Logics for $n$-ary Queries in Trees

Emmanuel Filiot

INRIA Lille Nord-Europe, Mostrare Project
University of Lille 1, LIFL

Ph.D. Defense, 2008, October

supervisors: Sophie Tison and Jean-Marc Talbot
eXtensible Markup Language

XML

- markup language to represent tree-shaped data
- XML data big bang!
- standard for data exchange and data storage
eXtensible Markup Language

XML
- markup language to represent tree-shaped data
- XML data big bang!
- standard for data exchange and data storage

Tree representation
eXtensible Markup Language

XML
- markup language to represent tree-shaped data
- XML data big bang!
- standard for data exchange and data storage

Tree representation
eXtensible Markup Language

XML
- markup language to represent tree-shaped data
- XML data big bang!
- standard for data exchange and data storage

Tree representation

Trees are ordered and unranked.
XML Queries

Queries

- access XML data, transform XML documents
- node selection in XML trees
- $n$-ary queries select set of $n$-tuples of nodes
  - $n = 1$: unary queries
  - $n = 2$: binary queries
**XML Queries**

**Queries**

- access XML data, transform XML documents
- node selection in XML trees
- $n$-ary queries select set of $n$-tuples of nodes
  - $n = 1$: unary queries
  - $n = 2$: binary queries

**Example (Select all directors)**
XML Queries

Queries

- access XML data, transform XML documents
- node selection in XML trees
- $n$-ary queries select set of $n$-tuples of nodes
  - $n = 1$: unary queries
  - $n = 2$: binary queries

Example (Select all triples $(\text{title}, \text{year}, \text{director name})$)
Logics and Automata to Query XML Trees

- FO, MSO (yardstick logics but high query evaluation complexity)
- FO-relatives
  - temporal logics (LibkinN03, BarceloL05, ABDGGMR05)
  - navigational language XPath (W3C, GottlobKP02, Marx04, tenCate06, ...)
- MSO-relatives
  - $\mu$-calculus (BarceloL05)
  - Monadic Datalog (GottlobK04)
  - query automata (NevenS99)
  - node-selecting automata (Neven00, FrickGK03, NiehrenPTT06)
- Combination Logics (Schwentick00, ArenasBL07)
- pattern-matching approach: XDuce/CDuce (HosoyaP03, BenzakenCF03), Spatial Logic TQL (CardelliG02, BonevaTT05)
Logics and Automata to Query XML Trees

- **FO, MSO** (yardstick logics but high query evaluation complexity)
- **FO-relatives**
  - temporal logics (LibkinN03, BarceloL05, ABDGGMR05)
  - navigational language XPath (W3C, GottlobKP02, Marx04, tenCate06, ...)
- **MSO-relatives**
  - \( \mu \)-calculus (BarceloL05)
  - Monadic Datalog (GottlobK04)
  - query automata (NevenS99)
  - node-selecting automata (Neven00, FrickGK03, NiehrenPTT06)
- **Combination Logics** (Schwentick00, ArenasBL07)
- **pattern-matching approach**: XDuce/CDuce (HosoyaP03, BenzakenCF03), Spatial Logic TQL (CardelliG02, BonevaTT05)

Only a few logics are well-suited to express \( n \)-ary queries
Objectives

Two popular approaches:

Navigational Approach

Pattern-matching approach
Objectives

Two popular approaches:

Navigational Approach

How to define a navigation-based \( n \)-ary query language?

Pattern-matching approach
Objectives

Two popular approaches:

Navigational Approach

How to define a navigation-based \( n \)-ary query language?

- expressiveness vs query evaluation complexity
- composition language: from binary to \( n \)-ary queries
- application to XPath-based \( n \)-ary query languages

Pattern-matching approach
Objectives

Two popular approaches:

**Navigational Approach**

How to define a navigation-based $n$-ary query language?

- expressiveness vs query evaluation complexity
- composition language: from binary to $n$-ary queries
- application to XPath-based $n$-ary query languages

**Pattern-matching approach**

- satisfiability problem
- is there an expressive **decidable** TQL fragment that can define $n$-ary queries?
Objectives

Two popular approaches:

**Navigational Approach**

How to define a navigation-based \( n \)-ary query language?

- expressiveness vs query evaluation complexity
- composition language: from binary to \( n \)-ary queries
- application to XPath-based \( n \)-ary query languages

**Pattern-matching approach**

- satisfiability problem
- is there an expressive **decidable** TQL fragment that can define \( n \)-ary queries?

- adaptation to ordered trees
- automata-based satisfiability algorithm
Outline

1. Composing Binary Queries
   - definitions
   - expressiveness, query evaluation
   - application to $n$-ary XPath logics

2. The Spatial Logic TQL
   - Examples, Definition
   - Expressiveness, Satisfiability
   - Tree Automata with Global Constraints

3. Summary and Perspectives
PART I: Composing Binary Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet $\Sigma = f, g, a, b \ldots$.

Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet \( \Sigma = f, g, a, b, \ldots \).

Unary Relations: nodes\((t)\)

Queries

\[ E.Filiot \] Logics for \( n \)-ary Queries in Trees

2008, October
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet \( \Sigma = \{f, g, a, b, \ldots\} \).

