Queries on Trees

Jérôme Champavère   Emmanuel Filiot   Olivier Gauwin
Édouard Gilbert   Sławek Staworko

INRIA Lille, Mostrare

2008
Framework

$n$-ary queries on unranked labeled finite ordered trees
Trees

t = \begin{array}{c}
a \\
c \\
c \\
d \end{array}

finite alphabet: \( \Sigma = \{a, b, c, d, e\} \)

t is the structure \((D, ch_*, ns_*, \text{label})\) with:

- \(D = \{\epsilon, 1, 2, 3\}\): prefix-closed finite subset of \(\mathbb{N}\)
- \(ch_* = \text{reflexive-transitive closure of } ch\), defined by:
  \[ ch(\pi_1, \pi_2) \iff \pi_2 = \pi_1 \cdot i \text{ for some } i \in \mathbb{N} \]
- \(ns_* = \text{reflexive-transitive closure of } ns\), defined by:
  \[ ns(\pi_1, \pi_2) \iff \pi_1 = \pi \cdot i \text{ and } \pi_2 = \pi \cdot (i + 1) \text{ for some } \pi, i \in \mathbb{N}^* \times \mathbb{N} \]
- \(\text{label} : D \to \Sigma\). Can also be seen as a partition \((\text{label}_a)_{a \in \Sigma}\) of \(D\).
  \[ \text{label}(1 \cdot 3) = d \quad \text{label}_d(1 \cdot 3) \]
**Queries**

*n*-ary queries

- \( q(t) \subseteq D_t^n \)

\( n=0 \): Boolean queries

- \( q(t) = \emptyset \) or \( q(t) = \{()\} \)
- \( q \) defines \( L_q = \{ t \mid q(t) = \{()\} \} \)

**Questions**

- expressivity
- complexity of:
  - model-checking: \( x \in q(t) \)
  - satisfiability: \( \exists t, q(t) \neq \emptyset \)
Existing material

Surveys

- Logics over unranked trees: an overview [Lib06]
- Automata, logic, and XML [Nev02b, Nev02a]
- Automata for XML – a survey [Sch07]
- Effective Characterizations of Tree Logics [Boj08a]
- Tree-walking automata [Boj08b]

Books

- Finite Model Theory [EF99, Lib04]
- Foundations of Databases [AHV95]
Outline

1. Classical logics (FO, MSO)
2. Queries by Tree Automata
   - Tree-walking automata
   - Schema Languages & Tree Automata
3. Conjunctive Queries over Trees
   - Definition, results and acyclic fragment
   - Twigs and Tree Patterns
4. Monadic Datalog
5. $\mu$-calculus
6. XPath
7. Temporal Logics
Part I

Classical Logics, Automata
Outline

1. Classical logics (FO, MSO)

2. Queries by Tree Automata
   - Tree-walking automata
   - Schema Languages & Tree Automata
Well-formed formulas based on:

- predicates from the structure: \( ch_*, ns_*, (\text{label}_a)_{a \in \Sigma} \)
- Boolean connectives: \( \land, \neg \)
- FO variables: \( x, y... \)
- quantifiers on FO variables: \( \exists x \)
We use free variables:

\[ q(x) = \exists y. \exists z. (\text{ch}_*(x, y) \land \text{ch}_*(y, z) \land \text{label}_a(z)) \]

This way we can define queries of any arity.
FO: Available predicates

Why $\text{ch}_*$ and $\text{ns}_*$?

- because $\text{ch}$ and $\text{ns}$ are definable from $\text{ch}_*$ and $\text{ns}_*$ in FO...
- ... but the converse is false

So in the following, we suppose that $\text{ch}$ and $\text{ns}$ are also available.

Also definable in FO:

- unary predicates: $\text{root}$, $\text{leaf}$, $\text{lc}$ (lastchild)
- binary predicates: $\text{fc}$ (firstchild)
FO: Complexity

Model-checking

- PSPACE-complete (combined complexity). [Sto74, Var82]
- Remark: PSPACE-hardness is even true for the quantified propositional logic [GO99].

Satisfiability

- non-elementary on trees
FO: Restrictions on the number of variables

\( \text{FO}^k = \text{FO} \) formulas using only \( k \) variables

Variables might be reused

\[
q(x) = \exists y. \exists z. (ch_*(x, y) \land ch_*(y, z) \land \text{label}_a(z)) \notin \text{FO}^2
\]

but is equivalent to

\[
q'(x) = \exists y. (ch_*(x, y) \land \exists x. (ch_*(y, x) \land \text{label}_a(x))) \in \text{FO}^2
\]

Theorem ([Imm82, Var95, GO99])

*The model-checking problem for \( \text{FO}^k \) (with \( k \geq 2 \)) is P-complete on any structure.*
**FO: Restrictions on the number of variables**

\[ \text{FO}^2 = \text{FO formulas using only 2 variables} \]

In \( \text{FO}^2 \), one cannot define \( ch \) and \( ns \) from \( ch_* \) and \( ns_* \) anymore. So \( ch \) and \( ns \) are added to the signature.

**Complexity**

Model-checking in \( \text{FO}^2 \) can be done in \( O(|t|^2\cdot |q|) \) [Imm82].

**Expressivity**

- FO is strictly more expressive than \( \text{FO}^2 \).
- Example of Boolean query: trees where the leaf language is \((ab)^*\).

Links between \( \text{FO}^2 \) and XPath will be shown in Part 3.
Expressivity

$A \rightarrow B \quad A \subsetneq B$

$A \ldots \rightarrow B \quad A \subseteq B$

$A \rightarrow B \quad A \nsubseteq B$

FO

FO^2
FO: Restrictions on the number of variables

Data values

- predicate $\sim$: $x \sim y$ if $x$ and $y$ are two attribute nodes that have the same value
- in XPath semantics: add tests of the form
  \[
  /bib//book/@type = //collection/@style
  \]

Decidability

- $\text{FO}^2 [\sim, ch, ns]$ is decidable [BDM$^+$06].
- $\text{FO}^2 [\sim, ch, ns, ch^*, ns^* ]$: open question
- $\text{FO}^3[\sim, ch, ns]$ is undecidable (even on strings) [BMS$^+$06].
FO: Restrictions on the number of variables

\[ \text{FO}_n^k = \text{FO formulas using} \ (k \ \text{bound variables}) \ + \ (n \ \text{free variables}) \]

We assume here that the \( n \) free variables are never quantified.

Some results on trees

- \( \text{FO}_2 = \text{FO}_2^3 \) [Mar05a] (his result is stronger)
- \( \text{FO}_3 = \text{FO}_3^3 \) [Mar05b]
- \( \text{FO}_n = \text{FO}_n^3 \)
  - translate into a \( \text{FO}_0 \) formula on alphabet \( \Sigma \times \mathbb{B}^n \),
  - \( \text{FO}_0 = \text{FO}_0^3 \) (consequence of [Mar05b], Th. 3)
  - backward translation: \( \text{label}_{(f, \vec{b})}(x) \) becomes \( \text{label}_f(x) \wedge \bigwedge_{i=1}^n x = x_i \)
MSO

\[ \text{MSO} = \text{FO} + \text{quantification over monadic predicates} \]

"monadic predicates" also seen as "sets"

\[ X(x) \quad x \in X \]

\[ \phi_{\text{odd}}(x, y) \]

= "\( y \) is a descendant of \( x \) and the path between them is of odd length"

= \[ \exists X. \exists Y. \quad (\forall z. (X(z) \iff \neg Y(z))) \land \]

\[ (\forall z. (X(z) \lor Y(z) \Rightarrow \text{ch}_* (x, z) \land \text{ch}_* (z, y))) \land \]

\[ (X(x) \land Y(y)) \land \]

\[ (\forall z. \forall v. (\text{ch}_* (x, z) \land \text{ch} (z, v) \land \text{ch}_* (v, y) \Rightarrow \]

\[ (X(z) \Rightarrow Y(v) \land Y(z) \Rightarrow X(v))) \]

Expressivity

MSO is strictly more expressive than FO (see \( \phi_{\text{odd}} \)).
Expressivity

$A \rightarrow B \quad A \subsetneq B$

$A \rightarrow B \quad A \subseteq B$

$A \rightarrow B \quad A \nsubseteq B$

MSO

FO

FO$^2$
MSO: Complexity

Model-checking

- combined complexity: $\text{PSPACE}_c$ [Sto74, Var82]
- data complexity: linear (by translation to automaton)

Satisfiability

- non-elementary on trees
Deciding membership to FO

**Theorem ([BS05])**

Given a regular tree language $L$, one can decide if $L$ is definable in $\text{FO}_{\text{ch},(\text{label}_a)_{a \in \Sigma}}$.

**Open decision problem**

Given a regular tree language $L$, is it possible to decide if $L$ is definable in FO?

In other words, FO-definability is known to be decidable for unordered trees, but unknown for ordered trees.

**Automata for FO**

For a definition of automata recognizing exactly FO-definable languages, see [Boj04, Chapter 2].
Outline

1. Classical logics (FO, MSO)

2. Queries by Tree Automata
   - Tree-walking automata
   - Schema Languages & Tree Automata
Tree Automata for Queries

- Branching & Stepwise Tree automata
- Query automata
- Tree-walking automata (TWA)
- Schema languages
Branching & Stepwise Tree Automata I

- Automata over $\Sigma \times \{0, 1\}^n$
  - Canonical languages
  - Same expressive power as MSO
- Automata with selecting states
  - Boolean values into the states
  - Existential run-based queries [NPTT05]
  - Selecting tree automata [FGK03]
- Stepwise tree automata [CNT04]
Branching & Stepwise Tree Automata II

- Decision problems
  - Membership: PTIME
  - Non-emptiness: PTIME

- From MSO to tree automata: non-elementary size
  - Upper bound [TW68]
  - Lower bound [FG02]
Query Automata [NS99, NS02]

- Two-way deterministic tree automata [Mor94] over (un)ranked trees extended with a selection function
- Equivalent to MSO
- Decision problems

<table>
<thead>
<tr>
<th></th>
<th>EXPTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-emptiness</td>
<td></td>
</tr>
<tr>
<td>Containment</td>
<td></td>
</tr>
<tr>
<td>Equivalence</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Classical logics (FO, MSO)

2. Queries by Tree Automata
   - Tree-walking automata
   - Schema Languages & Tree Automata
Most work is done on ranked trees
Context

- Most work is done on ranked trees
- Still some definitions on unranked cases, but few results
Context

