the rest of this course

Volume
size does mattes
(thousands of TBs of data)

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

Veracity
data is often
incomplete/inconsistent

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**

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

  Airlines

  Airports
Structured Data

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

**Airports**

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

**Airlines**

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

**Flights**

(VIE, LHR, BA)  
(VIE, LHR, OS)
**Structured Data**

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

<table>
<thead>
<tr>
<th>Airports</th>
<th>Flights</th>
<th>Airlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VIE, Vienna International, Vienna)</td>
<td>(VIE, LHR, BA)</td>
<td>(BA, British Airways)</td>
</tr>
<tr>
<td>(LHR, London Heathrow, London)</td>
<td>(VIE, LHR, OS)</td>
<td>(OS, Austrian Airlines)</td>
</tr>
<tr>
<td>(Airport_Code, Name, City)</td>
<td>(Origin, Destination, Airline)</td>
<td>(Airline_Code, Name)</td>
</tr>
</tbody>
</table>

- **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

![Diagram showing a table with records and attributes]
## Structured Data: “On the Fly” Example

### Airports

<table>
<thead>
<tr>
<th>Airports</th>
<th>Code</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>VIE</td>
<td>Vienna International</td>
<td>Vienna</td>
</tr>
<tr>
<td>LHR</td>
<td>LHR</td>
<td>London Heathrow</td>
<td>London</td>
</tr>
<tr>
<td>LGW</td>
<td>LGW</td>
<td>London Gatwick</td>
<td>London</td>
</tr>
<tr>
<td>LCA</td>
<td>LCA</td>
<td>Larnaca International</td>
<td>Larnaca</td>
</tr>
<tr>
<td>GLA</td>
<td>GLA</td>
<td>Glasgow</td>
<td>Glasgow</td>
</tr>
<tr>
<td>EDI</td>
<td>EDI</td>
<td>Edinburgh</td>
<td>Edinburgh</td>
</tr>
</tbody>
</table>

### Airlines

<table>
<thead>
<tr>
<th>Airlines</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>BA</td>
<td>British Airways</td>
</tr>
<tr>
<td>OS</td>
<td>OS</td>
<td>Austrian Airlines</td>
</tr>
<tr>
<td>U2</td>
<td>U2</td>
<td>EasyJet</td>
</tr>
</tbody>
</table>

### Flights

<table>
<thead>
<tr>
<th>Flights</th>
<th>Origin</th>
<th>Destination</th>
<th>Airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td></td>
<td>British Airways</td>
</tr>
<tr>
<td>VIE</td>
<td>LHR</td>
<td></td>
<td>Austrian Airlines</td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td></td>
<td>British Airways</td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td></td>
<td>EasyJet</td>
</tr>
</tbody>
</table>
“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
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
  o Each row is a semantic entity - person
  o All entities are grouped in a class - persons

• But not too much structure
  o Entities have no regular structure
  o Structure of future entities is unpredictable
Why Semi-structured Data?

- There are data sources that we would like to treat as databases, but which cannot be constraint by a schema

- Flexible format for exchanging data between different places

... the WEB

**GOAL:** Reconcile document view (web) with strict structures (databases)
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
Trees as Data Model

- SSD can be represented as a (labelled) tree:
  - leaf nodes standing for single data items
  - inner nodes have no label
  - edges labelled with elements

- Such a model is called **self-describing** - information that is usually associated with a schema is contained within the data

