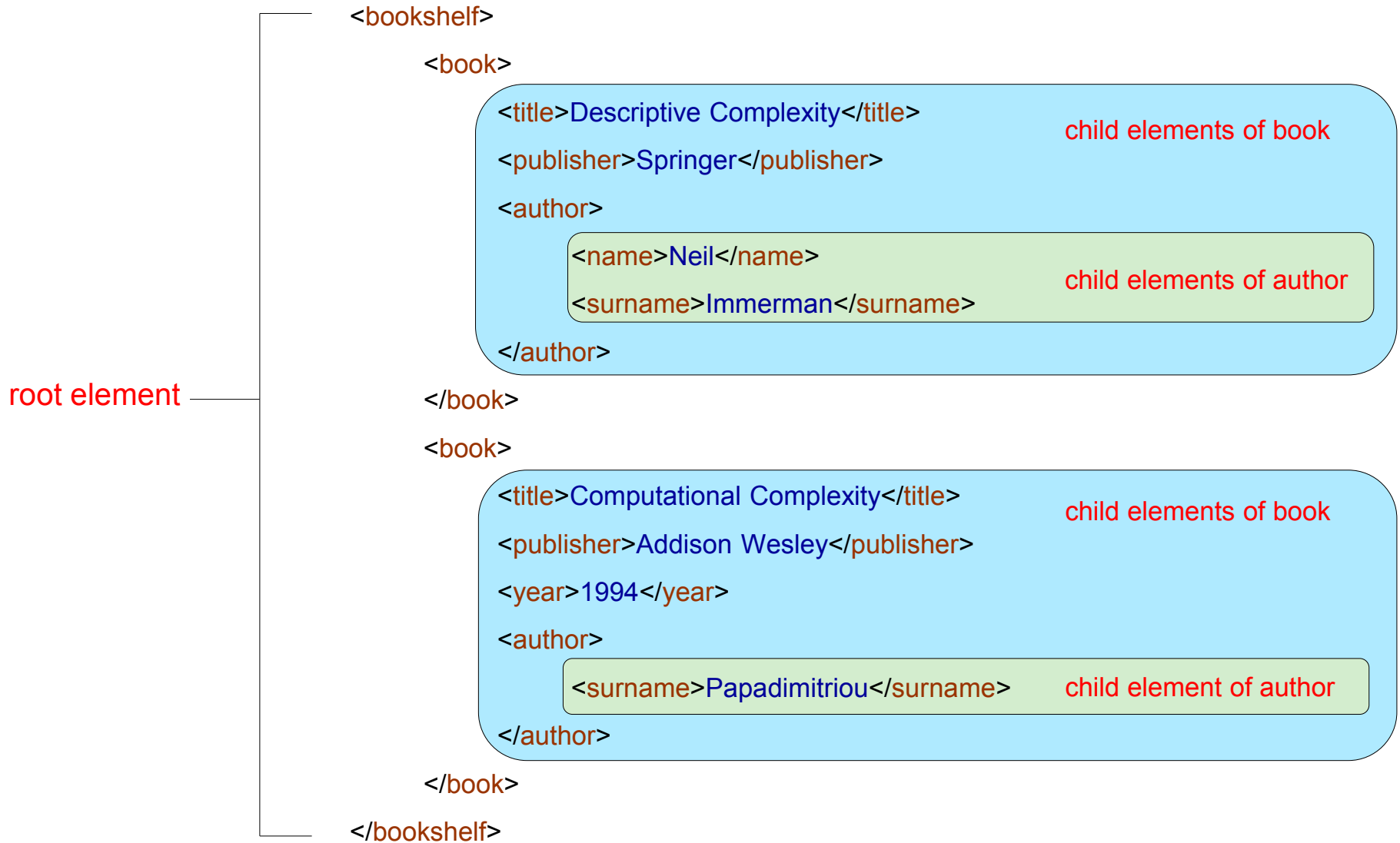
- There are **various formalisms** to store semi-structured data
 - Object Exchange Model (OEM)
 - JavaScript Object Notation (JSON)
 - eXtensible Markup Language (XML)
- Different syntax
- Different mechanisms for self-describing
- Different description mechanisms
 - Which attributes are allowed/required
 - Which values are allowed/required
- Different query languages and manipulation mechanisms

but the goal is the same:

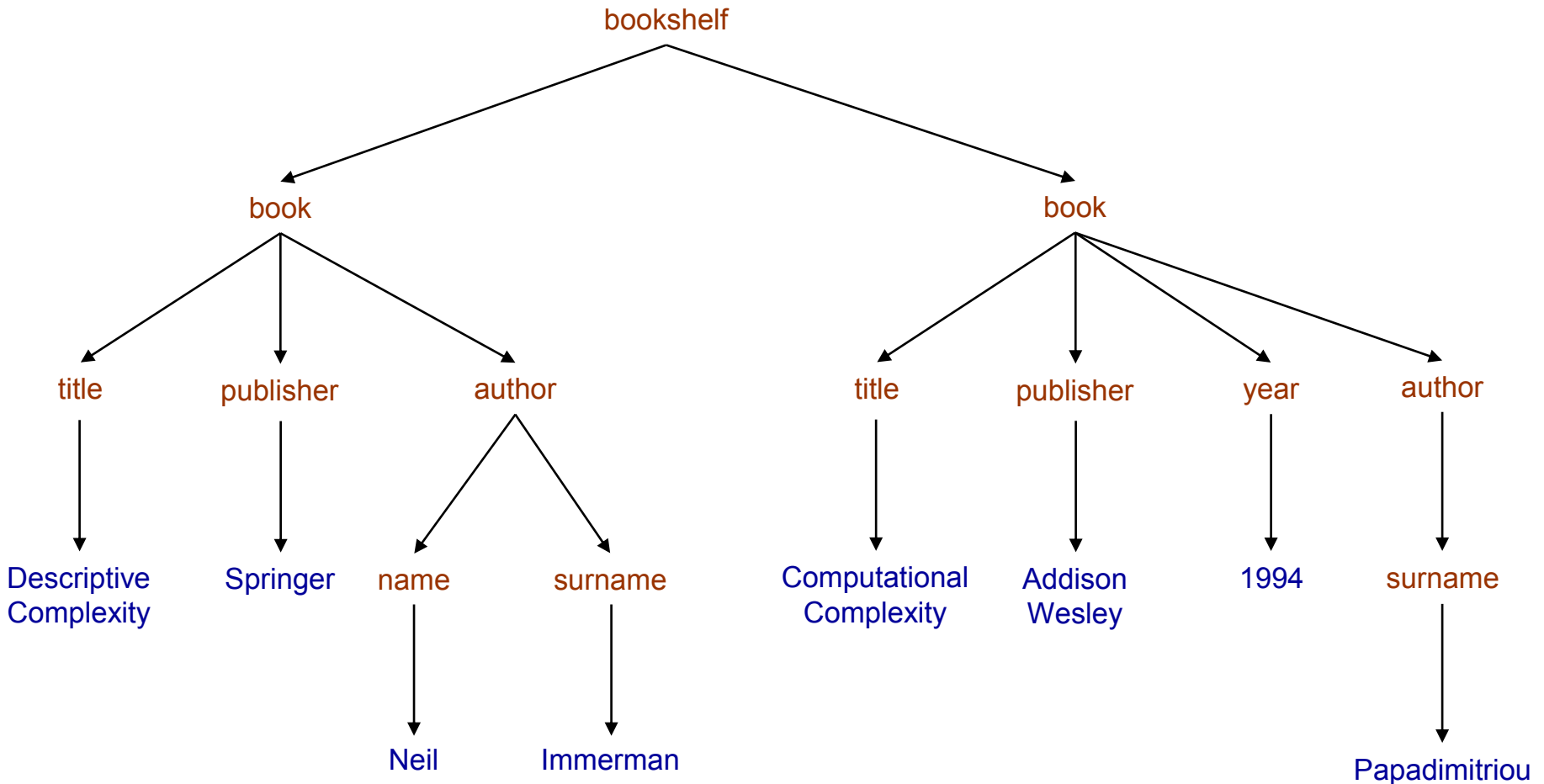
store SSD

Foundations of XML

XML at First Glance



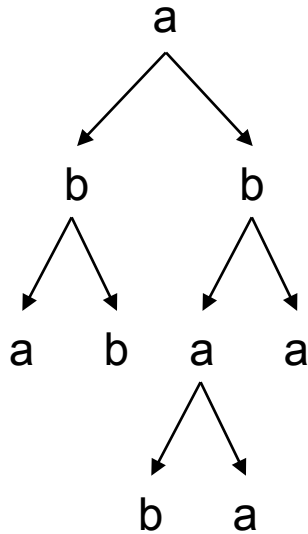
XML Documents as Trees



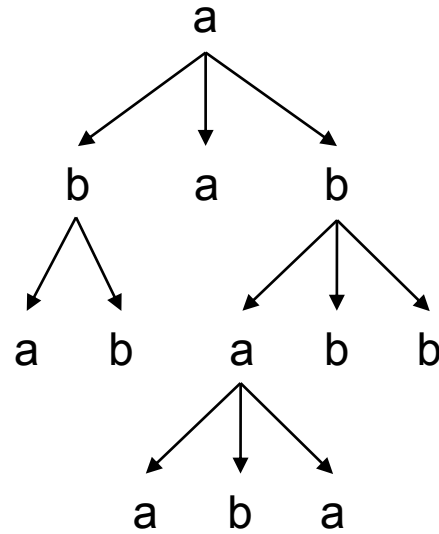
labeled ordered unranked tree

Ranked vs. Unranked Trees

Typically in computer science one works with **ranked trees**, e.g.,



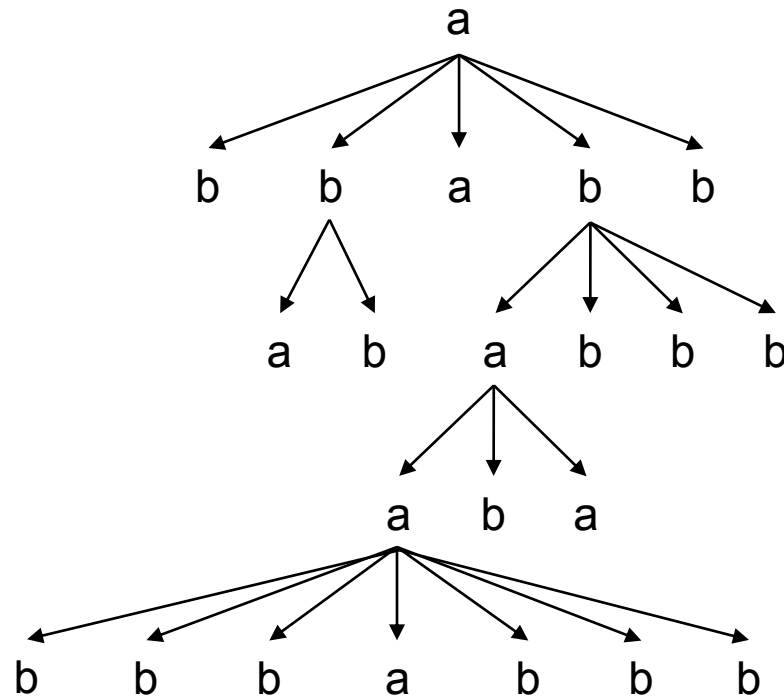
binary trees



ternary trees

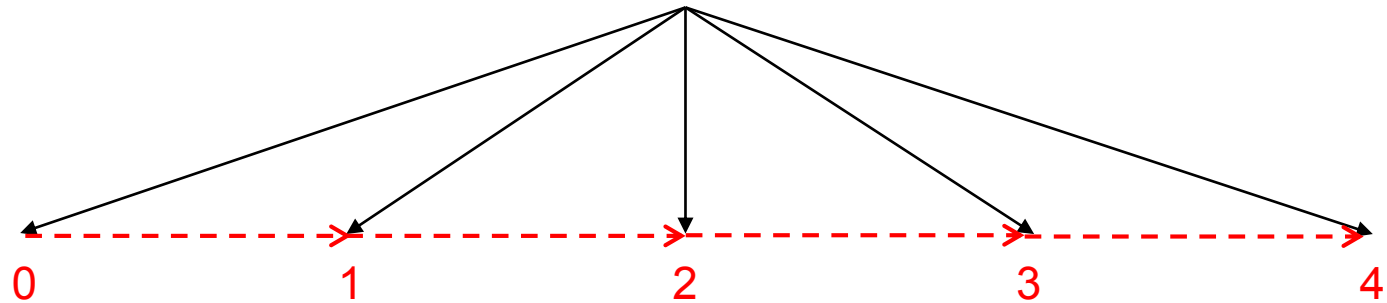
Ranked vs. Unranked Trees

But for XML we need **unranked trees** – nodes can have arbitrarily many children