- Most work is done on ranked trees
- Still some definitions on unranked cases, but few results
- Thus we will work on trees of rank 2
Most work is done on ranked trees
Still some definitions on unranked cases, but few results
Thus we will work on trees of rank 2
Structure: label, $ch_1$, $ch_2$
A tree-walking automaton (TWA): [AU71]:

- A tree is accepted whenever the “accept” action is used.
A tree-walking automaton (TWA)\cite{AU71}:

- the automaton is located in some node (at first, the root) of the tree and in a given state

- a tree is accepted whenever the “accept” action is used
Tree-walking automata on ranked trees

- A tree-walking automaton (TWA)[AU71]:
  - the automaton is located in some node (at first, the root) of the tree and in a given state
  - if some conditions are verified (label, being a leaf, being the left child of one’s parent), decide of an action

  - a tree is accepted whenever the “accept” action is used
A tree-walking automaton (TWA)[AU71]:

- the automaton is located in some node (at first, the root) of the tree and in a given state
- if some conditions are verified (label, being a leaf, being the left child of one’s parent), decide of an action
- actions: accept, reject, move to parent with state $q$, move to left child with state $q'$, ...
- a tree is accepted whenever the “accept” action is used
A tree-walking automaton (TWA)[AU71]:

- the automaton is located in some node (at first, the root) of the tree and in a given state
- if some conditions are verified (label, being a leaf, being the left child of one’s parent), decide of an action
- actions: accept, reject, move to parent with state $q$, move to left child with state $q'$, . . .
- a tree is accepted whenever the “accept” action is used
A tree-walking automaton (TWA)[AU71]:

- the automaton is located in some node (at first, the root) of the tree and in a given state
- if some conditions are verified (label, being a leaf, being the left child of one’s parent), decide of an action
- actions: accept, reject, move to parent with state \( q \), move to left child with state \( q' \), . . .
- a tree is accepted whenever the “accept” action is used a tree can be rejected by looping, the “reject” action is not necessary

Expressiveness:
A tree-walking automaton (TWA)[AU71]:

- the automaton is located in some node (at first, the root) of the tree and in a given state
- if some conditions are verified (label, being a leaf, being the left child of one’s parent), decide of an action
- actions: accept, reject, move to parent with state $q$, move to left child with state $q'$, . . .
- a tree is accepted whenever the “accept” action is used a tree can be rejected by looping, the “reject” action is not necessary

Expressiveness:
- any tree-walking automaton can be represented as a branching automaton, but with exponential blowup
Tree-walking automata
on ranked trees

- A tree-walking automaton (TWA) [AU71]:
  - the automaton is located in some node (at first, the root) of the tree and in a given state
  - if some conditions are verified (label, being a leaf, being the left child of one’s parent), decide of an action
  - actions: accept, reject, move to parent with state $q$, move to left child with state $q'$, ...
  - a tree is accepted whenever the “accept” action is used a tree can be rejected by looping, the “reject” action is not necessary

- Expressiveness:
  - any tree-walking automaton can be represented as a branching automaton, but with exponential blowup
  - but the opposite is false: TWA are not as expressive as MSO [BC05]
Expressiveness (ranked case)

\[ A \rightarrow B \quad A \subsetneq B \]
\[ A \rightarrow B \quad A \subseteq B \]
\[ A \rightarrow B \quad A \nsubseteq B \]

[BC04]
Deterministic TWA and their expressiveness

- Formulae recognised by deterministic TWA are stable by negation
- Formulae recognised by non-deterministic ones are not
Deterministic TWA and their expressiveness

- Formulae recognised by deterministic TWA are stable by negation
- Formulae recognised by non-deterministic ones are not
Deterministic TWA and their expressiveness

- Formulae recognised by deterministic TWA are stable by negation.
- Formulae recognised by non-deterministic ones are not $\Rightarrow$
  deterministic TWA are strictly less expressive than non-deterministic ones [MSS06]
- $\text{FO} \subseteq \text{TWA} \subsetneq \text{MSO}$ [BC04]
Deterministic TWA and their expressiveness

- Formulae recognised by deterministic TWA are stable by negation
- Formulae recognised by non-deterministic ones are not \( \Rightarrow \)
- Deterministic TWA are strictly less expressive than non-deterministic ones [MSS06]
- \( \text{FO} \subseteq \text{TWA} \subsetneq \text{MSO} \) [BC04]
- \( \text{FO} \nsubseteq \text{DTWA} \nsubseteq \text{FO} \) [BC05]
Expressiveness (ranked case)

\[ A \rightarrow B \quad A \subsetneq B \]

\[ A \rightarrow B \quad A \subseteq B \]

\[ A \rightarrow B \quad A \notin B \]

\[ \text{FO} \]

\[ \text{detTWA} \]

\[ \text{TWA} \]

\[ \text{MSO} \]

[BC04]

[BC05]

[MSS06]
Add a finite number of pebble marked \{1, \ldots, n\} to the automaton.
Add a finite number of pebble marked \( \{1, \ldots, n\} \) to the automaton

New tests: is there a pebble on current node?
Add a finite number of pebble marked \( \{1, \ldots, n\} \) to the automaton

- New tests: is there a pebble on current node?
- New actions: add a pebble to current position, remove a pebble from the current (or any) state
Add a finite number of pebble marked \{1, \ldots, n\} to the automaton

- New tests: is there a pebble on current node?
- New actions: add a pebble to current position, remove a pebble from the current (or any) state
- Stack discipline: if pebble 1 to \(i\) already can only add pebble \(i + 1\) or remove pebble \(i\)
Expressiveness increases with number of pebble [BSSS06]

\[ \forall n \in \mathbb{N} \quad \text{PTWA}_n \subsetneq \text{PTWA}_{n+1} \]

\[ \text{detPTWA} \subseteq \text{PTWA}[EH99] \]

it is not known if \( \text{detPTWA} = \text{PTWA} \)

but there is no \( c \) s.t. \( \text{PTWA}_k \subseteq \text{detPTWA}_{ck} \)

Expressiveness without stack discipline

\[ \text{MSO} \nsubseteq \text{TWA}_{\text{no stack}} \]

\( \text{TWA}_{\text{no stack}} \) emptiness is undecidable
Expressiveness (ranked case)

A → B  A ⊊ B
A ←→ B  A ⊆ B
A → B  A ̸∈ B

[BC04]

TWA

[BC04]  detTWA

[BC05]  FO

[TWA]_{pebble}

det[TWA]_{pebble}

[BSSS06]
Unbounded pebble TWA

- We now allow an unbounded number of pebble (with stack discipline)

Expressiveness

Unbounded pebble TWA emptiness is undecidable
Invisible pebble TWA $\equiv$ MSO
Unbounded pebble TWA

- We now allow an unbounded number of pebble (with stack discipline)
- We can consider invisible pebble: only the top pebble presence can be tested [EHS07]

Expressiveness

Unbounded pebble TWA emptiness is undecidable
Invisible pebble TWA = MSO
Alternating tree-walking automata

- Two players $\forall, \exists$
Alternating tree-walking automata

- Two players $\forall, \exists$
- Each state belongs to a player: $Q = Q_\forall \cup Q_\exists$
Alternating tree-walking automata

- Two players $\forall, \exists$
- Each state belongs to a player: $Q = Q_\forall \cup Q_\exists$
- If $q \in Q_\forall$, then $\forall$ plays the next move in a given set of rules, otherwise, $\exists$ does
Alternating tree-walking automata

- Two players $\forall, \exists$
- Each state belongs to a player: $Q = Q_\forall \sqcup Q_\exists$
- If $q \in Q_\forall$, then $\forall$ plays the next move in a given set of rules, otherwise, $\exists$ does
- A tree is accepted if $\exists$ wins, rejected if $\forall$ does
Alternating tree-walking automata

- Two players $\forall, \exists$
- Each state belongs to a player: $Q = Q\forall \sqcup Q\exists$
- If $q \in Q\forall$, then $\forall$ plays the next move in a given set of rules, otherwise, $\exists$ does
- A tree is accepted if $\exists$ wins, rejected if $\forall$ does
- $\exists$ wins if an accept rule is played by someone or if $\forall$ has no possible move, otherwise $\forall$ wins
Alternating tree-walking automata

- Two players $\forall, \exists$
- Each state belongs to a player: $Q = Q_\forall \cup Q_\exists$
- If $q \in Q_\forall$, then $\forall$ plays the next move in a given set of rules, otherwise, $\exists$ does
- A tree is accepted if $\exists$ wins, rejected if $\forall$ does
- $\exists$ wins if an accept rule is played by someone or if $\forall$ has no possible move, otherwise $\forall$ wins
Alternating tree-walking automata

- Two players $\forall, \exists$
- Each state belongs to a player: $Q = Q_\forall \cup Q_\exists$
- If $q \in Q_\forall$, then $\forall$ plays the next move in a given set of rules, otherwise, $\exists$ does
- A tree is accepted if $\exists$ wins, rejected if $\forall$ does
- $\exists$ wins if an accept rule is played by someone or if $\forall$ has no possible move, otherwise $\forall$ wins

Expressiveness

Alternating TWA = MSO
Caterpillar expressions [BKW00]
Caterpillar expressions [BKW00]

- Caterpillar expressions describe runs of tree-walking automata
Caterpillar expressions [BKW00]

- Caterpillar expressions describe runs of tree-walking automata
- Caterpillar alphabet on $\Sigma$
  - Commands letter goleft, goright and goparent
Caterpillar expressions \cite{BKW00}

- Caterpillar expressions describe runs of tree-walking automata
- Caterpillar alphabet on $\Sigma$
  - Commands letter goleft, goright and goparent
  - Tests letter leaf, isleft, isright and labels $a \in \Sigma$
Caterpillar expressions [BKW00]

- Caterpillar expressions describe runs of tree-walking automata
- Caterpillar alphabet on $\Sigma$
  - Commands letter `goleft`, `goright` and `goparent`
  - Tests letter `leaf`, `isleft`, `isright` and labels $a \in \Sigma$
- Caterpillar words describe paths in TWA: `isleft a goleft b` describes paths going from a left child labeled $a$ to its left child labeled $b$
Caterpillar expressions [BKW00]

- Caterpillar expressions describe runs of tree-walking automata
- Caterpillar alphabet on $\Sigma$
  - Commands letter goleft, goright and goparent
  - Tests letter leaf, isleft, isright and labels $a \in \Sigma$
- Caterpillar words describe paths in TWA: isleft $a$ goleft $b$
  describes paths going from a left child labeled $a$ to its left child labeled $b$
- Caterpillar expressions: regular expressions on caterpillar alphabet
Cutting caterpillar expressions