- Data carries its own description
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)
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}
  }
}
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>
```
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 = eXtensible Markup Language

- W3C standard for document markup since 1998
- Generic syntax to markup data with human- and machine-readable tags
- One of the most common data formats
- Several XML-related W3C standards
  - XML Schema: define the markup permitted in a document
  - XPath: navigation mechanism
  - XSLT: transformation language
  - XQuery: query language

An exciting topic for database theorists: it brings techniques from formal language theory and merges them nicely with logic
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 – XPath
  - Flavour of traditional first-order logic, or
  - Temporal logics for describing navigation

- 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 \( \Lambda \)

An **ordered unranked tree** \( T \) is a structure

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

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

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

- They are not definable in first-order logic
- However, if the adopted logic is powerful enough to define them, then we can simply use $\prec_{\text{ch}}$ and $\preceq_{\text{ns}}$
Check that in a tree $T$ over the alphabet \{\(\alpha,\beta\)\} every \(\alpha\)-labeled node always has a \(\beta\)-labeled descendant

\[
Q = \forall x (P_\alpha(x) \rightarrow \exists y (x \prec_{ch^*} y \land P_\beta(y))
\]
Ordered Unranked Trees: Querying

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

(i) labeled \(\alpha\),

(ii) have a descendant \(d\) labeled \(b\), and

(iii) \(d\) has a younger sibling labeled \(\gamma\)

\[
Q(x) = P_\alpha(x) \land \exists y \exists z (x \prec_{ch^*} y \land P_\beta(y) \land y \prec_{ns^*} z \land P_\gamma(z))
\]
Ordered Unranked Trees: Querying

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

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

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

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

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

\[
Q(x,y) = \exists S \exists R \quad (x \in S \land y \in R)
\land \forall z ((x \prec_{ch} z \prec_{ch} y) \rightarrow (z \in S \iff \neg (z \in R)))
\land \forall z \forall w ((x \prec_{ch} z \prec_{ch} w \prec_{ch} y) \rightarrow ((z \in S \rightarrow w \in R) \land (z \in R \rightarrow w \in S)))
\]
Ordered Unranked Trees: Querying

- For querying labeled ordered unranked trees we use:
  - **First-order logic (FO)**
    - Boolean connectives $\lor$, $\land$, $\neg$
    - 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 $\phi$ of $L$ such that $T \in T \iff T \models \phi$

- 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 $T$, $v \in Q(T) \iff T \models \varphi(v)$

the set of nodes in $T$ selected by $Q$
Unranked Tree Automata

- A nondeterministic unranked tree automaton (NUTA) over $\Lambda$-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 $D$ is a function $\lambda_A : D \rightarrow S$ such that: if $v$ is a node with $n$ children, and is labeled $\alpha$, then the string $\lambda_A(v \cdot 0) \ldots \lambda_A(v \cdot (n-1)) \in \delta(\lambda_A(v), \alpha)$

\[ s_1 \ldots s_n \in \delta(s, \alpha) \]
Unranked Tree Automata

- A nondeterministic unranked tree automaton (NUTA) over $\Lambda$-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 $D$ is a function $\lambda_A : D \rightarrow S$ such that: if $v$ is a node with $n$ children, and is labeled $\alpha$, then the string $\lambda_A(v\cdot 0)\ldots\lambda_A(v\cdot(n-1)) \in \delta(\lambda_A(v),\alpha)$

- 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 $\Lambda$-labeled trees where 0,1 appear only at leaves, while $\wedge, \vee$ can appear everywhere except at leaves.

- We define $A = (\{s_0, s_1\}, \{s_1\}, \delta)$, where

$$\begin{align*}
\delta(s_0, 0) &= \delta(s_1, 1) = \{\varepsilon\} \\
\delta(s_0, 1) &= \delta(s_1, 0) = \emptyset \\
\delta(s_0, \wedge) &= (s_0 \cup s_1)^* \cdot s_0 \cdot (s_0 \cup s_1)^* \\
\delta(s_1, \wedge) &= s_1^* \\
\delta(s_0, \vee) &= s_0^* \\
\delta(s_1, \vee) &= (s_0 \cup s_1)^* \cdot s_1 \cdot (s_0 \cup s_1)^*
\end{align*}$$
We can now present an interesting result:

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

$$T \text{ is MSO-definable } \iff T \text{ is regular}$$

...but, what about unary queries?

we need an extended automata model
Query Automata

- A nondeterministic query automaton (NQA) over $\Lambda$-labeled trees is a pair
  
  \[ A = (B, P) \]

  where:
  - $B$ is an NUTA $(S, F, \delta)$
  - $P$ is a subset of $S \times \Lambda$

- Such an automaton defines a unary query $Q_A$ over unranked trees:
  
  \[ v \in Q_A(T) \iff (\lambda_B(v), \text{label}(v)) \in P, \text{ for some accepting run } \lambda_B \text{ of } B \text{ on } T \]
We have similar characterization for unary queries:

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

$Q$ is MSO-definable $\iff Q$ 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 $\varphi$

**Question:** $T \vDash \varphi$?

- The same problem can be defined for unary formulas
  - Given a tree $T$, a unary formula $\varphi(x)$, and a node $v$: does $T \vDash \varphi(v)$?

- As usual, we consider the data and combined complexity
  - Data complexity: $T$ is input, $\varphi$ is fixed
  - Combined complexity: both $T$ and $\varphi$ are part of the input
 Complexity of MSO

**Theorem:** It holds that:

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

**Proof idea:** By translation to automata:

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

\[2^{2^{|\varphi|}}\text{ depends on } \varphi\]
Complexity of MSO

Theorem: It holds that:

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

Proof idea: By translation to automata:

- Convert the given sentence $\varphi$ into a NUTA $A_\varphi$ such that $T \models \varphi \iff T \in L(A_\varphi)$
- 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:

- 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 $\varphi$ into a NUTA $A_\varphi$ such that $T \models \varphi \iff T \in L(A_\varphi)$
- 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.

- \( \mu \)-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?

\[ \exists x \ (\text{Flight}(\text{Vienna},x) \land \text{Flight}(x,\text{Glasgow})) \]

✓
Reachability

Is Glasgow reachable from Vienna?

∃x (Flight(Vienna,x) ∧ Flight(x,Glasgow))
Reachability

Is Glasgow reachable from Vienna?

$\exists x \exists y \ (\text{Flight(Vienna},x) \land \text{Flight}(x,y) \land \text{Flight}(y,\text{Glasgow}))$
Is Glasgow reachable from Vienna?

\[ \exists x \exists y \ (\text{Flight}(\text{Vienna}, x) \land \text{Flight}(x, y) \land \text{Flight}(y, \text{Glasgow})) \]
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 \((Vienna, Glasgow)\)
Reachability

Is Glasgow reachable from Vienna?

Here is a possible strategy:

• Compute all the pairs of cities (c1, c2) such that c2 is reachable from c1
• Check if there is a pair (Vienna, Glasgow)

**not possible via an FO query – we need recursion**
Reachability

Is Glasgow reachable from Vienna?

Edinburgh

- London
- Glasgow
- Paris
- Larnaca
- Vienna

Reachable(x,y) :- Flight(x,y)
Reachable(x,z) :- Flight(x,y), Reachable(y,z)
Goal :- Reachable(Vienna,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)
Mark(ν) means that

- Select(ν) holds
- For every u such that ν <_ns* u, Select(u) holds
Monadic Datalog

\[
\text{Select}(x) \leftarrow P_\alpha(x), \text{Leaf}(x)
\]

\[
\text{Select}(x) \leftarrow P_\alpha(x), x \prec_{fc} y, \text{Mark}(y)
\]
Monadic Datalog

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

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

\[
\begin{align*}
\text{Select}(x) & : \quad P_\alpha(x), \text{Leaf}(x) \\
\text{Select}(x) & : \quad P_\alpha(x), x \prec_{fc} y, \text{Mark}(y)
\end{align*}
\]
Monadic Datalog

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

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

Goal(x) :- P_α(x), Leaf(x)

Goal(x) :- P_α(x), x \prec_{fc} y, Mark(y)

Mark(v) means that

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

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

Mark(x) :- Goal(x), x \prec_{ns} y, Mark(y)
Monadic Datalog

\[ R = \{ \prec_{fc}, \text{Leaf}, \text{LastChild}, \text{Root}, \{ \text{P}_s \}_{s \in \Lambda} \} \]

**Theorem:** Consider a unary query \( Q \) on labeled ordered unranked trees. Then:

\[ Q \text{ is MSO-definable} \iff Q \text{ is definable in Monadic Datalog over } R \]

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