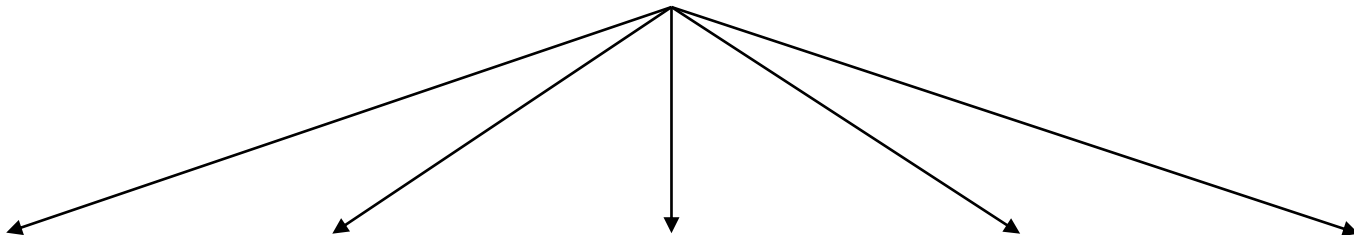
Ordered vs. Unordered Trees

In **ordered trees**, siblings are ordered (from the oldest to the youngest)



a “build-in” binary relation provides access to this ordering

In **unordered trees**, such an order among siblings does not exist



XML Development

- Clean and simple model – **labeled ordered unranked trees**
- Declarative languages – **based on logic**
- Procedural languages – **automata-theoretic constructions**
- Key advantages (like the relational model)
 - Simple and clean mathematical model (based on **logic**)
 - Separation of declarative and procedural

Ordered Unranked Trees: Definition

Fix a finite alphabet Λ

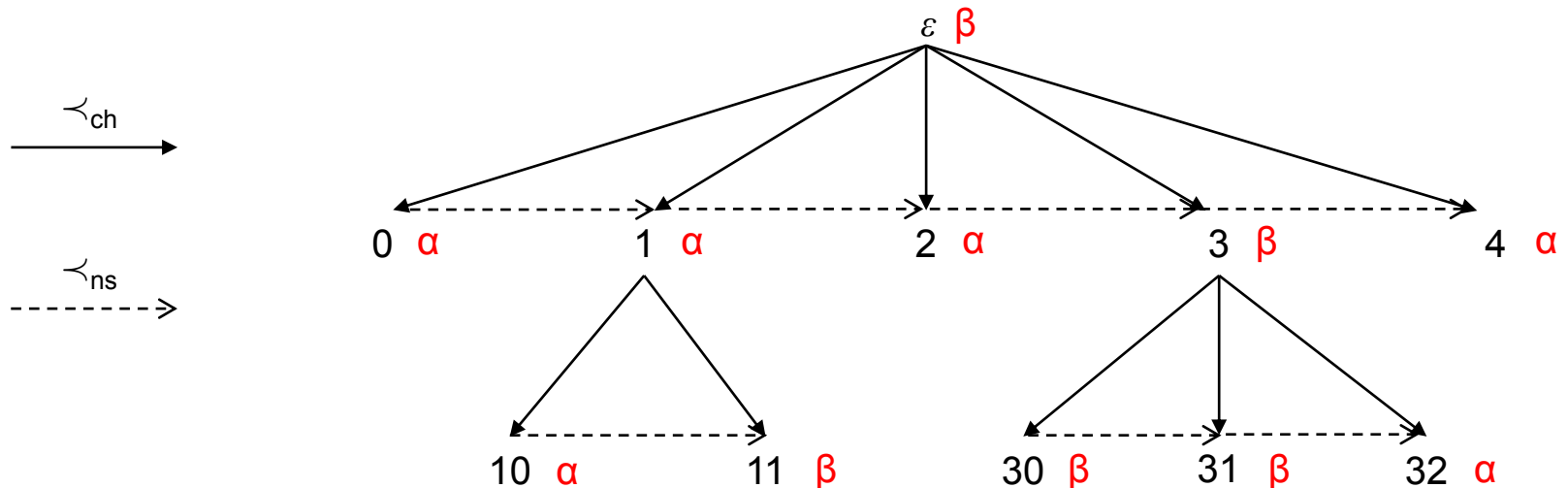
An **ordered unranked tree** T is a structure

$$(\mathbf{D}, \prec_{\text{ch}^*}, \prec_{\text{ns}^*}, \{P_s\}_{s \in \Lambda})$$

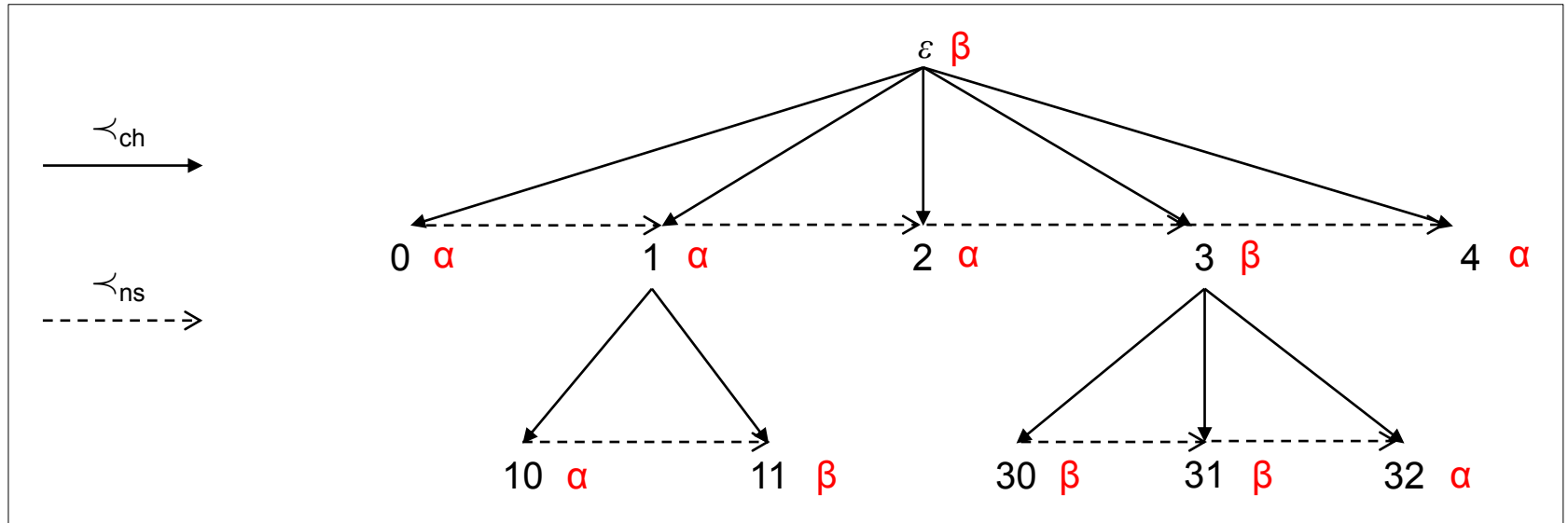
- \mathbf{D} is a finite prefix-closed subset of N^* such that $s \cdot i \in \mathbf{D} \Rightarrow s \cdot j \in \mathbf{D}$ for every $j < i$
- \prec_{ch^*} is the descendant relation
- \prec_{ns^*} is the sibling relation
- P_s 's are interpreted as disjoint sets whose union is the entire domain \mathbf{D}

Ordered Unranked Trees: Example

- Let $\Lambda = \{\alpha, \beta\}$
- Consider the ordered unranked tree $T = (\mathbf{D}, \prec_{\text{ch}}^*, \prec_{\text{ns}}^*, \{P_s\}_{s \in \Lambda})$, where
 - $\mathbf{D} = \{\varepsilon, 0, 1, 2, 3, 4, 10, 11, 30, 31, 32\}$
 - $\prec_{\text{ch}} = \{(\varepsilon, 0), (\varepsilon, 1), (\varepsilon, 2), (\varepsilon, 3), (\varepsilon, 4), (1, 10), (1, 11), (3, 30), (3, 31), (3, 32)\}$
 - $\prec_{\text{ns}} = \{(0, 1), (1, 2), (2, 3), (3, 4), (10, 11), (30, 31), (31, 32)\}$
 - $P_\alpha = \{0, 1, 2, 4, 10, 32\}$
 - $P_\beta = \{\varepsilon, 3, 11, 30, 31\}$



Ordered Unranked Trees: Basic Predicates



In $T = (\mathbf{D}, \prec_{ch}^*, \prec_{ns}^*, \{P_s\}_{s \in \Lambda})$ we use the **transitive closures** of \prec_{ch} and \prec_{ns}

- They are not definable in first-order logic
- However, if the adopted logic is powerful enough to define them, then we can simply use \prec_{ch} and \prec_{ns}

Ordered Unranked Trees: Querying

Check that in a tree T over the alphabet $\{\alpha, \beta\}$ every α -labeled node always has a β -labeled descendant

$$Q = \forall x (P_\alpha(x) \rightarrow \exists y (x \prec_{\text{ch}^*} y \wedge P_\beta(y)))$$

Ordered Unranked Trees: Querying

Select the nodes in a tree T over the alphabet $\{\alpha, \beta, \gamma\}$ that are

- (i) labeled α ,
- (ii) have a descendant d labeled β , and
- (iii) d has a younger sibling labeled γ

$$Q(x) = P_{\alpha}(x) \wedge \exists y \exists z (x \prec_{\text{ch}^*} y \wedge P_{\beta}(y) \wedge y \prec_{\text{ns}^*} z \wedge P_{\gamma}(z))$$