- New letters:
  - test $\langle c \rangle$ (nest) where $c$ is a caterpillar expression: true if $c$ applied to current node selects at least one path
New letters:

- test $\langle c \rangle$ (nest) where $c$ is a caterpillar expression: true if $c$ applied to current node selects at least one path
- command cut transform the whole tree into the subtree of the current node — local transform, does not apply outside nests
New letters:

- test $\langle c \rangle$ (nest) where $c$ is a caterpillar expression: true if $c$ applied to current node selects at least one path
- command cut transform the whole tree into the subtree of the current node — local transform, does not apply outside nests

Expressiveness: if nesting is forbidden under scope of negation, \( \text{posCAT} = \text{PTWA} \)
Cutting caterpillar expressions

- New letters:
  - test $\langle c \rangle$ (nest) where $c$ is a caterpillar expression: true if $c$ applied to current node selects at least one path
  - command cut transform the whole tree into the subtree of the current node — local transform, does not apply outside nests

- Expressiveness: if nesting is forbidden under scope of negation, posCAT = PTWA

- Expressiveness: if nesting is allowed under the scope of a negation, as expressive as nested TWA (not defined here)
Expressiveness (ranked case)

\[ A \rightarrow B \quad A \subseteq B \]

\[ A \leftrightarrow B \quad A \subseteq B \]

\[ A \rightarrow B \quad A \not\subseteq B \]

\[ \text{MSO} \]

\[ \text{nested TWA} \]

\[ \text{TWA}_{\text{pebble}} \]

\[ \text{detTWA}_{\text{pebble}} \]

\[ \text{TWA} \]

\[ [\text{BC04}] \]

\[ [\text{BSSS06}] \]

\[ \text{FO} \]

\[ [\text{BC05}] \]

\[ [\text{BC04}] \]

\[ [\text{BC05}] \]
FO: Extensions

Notation: \( \bar{z} = (z_1, \ldots, z_n) \)

Adding Transitive Closure: \( TC^n \)

\[
TC^n[\varphi(\bar{x}, \bar{y})](\bar{u}, \bar{v})
\]

iff

\[
\exists k, \exists (\bar{w}_i)_{i \in [1..k]}, \varphi(\bar{u}, \bar{w}_1) \land \varphi(\bar{w}_1, \bar{w}_2) \land \ldots \land \varphi(\bar{w}_k, \bar{v})
\]

By \( TC^n \), we mean “parameter-free” transitive closure, i.e., \( \bar{x} \) and \( \bar{y} \) are exactly the free variable of \( \varphi \).

We write \( TC_p^n \) for the non-parameter-free transitive closure (i.e., \( \varphi \) can have extra free variables).
**FO: Extensions**

\[ FO + TC_1^p = \text{nested TWA} \quad [tCS08] \]

FO + TC\(^1\) is often written FO\(^*\), and FO + TC\(_p^1\) is written FO(\(MTC\)).

**FO + TC\(^1\) \subseteq FO + TC\(_p^1\):** it is unknown whether it is strict.

**FO + TC\(^1\) \subseteq MSO**

because \( TC_1^1[\varphi(x, y)](u, v) \iff \forall X. (u \in X \land \forall (x, y). (x \in X \land \varphi(x, y) \Rightarrow y \in X) \Rightarrow v \in X) \)

**FO \nsubseteq FO + TC\(^1\) \nsubseteq MSO**

- Transitive closure is not expressible in FO [Fag75].
- Adding TC\(^1\) to FO is not enough to reach MSO [tCS08].

For properties of FO + TC\(^1\) see [Kep06].
Expressiveness (ranked case)

\[
\begin{align*}
A & \rightarrow B & A & \not\subseteq B \\
A & \not\rightarrow B & A & \subseteq B \\
A & \rightarrow B & A & \not\in B
\end{align*}
\]

\[
\begin{align*}
\text{FO + TC}^* \\
\text{MSO} \\
\text{FO + TC}^1 \\
\text{TWA}_{\text{pebble}} \\
\text{detTWA}_{\text{pebble}} \\
\text{TWA} \\
\text{FO}
\end{align*}
\]

[BC04] [BC05] [BSSS06] [BC04] [tCS08] [TK06]
FO: *Extensions*

FO + $TC^2$

**FO + $TC^2$ $\not\subseteq$ MSO**

(cf next slide)

**MSO $\subseteq$ FO + $TC^2$?**

This is an open question. It could be the case that MSO $\not\subseteq$ FO + $TC^k$, for all $k$. 
For instance $L = \{ f(X, X) \mid X \in T_\Sigma \}$ is defined by:

$$
\varphi = \text{label}_f(\epsilon) \land \\
\exists u_1. \exists v_1. \text{fc}(\epsilon, u_1) \land \text{ns}(u_1, v_1) \land \text{samelabel}(u_1, v_1) \land \\
\neg(\exists w. \text{ns}(v_1, w)) \land \\
\forall u_2. \text{ch}_*(u_1, u_2) \Rightarrow \exists v_2. \text{TC}^2[\psi(\bar{x}, \bar{y})](u_1, u_2, v_1, v_2)
$$

where $\psi$ encodes a step isomorphism:

$$
\psi(\bar{x}, \bar{y}) = \text{samelabel}(x_2, y_2) \land \\
(f_c(x_1, x_2) \land f_c(y_1, y_2)) \lor \text{ns}(x_1, x_2) \land \text{ns}(y_1, y_2))
$$

with:

$$
\text{samelabel}(x, y) = \bigvee_{a \in \Sigma} \text{label}_a(x) \land \text{label}_a(y)
$$
Expressiveness (ranked case)

A \rightarrow B \quad A \subseteq B
A \quad \not\in \quad B
A \rightarrow B \quad A \subseteq B

FO + TC\^*

\[TK06\]

tree isomorphism

FO + TC\^2

[BC04]

TWA_{pebble}

[BC05]

detTWA_{pebble}

[BC04]

TWA

[BC05]

detTWA

[BC05]
FO: *Extensions*

\[
\text{FO} + det\ TC^1 = det\ TWA_{\text{pebble}} \quad [\text{EH06}]
\]

Deterministic Transitive Closure of \( \varphi = TC \) on the functional part of \( \varphi \)

\[
\text{FO} + det\ TC^1 \subseteq \text{FO} + TC^1
\]

because

\[
det\ TC^1[\varphi(x, y)](u, v) \Leftrightarrow TC^1[\varphi(x, y) \land \forall z. \varphi(x, z) \Rightarrow z = y](u, v)
\]

\[
\text{FO} + det\ TC^1 \subsetneq \text{FO} + TC^1?
\]

Open question (see [Kep06]).

For some properties of \( \text{FO} + det\ TC^1 \) (linear order, even...) see [Kep06, EI95].
FO: Extensions

\[ \text{FO} + posTC^1 = \text{TWA}_{\text{pebble}} \quad [\text{EH06}] \]

formulas of \( \text{FO} + TC^1 \) with \( TC^1 \) operators under an even number of negations

\[
\text{FO} + detTC^1 \subseteq \text{FO} + posTC^1 \subseteq \text{FO} + TC^1
\]

- inclusions due to TWA characterisations
- whether these 2 inclusions are strict is still open

\[
\text{FO} + TC^1 \subsetneq \text{MSO}
\]

- separation language based on the branching structure [tCS08]
Expressiveness (ranked case)

\[ A \rightarrow B \quad A \subsetneq B \]
\[ A \rightarrow B \quad A \subseteq B \]
\[ A \rightarrow B \quad A \nsubseteq B \]

\[ \text{FO} + TC^* \]
\[ \text{FO} + TC^2 \]
\[ \text{MSO} \]

\[ \text{FO} + TC^1 \quad \text{[tCS08]} \]
\[ \text{FO} + posTC^1 \quad \text{[EH06]} = \text{TWA}_{\text{pebble}} \]
\[ \text{FO} + detTC^1 \quad \text{[EH06]} = \text{detTWA}_{\text{pebble}} \]
\[ \text{TWA} \quad \text{[Kep06, BSSS06]} \]
\[ \text{detTWA} \quad \text{[BC05]} \]
\[ \text{FO} \quad \text{[BC05]} \]

\[ \text{tree isomorphism} \]

\[ \text{queries on trees} \]
Outline

1. Classical logics (FO, MSO)

2. Queries by Tree Automata
   - Tree-walking automata
   - Schema Languages & Tree Automata
XML Schema Languages

- Describe a set of XML documents
- Theoretical framework: no data, only structure
- Closer to tree grammars [MLM01] than to tree automata
- Tree automata: reference model for the expressiveness
Local tree languages

\[ L = \{ a \rightarrow q_a, a'(q_b) \rightarrow q_{a'}, b(q_c q_d) \rightarrow q_b, b(q_d q_c) \rightarrow q_b, c(\epsilon) \rightarrow q_c, d(\epsilon) \rightarrow q_d \} \]

- \( a \in L! \)

Restriction: no competing states

Deterministic content models

- One-unambiguous regular expressions [BKW98]
- \( ab + ac \): which \( a \) to match depends on the next symbol

Polynomial complexity for other usual decision problems (membership, emptiness, containment), except intersection [MNS04]

Lack of expressivity
Extended DTDs (EDTDs) [MNSB06, Sch07]

- Alphabet extended with types (each type is associated to a unique symbol)

\[
L = \{ \begin{array}{c}
    a \\
    a' \\
    b_1 \\
    b_2 \\
    c \\
    d \\
    d' \\
    c \\
\end{array} \}
\]

- Typing problem:
  - Valid assignment of types to the elements w.r.t. EDTD
  - (Consistent) combination of unary queries

- As expressive as (parallel) unranked tree automata of [BKWM01], thus equivalent to regular tree languages
  - Examples of such schema languages: Relax NG [CM01], XDuce [HP03]
  - Restricted EDTDs: single-type, restrained-competition
Extended DTDs (EDTDs) [MNSB06, Sch07] II

- **Single-type EDTDs**

  \[ a(q_{b_1} + q_{b_2}) \rightarrow q_a \]
  \[ b^1(\epsilon) \rightarrow q_{b_1} \]
  \[ b^2(\epsilon) \rightarrow q_{b_2} \]

  \[ a^1(q_{b_1}) \rightarrow q_{a_1} \]
  \[ a^2(q_{b_2}) \rightarrow q_{a_2} \]

  \[ b^1(\epsilon) \rightarrow q_{b_1} \]
  \[ b^2(\epsilon) \rightarrow q_{b_2} \]