```xml
<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>
```
Fix a finite alphabet $\Lambda$

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

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

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

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

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

$$\delta(s_x, x) = \varepsilon, \text{ for every } x \in \{\text{title, publisher, year, name, surname}\}$$
$$\delta(s_{\text{bookshelf}}, \text{bookshelf}) = s_{\text{book}} \cdot s_{\text{book}}^*$$
$$\delta(s_{\text{book}}, \text{book}) = s_{\text{title}} \cdot s_{\text{publisher}} \cdot (s_{\text{year}} \cup \varepsilon) \cdot (s_{\text{author}} \cdot s_{\text{author}}^*)$$
$$\delta(s_{\text{author}}, \text{author}) = (s_{\text{name}} \cup \varepsilon) \cdot s_{\text{surname}}$$

$L(A_D) = \{T \mid T \text{ is valid w.r.t. } D\}$
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: \( \text{MSO} = \text{NUTA} \)
  - Unary queries that select nodes in trees: \( \text{MSO} = \text{NQA} \)

- **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**)
Ordered Unranked Trees: Querying

For querying labeled ordered unranked trees we use:

- **First-order logic (FO)** – often studied in connection with XPath
  - Boolean connectives $\lor$, $\land$, $\neg$
  - Quantifiers $\exists x$ and $\forall x$ that range over **nodes** of trees

- **Monadic second-order logic (MSO)** – the yardstick logic
  - **FO** plus quantifiers $\exists S$ and $\forall S$ that range over **sets of nodes**
  - New formulae $x \in S$
XPath at First Glance


/title/Descriptive Complexity
/publisher/Springer
/author/name/Neil
/author/surname/Immerman

/title/Computational Complexity
/publisher/Addison Wesley
/year/1994
/author/surname/Papadimitriou
/child::bookshelf/child::book
XPath at First Glance

/bookshelf/book

/book

/title

/publisher

/author

/book

/title

/publisher

/year

/author


/child::bookshelf/child::book[position() = 1]
/descendant::author/child::surname
XPath at First Glance

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

/descendant::book/child::author[child::name]/child::surname
XPath at First Glance

/book
/book

/title
/titel
/publisher
/publisher
/author
/author

/name
/surname
/name
/surname

/empty
/empty

/Computational Complexity
/Addison Wesley

/Descriptive Complexity
/Springer

/Neil
/Immerman

/empty
/empty

/1994
/surname

/Papadimitriou

/descendant::book/child::author[position() = 2][child::name]
Location Paths

- XPath uses location paths to select nodes in a tree
- A location path is a series of location steps separated by the symbol `/`
- Each location step has the form