Ordered Unranked Trees: Querying

Check that in a tree T over the alphabet $\{\alpha, \beta\}$ every α -labeled node always has a β -labeled descendant, but **using only** \prec_{ch}

any **set of nodes** that contains x and is closed under the \prec_{ch} relation, also contains y

$$Q = \forall x(P_\alpha(x) \rightarrow \exists y(\text{desc}(x,y) \wedge P_\beta(y)))$$

$$\text{desc}(x,y) = \forall S((x \in S \wedge \forall z \forall w((z \in S \wedge z \prec_{\text{ch}} w) \rightarrow w \in S)) \rightarrow y \in S)$$

second-order quantifier
ranging over sets of nodes

first-order quantifiers
ranging over nodes

monadic second-order logic (MSO)

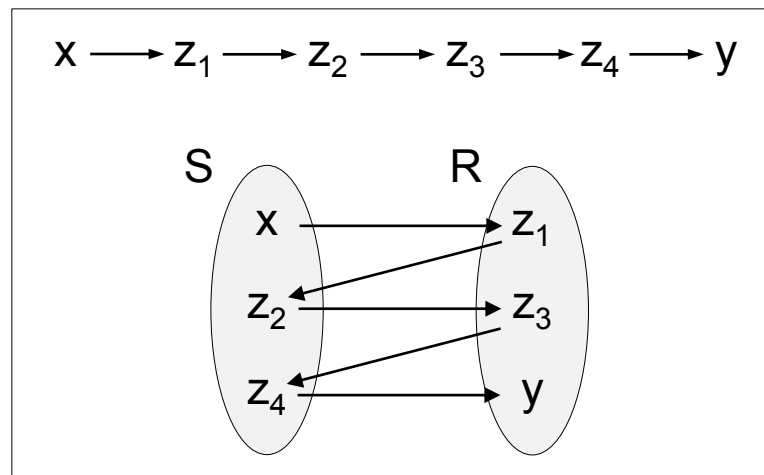
Ordered Unranked Trees: Querying

Compute the pairs of nodes (x,y) such that y is a descendant of x
and the path between them is of odd length

\equiv

There **exist two sets of nodes** S and R that

- (i) partition the path from x to y
- (ii) $x \in S$ and $y \in R$
- (iii) the successor element of each element in S is in R , and vice versa



Ordered Unranked Trees: Querying

Compute the pairs of nodes (x,y) such that y is a descendant of x
and the path between them is of odd length

$Q(x,y) =$


$$\exists S \exists R \left(\begin{array}{l} \forall z ((x \prec_{\text{ch}^*} z \prec_{\text{ch}^*} y) \rightarrow (z \in S \leftrightarrow \neg(z \in R))) \\ \wedge (x \in S \wedge y \in R) \\ \wedge \forall z \forall w ((x \prec_{\text{ch}^*} z \prec_{\text{ch}} w \prec_{\text{ch}^*} y) \rightarrow ((z \in S \rightarrow w \in R) \wedge (z \in R \rightarrow w \in S))) \end{array} \right)$$

Ordered Unranked Trees: Querying

- For querying labeled ordered unranked trees we use:
 - **First-order logic (FO)**
 - Boolean connectives \vee, \wedge, \neg
 - Quantifiers $\exists x$ and $\forall x$ that range over **nodes** of trees
 - **Monadic second-order logic (MSO)**
 - FO plus quantifiers $\exists S$ and $\forall S$ that range over **sets of nodes**
 - New formulae $x \in S$
- Most commonly they define:
 - Boolean (yes/no) queries – in fact, they define **sets of trees**
 - Unary queries that **select nodes** in trees

Ordered Unranked Trees: Definability in Logic

Let L be some logic (such as **FO** or **MSO**)

- A Boolean query Q (i.e., a set of trees T) is **L-definable** if there is a sentence φ of L such that $T \in T \Leftrightarrow T \models \varphi$
- A unary query $Q(x)$ is **L-definable** if there is a formula $\varphi(x)$ of L such that for every tree T and node v in T , $v \in Q(T) \Leftrightarrow T \models \varphi(v)$


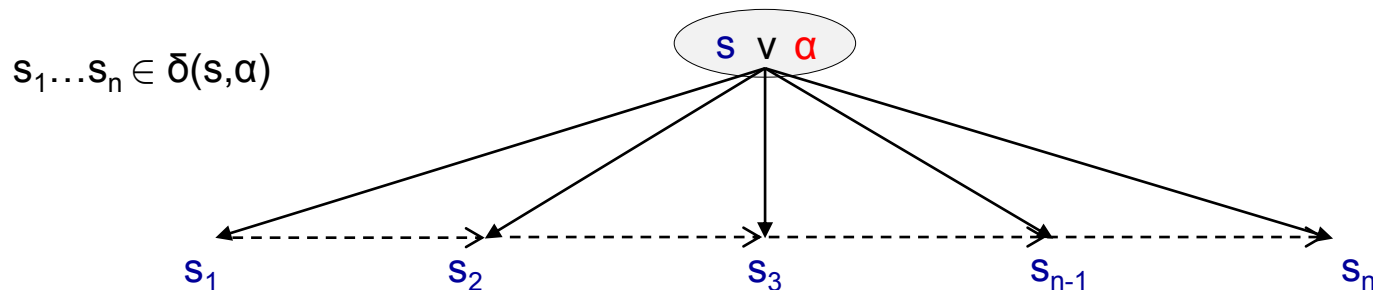
the set of nodes in T selected by Q

Unranked Tree Automata

- A **nondeterministic unranked tree automaton (NUTA)** over Λ -labeled trees is a triple

$$A = (S, F, \delta)$$

- S is a finite set of states
 - $F \subseteq S$ is the set of final states
 - $\delta : S \times \Lambda \rightarrow 2^{S^*}$ such that $\delta(s, \alpha)$ is a regular language over S
- A **run of A on a tree T** with domain \mathbf{D} is a function $\lambda_A : \mathbf{D} \rightarrow S$ such that: if v is a node with n children, and is labeled α , then the string $\lambda_A(v \cdot 0) \dots \lambda_A(v \cdot (n-1)) \in \delta(\lambda_A(v), \alpha)$



Unranked Tree Automata

- A **nondeterministic unranked tree automaton (NUTA)** over Λ -labeled trees is a triple

$$A = (S, F, \delta)$$

- S is a finite set of states
 - $F \subseteq S$ is the set of final states
 - $\delta : S \times \Lambda \rightarrow 2^{S^*}$ such that $\delta(s, \alpha)$ is a regular language over S
- A **run of A on a tree T** with domain \mathbf{D} is a function $\lambda_A : \mathbf{D} \rightarrow S$ such that: if v is a node with n children, and is labeled α , then the string $\lambda_A(v \cdot 0) \dots \lambda_A(v \cdot (n-1)) \in \delta(\lambda_A(v), \alpha)$
 - A run is **accepting** if $\lambda_A(\varepsilon) \in F$, i.e., the root is in an accepting state
 - A **tree T is accepted** by A if there exists an accepting run of A on T
 - We denote by $L(A)$ the set of all trees accepted by A – a set of trees accepted by an NUTA is called **regular**

Unranked Tree Automata: Example

- Let $\Lambda = \{\wedge, \vee, 0, 1\}$, and consider Λ -labeled trees where 0,1 appear only at leaves, while \wedge, \vee can appear everywhere except at leaves
- We define $A = (\{s_0, s_1\}, \{s_1\}, \bar{\delta})$, where

$$\bar{\delta}(s_0, 0) = \bar{\delta}(s_1, 1) = \{\varepsilon\}$$

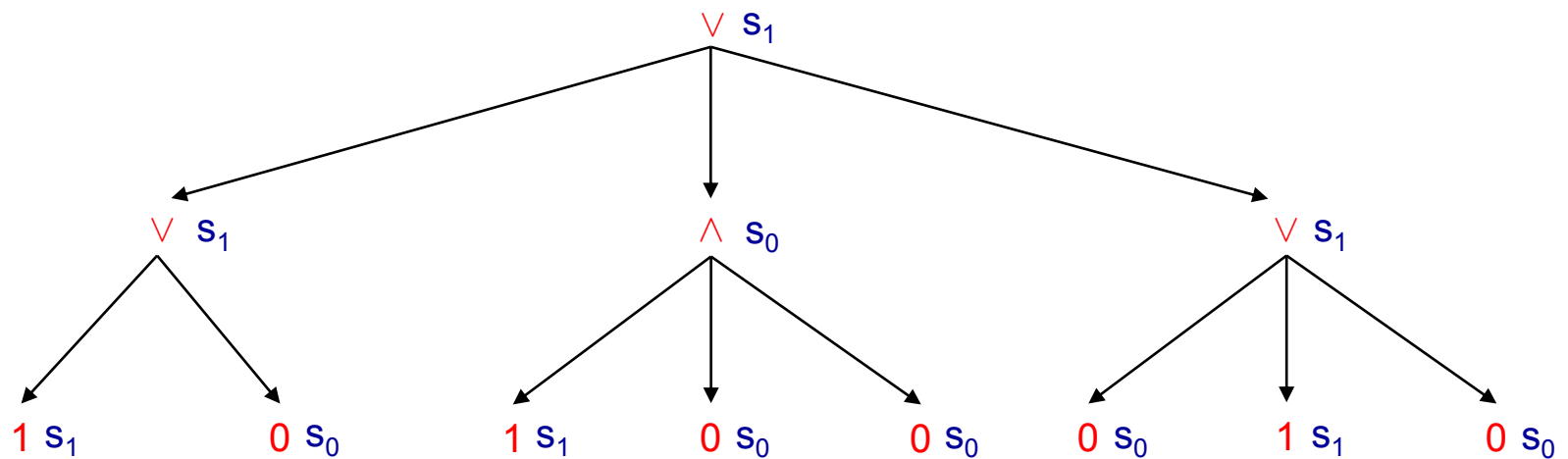
$$\bar{\delta}(s_0, 1) = \bar{\delta}(s_1, 0) = \emptyset$$

$$\bar{\delta}(s_0, \wedge) = (s_0 \cup s_1)^* \cdot s_0 \cdot (s_0 \cup s_1)^*$$

$$\bar{\delta}(s_1, \wedge) = s_1^*$$

$$\bar{\delta}(s_0, \vee) = s_0^*$$

$$\bar{\delta}(s_1, \vee) = (s_0 \cup s_1)^* \cdot s_1 \cdot (s_0 \cup s_1)^*$$



MSO = NUTA

We can now present an interesting result:

Theorem: Consider a set **T** of labeled ordered unranked trees. Then:

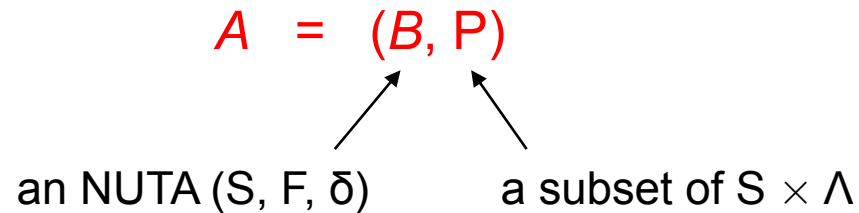
T is **MSO**-definable \Leftrightarrow **T** is regular

...but, what about unary queries?

we need an extended automata model

Query Automata

- A **nondeterministic query automaton (NQA)** over Λ -labeled trees is a pair

$$A = (B, P)$$


an NUTA (S, F, δ) a subset of $S \times \Lambda$

- Such an automaton defines a unary query Q_A over unranked trees:

$$v \in Q_A(T) \Leftrightarrow (\lambda_B(v), \text{label}(v)) \in P, \text{ for some accepting run } \lambda_B \text{ of } B \text{ on } T$$

MSO = NQA

We have similar characterization for unary queries:

Theorem: Consider a unary query Q on labeled ordered unranked trees. Then:

Q is **MSO**-definable $\Leftrightarrow Q$ is of the form Q_A for some NQA A

Ordered Unranked Trees: Recap

- XML documents are modeled as labeled ordered unranked trees
- **MSO** is the yardstick logic for querying ordered unranked trees
- Most commonly we consider:
 - Boolean queries that they define sets of trees: **MSO** = NUTA
 - Unary queries that select nodes in trees: **MSO** = NQA

...but, what about the complexity of **MSO** over trees?