- **Element Declaration Consistent constraint (W3C XML Schemas)**
- **Unique top-down typing**
- **Validation with deterministic tree-walking automata**

- **Restrained-competition EDTDs**

  \[ a(q_{b_1} \cdot q_{b_2}) \rightarrow q_a \]
  \[ b^1(\epsilon) \rightarrow q_{b_1} \]
  \[ b^2(\epsilon) \rightarrow q_{b_2} \]

- **Unique top-down left-to-right typing**
- **Validation with deterministic top-down tree automata**
Expressiveness of Schemas

\[ \text{EDTD} = \text{MSO} = \text{UTA} \]

(Homogeneous) regular tree languages

Path-closed tree languages

Local tree languages

EDTD

Top-down DetTA

Restrained-competition EDTD

Single-type EDTD

DTD

DTWA
Part II

Conjunctive Queries, Monadic Datalog
Outline

3 Conjunctive Queries over Trees
   • Definition, results and acyclic fragment
   • Twigs and Tree Patterns

4 Monadic Datalog
Conjunctive Queries

... seen as FO formulas

\[ \exists \bar{x}. \phi(\bar{x}, \bar{y}) \text{ where } \phi \text{ is a conjunction of atomic predicates.} \]

For instance:

\[ \exists x \exists y \exists w \ R_1(x) \land R_2(x, y) \land R_3(x, w, z) \]

... seen as rules

\[ \text{answer}(z) \leftarrow R_1(x), R_2(x, y), R_3(x, w, z) \]

... seen as terms of the Projection/Join algebra

\[ \pi_Z( R_1(X) \bowtie R_2(X, Y) \bowtie R_3(X, W, Z) ) \]

These 3 formalisms are equivalent (see [AHV95]).
Conjunctive Queries over Trees

XPath axis $\mathcal{X}$: $\mathit{ch}, \mathit{ch}^*, \mathit{ch}^+, \mathit{ns}, \mathit{ns}^*, \mathit{ns}^+$, following and their inverse

following $= (\mathit{ch}^*)^{-1} \circ \mathit{ns}^+ \circ \mathit{ch}^*$

Example

$\exists x \exists y \; \mathit{ch}^+(x, y) \land \mathit{ch}^+(x, z) \land \text{following}(x, z)$
Theorem ([GKS04])

*Evaluation of Boolean CQ over $\mathcal{X}$ is NP-complete, even on a fixed tree.*

**Tractable fragments**

- $\mathcal{X}$ underbar property
- Acyclic conjunctive queries
- Twigs
**X** property

- $R$: a binary relation on the domain $D_t$ of a tree $t$
- a total order $\prec$ on $D_t$

### Definition

The relation $R$ satisfies the **X** property wrt $\prec$ if $\forall n_1, n_2, n_3, n_4$ st $n_1 \prec n_2$ and $n_3 \prec n_4$:

A set of relations $R_1, \ldots, R_n$ satisfies **X** wrt $\prec$ if every $R_i$ does.
X property: Example

- \{ch^+, ch^*\} for the preorder \(<_\text{pre}(ch^+(x,y) \Rightarrow x <_\text{pre} y)\)
- \{ch, ns, ns^+, ns^*\} for \(<_\text{bfir}\)
- but not following for \(<_\text{pre}\)
Theorem (Gottlob, Koch, Schulz, 2004)

For all $F \subseteq \mathcal{X}$, $CQ[F]$ Boolean queries can be evaluated in PTIME iff there is a total order $<$ such that $F$ satisfies the $\mathcal{X}$ property wrt $<$. 

Question: generalization to $n$-ary queries? Which complexity measure?

→ polynomial in the number of answers.
Acyclic Conjunctive Queries (ACQ)

**Acyclic:** the query graph is acyclic

\[ \exists x \exists y \exists z, \text{ns}(x, y) \land \text{ch}_*(y, z) \]
Expressiveness

- [GKS04]

\[ CQ[\mathcal{X}] \subsetneq \bigcup \text{ACQ}[\mathcal{X}] \subseteq \text{FO}[\mathcal{X}] \]

exponential

- [Mar05b], over unranked trees,

\[ \bigcup \text{ACQ}[\text{FO}_2] = \text{FO}_{\text{nary}} \]
ACQ Evaluation

- Yannakakis algorithm: \( O(|q| \cdot |db| \cdot |q(db)|) \)

\[ \exists x \ R(x, y) \land R'(x, z) \]

- on trees \( t \) with predicates \( \mathcal{X} \): \( O(|q| \cdot |t|^2 \cdot |q(t)|) \)

\[ \exists x \ \text{ch}_*(x, y) \land \text{ns}_*(x, z) \]
3 Conjunctive Queries over Trees
- Definition, results and acyclic fragment
- Twigs and Tree Patterns

4 Monadic Datalog
Twigs: Testing containment [MS02]

Tree pattern

- (unordered and unranked) tree labeled with elements from \( \Sigma \cup \{\ast\} \)
- \( child \) and \( descendant \) edges
- \( n \) distinguished querying nodes (\( n \)-ary query)
- unary tree patterns (\( n = 1 \)) equivalent to XPath\((\ast, [], //, /)\)

\[
\begin{align*}
  p_1 & \subseteq p_2 \text{ if and only if } \text{Ans}(p_1, t) \subseteq \text{Ans}(p_2, t) \text{ for every } t \in T_{\Sigma} \\
\end{align*}
\]
Booleanize your twigs

**Boolean tree patterns**

Tree patterns $p$ with no querying nodes ($n = 0$)

$$\text{Mod}(p) = \{ t \in T_\Sigma | t \text{ satisfies } p \}$$

Then, $p_1 \subseteq p_2$ if and only if $\text{Mod}(p_1) \subseteq \text{Mod}(p_2)$.

**Proposition**

For any two $n$-ary tree patterns $p_1$ and $p_2$: $p_1 \subseteq p_2 \iff p_1^B \subseteq p_2^B$.
Canonical models of Boolean twigs

\[ p : \begin{array}{c}
  a \\
  \downarrow \\
  b \\
  \downarrow \\
  c \\
\end{array} \quad \rightarrow \quad \begin{array}{c}
  a \\
  \downarrow \\
  b \\
  \downarrow \\
  c \\
\end{array} \quad \begin{array}{c}
  a \\
  \downarrow \\
  b \\
  \downarrow \\
  \ast \\
\end{array} \quad \begin{array}{c}
  a \\
  \downarrow \\
  b \\
  \downarrow \\
  \ast \\
\end{array} \quad \ldots \]
Canonical models of Boolean twigs
Canonical models of Boolean twigs

\[ p : \begin{array}{c} a \\ \Downarrow \begin{array}{c} b \\ c \end{array} \\ * \\ \Downarrow \begin{array}{c} z \\ z \\ c \end{array} \\ \\ a \\
\end{array} \rightarrow \begin{array}{c} b \\ c \\ z \\ z \\ c \\ \end{array} \]

\[ \text{mod}(p) \]

**Proposition**

For any Boolean tree patterns \( p_1 \) and \( p_2 \):

\[ p_1 \subseteq p_2 \iff \text{mod}(p_1) \subseteq \text{Mod}(p_2). \]
Testing containment of Boolean twigs: Outline

\[ \begin{array}{ccc}
\text{unranked} & \text{ranked} & \text{unranked} \\
\begin{array}{c}
a \\
\downarrow \quad \quad \downarrow \\
b & c \\
\downarrow & \downarrow \\
\ast & \ast \\
p & L_p & \mod(p) \\
\end{array} & \begin{array}{c}
a \\
\downarrow \quad \quad \downarrow \\
b & * \\
\downarrow & \downarrow \\
\ast & * \\
U_p & L_p & c \\
\end{array} & \begin{array}{c}
a \\
\downarrow \quad \quad \downarrow \\
b & z \\
\downarrow & \downarrow \\
\ast & \ast \\
U_p & \mod(p) & c \\
\end{array}
\end{array} \]
Main idea

\[ p_1 \subseteq p_2 \iff \text{mod}(p_1) \subseteq \text{Mod}(p_2) \iff U_{p_1}(L_{p_1}) \subseteq \text{Mod}(p_2) \]
\[ \iff L_{p_1} \subseteq U_{p_1}^{-1}(\text{Mod}(p_2)) \iff A_{p_1} \subseteq A_{p_2} , \]

where:

- \( A_{p_1} \): DFTA defining \( L_{p_1} \)
- \( A_{p_2} \): AFTA defining \( U_{p_1}^{-1}(\text{Mod}(p_2)) \)

complexity \( O(|p_1|^2|p_2|) \)
Testing containment: Conclusions

Positive results

$p_1 \subseteq p_2$ can be decided in time $O(|p_1||p_2|w^d)$, where:
- $d$ is the number of $//$/edges in $p_1$
- $w$ is the maximal length of $*/ */\ldots/*$ in $p_2$

Negative results

Deciding containment is coNP-complete. The result holds even if we:
- bound the number of occurrences of $*$
- bound the degree of the nodes of tree patterns
Efficient evaluation of tree patterns

**TwigStack [BKS02]**

- Interval representation used with a variant of B-tree index
- Two phase approach:
  1. Find and stack (partial) solutions to leaf-to-root paths
  2. Join partial solutions
- Linear in the size of the input and output
- I/O and CPU optimal if only /-/edges used

**Twig²Stack [CLT⁺06]**

- Generalized tree pattern queries
- One phase bottom-up approach
- May stack elements that are not solutions
- In the worst case the whole document may be stored in main memory
- HollisticTwigStack [JLH⁺07] addresses this shortcoming
Outline

3 Conjunctive Queries over Trees
- Definition, results and acyclic fragment
- Twigs and Tree Patterns

4 Monadic Datalog
Overview

- Few words on datalog
- Least fixed point
- Monadic datalog over trees
Datalog in (Very) Few Words

- Language used in deductive databases
- Extends conjunctive queries with recursion
- Example: transitive closure of a graph

\[
TC(x, y) : \neg \text{Edge}(x, y).
\]
\[
TC(x, y) : \neg \text{Edge}(x, z), TC(z, y).
\]

Model theoretic point of view:

\[
\forall x, y (\text{Edge}(x, y) \rightarrow TC(x, y))
\]
\[
\forall x, y, z ((\text{Edge}(x, z) \land TC(z, y)) \rightarrow TC(x, y))
\]