Unary Relations: \( \text{nodes}(t) \) , \( \text{root}(t) \)

Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet $\Sigma = f, g, a, b \ldots$.

Unary Relations: $\text{nodes}(t)$, $\text{root}(t)$, $(\text{lab}_a(t))_{a \in \Sigma}$

Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet
\[ \Sigma = f, g, a, b, \ldots \]

Unary Relations: \( \text{nodes}(t), \text{root}(t), (\text{lab}_a(t))_{a \in \Sigma} \)

Binary Relations: \( \text{ns} \)

Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet \( \Sigma = f, g, a, b \ldots \).

Unary Relations: \( \text{nodes}(t) \), \( \text{root}(t) \), \( (\text{lab}_a(t))_{a \in \Sigma} \)

Binary Relations: \( \text{ns} \), \( \text{ns}^* \)

Queries

E.Filiot

Logics for \( n \)-ary Queries in Trees

2008, October
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet \( \Sigma = f, g, a, b \ldots \)

Unary Relations: \( \text{nodes}(t), \text{root}(t), (\text{lab}_a(t))_{a \in \Sigma} \)

Binary Relations: \( \text{ns}, \text{ns}^*, \text{ch} \)

Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet $\Sigma = f, g, a, b, \ldots$.

Unary Relations: $\text{nodes}(t), \text{root}(t), (\text{lab}_a(t))_{a \in \Sigma}$

Binary Relations: $\text{ns}, \text{ns}^*, \text{ch}, \text{ch}^*$

Queries
Trees and Queries

Trees

Trees are finite, unranked and ordered over a finite alphabet $\Sigma = f, g, a, b, \ldots$.

Unary Relations: $\text{nodes}(t)$, $\text{root}(t)$, $(\text{lab}_a(t))_{a \in \Sigma}$

Binary Relations: $\text{ns}$, $\text{ns}^*$, $\text{ch}$, $\text{ch}^*$

Queries

Let $n \in \mathbb{N}$. An $n$-ary query $q$ maps trees $t$ to $n$-tuples of nodes

$$q(t) \subseteq \text{nodes}(t)^n$$
The navigational language XPath

- to navigate and select sets of nodes in XML trees
- by defining path expressions
- complex counting conditions
- \textit{CoreXPath}: navigational core (GottlobKP02)
The navigational language XPath

Example: select all director names

\texttt{ch :: DVD/ch :: director/ch :: name}
The navigational language XPath

Example: select all director names

\( ch :: DVD/ch :: director/ch :: name \)
The navigational language XPath

Example: select all director names

\texttt{ch :: DVD/ch :: director/ch :: name}
The navigational language XPath

Example: select all director names

ch :: DVD/ch :: director/ch :: name
The navigational language XPath

Example: select all awarded director names

\[
\text{ch :: DVD[ch :: awards]/ch :: director/ch :: name}
\]
Expressions of CoreXPath and their semantics

Axis
- self, ch, ch^+, ns, ns^+
- ch^{-1}, (ch^{-1})^+, ns^{-1}, (ns^{-1})^+

Steps
- Axis::a
- Axis::*

Composition
- $P_1/P_2$

Union
- $P_1 \cup P_2$

Tests
- $P[T]

Path existence
- $P$

Negation
- not $T$

Conjunction
- $T_1$ and $T_2$
Expressions of CoreXPath and their semantics

Axis
self, ch, ch+, ns, ns+
ch⁻¹, (ch⁻¹)+, ns⁻¹, (ns⁻¹)+

\[[.]\]^t \subseteq \text{nodes}(t) \times \text{nodes}(t)

Steps
\[[\text{Axis::a}]\]^t = \{(v_1, v_2) \mid v_1 \text{ Axis } v_2 \text{ and } v_2 \in \text{lab}_a(t)\}
\[[\text{Axis::*}]\]^t = \{(v_1, v_2) \mid v_1 \text{ Axis } v_2\}

Composition
\[[P_1/P_2]\]^t = \[[P_1]\]^t \circ \[[P_2]\]^t

Union
\[[P_1 \cup P_2]\]^t = \[[P_1]\]^t \cup \[[P_2]\]^t

Tests
\[[P[\mathcal{T}]]\]^t = \{(v_1, v_2) \in \[[P]\]^t \mid v_2 \in \[[\mathcal{T}]\]_{\text{test}}\}

\[[.]\]_{\text{test}}^t \subseteq \text{nodes}(t)

Path existence
\[[P]\]_{\text{test}}^t = \{v \mid (v, v') \in \[[P]\]^t\}

Negation
\[[\text{not } \mathcal{T}]\]_{\text{test}}^t = \text{nodes}(t) - \[[\mathcal{T}]\]_{\text{test}}^t

Conjunction
\[[\mathcal{T}_1 \text{ and } \mathcal{T}_2]\]_{\text{test}}^t = \[[\mathcal{T}_1]\]_{\text{test}}^t \cap \[[\mathcal{T}_2]\]_{\text{test}}^t
How to turn XPath into an $n$-ary query language?
How to turn XPath into an $n$-ary query language?