Complexity of MSO

BQE(MSO)

Input: a labeled ordered unranked tree T , an **MSO** sentence φ

Question: $T \models \varphi$?

- The same problem can be defined for **unary formulas**
 - Given a tree T , a unary formula $\varphi(x)$, and a node v : does $T \models \varphi(v)$?
- As usual, we consider the **data** and **combined complexity**
 - Data complexity: T is input, φ is fixed
 - Combined complexity: both T and φ are part of the input

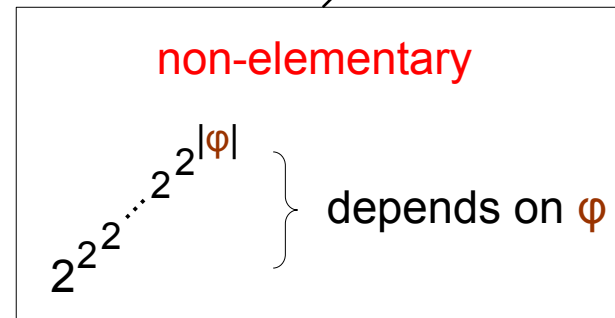
Complexity of MSO

Theorem: It holds that:

- BQE(**MSO**) is in PTIME in data complexity (in fact, linear time)
- BQE(**MSO**) is non-elementary in combined complexity

Proof idea: By translation to automata:

- Convert the given sentence φ into a NUTA A_φ such that $T \models \varphi \Leftrightarrow T \in L(A_\varphi)$
- To decide whether $T \in L(A_\varphi)$ is feasible in time $O(|T| \cdot |A_\varphi|^2)$



Complexity of MSO

Theorem: It holds that:

- $\text{BQE}(\mathbf{MSO})$ is in PTIME in data complexity (in fact, linear time)
- $\text{BQE}(\mathbf{MSO})$ is non-elementary in combined complexity

Proof idea: By translation to automata:

- Convert the given sentence φ into a NUTA A_φ such that $T \models \varphi \Leftrightarrow T \in L(A_\varphi)$
- To decide whether $T \in L(A_\varphi)$ is feasible in time $O(|T| \cdot |A_\varphi|^2)$

Even a bigger problem: there is no algorithm (even if we avoid automata) for checking whether $T \models \varphi$ that runs in time $O(|T| \cdot f(|\varphi|))$ and f is an elementary function (unless $P = NP$)

Complexity of MSO

Theorem: It holds that:

- $\text{BQE}(\mathbf{MSO})$ is in PTIME in data complexity (in fact, linear time)
- $\text{BQE}(\mathbf{MSO})$ is non-elementary in combined complexity

Proof idea: By translation to automata:

- Convert the given sentence φ into a NUTA A_φ such that $T \models \varphi \Leftrightarrow T \in L(A_\varphi)$
- To decide whether $T \in L(A_\varphi)$ is feasible in time $O(|T| \cdot |A_\varphi|^2)$

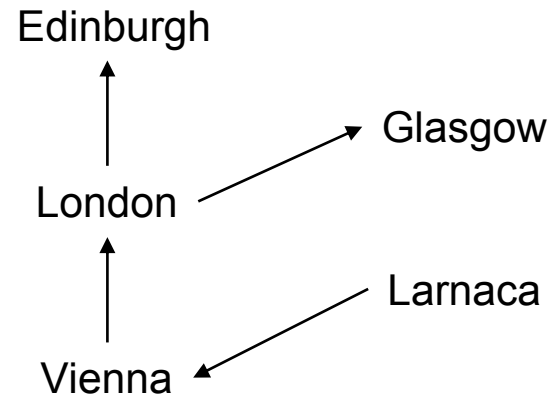
We need logics that have the **same power** as **MSO**, but
permit **faster** evaluation algorithms

Alternative Logics for MSO

- **Efficient Tree Logic (ETL)** – obtained by posing some syntactic restrictions on **MSO** formulae, and at the same time adding new constructors for formulae that are not in **MSO**, but are **MSO**-definable
 - **μ -calculus** – extension of a temporal logic with the least fixed-point operator
- **Monadic Datalog** – fragment of Datalog, a database query language that essentially extends existential positive **FO** with the least fixed-point operator

Reachability

Is Glasgow reachable from Vienna?

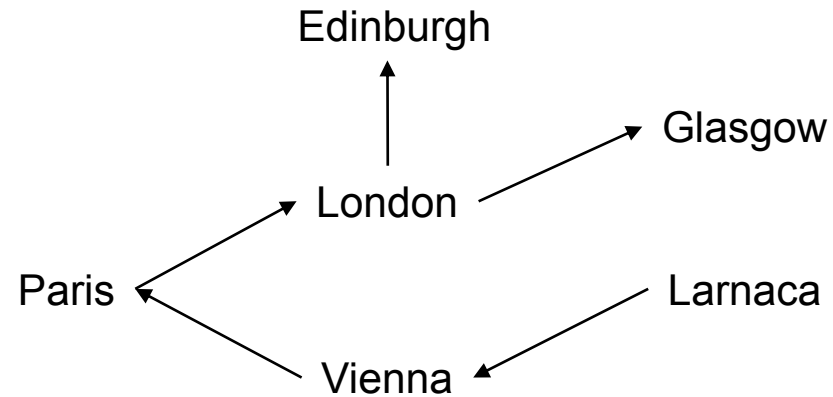


$\exists x (\text{Flight}(\text{Vienna}, x) \wedge \text{Flight}(x, \text{Glasgow}))$



Reachability

Is Glasgow reachable from Vienna?

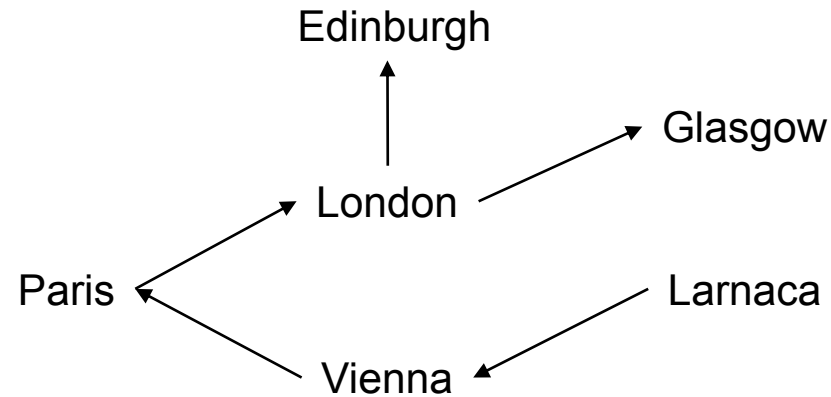


$\exists x (\text{Flight}(\text{Vienna}, x) \wedge \text{Flight}(x, \text{Glasgow}))$

x

Reachability

Is Glasgow reachable from Vienna?

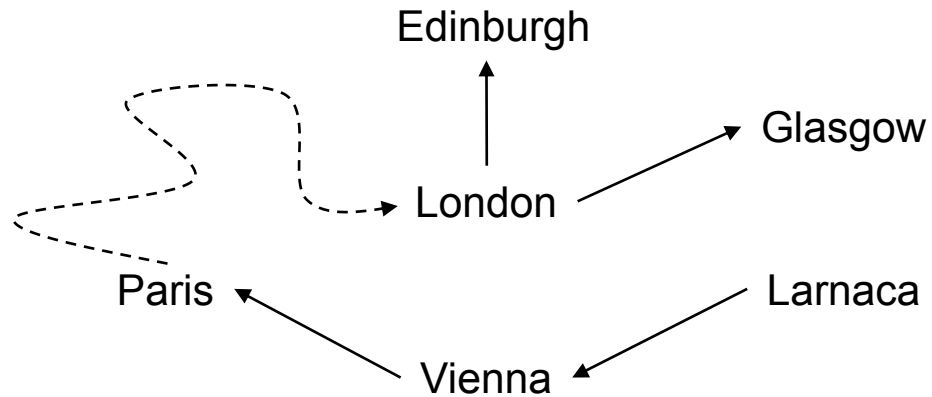


$\exists x \exists y (Flight(Vienna, x) \wedge Flight(x, y) \wedge Flight(y, Glasgow))$



Reachability

Is Glasgow reachable from Vienna?

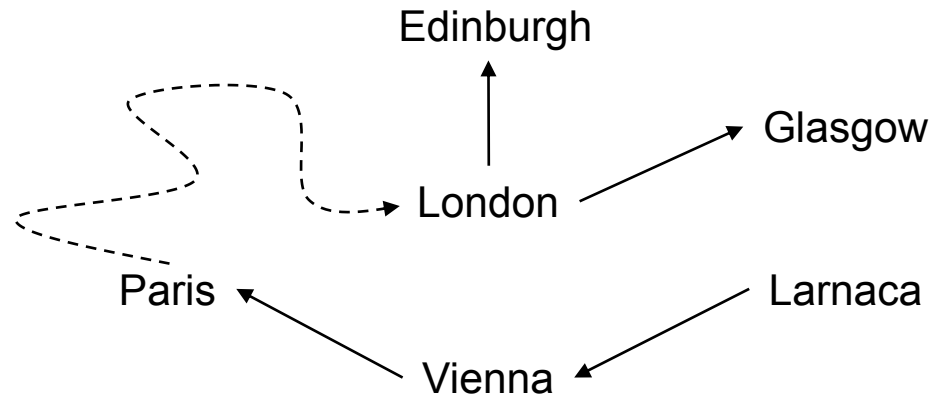


$\exists x \exists y (Flight(Vienna, x) \wedge Flight(x, y) \wedge Flight(y, Glasgow))$

x

Reachability

Is Glasgow reachable from Vienna?

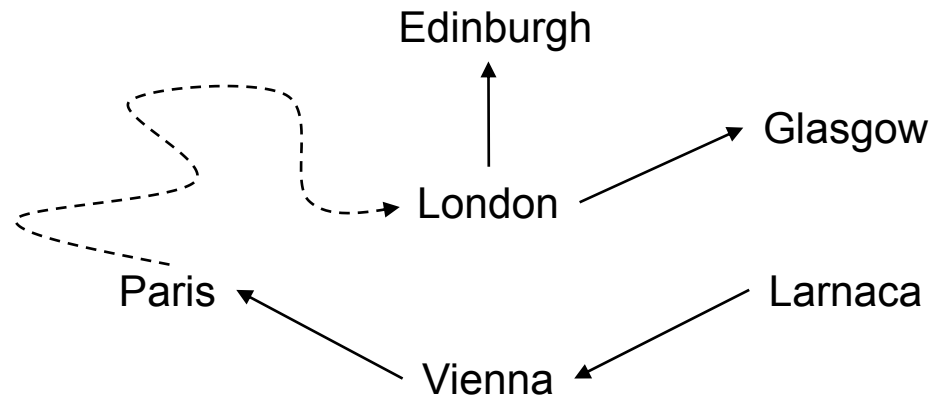


Here is a possible strategy:

- Compute all the pairs of cities (c_1, c_2) such that c_2 is reachable from c_1
- Check if there is a pair $(\text{Vienna}, \text{Glasgow})$

Reachability

Is Glasgow reachable from Vienna?