- Remark: no function symbols (finite models), no negation

See chapter 12 of [AHV95] for more details
Least Fixed Point I

- $P$ is a *fixed point* of operator $F$ if $F(P) = P$
- The *least fixed point* $lfp(F)$ is the least element of the set of fixed points of $F$ w.r.t. inclusion
- Every monotone operator $F$ (i.e., $P \subseteq Q \Rightarrow F(P) \subseteq F(Q)$) has a least fixed point (Knaster-Tarski, cited by [Lib04]):

$$lfp(F) = \bigcap\{P | F(P) = P\}$$

- Computing the least fixed point (standard closure):

\[
\begin{align*}
P^0 &= \emptyset \\
P^{i+1} &= F(P^i) \\
lfp(F) &= P^\infty = \bigcup_{i=0}^{\infty} P^i
\end{align*}
\]

Stabilizes after $n$ steps on finite structures, i.e., $P^\infty = P^n$
Least Fixed Point II

- Datalog immediate consequence operator $T_P$ (from [GK04]):

\[
T_P(Q) := Q \cup \{ f \mid \exists \phi, \exists h: - b_1, \ldots, b_n \in \mathcal{P} \\
\phi(h) = f \\
\phi(b_1), \ldots, \phi(b_n) \in Q \}
\]

- Example: program $\mathcal{P} = \{ TC(x, y) :- Edge(x, y). \\
TC(x, y) :- Edge(x, z), TC(z, y). \}$ and database $Q = \{ Edge(1, 2), Edge(2, 3), Edge(3, 1) \}$

\[
T^0_P = Q = \{ Edge(1, 2), Edge(2, 3), Edge(3, 1) \} \\
T^1_P = T^0_P \cup \{ TC(1, 2), TC(2, 3), TC(3, 1) \} \\
T^2_P = T^1_P \cup \{ TC(1, 3), TC(2, 1), TC(3, 2) \} \\
T^3_P = T^2_P
\]

Finally, $\text{lfp}(T_P)^{\text{notation}} = T^\omega_P = T^3_P = T^2_P = \{ Edge(1, 2), Edge(2, 3), Edge(3, 1), TC(1, 2), TC(2, 3), TC(3, 1), \ldots \}$
Monadic Datalog over Trees

- Datalog with unary head predicates
- Built-in predicates (for binary trees): root, leaf, \((\text{label}_a)_{a \in \Sigma}\), ch\(_1\), ch\(_2\)
- Example of query: select all nodes labeled by \(a\) at even height

\[
\begin{align*}
Q_0(x) & :\leftarrow \text{root}(x). \\
Q_{(i+1) \mod 2}(x) & :\leftarrow Q_i(y), \text{ch}_k(y, x). \quad \text{(for } k \in \{1, 2\}) \\
\text{Ans}(x) & :\leftarrow Q_0(x), \text{label}_a(x).
\end{align*}
\]

The query predicate is Ans
Monadic Datalog over Trees: Complexity

Model Checking

*Over ranked as well as unranked trees, monadic datalog has* $O(|\mathcal{P}| \ast |\text{dom}|)$ *combined complexity* (theo. 4.2 of [GK04])

Proved by rewriting of $\mathcal{P}$ such that it is ground.

Satisfiability

*Monadic datalog (over arbitrary finite structures) is NP-complete w.r.t. combined complexity* (prop. 3.4 of [GK04])

- Membership: guess a proof tree
- Hardness: boolean conjunctive queries

For trees, satisfiability can be reduced to the emptiness problem for context-free languages [?]. What about the complexity?
Equivalence with MSO

A tree language is definable in monadic datalog exactly if it is definable in MSO (coro. 4.7 of [GK04])

Sketch of proof (for monadic queries):

⇒ Encode the query defined by a monadic datalog program into an MSO formula (prop. 3.3 of [GK04])

⇐ More intricate, different ways of proving it:

1. Using $\equiv^\text{MSO}_k$-types (theo. 4.4 of [GK04])
2. Simulating query automata of Neven & Schwentick [NS02] (Section 4.3 of [GK04])
3. Encoding tree automata with selecting states? (next slides)
Encoding a Tree Automaton $A$ into a Monadic Datalog Program $\mathcal{P}$

$R_q(x)$ in $lfp$ of $\mathcal{P}$ if a run of $A$ can evaluate node $x$ in state $q$:

$$a \rightarrow q \in \text{rules}(A)$$

$$R_q(x) :\neg \text{leaf}(x), \text{label}_a(x).$$

$$f(q_1, q_2) \rightarrow q \in \text{rules}(A)$$

$$R_q(x) :\neg R_{q_1}(y), R_{q_2}(z), \text{ch}_1(x, y), \text{ch}_2(x, z), \text{label}_f(x).$$
Encoding a Tree Automaton $A$ into a Monadic Datalog Program $\mathcal{P}$ II

$L2F_q(x)$, aka $\text{ LeadsToFinal}_q(x)$, in $\text{lfp}$ of $\mathcal{P}$ if state $q$ is used in a successful run of $A$:

\[
q \in \text{final}(A) \\
\frac{}{L2F_q(x) : - \text{root}(x)}.
\]

\[
f(q_1, q_2) \rightarrow q \in \text{rules}(A) \\
\frac{}{L2F_{q_1}(y) : - L2F_q(x), \text{ch}_1(x, y), \text{ch}_2(x, z), \text{label}_f(x), R_{q_2}(z).}
\]

\[
L2F_{q_2}(z) : - L2F_q(x), \text{ch}_1(x, y), \text{ch}_2(x, z), \text{label}_f(x), R_{q_1}(y).
\]
Encoding a Tree Automaton $A$ into a Monadic Datalog Program $\mathcal{P}$ III

$\text{Ans}(x)$ in $lfp$ of $\mathcal{P}$ if $x$ is selected by automaton $A$, i.e., $x$ is evaluated in state $q \in S$, where $S \subseteq \text{states}(A)$ is the set of selecting states:

\[
q \in S \quad \frac{}{\text{Ans}(x) \leftarrow R_q(x), L2F_q(x)}.
\]

**Proposition:** Monadic datalog program $\mathcal{P}$ with $\text{Ans}$ as query predicate simulates tree automaton with selecting states $A$
Part III

$\mu$-calculus, Modal Logics (Temporal Logics, XPath...)

PhDs+Sławek (Mostrare)
Outline

5 $\mu$-calculus

6 XPath

7 Temporal Logics
The structure used here is the one used by Barceló and Libkin. Most of the results are taken from [BL05a, ABL07].

Tree \( t \) with two relations (or more) on position: child \( \prec_{ch} \) and next sibling \( \prec_{ns} \).

Formulae of \( L_\mu[\prec] \):

- constants \( a \)
- second order variables \( X \)
- \( \top, \bot, \neg \varphi, \varphi \lor \varphi' \)
- \( \diamond (\prec) \varphi \)
- \( \mu X. \varphi \) where \( X \) can only appear positively in \( \varphi \)
Given a tree $t$, nodes $s, s' \in \text{Domain}(t)$ and a valuation $v : \mathcal{X} \rightarrow \mathcal{P}(\text{Domain}(t))$

- logic operators are interpreted as usual
  - $(t, v, s) \models a$ iff $t(s) = a$
  - $(t, v, s) \models X$ iff $s \in v(X)$
  - $(t, v, s) \models \Diamond (\prec) \phi$ iff $(t, v, s') \models \Diamond (\prec) \phi$ for some $s'$ such that $s \prec s'$
  - $(t, v, s) \models \mu X.\phi$ iff $s \in S$ where $S$ is the least fix point of $F\phi$, defined by $F\phi(P) = \{ s' \mid (t, v[P/X], s') \models \phi \}$
Interpretation

- \((t, v, s) \models \mu X.\phi(X)\) iff \(s \in S\) where \(S\) is the least fix point of \(F\)
- Problem: is there a least fix point?
- The function \(P \mapsto \{s' \mid (t, v[P/X], s') \models \phi\}\) is monotonically increasing because \(X\) can only appear positively in \(\mu X.\phi\)
Interpretation

- $(t, v, s) \models \mu X.\phi(X)$ iff $s \in S$ where $S$ is the least fix point of $F$
- Problem: is there a least fix point?
- The function $P \mapsto \{s' \mid (t, v[P/X], s') \models \phi\}$ is monotonically increasing because $X$ can only appear positively in $\mu X.\phi$
  - $F_a, F_T, F_\perp, F_Y$ are constant
Interpretation

- $(t, v, s) \models \mu X. \phi(X)$ iff $s \in S$ where $S$ is the least fix point of $F$
- Problem: is there a least fix point?
- The function $P \mapsto \{s' \mid (t, v[P/X], s') \models \phi\}$ is monotonically increasing because $X$ can only appear positively in $\mu X. \phi$
  - $F_a, F_T, F_\bot, F_Y$ are constant
  - $F_X(P) = P$ is increasing
Interpretation

- \((t, v, s) \models \mu X.\phi(X)\) iff \(s \in S\) where \(S\) is the least fix point of \(F\)
- Problem: is there a least fix point?
- The function \(P \mapsto \{s' \mid (t, v[P/X], s') \models \phi\}\) is monotonically increasing because \(X\) can only appear positively in \(\mu X.\phi\)
  - \(F_a, F_T, F_\bot, F_Y\) are constant
  - \(F_X(P) = P\) is increasing
  - if \(F_\phi\) and \(F'_\phi\) both are increasing (resp. decreasing), then \(F_{\phi \lor \phi'}(P) = F_\phi(P) \cup F'_\phi(P)\) is increasing (resp. decreasing)
Interpretation

- $(t, v, s) \models \mu X.\phi(X)$ iff $s \in S$ where $S$ is the least fix point of $F$
- Problem: is there a least fix point?
- The function $P \mapsto \{s' \mid (t, v[P/X], s') \models \phi\}$ is monotonically increasing because $X$ can only appear positively in $\mu X.\phi$
  - $F_a, F_T, F_\perp, F_Y$ are constant
  - $F_X(P) = P$ is increasing
  - if $F_\phi$ and $F'_\phi$ both are increasing (resp. decreasing), then $F_{\phi \lor \phi'}(P) = F_\phi(P) \cup F'_\phi(P)$ is increasing (resp. decreasing)
  - if $F_\phi$ is increasing (resp. decreasing) then $F_{\phi(\prec)}\phi$ is increasing (resp. decreasing)
Interpretation