```
axis::node-test[expression-1][expression-2]...
```

- `defines the relationship to be followed`
- `defines what kind of nodes must be selected`
- `zero or more predicates, which filter the selected nodes according to arbitrary selection criteria`
The Anatomy of a Location Path

child::bookshelf/child::book[position() = 1]

axis node-test axis node-test predicate

location step location step

location path

NOTE: The first location step does not have a predicate
FO over Ordered Unranked Trees

• **First-order logic (FO)** – often studied in connection with XPath
  – Boolean connectives $\lor$, $\land$, $\neg$
  – Quantifiers $\exists x$ and $\forall x$ that range over nodes of trees

• The navigational features of XPath can be described in FO

• Can we define alternative logics for **FO** over ordered unranked trees with good evaluation properties?
  – LTL-like logics
  – CTL-like logics
Tree Temporal Logic – $\text{TL}^\text{tree}$

Syntax: with $d \in \{\text{ch,ns}\}$

$$\phi, \phi' := \alpha, \alpha \in \Lambda \mid \phi \lor \phi' \mid \neg \phi \mid X_d \phi \mid \text{inv}X_d \phi \mid \phi U_d \phi' \mid \phi S_d \phi'$$

$$(T,v) \models X_{\text{ch}} \phi$$
Syntax: with $d \in \{\text{ch,ns}\}$

$$\varphi, \varphi' := \alpha, \alpha \in \Lambda \mid \varphi \lor \varphi' \mid \neg \varphi \mid X_d \varphi \mid \text{inv}X_d \varphi \mid \varphi U_d \varphi' \mid \varphi S_d \varphi'$$

$$(T,v) \models \text{inv}X_{\text{ch}} \varphi$$
Syntax: with $d \in \{\text{ch}, \text{ns}\}$

$$\varphi, \varphi' := \alpha, \alpha \in \Lambda | \varphi \lor \varphi' | \neg \varphi | X_d \varphi | \text{invX}_d \varphi | \varphi U_d \varphi' | \varphi S_d \varphi'$$

$$(T,v) \models \varphi U_{\text{ch}} \varphi'$$
Tree Temporal Logic – TL\textsubscript{tree}

Syntax: with $d \in \{\text{ch}, \text{ns}\}$

\[
\varphi, \varphi' \; := \; \alpha, \alpha \in \Lambda \mid \varphi \lor \varphi' \mid \neg \varphi \mid X_d \varphi \mid \text{inv}X_d \varphi \mid \varphi U_d \varphi' \mid \varphi S_d \varphi'
\]

\[(T,v) \models \varphi S_{\text{ch}} \varphi'\]

(analogously for $X_{\text{ns}} \varphi \mid \text{inv}X_{\text{ns}} \varphi \mid \varphi U_{\text{ns}} \varphi' \mid \varphi S_{\text{ns}} \varphi'$)
Tree Temporal Logic – $\text{TL}^{\text{tree}}$

Syntax: with $d \in \{\text{ch}, \text{ns}\}$

$$\varphi, \varphi' ::= \alpha, \alpha \in \Lambda \mid \varphi \lor \varphi' \mid \neg \varphi \mid X_d \varphi \mid \text{inv}_X \varphi \mid \varphi U_d \varphi' \mid \varphi S_d \varphi'$$

**Theorem:** Consider a Boolean/unary query $Q$ on labeled ordered unranked trees. Then:

$$Q \text{ is } \text{FO}-\text{definable} \iff Q \text{ is } \text{TL}^{\text{tree}}-\text{definable}$$
Important Algorithmic Problems for XPath

XPathSAT

**Input:** an XPath expression $E$, a DTD $D$

**Question:** is there a tree $T$ valid w.r.t. $D$ so that $E$ selects at least one node in it?

XPathCONT

**Input:** two XPath expressions $E$, $E'$ and a DTD $D$

**Question:** does $E \subseteq_D E'$, i.e., for every tree $T$ valid w.r.t. $D$, each node selected by $E$ is also selected by $E'$?
**Theorem:** Given an XPath expression $E$, and a DTD $D$, the problem of deciding whether $E$ is satisfiable w.r.t. $D$ is feasible in time $|D| \cdot 2^{O(|E|)}$

**Proof idea:** exploit automata

- Translate $E$ into a query automaton $A_E$ of exponential size in time $2^{O(|E|)}$
- Translate $D$ into an automaton $A_D$ in linear time
- Let $A = A_E \times A_D$ be the product of the two automata – exponential size
- Test $A$ for emptiness – this can be done in polynomial time in the size of $A$
**Theorem:** Given two XPath expressions $E$, $E'$ and a DTD $D$, the problem of deciding whether $E \subseteq_D E'$ is feasible in time $|D| \cdot 2^{O(|E| + |E'|)}$

**Proof idea:** exploit TL$^\text{tree}$ and automata

- Translate $E$ and $E'$ into TL$^\text{tree}$ formulae $\varphi$ and $\psi$, respectively
- Construct a query automaton $A_{(\varphi \land \neg \psi)}$ for $\varphi \land \neg \psi$
- Translate $D$ into an automaton $A_D$
- Let $A = A_{(\varphi \land \neg \psi)} \times A_D$ – a query automaton of size $|D| \cdot 2^{O(|E| + |E'|)}$
- Test $A$ for emptiness – this can be done in polynomial time in the size of $A$
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

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

select the nodes with at least two children labeled $\alpha$
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

```
α ——— β ——— α ——— α ——— β
```

no way to say that there are at least two children labeled $\alpha$

- 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


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


  How to evaluate XPath most efficiently


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


  Extending automata to formalisms that select nodes in trees


  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


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


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


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


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


  Theoretical reconstructions of XML Schema
Associated Papers


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


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


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


The NP bound for sets and linear constraints
Associated Papers


  Extending CQ containment from relations to databases


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


  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

  
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