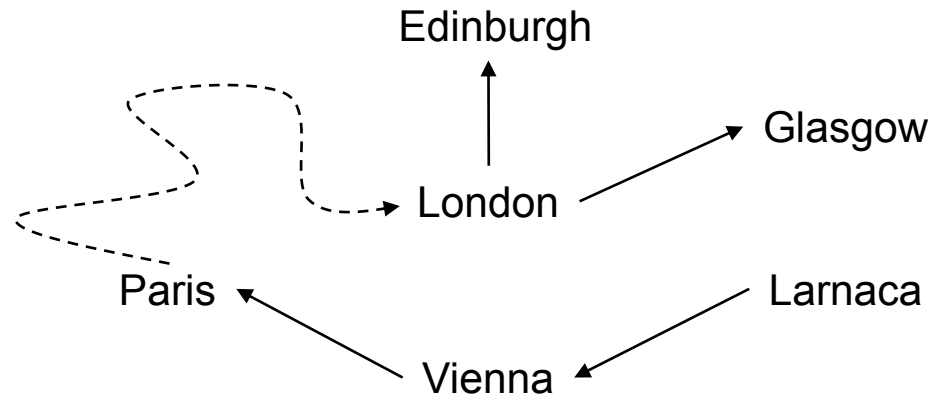
Here is a possible strategy:

- Compute all the pairs of cities (c_1, c_2) such that c_2 is reachable from c_1
- Check if $(vienna, Glasgow)$ is in the set

not possible via an FO query – we need recursion

Reachability

Is Glasgow reachable from Vienna?



$\text{Reachable}(x,y) \text{ :- Flight}(x,y)$

$\text{Reachable}(x,z) \text{ :- Flight}(x,y), \text{Reachable}(y,z)$

$\text{Goal} \text{ :- Reachable}(\text{Vienna}, \text{Glasgow})$

Reachability

DATALOG

essentially, positive **FO** with least fixed-point

Reachable(x,y) :- Flight(x,y)

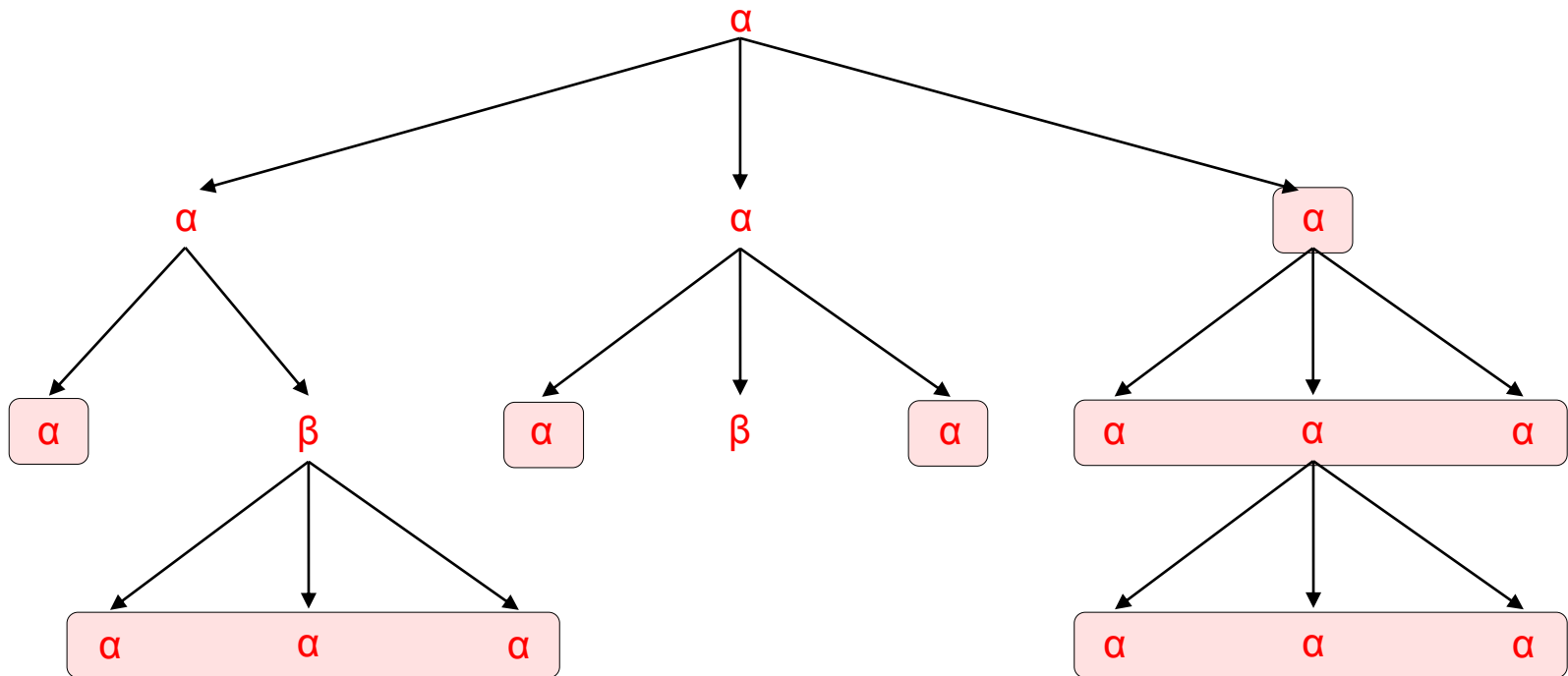
Reachable(x,z) :- Flight(x,y), Reachable(y,z)

Goal :- Reachable(Vienna,Glasgow)

Monadic Datalog

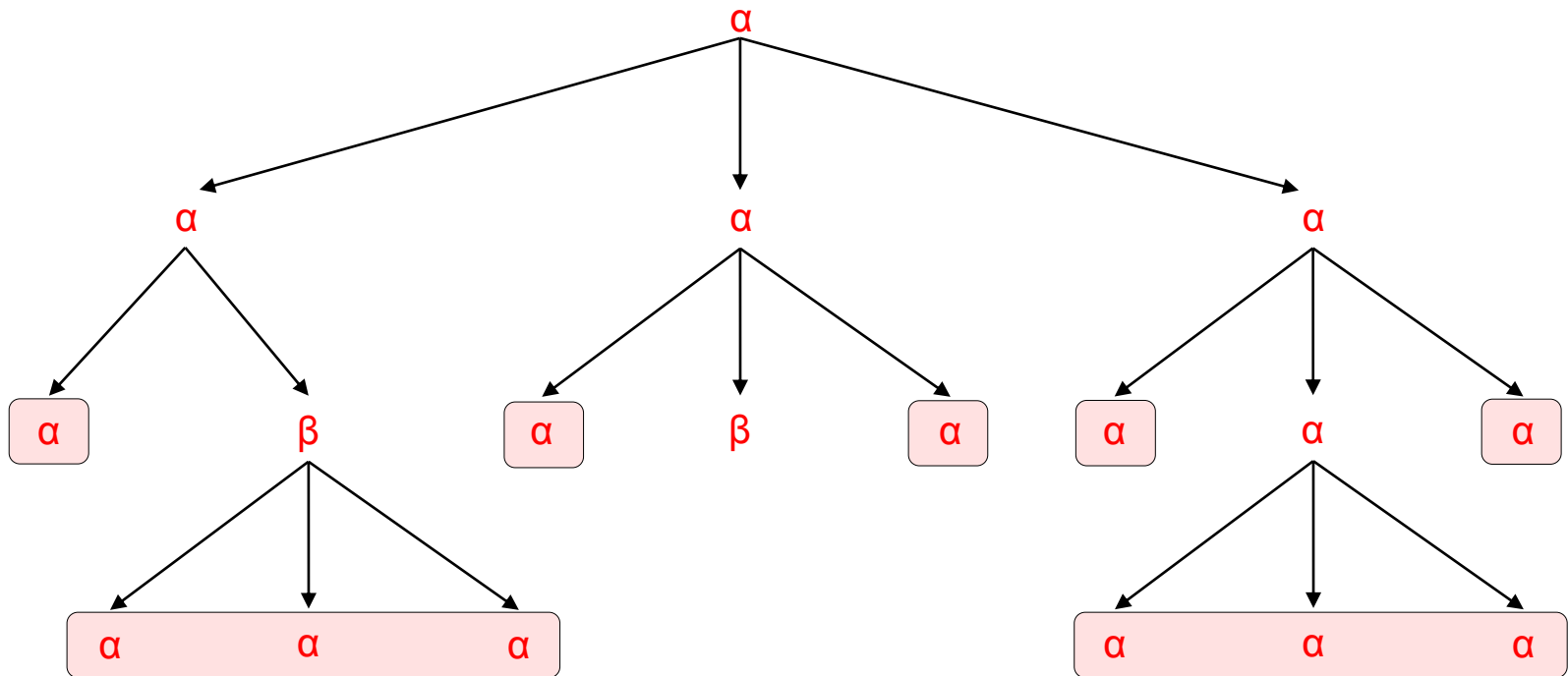
all the introduced (or intentional) predicates are unary