- \((t, v, s) \models \mu X.\phi(X)\) iff \(s \in S\) where \(S\) is the least fix point of \(F\)
- Problem: is there a least fix point?
- The function \(P \mapsto \{s' \mid (t, v[P/X], s') \models \phi\}\) is monotonically increasing because \(X\) can only appear positively in \(\mu X.\phi\)
  - \(F_a, F_T, F_\bot, F_Y\) are constant
  - \(F_X(P) = P\) is increasing
  - if \(F_\phi\) and \(F'_\phi\) both are increasing (resp. decreasing), then \(F_{\phi \lor \phi'}(P) = F_\phi(P) \cup F'_\phi(P)\) is increasing (resp. decreasing)
  - if \(F_\phi\) is increasing (resp. decreasing) then \(F_\Diamond(\prec)\phi\) is increasing (resp. decreasing)
  - if \(F_\phi\) is increasing (resp. decreasing), then \(F_{\neg\phi}(P) = F_\phi(P)\) is decreasing (resp. increasing)
Interpretation

- \((t, \nu, s) \models \mu X.\phi(X)\) iff \(s \in S\) where \(S\) is the least fix point of \(F\)
- Problem: is there a least fix point?
- The function \(P \mapsto \{s' \mid (t, \nu[P/X], s') \models \phi\}\) is monotonically increasing because \(X\) can only appear positively in \(\mu X.\phi\)
  - \(F_a, F_T, F_\bot, F_Y\) are constant
  - \(F_X(P) = P\) is increasing
  - if \(F_\phi\) and \(F'_\phi\) both are increasing (resp. decreasing), then \(F_{\phi \lor \phi'}(P) = F_\phi(P) \cup F'_\phi(P)\) is increasing (resp. decreasing)
  - if \(F_\phi\) is increasing (resp. decreasing) then \(F_{\phi(\prec)}\) is increasing (resp. decreasing)
  - if \(F_\phi\) is increasing (resp. decreasing), then \(F_{\neg \phi}(P) = F_\phi(P)\) is decreasing (resp. increasing)
  - if \(F_\phi\) is increasing then \(F_{\mu X.\phi}(P)\) is increasing
Unary and boolean queries

A formula $\phi$ from $L_\mu$ can be used as a unary query which selects in $t$ the nodes $s$ such that

$$(t, .., s) \models \phi$$
Unary and boolean queries

A formula $\phi$ from $L_\mu$ can be used as a unary query which selects in $t$ the nodes $s$ such that

$$(t, \ldots, s) \models \phi$$

A formula $\phi$ from $L_\mu$ can be used as a boolean query which accepts a tree $t$ iff

$$(t, \ldots, \varepsilon) \models \phi$$

Example

Selects nodes which are ancestors of a node labelled by $a$:

$$\mu X. (a \lor \diamond (\prec_{ch} X)).$$
Expressiveness of boolean queries

- $L_\mu[\prec_{\text{ch}}, \prec_{\text{ns}}]$ cannot express first child...
  - $L_\mu[\prec_{\text{ch}}, \prec_{\text{ns}}, \prec_{\text{fc}}]$
  - $L_\mu^{\text{full}}[\prec_{\text{ch}}, \prec_{\text{ns}}]$: one can use $\diamond (\sim \phi)$, where $s \sim s'$ iff $s' \prec s$
- $L_\mu[\prec_{\text{ch}}, \prec_{\text{ns}}, \prec_{\text{fc}}] = L_\mu^{\text{full}}[\prec_{\text{ch}}, \prec_{\text{ns}}] = \text{MSO}$
One can rewrite any $L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}]$ formula into a MSO query as follow:

- $\langle a \rangle(x) = \text{label}_a(x)$

- $\langle \mu X. \phi \rangle(z) = \exists X \ (z \in X \land \forall x \in X \Rightarrow \langle \phi \rangle(x) \land (\forall Y (\forall y \in Y \Rightarrow \langle \phi \rangle(y)) \Rightarrow X \subseteq Y))$
One can rewrite any $L_{\mu}[\prec_{\text{ch}}, \prec_{\text{ns}}, \prec_{\text{fc}}]$ formula into a MSO query as follow:

- $\langle a \rangle(x) = \text{label}_a(x)$
- $\langle X \rangle(x) = x \in X$
- $\langle \mu X. \phi \rangle(z) = \exists X \ (z \in X \land \forall x \in X \Rightarrow \langle \phi \rangle(x) \land (\forall Y (\forall y \in Y \Rightarrow \langle \phi \rangle(y)) \Rightarrow X \subseteq Y))$
$L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}] \subseteq \text{MSO}$

One can rewrite any $L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}]$ formula into a MSO query as follow:

- $\langle a \rangle(x) = \text{label}_a(x)$
- $\langle X \rangle(x) = x \in X$
- $\langle \diamond (\prec_{ch}) \phi \rangle(x) = \exists y \mid ch(y,x) \land \langle \phi \rangle(y)$, ...
- $\langle \mu X. \phi \rangle(z) = \exists X \quad (z \in X \land \forall x \in X \Rightarrow \langle \phi \rangle(x) \land (\forall Y (\forall y \in Y \Rightarrow \langle \phi \rangle(y)) \Rightarrow X \subseteq Y))$
$L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}] \subseteq MSO$

One can rewrite any $L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}]$ formula into a MSO query as follow:

- $\langle a \rangle(x) = label_a(x)$
- $\langle X \rangle(x) = x \in X$
- $\langle \diamond(\prec_{ch})\phi \rangle(x) = \exists y \mid ch(y, x) \land \langle \phi \rangle(y)$, ...
- $\langle \mu X.\phi \rangle(z) = \exists X \ (z \in X \land \forall x \in X \Rightarrow \langle \phi \rangle(x) \land (\forall Y (\forall y \in Y \Rightarrow \langle \phi \rangle(y)) \Rightarrow X \subseteq Y))$
One can rewrite any $L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}]$ formula into a MSO query as follow:

- $\langle a \rangle(x) = \text{label}_a(x)$
- $\langle X \rangle(x) = x \in X$
- $\langle \Diamond(\prec_{ch})\phi \rangle(x) = \exists y \mid ch(y, x) \land \langle \phi \rangle(y)$, ...
- $\langle \mu X.\phi \rangle(z) = \exists X \ (z \in X \land \forall x \in X \Rightarrow \langle \phi \rangle(x) \land (\forall Y(\forall y \in Y \Rightarrow \langle \phi \rangle(y)) \Rightarrow X \subseteq Y))$

Finally, the whole query will be $\exists x \ \text{root}(x) \land \langle \phi \rangle(x)$
Given a MSO query, let \( A \) be an equivalent deterministic automaton. We can encode \( A \) with a \( L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}] \) formula.

**Example**

On ranked trees, \( \prec_{ch1}, \prec_{ch2} \),

Automaton \( Q = \{q_a, q_b\}, Q_F = \{q_a\} \)

\( a \rightarrow q_a, b \rightarrow q_b, f(q_a, q_b) \rightarrow q_a, f(q_b, q_a) \rightarrow q_b \)

\[
\mu X_a. a \lor f \land \Diamond (\prec_{ch1}) X_a \land \Diamond (\prec_{ch2}) (\mu X_b. b \lor f \land \Diamond (\prec_{ch1}) X_a \land \Diamond (\prec_{ch2}))
\]
Expressiveness
of unary queries

- $L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}]$ cannot express root...
- we need to use $L_\mu^{full}[\prec_{ch}, \prec_{ns}]$
- $L_\mu^{full}[\prec_{ch}, \prec_{ns}] = MSO$
Expressiveness of unary queries

- \( L_\mu[\prec_{\text{ch}}, \prec_{\text{ns}}, \prec_{\text{fc}}] \) cannot express root...
- we need to use \( L_\mu^{\text{full}}[\prec_{\text{ch}}, \prec_{\text{ns}}] \)
- \( L_\mu^{\text{full}}[\prec_{\text{ch}}, \prec_{\text{ns}}] = \text{MSO} \)
Expressiveness of unary queries

- \( L_\mu[\prec_{ch}, \prec_{ns}, \prec_{fc}] \) cannot express root...
- We need to use \( L^\text{full}_\mu[\prec_{ch}, \prec_{ns}] \)
- \( L^\text{full}_\mu[\prec_{ch}, \prec_{ns}] = MSO \)

Proofs: similar to Boolean queries, but with query automata instead
Complexities

- Because the structure of trees are acyclic, model checking of $L^\text{full}_\mu [\prec_{\text{ch}}, \prec_{\text{sb}}]$ can be computed in $O(|\phi|^2 |t|)$. Can be reduced for a subclass of $L_\mu$ (as expressive as MSO) to $O(|\phi| |t|)$.

- Satisfiability of $L^\text{full}_\mu [\prec_{\text{ch}}, \prec_{\text{sb}}]$ is EXPTIME (slightly better bounds in the case of tree than in the general case).
Outline

5 \( \mu \)-calculus

6 XPath

7 Temporal Logics
First-order modal logics
on Unranked Trees

Strong links between:

- XPath
- Modal Logics (temporal, propositional...)
- FO
First-order modal logics
on Unranked Trees

Strong links between:
- XPath
- Modal Logics (temporal, propositional...)
- FO

→ remember the first slides about the model and FO
First-order modal logics
on Unranked Trees

Strong links between:

- XPath
- Modal Logics (temporal, propositional...)
- FO

→ remember the first slides about the model and FO
→ we won’t talk about $\mathcal{L}$-definability (i.e., given an automaton, is it equivalent to a formula of the logic $\mathcal{L}$?). See [Boj08a] for a survey.
Binary vs Unranked Trees

FO-definable queries on binary trees?

- “select trees with even number of nodes”
Binary vs Unranked Trees

FO-definable queries on binary trees?

- “select trees with even number of nodes” ✓ (always false)
FO-definable queries on binary trees?

- “select trees with even number of nodes” ✓ (always false)
- “select trees with even number of a-nodes”
## FO-definable queries on binary trees?

- “select trees with even number of nodes” ✔ (always false)
- “select trees with even number of a-nodes” ✗
Binary vs Unranked Trees

FO-definable queries on binary trees?

- “select trees with even number of nodes” ✓ (always false)
- “select trees with even number of a-nodes” ×
- “select trees that have a leaf of even depth”
### FO-definable queries on binary trees?

- “select trees with even number of nodes” ✓ (always false)
- “select trees with even number of $a$-nodes” ×
- “select trees that have a leaf of even depth” ✓ (zigzag technic)
Binary vs Unranked Trees

FO-definable queries on binary trees?

- “select trees with even number of nodes” ✓ (always false)
- “select trees with even number of a-nodes” ×
- “select trees that have a leaf of even depth” ✓ (zigzag technic)

not clear whether the last query is FO-definable on unranked trees.
XPath 1.0: a W3C recommendation (since 1999)

Example:
/descendant::a[position() > last() * 0.5 or self::* = 100]

Features:

- select nodes (monadic queries)
- navigation through axis (child... following, preceding)
- node test and filters: /ax1::ntst1[f1][f2[f3]]/...
- context-sensitive functions (position, last...)
- element types (element, attribute, instruction, comments)
- arithmetic operators (+,−,...)
- data operators/comparators (string-length...)
- aggregators (count, sum...)
- identifiers functions...
- type conversion functions...
XPath axes [Shi08]
XPath 1.0

- first implementations: exponential time in the size of the query
- PTIME combined complexity obtained in [GKP02, GKP03a]: \( O(|D|^2 |Q|^4) \) in time, \( O(|D|^2 |Q|^2) \) in space.
XPath 1.0

- first implementations: exponential time in the size of the query
- PTIME combined complexity obtained in [GKP02, GKP03a]: $O(|D|^2 |Q|^4)$ in time, $O(|D|^2 |Q|^2)$ in space.

Questions:
- linear time fragment?
- expressiveness? links to other logics?
CoreXPath
The navigational core of XPath

- defined by Gottlob, Koch and Pichler [GKP02, GKP03a]
- restriction to navigation through axis, filters, and nodetests

\[
\begin{align*}
\text{locpath} & ::= \text{axis} :: \text{ntst} | \text{axis} :: \text{ntst[fexpr]} | /\text{locpath} | \text{locpath}/\text{locpath} \\
\text{fexpr} & ::= \text{locpath} | \text{not fexpr} | \text{fexpr and fexpr} | \text{fexpr or fexpr} \\
\text{axis} & ::= \text{self} | \text{ch} | \text{ch}_+ | \text{ch}_- | \text{ch}^{-1} | \text{ch}_{-1} | \text{ns}_+ | \text{ns}^{-1} \\
\text{ntst} & ::= a, a \in \Sigma | * \\
\end{align*}
\]

document order axis following and preceding are syntactic sugar:

- following :: ntst[fexpr] ≡ ch\_{-1} :: */ns\_+ :: */ch\_* :: ntst[fexpr]
- preceding :: ntst[fexpr] ≡ ch\_{-1} :: */ns\_+\_{-1} :: */ch\_* :: ntst[fexpr]
CoreXPath complexity [GKP03b]

- query evaluation becomes linear: $O(|D| \cdot |Q|)$
- it is P-hard wrt. combined complexity...
- ... even when $t$ is limited to depth 3 and only axes $ch$, $ch^{-1}$, $ch_*$ are allowed
- Positive-CoreXPath is LOGCFL-complete
- satisfiability is EXPTIME-complete
CoreXPath expressiveness

CoreXPath $\subseteq$ FO

<table>
<thead>
<tr>
<th>CoreXPath</th>
<th>/ch$_+$ :: a</th>
<th>[ch :: b]</th>
<th>/ch :: c</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td>$y$</td>
<td>$z$</td>
<td>$x$</td>
</tr>
<tr>
<td>$\phi(x) =$</td>
<td>$\exists y. \text{label}_a(y) \land \exists z. \text{label}_c(z) \land \text{label}_c(x)$</td>
<td>$\land \text{ch}(y, z)$</td>
<td>$\land \text{ch}(y, x)$</td>
</tr>
</tbody>
</table>
CoreXPath expressiveness

CoreXPath ⊆ FO

<table>
<thead>
<tr>
<th>CoreXPath</th>
<th>/ch⁺ :: a</th>
<th>[ch :: b]</th>
<th>/ch :: c</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td>y</td>
<td>z</td>
<td>x</td>
</tr>
<tr>
<td>φ(x) =</td>
<td>\exists y. labelₐ(y) \land \exists z. label₇(z) \land label₇(x) \land ch(y, z) \land ch(y, x)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FO ∉ CoreXPath

- example: select root if the leaf language is \((ab)^*\).
- in fact, CoreXPath = FO₂¹ [Mar05b]
Expressiveness

\[ A \rightarrow B \quad A \subseteq B \]

\[ A \rightarrow B \quad A \subseteq B \]

\[ A \rightarrow B \quad A \not\subseteq B \]

\[ \text{FO}_1 \]

\[ \text{FO}_1^2 = \text{CoreXPath} \]
CondXPath \[\text{Mar04}\]
Conditional XPath

\[
\text{CondXPath} = \text{CoreXPath} + \text{axis: } \text{ns, ns}^*, \text{ns}^{-1}, \text{ns}^{-1} + \text{until operator: } (\text{axis} :: \text{ntst}[fexpr])^+ \text{ with } \text{axis} \in \{\text{ch, ch}^{-1}, \text{ns}, \text{ns}^{-1}\}
\]

CondXPath has the same complexity as CoreXPath (for both query evaluation and satisfiability).
CondXPath \[\text{Mar04}\]
Conditional XPath

\[
\text{CondXPath} = \\
\text{CoreXPath} \\
+ \text{axis: } \text{ns, ns}_*, \text{ns}^{-1}, \text{ns}_*^{-1} \\
+ \text{until operator: } (\text{axis :: ntst[fexpr]})^+ \text{ with } \text{axis} \in \{\text{ch, ch}^{-1}, \text{ns, ns}^{-1}\}
\]

CondXPath has the same complexity as CoreXPath (for both query evaluation and satisiability).

\[
\text{CondXPath} \subseteq \text{FO}
\]

For instance \((\text{ch :: a}[\text{ns}_* :: \text{b}])^+\) translates to the FO formula:

\[
\phi(x, y) = \\
\exists z. \text{ns}_*(y, z) \land \text{label}_b(z) \land \\
\neg(\exists s. \text{ch}_*(x, s) \land \text{ch}_*(s, y) \land (\neg\text{label}_a(s) \lor \neg\exists s'. \text{ns}_*(s, s') \land \text{label}_b(s')))
\]
How to prove that $\text{FO} \subseteq \text{CondXPath}$?
How to prove that $FO \subseteq \text{CondXPath}$? Marx uses an intermediate logic: $X_{\text{until}}$. 
Syntax

\[ \varphi ::= a \mid \top \mid \neg \varphi \mid \varphi \land \varphi' \mid \theta(\varphi, \varphi') \quad (a \in \Sigma, \ \theta \in \{\downarrow, \leftarrow, \Rightarrow, \uparrow}\) \]

Arrows are interpreted as transitive closures of corresponding axis.

Semantics

\[
\begin{align*}
(t, \pi) \models a & \iff \text{label}^t_a(\pi) \\
(t, \pi) \models \neg \varphi & \iff (t, \pi) \not\models \varphi \\
(t, \pi) \models \varphi \land \varphi' & \iff (t, \pi) \models \varphi \text{ and } (t, \pi) \models \varphi' \\
(t, \pi) \models \theta(\varphi, \varphi') & \iff \text{there exists } \pi' \text{ s.t. } \theta^+(\pi, \pi') \text{ and } (t, \pi') \models \varphi \\
& \quad \text{and for all } \pi'' \text{ s.t. } \pi \theta^+ \pi'' \theta^+ \pi', (t, \pi'') \models \varphi'
\end{align*}
\]
1. From $X_{until}$ to CondXPath

<table>
<thead>
<tr>
<th>$X_{until}$</th>
<th>$\rightarrow$</th>
<th>CondXPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r(a)$</td>
<td>$\rightarrow$</td>
<td>$self :: a$</td>
</tr>
<tr>
<td>$r(\neg \phi)$</td>
<td>$\rightarrow$</td>
<td>$not r(\phi)$</td>
</tr>
<tr>
<td>$r(\phi \land \phi')$</td>
<td>$\rightarrow$</td>
<td>$r(\phi)$ and $r(\phi')$</td>
</tr>
<tr>
<td>$r(\theta(\phi, \phi'))$</td>
<td>$\rightarrow$</td>
<td>$\theta :: *[r(\phi)]$ or $(\theta :: *[r(\phi')])^+/\theta :: *[r(\phi)]$</td>
</tr>
</tbody>
</table>
2. From FO to $X_{\text{until}}$
Separation technic

**Theorem ([GHR94], adapted in [Mar04])**

*If every $X_{\text{until}}$ formula is separable over trees, then $X_{\text{until}}$ is FO-expressive.*

"$\varphi$ separable" means:
equivalent to a Boolean combination of pure past/present/future/left/right formula
2. From FO to $X_{\text{until}}$

Separation technic

**Theorem ([Mar04])**

*Each $X_{\text{until}}$ formula is separable.*

Query rewriting... with blowup.
Alternative proof

**Theorem ([Mar05b])**

- *any expansion of CoreXPath which is closed under complementation is FO-expressive*
- *CondXPath is closed under complementation*
Expressiveness

A \rightarrow B \quad A \varsubsetneq B

A \dashrightarrow B \quad A \subseteq B

A \rightarrow B \quad A \notin B

FO_1 = \text{CondXPath}

FO_1^2 = \text{CoreXPath}
RegularXPath ≈ [tC06]

RegularXPath =

CoreXPath
+ axis: \( ns, ns_*, ns^{-1}, ns_*^{-1} \)
+ transitive closure: \((\text{RegularXPath expression})^*\)

RegularXPath \(≈\) =

RegularXPath
+ loop predicate: \([\text{loop}(\varphi)]_t = \{ \pi \in D_t \mid (\pi, \pi) \in [\varphi]_t \}\)

Both have PTIME combined complexity for query evaluation.
**Theorem**

RegularXPath\(\approx\) and FO + TC\(^1\) have the same expressive power.

In a preceding section, we saw that FO + TC\(^1\) is strictly less expressive than MSO [tCS08].

**Corollary**

The class of binary relations definable in RegularXPath\(\approx\) is closed under intersection and complementation.