Select all nodes v such that their descendants (including v) are labeled α



Monadic Datalog

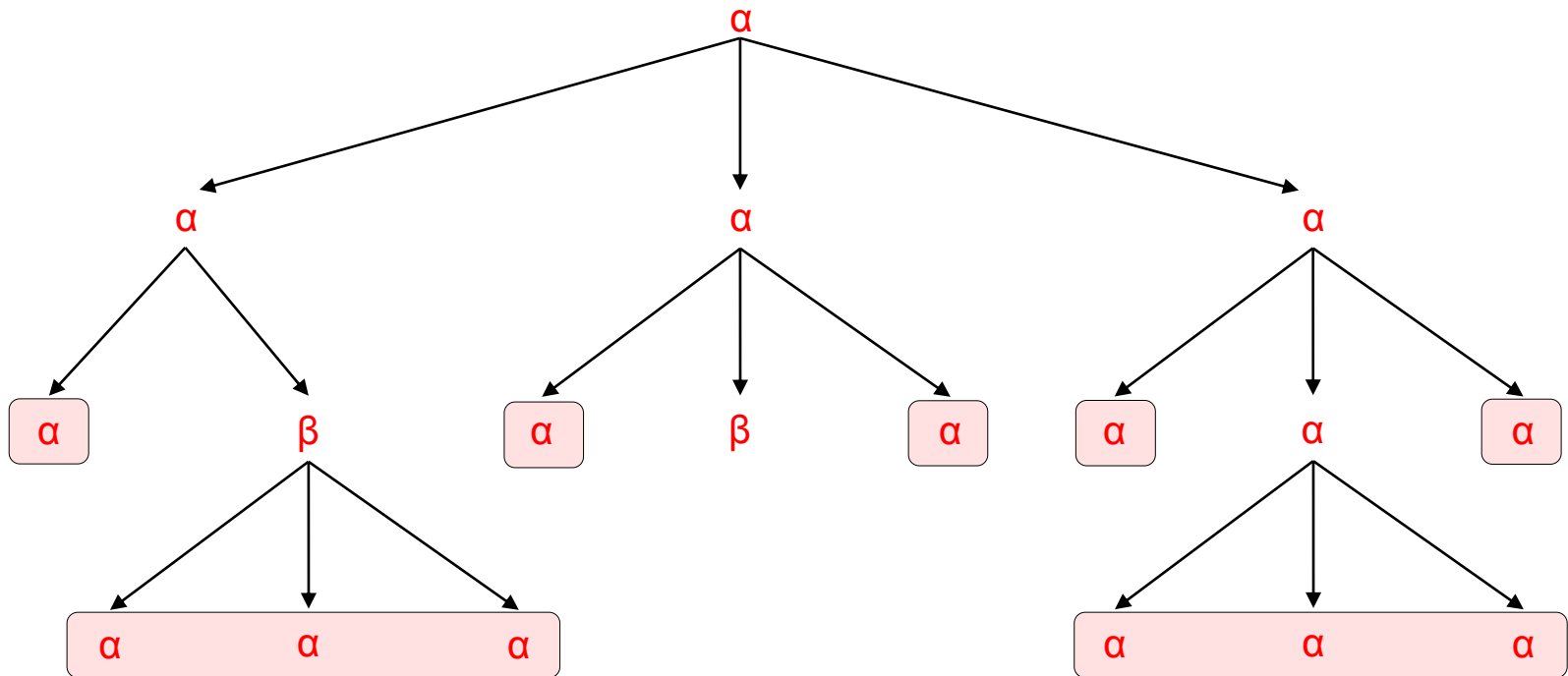
Select(x) :- $P_\alpha(x)$, Leaf(x)



Monadic Datalog

Mark(v) means that

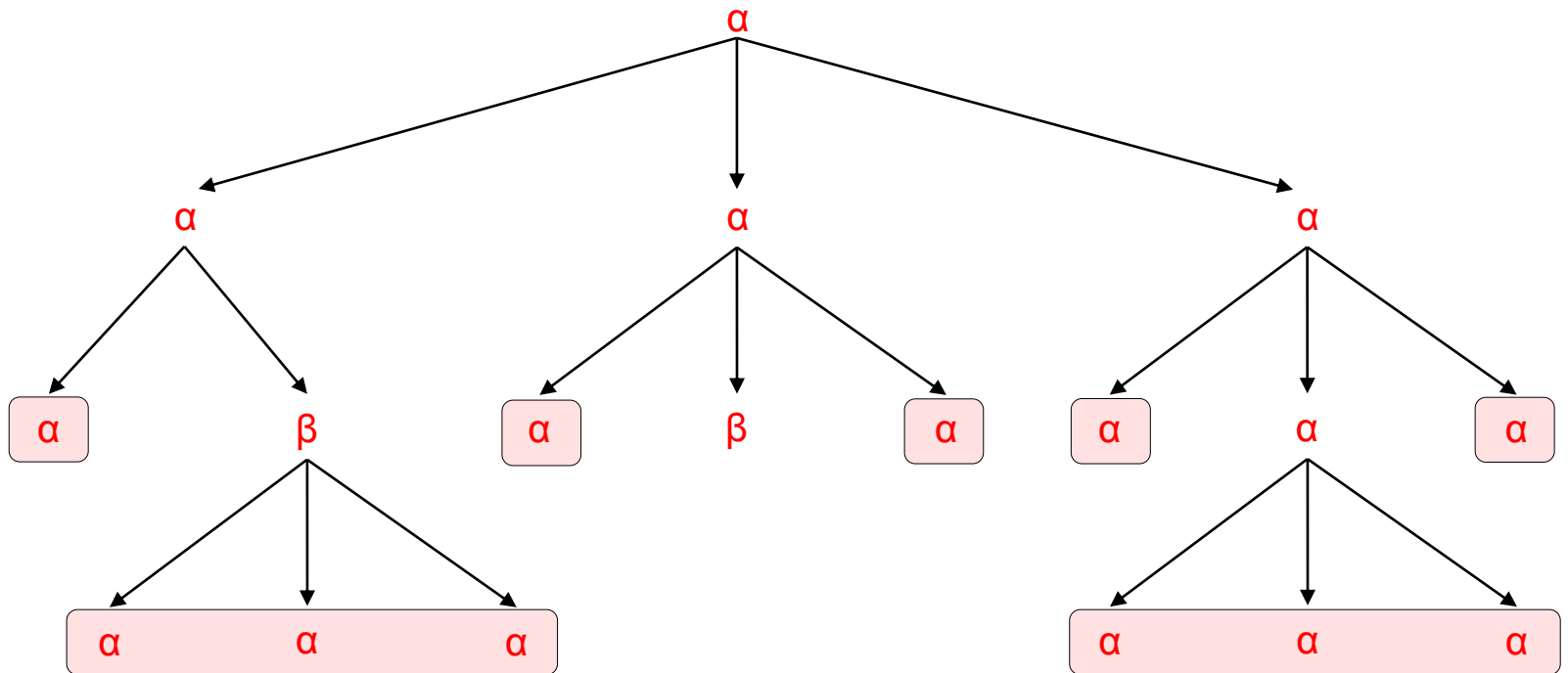
- $\text{Select}(v)$ holds
- For every u such that $v \prec_{\text{ns}^*} u$, $\text{Select}(u)$ holds



Monadic Datalog

$\text{Select}(x) \text{ :- } P_\alpha(x), \text{Leaf}(x)$

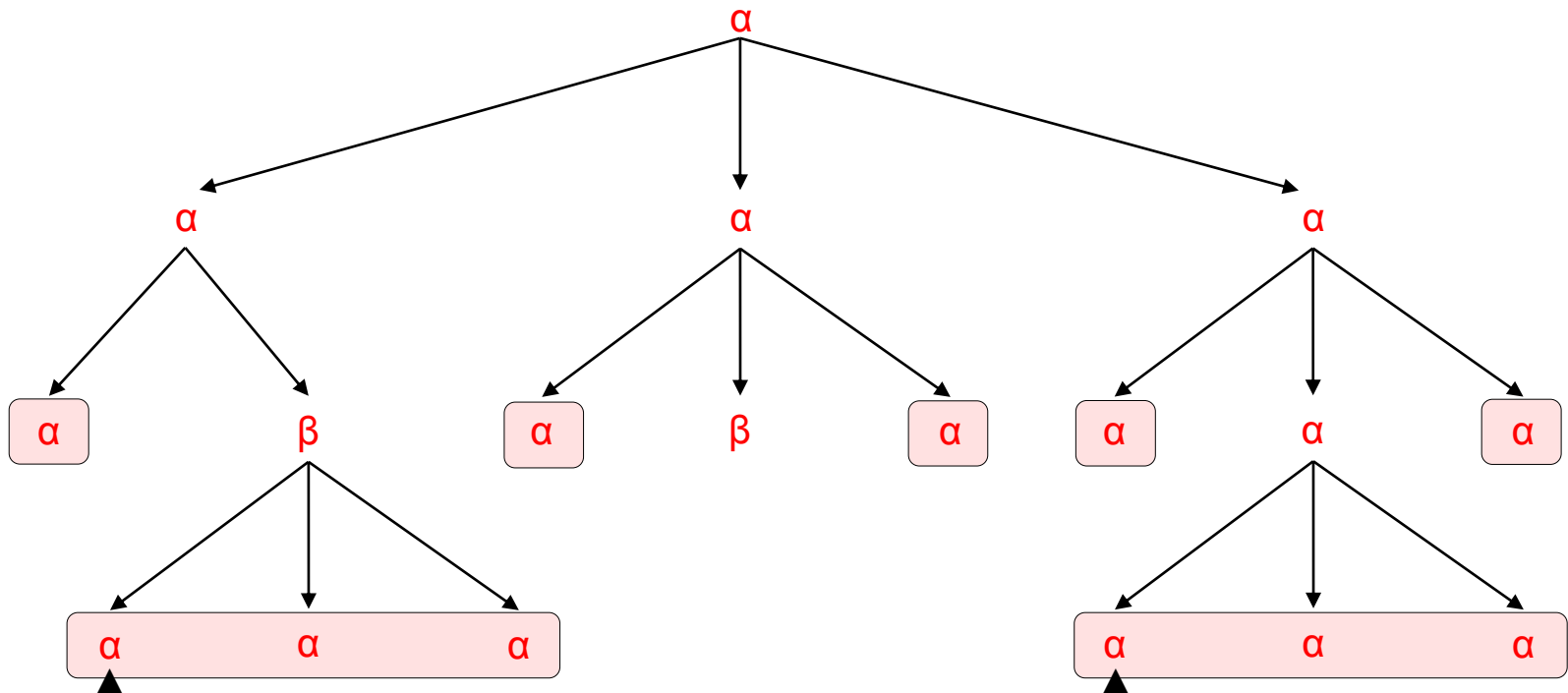
$\text{Select}(x) \text{ :- } P_\alpha(x), x \prec_{\text{fc}} y, \text{Mark}(y)$



Monadic Datalog

Select(x) :- $P_\alpha(x)$, Leaf(x)

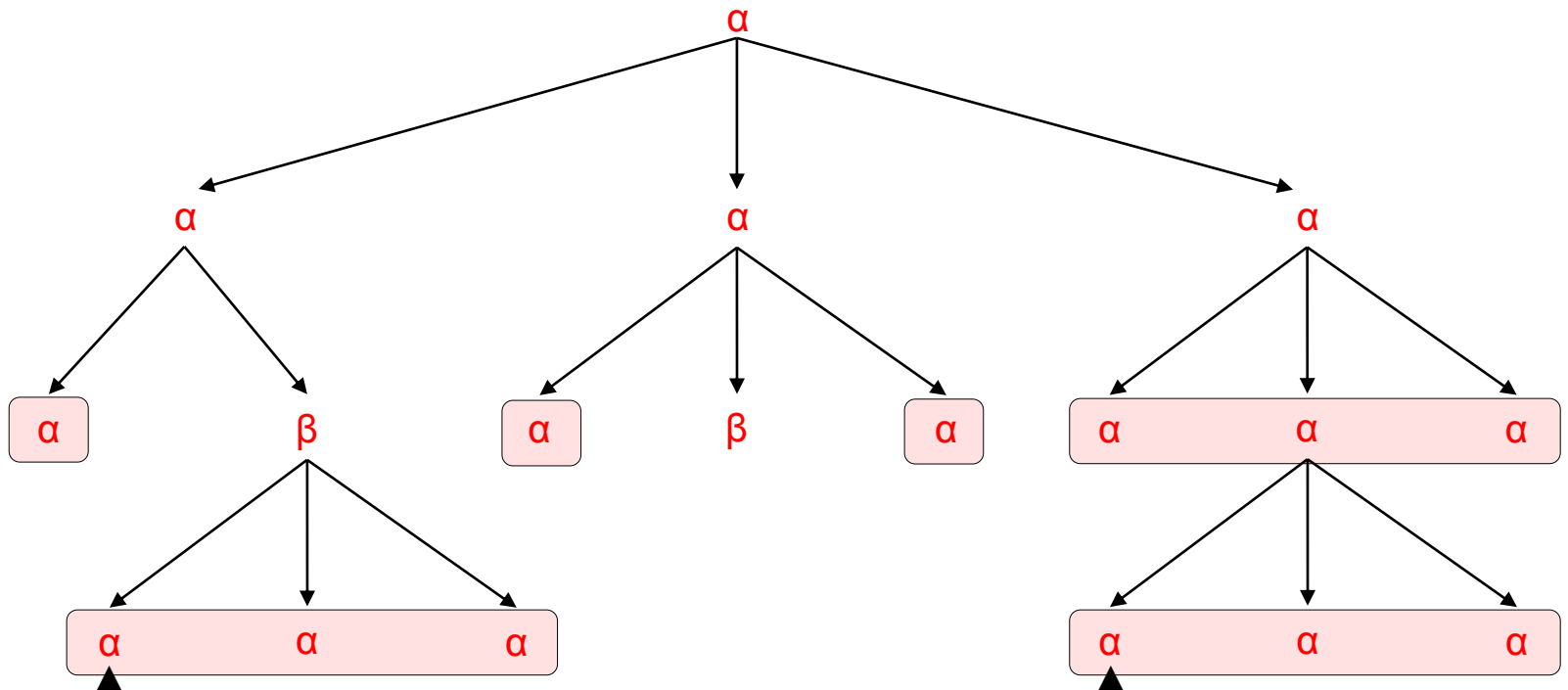
Select(x) :- $P_\alpha(x)$, $x \prec_{fc} y$, Mark(y)



Monadic Datalog

$\text{Select}(x) \text{ :- } P_\alpha(x), \text{Leaf}(x)$

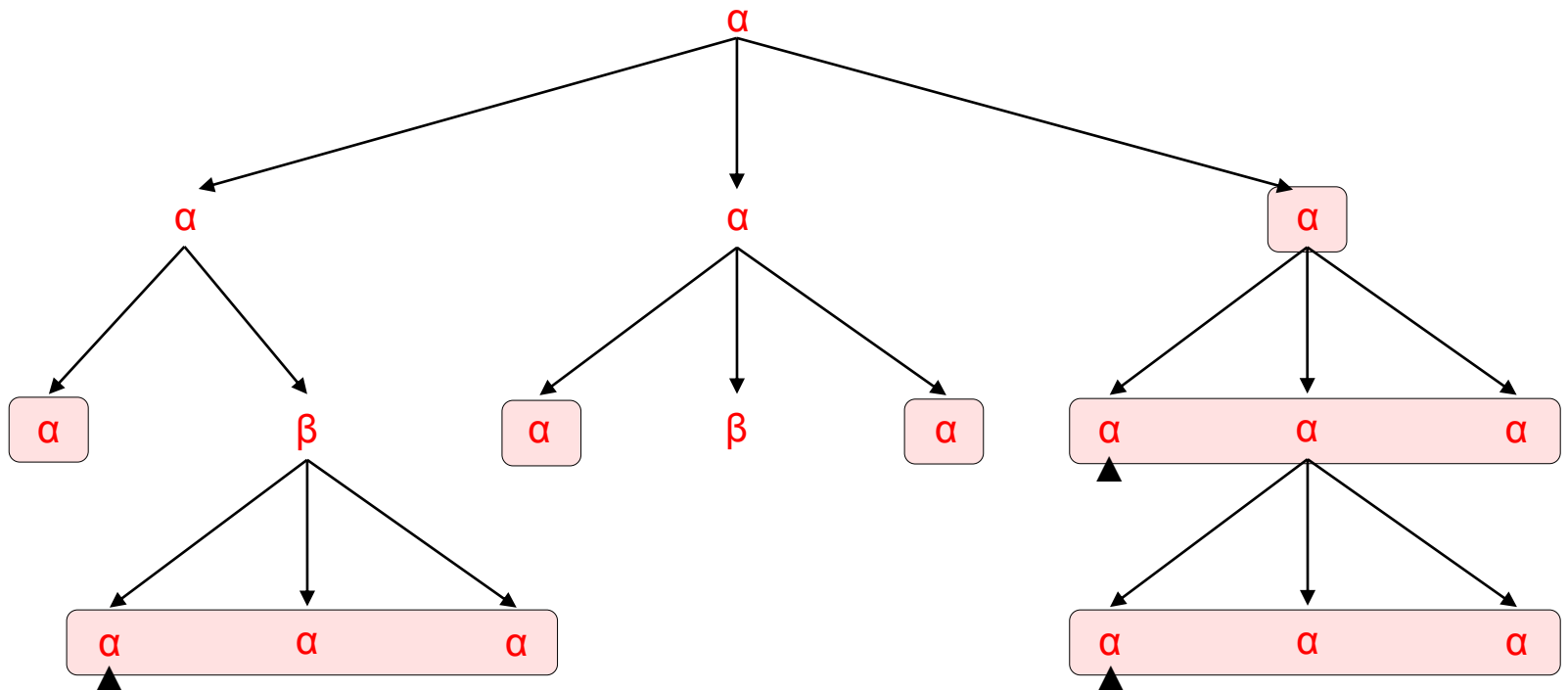
$\text{Select}(x) \text{ :- } P_\alpha(x), x \prec_{\text{fc}} y, \text{Mark}(y)$



Monadic Datalog

Select(x) :- $P_\alpha(x)$, Leaf(x)

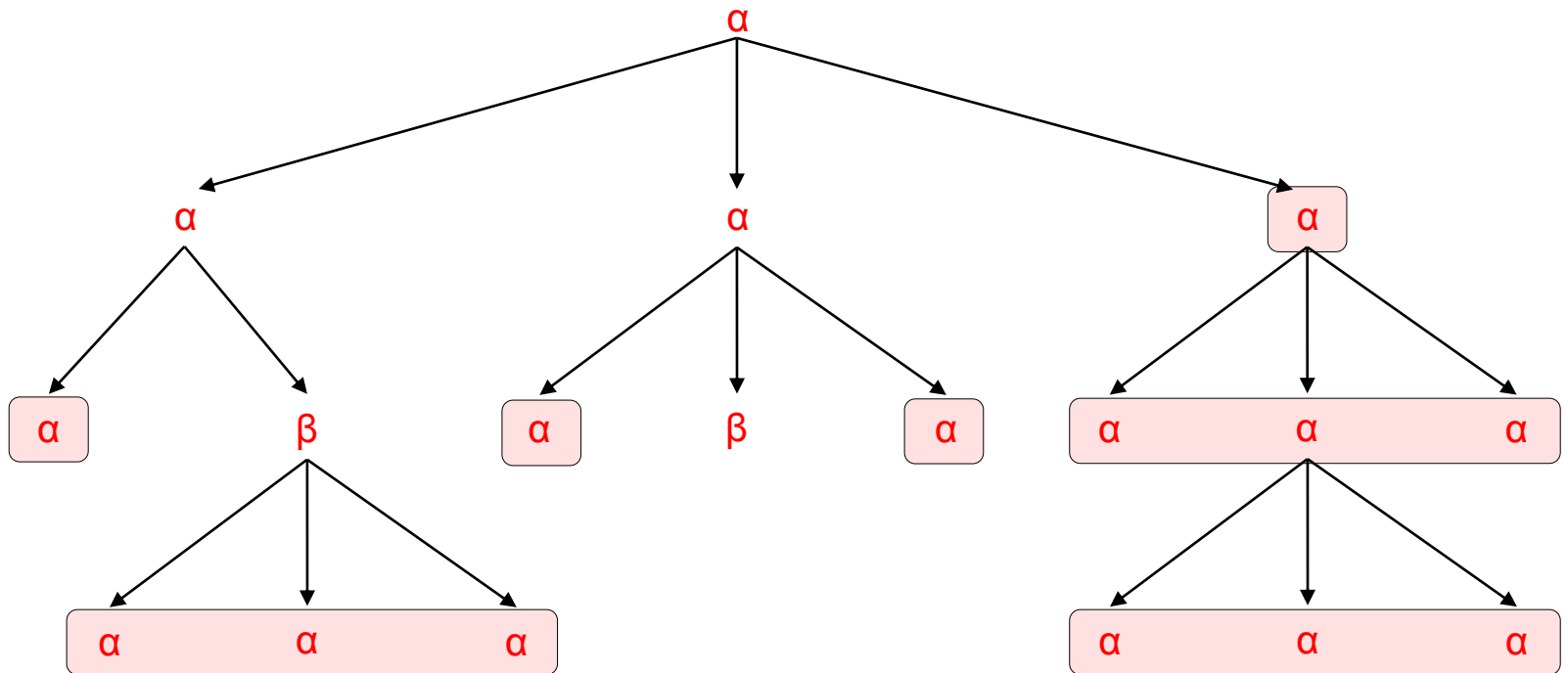
Select(x) :- $P_\alpha(x)$, $x \prec_{fc} y$, Mark(y)



Monadic Datalog

Mark(x) :- LastChild(x), Select(x)

Mark(x) :- Select(x), $x \prec_{ns} y$, Select(y)



Monadic Datalog

all the introduced (or intentional) predicates are unary

Select all nodes v such that their descendants (including v) are labeled α

$\text{Goal}(x) \text{ :- } P_\alpha(x), \text{Leaf}(x)$

$\text{Goal}(x) \text{ :- } P_\alpha(x), x \prec_{fc} y, \text{Mark}(y)$

Mark(v) means that

- $\text{Select}(v)$ holds
- For every u such that $v \prec_{ns^*} u$, $\text{Select}(u)$ holds

$\text{Mark}(x) \text{ :- } \text{LastChild}(x), \text{Goal}(x)$

$\text{Mark}(x) \text{ :- } \text{Goal}(x), x \prec_{ns} y, \text{Mark}(y)$

Monadic Datalog

$$\mathbf{R} = \{<_{\text{fc}}, \text{Leaf}, \text{LastChild}, \text{Root}, \{P_s\}_{s \in \Lambda}\}$$

Theorem: Consider a unary query Q on labeled ordered unranked trees. Then:

Q is **MSO**-definable $\Leftrightarrow Q$ is definable in Monadic Datalog over \mathbf{R}

Theorem: A Monadic Datalog query Q can be evaluated on a tree T in time $O(|T| \cdot |Q|)$

Monadic Datalog is heavily used in Web data extraction:
real-life languages are based on Monadic Datalog, which
combines expressiveness and good evaluation properties

XML Schemas

- Usually, we are not interested in documents containing arbitrary elements, but only in documents that satisfy some specific constraints
- **Schema** – the markup permitted in an XML document
- Many different XML **schema languages** available:
 - Document Type Definitions (DTDs)
 - W3C XML Schema
 - REgular LAnguage for XML Next Generation (RELAX NG)
 - Schematron
 - ...