It is only conjectured that adding loop increases expressivity, i.e., that RegularXPath \(\subseteq\) RegularXPath\(\approx\).
Expressiveness

\[ A \rightarrow B \quad A \subsetneq B \]
\[ A \not\rightarrow B \quad A \subseteq B \]
\[ A \rightarrow B \quad A \not\in B \]

\[ \text{MSO} \]

\[ \text{FO} + TC^1 = \text{RegularXPath} \approx \]

\[ [tCS08] \]

\[ \text{FO} = \text{CondXPath} \]

\[ [BC05, BSSS06] \]

\[ \text{FO}^2 = \text{CoreXPath} \]
RegularXPath variants

- $\mu$RegularXPath adds a fixed-point operator [tC06] $\rightarrow$ MSO
- RegularXPath$(W)$ adds a “subtree relativisation operator” [tCS08]

Beware: RegularXPath$(W) = \text{FO} + TC^1_p$, whereas
RegularXPath$\sim = \text{FO} + TC^1$. Remind that it is not known whether the
inclusion $\text{FO} + TC^1 \subseteq \text{FO} + TC^1_p$ is strict.
XPath 2.0 adds the following features to XPath:

- **for loops**: `for $i in R return S`
- Boolean intersection (`intersect`) and complementation (`except`) on path expressions
- variables: *n*-ary queries
- node comparison tests (`is`)
CoreXPath 2 [tCM07]

CoreXPath 2

- *for* loops are interpreted as sets of nodes, not sequences
- no positional/aggregate: `position()`, `last()`, `count()`
- no value comparison operators
CoreXPath 2 [tCM07]

CoreXPath 2

- for loops are interpreted as sets of nodes, not sequences
- no positional/aggregate: position(), last(), count()
- no value comparison operators

- adding the last 2 features leads to undecidability.
- equivalence of CoreXPath 2 queries is decidable.
- of course, CoreXPath 2 is FO-expressive (adding except to CoreXPath is already sufficient).
- CoreXPath 2 $\leftrightarrow$ FO translations in linear time
Outline

5 $\mu$-calculus

6 XPath

7 Temporal Logics
Preliminaries: Linear Temporal Logic (LTL)

Syntax

\[ \varphi ::= a \mid \neg \varphi \mid \varphi \lor \psi \mid X \varphi \mid X^- \varphi \mid \varphi U \psi \mid \varphi S \psi \]

Semantics

Structure: \( s = s_0 s_1 \cdots s_n \) a string over \( \Sigma \)

Interpretation: \( (s, i) \models \varphi \) (\( \varphi \) is satisfied in \( s \) at position \( i \))

label \( (s, i) \models a \) iff \( s_i = a \) (i.e. \( label_s(i) = a \))

next \( (s, i) \models X \varphi \) iff \( (s, i + 1) \models \varphi \)

prev \( (s, i) \models X^- \varphi \) iff \( (s, i - 1) \models \varphi \)

until \( (s, i) \models \varphi U \psi \) iff \( \exists j \geq i. (s, j) \models \psi \land \forall k \in \{i, \ldots, j - 1\}. (s, k) \models \varphi \)

since \( (s, i) \models \varphi S \psi \) iff \( \exists j \leq i. (s, j) \models \psi \land \forall k \in \{j + 1, \ldots, i\}. (s, k) \models \varphi \)
Querying and expressivity

**LTL Boolean queries**

\[ QA(\varphi, s) = \text{true} \iff (s, 0) \models \varphi \]

**LTL unary queries**

\[ QA(\varphi, s) = \{ i \in \{0, \ldots, |s|\} : (s, i) \models \varphi \} \]

**Kamp’s Theorem.**

Over strings, \( LTL = FO \)
Tree Temporal Logic $\text{TL}^{\text{tree}}$

### Syntax

\[
\varphi ::= a \mid \neg \varphi \mid \varphi \lor \psi \mid \text{X}_\theta \varphi \mid \text{X}^\neg \varphi \mid \varphi \cup \psi \mid \varphi \cup_{\theta} \psi \mid \varphi \cup_{\theta} \psi \quad (\theta \in \{\downarrow, \leftarrow\})
\]

### Semantics

\[(t, \pi) \models \varphi \text{ reads “} \varphi \text{ is satisfied in } t \text{ at node } \pi \text{”}\]

\[(t, \pi) \models a \iff \text{label}_t(\pi) = a\]

\[(t, \pi) \models \text{X}^\downarrow \varphi \iff \exists \pi' \text{ such that } \pi \downarrow \pi' \text{ and } (t, \pi') \models \varphi.\]

etc.

### Theorem [Mar05a]

Over unranked ordered trees, $\text{TL}^{\text{tree}} = \text{FO}$ (Boolean and unary queries)
Computational tree logic $\text{CTL}^*_{\text{past}}$

**Syntax**

Node formulas: $\Phi ::= a \mid \neg \Phi \mid \Phi \lor \Psi \mid E \downarrow \varphi \mid E \rightarrow \varphi$

Path formulas: $\varphi ::= \Phi \mid \neg \varphi \mid \varphi \lor \psi \mid X \varphi \mid X^- \varphi \mid \varphi U \psi \mid \varphi S \psi$

**Semantics**

$(t, \pi) \models \Phi$ reads “$\Phi$ is satisfied in $t$ at node $\pi$”

$(t, \pi) \models E \downarrow \varphi$ iff $\exists \pi_1 \downarrow \cdots \downarrow \pi_{i-1} \downarrow \pi \downarrow \pi_{i+1} \downarrow \cdots \downarrow \pi_k$ such that $(\pi_1 \cdots \pi_k, i) \models p \varphi$ where:

$(\pi_1 \cdots \pi_k, i) \models p \Phi$ iff $(t, \pi_i) \models \Phi$ etc.

**Theorem [BL05b]**

Over unranked ordered trees, $\text{CTL}^*_{\text{past}} = \text{FO}$ (Boolean and unary queries)
Propositional Dynamic Logic for trees PDL\textsubscript{tree} [ABD\textsuperscript{+}05]

Syntax

Path formulas:

\[ \sigma ::= \equiv \mid \rightarrow \mid \downarrow \mid \uparrow \mid \sigma/\sigma' \mid \sigma \cup \sigma' \mid \sigma^* \mid \varphi? \]

Propositions:

\[ \varphi ::= a \mid \neg \varphi \mid \varphi \lor \psi \mid X_\sigma \varphi \]

Semantics

\( \sigma \) defines a binary relation \([\sigma]_t\) on nodes of \( t \)

\( (t, \pi) \models X_\sigma \varphi \) iff \( \exists \pi' \) such that \( \pi \models [\sigma]_t \pi' \) and \( (t, \pi') \models \varphi \)
Theorem

\( \text{PDL}_{\text{tree}} \) is equivalent to Regular XPath.

Theorem

\( \text{PDL}_{\text{tree}} \) restricted to

\[ \sigma ::= \equiv \leftarrow \rightarrow \downarrow \uparrow \sigma^* \mid \sigma/\varphi? \]

is equivalent to Conditional XPath which is equivalent to FO.

Theorem

\( \text{PDL}_{\text{tree}} \) restricted to

\[ \sigma ::= \equiv \leftarrow \rightarrow \downarrow \uparrow \sigma^* \]

is equivalent to Core XPath which is equivalent to FO\(^2\).
References
[ABD⁺05] Loredana Afanasiev, Patrick Blackburn, Ioanna Dimitriou, Bertrand Gaiffe, Evan Goris, Maarten Marx, and Maarten de Rijke.
PDL for ordered trees.

Combining temporal logics for querying XML documents.

Foundations of Databases.
1995.

Translations on a context-free grammar.

Tree-walking automata cannot be determinized.

Tree-walking automata do not recognize all regular languages.

Two-variable logic on data trees and XML reasoning.

[BKS02] Nicolas Bruno, Nick Koudas, and Divesh Srivastava.
Holistic twig joins: optimal xml pattern matching.

[BKW98] Anne Brügge mann-Klein and Derick Wood.
One-unambiguous regular languages. 

[BKW00] Anne Brüggemann-Klein and Derick Wood. 
Caterpillars: A context specification technique. 

Regular tree and regular hedge languages over unranked alphabets: Version 1, April 07 2001.

[BL05a] Pablo Barceló and Leonid Libkin. 
Temporal logics over unranked trees. 

[BL05b] Pablo Barcelo and Leonid Libkin. 
Temporal logics over unranked trees. 
Two-variable logic on words with data. 

*Decidable Properties of Tree Languages.* 

[Boj08a] Mikołaj Bojańczyk. 
Effective characterizations of tree logics, 2008. 
PODS’08 Keynote.

[Boj08b] Mikołaj Bojańczyk. 
Tree-walking automata. 
Tutorial at LATA’08, 2008.

[BS05] Michael Benedikt and Luc Segoufin. 
Regular tree languages definable in FO and FOmod.


[CNT04] Julien Carme, Joachim Niehren, and Marc Tommasi.
Querying unranked trees with stepwise tree automata.

*Finite Model Theory.*

Tree-walking pebble automata.

[EH06] Joost Engelfriet and Hendrik Jan Hoogeboom.
Nested pebbles and transitive closure.


[FG02] Markus Frick and Martin Grohe. 
The complexity of first-order and monadic second-order logic revisited. 

[FGK03] Markus Frick, Martin Grohe, and Christoph Koch. 
Query evaluation on compressed trees. 

*Temporal Logic (Volume 1: Mathematical Foundations and Computational Aspects)*. 

[GK04] Georg Gottlob and Christoph Koch. 
Monadic datalog and the expressive power of languages for web information extraction.


Maarten Marx. First order paths in ordered trees.

M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory.

Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for simple regular expressions.

[Mor94] Etsuro Moriya.  
On two-way tree automata.  

[MS02] Gerome Miklau and Dan Suciu.  
Containment and equivalence for an xpath fragment.  

Complementing deterministic tree-walking automata.  

[Nev02a] Frank Neven.  
Automata, logic, and XML.  

[Nev02b] Frank Neven.  
Automata theory for XML researchers.
N-ary queries by tree automata.

[NS99] Frank Neven and Thomas Schwentick.
Query automata.

[NS02] Frank Neven and Thomas Schwentick.
Query automata over finite trees.

[Sch07] Thomas Schwentick.
Automata for XML—a survey.
John W. Shipman.  
*XSLT Reference*.  
2008.

L. J. Stockmeyer.  
*The Complexity of Decision Problems in Automata Theory*.  

Balder ten Cate.  
The expressiveness of XPath with transitive closure.  

Balder ten Cate and Maarten Marx.  
Axiomatizing the logical core of XPath 2.0.  

Balder ten Cate and Luc Segoufin.
XPath, transitive closure logic, and nested tree walking automata.


**[TK06]** Hans-Jörg Tiede and Stephan Kepser.
Monadic second-order logic and transitive closure logics over trees.

**[TW68]** J. W. Thatcher and J. B. Wright.
Generalized finite automata with an application to a decision problem of second-order logic.

**[Var82]** Moshe Y. Vardi.
The complexity of relational query languages.
[Var95] Moshe Y. Vardi.
On the complexity of bounded-variable queries.