DTDs: An Example

```
<bookshelf>
  <book>
    <title>Descriptive Complexity</title>
    <publisher>Springer</publisher>
    <author>
      <name>Neil</name>
      <surname>Immerman</surname>
    </author>
  </book>
  <book>
    <title>Computational Complexity</title>
    <publisher>Addison Wesley</publisher>
    <year>1994</year>
    <author>
      <surname>Papadimitriou</surname>
    </author>
  </book>
</bookshelf>
```

the XML document is valid w.r.t. the DTD

```
<!DOCTYPE bookshelf [
  <!ELEMENT bookshelf (book+)>
  <!ELEMENT book (title, publisher, year?, author+)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT author (name?, surname)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT surname (#PCDATA)>
]>
```

DTDs: Formal Definition

Fix a finite alphabet Λ

A **document type definition (DTD)** D is function-symbol pair

$$(f : \Lambda \rightarrow \text{regular expressions over } \Lambda, s \in \Lambda)$$

For example, the previous DTD is written as $(f, \text{bookshelf})$, where

$$f(\text{bookshelf}) = \text{book} \cdot \text{book}^*$$

$$f(\text{book}) = \text{title} \cdot \text{publisher} \cdot (\text{year} \cup \varepsilon) \cdot (\text{author} \cdot \text{author}^*)$$

$$f(\text{author}) = (\text{name} \cup \varepsilon) \cdot \text{surname}$$

$$f(\text{title}) = f(\text{publisher}) = f(\text{year}) = f(\text{name}) = f(\text{surname}) = \varepsilon$$

DTDs into Tree Automata (and MSO)

- The previous DTD is written as $D = (f, \text{bookshelf})$, where

$$f(\text{bookshelf}) = \text{book} \cdot \text{book}^*$$

$$f(\text{book}) = \text{title} \cdot \text{publisher} \cdot (\text{year} \cup \varepsilon) \cdot (\text{author} \cdot \text{author}^*)$$

$$f(\text{author}) = (\text{name} \cup \varepsilon) \cdot \text{surname}$$

$$f(\text{title}) = f(\text{publisher}) = f(\text{year}) = f(\text{name}) = f(\text{surname}) = \varepsilon$$

- Let $A_D = (\{s_{\text{bookshelf}}, s_{\text{book}}, s_{\text{title}}, s_{\text{publisher}}, s_{\text{year}}, s_{\text{author}}, s_{\text{name}}, s_{\text{surname}}\}, \{s_{\text{bookshelf}}\}, \bar{\delta})$, where

$$\bar{\delta}(s_x, x) = \varepsilon, \text{ for every } x \in \{\text{title}, \text{publisher}, \text{year}, \text{name}, \text{surname}\}$$

$$\bar{\delta}(s_{\text{bookshelf}}, \text{bookshelf}) = s_{\text{book}} \cdot s_{\text{book}}^*$$

$$\bar{\delta}(s_{\text{book}}, \text{book}) = s_{\text{title}} \cdot s_{\text{publisher}} \cdot (s_{\text{year}} \cup \varepsilon) \cdot (s_{\text{author}} \cdot s_{\text{author}}^*)$$

$$\bar{\delta}(s_{\text{author}}, \text{author}) = (s_{\text{name}} \cup \varepsilon) \cdot s_{\text{surname}}$$

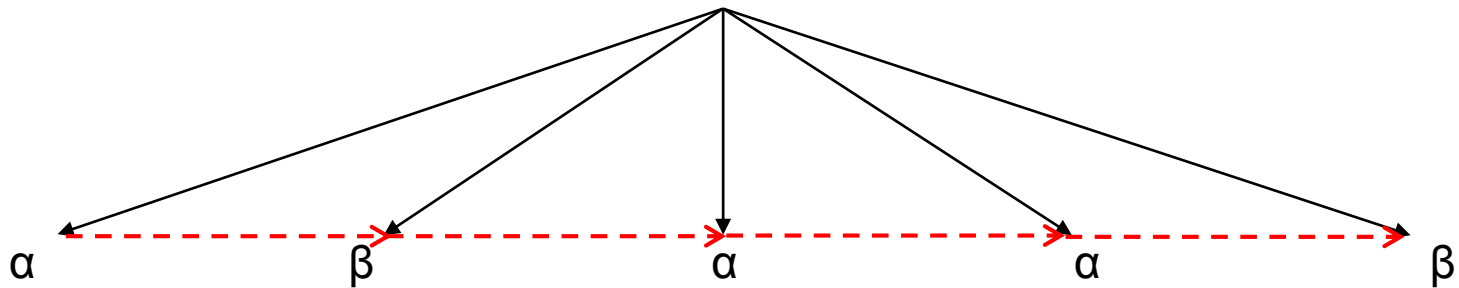
$$L(A_D) = \{T \mid T \text{ is valid w.r.t. } D\}$$

Recap

- XML documents are modeled as labeled ordered unranked trees
- **MSO** is the yardstick logic for querying ordered unranked trees
 - Boolean queries that they define sets of trees: **MSO** = NUTA
 - Unary queries that select nodes in trees: **MSO** = NQA
- **MSO** over trees can be evaluated in linear time in data complexity, but the combined complexity is non-elementary
- Monadic Datalog – an alternative logic for **MSO** with good evaluation properties
- DTDs are captured by NUTA (**MSO**)

A Quick Note on Unordered Trees

- Like ordered trees but the sibling ordering (\prec_{ns}) is **no longer available**
- Without order, **counting** has to be introduced explicitly – order buys counting

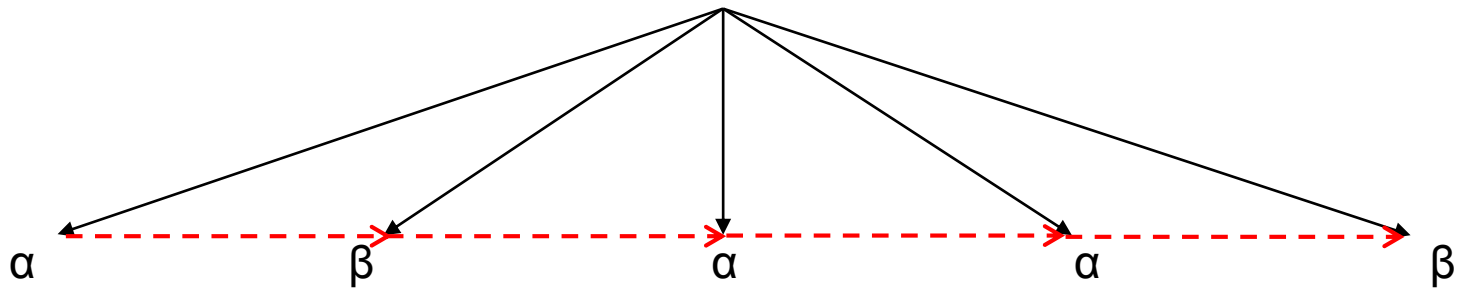


select the nodes with at least two children labeled α

$$Q(x) = \exists y \exists z (x \prec_{ch} y \wedge P_{\alpha}(y) \wedge y \prec_{ns^*} z \wedge P_{\alpha}(z))$$

A Quick Note on Unordered Trees

- Like ordered trees but the sibling ordering (\prec_{ns}) is **no longer available**
- Without order, **counting** has to be introduced explicitly – order buys counting



select the nodes with at least two children labeled α

- We have counting NUTA and counting query automata

Associated Papers

- Frank Neven: Automata, Logic, and XML. CSL 2002: 2-26

A survey of automata theoretic techniques in XML, particularly XML standards

- Leonid Libkin: Logics for Unranked Trees: An Overview. Logical Methods in Computer Science 2(3) (2006)

A survey of logical techniques for languages used in XML (schema, navigation, querying)

- Georg Gottlob, Christoph Koch, Reinhard Pichler: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. 30(2): 444-491 (2005)

How to evaluate XPath most efficiently

- Georg Gottlob, Christoph Koch, Reinhard Pichler, Luc Segoufin: The complexity of XPath query evaluation and XML typing. Journal of the ACM 52(2): 284-335 (2005)

Pinpointing exact complexity of many problems related to XPath evaluation and XML schemas

Associated Papers

- Frank Neven, Thomas Schwentick: Query automata over finite trees. Theor. Comput. Sci. 275(1-2): 633-674 (2002)

Extending automata to formalisms that select nodes in trees

- Georg Gottlob, Christoph Koch: Monadic datalog and the expressive power of languages for Web information extraction. Journal of the ACM 51(1): 74-113 (2004)

Capturing MSO with a very efficient language, and applications in Web data extraction

- Pablo Barceló, Leonid Libkin: Temporal logics over unranked trees. LICS 2005: 31-40

How XML languages are related to logics used in software/hardware verification

- Leonid Libkin, Cristina Sirangelo: Reasoning about XML with temporal logics and automata. J. Applied Logic 8(2): 210-232 (2010)

... and how to exploit the connection to verify properties of XML

Associated Papers

- Thomas Schwentick: XPath query containment. SIGMOD Record 33(1): 101-109 (2004)

A survey of techniques for testing containment and equivalence of XPath queries

- Wenfei Fan, Leonid Libkin: On XML integrity constraints in the presence of DTDs. Journal of the ACM 49(3): 368-406 (2002)

Explaining why unary keys and foreign keys are in NP for XML, and why beyond unary they are undecidable

- Marcelo Arenas, Wenfei Fan, Leonid Libkin: On the Complexity of Verifying Consistency of XML Specifications. SIAM J. Comput. 38(3): 841-880 (2008)

Pushing this further to more expressive constraints used in XML, such as those in XML Schema

- Wim Martens, Frank Neven, Thomas Schwentick: Simple off the shelf abstractions for XML schema. SIGMOD Record 36(3): 15-22 (2007)

Theoretical reconstructions of XML Schema

Associated Papers

- Wim Martens, Frank Neven, Thomas Schwentick, Geert Jan Bex: Expressiveness and complexity of XML Schema. ACM Trans. Database Syst. 31(3): 770-813 (2006)

An automaton model for XML schema, and its use in efficient typing of documents

- Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin: Two-variable logic on data trees and XML reasoning. Journal of the ACM 56(3) (2009)

Decidability/undecidability boundary for 2 vs 3 variables over data trees

- Tony Tan: Extending two-variable logic on data trees with order on data values and its automata. ACM Trans. Comput. Log. 15(1): 8 (2014)

Pushing this to more expressive formalisms, with better (more readable) algorithms

- Claire David, Leonid Libkin, Tony Tan: Efficient reasoning about data trees via integer linear programming. ACM Trans. Database Syst. 37(3): 19 (2012)

The NP bound for sets and linear constraints

Associated Papers

- Henrik Björklund, Wim Martens, Thomas Schwentick: Conjunctive query containment over trees. *J. Comput. Syst. Sci.* 77(3): 450-472 (2011)

Extending CQ containment from relations to databases

- Wojciech Czerwinski, Wim Martens, Pawel Parys, Marcin Przybylko: The (Almost) Complete Guide to Tree Pattern Containment. *PODS 2015*: 117-130

CQs over trees are essentially patterns: A detailed study of their containment

- Maarten Marx: Conditional XPath. *ACM Trans. Database Syst.* 30(4): 929-959 (2005)

An XPath extension that captures all first-order queries over XML documents

- Balder ten Cate, Maarten Marx: Navigational XPath: calculus and algebra. *SIGMOD Record* 36(2): 19-26 (2007)

Providing algebraic counterpart for XPath fragments

Associated Papers

- Loredana Afanasiev, Maarten Marx: An analysis of XQuery benchmarks. Inf. Syst. 33(2): 155-181 (2008)

The title says it all

- Luc Segoufin, Cristina Sirangelo: Constant-Memory Validation of Streaming XML Documents Against DTDs. ICDT 2007: 299-313

Analyzing which DTDs can be checked over streamed documents