- use path expressions $p$ to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples
How to turn XPath into an $n$-ary query language?

- use path expressions $p$ to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples

Example (All triples (title, year, director name))

$\phi(x, y, z) =$

![Diagram showing the structure of DVDs and DVDs with nodes for title, year, director, writer, awards, and name, birthday, award, birthday, name, birthday]
How to turn XPath into an $n$-ary query language?

- use path expressions $p$ to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples

Example (All triples (title, year, director name))

$\phi(x, y, z) = ch^* :: title$
How to turn XPath into an \( n \)-ary query language?

- use path expressions \( p \) to navigate
- use node variables \( x_1, x_2, \ldots, x_n \) to select \( n \)-tuples

Example (All triples \((\text{title}, \text{year}, \text{director name})\))

\[
\phi(x, y, z) = \text{ch}^* :: \text{title}/x
\]
How to turn XPath into an \( n \)-ary query language?

- use path expressions \( p \) to navigate
- use node variables \( x_1, x_2, \ldots, x_n \) to select \( n \)-tuples

Example (All triples (title, year, director name))

\[ \phi(x, y, z) = \text{ch}^* :: \text{title}/x/\text{ns} :: \text{year} \]
How to turn XPath into an \( n \)-ary query language?

- use path expressions \( p \) to navigate
- use node variables \( x_1, x_2, \ldots, x_n \) to select \( n \)-tuples

Example (All triples \((\text{title}, \text{year}, \text{director name})\))

\[
\phi(x, y, z) = \text{ch}^* :: \text{title}/x/\text{ns} :: \text{year}/y
\]
How to turn XPath into an \( n \)-ary query language?

- use path expressions \( p \) to navigate
- use node variables \( x_1, x_2, \ldots, x_n \) to select \( n \)-tuples

Example (All triples (\texttt{title},\texttt{year},\texttt{director name}))

\[
\phi(x, y, z) = \text{ch}^* :: \text{title}/x/\text{ns} :: \text{year}/\text{ns} :: \text{director}/\text{ch} :: \text{name}
\]
How to turn XPath into an $n$-ary query language?

- use path expressions $p$ to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples

Example (All triples (title, year, director name))

$$\phi(x, y, z) = ch^* :: title/x/\text{ns} :: year/\text{ns} :: director/ch :: name/z$$
How to turn XPath into an $n$-ary query language?

- use path expressions to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples
How to turn XPath into an \( n \)-ary query language?

- use path expressions to navigate
- use node variables \( x_1, x_2, \ldots, x_n \) to select \( n \)-tuples

Example (All triples (title, year, director name))

\[
\phi(x, y, z) =
\]
How to turn XPath into an $n$-ary query language?

- use path expressions to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples

Example (All triples $\langle \text{title}, \text{year}, \text{director name} \rangle$)

$$ \phi(x, y, z) = \text{ch} :: \text{DVD} $$
How to turn XPath into an $n$-ary query language?

- use path expressions to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples

Example (All triples $(title, year, director, name)$)

$$\phi(x, y, z) = ch :: DVD[ch :: title/x]$$
How to turn XPath into an $n$-ary query language?

- use path expressions to navigate
- use node variables $x_1, x_2, \ldots, x_n$ to select $n$-tuples

Example (All triples (title, year, director name))

$$\phi(x, y, z) = ch :: DVD[ch :: title/x][ch :: year/y]$$
How to turn XPath into an \(n\)-ary query language?

- use path expressions to navigate
- use node variables \(x_1, x_2, \ldots, x_n\) to select \(n\)-tuples

Example (All triples \((\text{title}, \text{year}, \text{director name})\))

\[
\phi(x, y, z) = \text{ch} :: \text{DVD}[\text{ch} :: \text{title}/x][\text{ch} :: \text{year}/y][\text{ch} :: \text{director}/\text{ch} :: \text{name}/z]
\]
Idea of the composition language

- use path expressions to navigate
- use variables $x_1, x_2, \ldots, x_n$ to select output $n$-tuples
- composition operator $\circ$ to compose queries
Idea of the composition language

- use binary queries from some binary query language \( L \) to navigate
- use variables \( x_1, x_2, \ldots, x_n \) to select output \( n \)-tuples
- composition operator \( \circ \) to compose queries
Idea of the composition language

- use binary queries from some binary query language $L$ to navigate
- use variables $x_1, x_2, \ldots, x_n$ to select output $n$-tuples
- composition operator $\circ$ to compose queries

Example (Composition of CoreXPath expressions)

\[
\begin{align*}
\text{ch* :: title} & / x / \text{ns :: year} / y / \text{ns :: director/ch :: name} / z \\
q_1 & \circ x \circ q_2 & \circ y \circ q_3 & \circ z
\end{align*}
\]

where $q_1, q_2, q_3 \in \text{CoreXPath}$.
The composition language \textbf{Comp}(L)

**Syntax of composition formulas \textbf{Comp}(L)**

We start from \( L \) a binary query language, and \( \text{Var} \) a set of variables.

\[
\phi \;:=\; q \quad q \in L \\
\quad | \quad x \quad \text{variable} \\
\quad | \quad \phi \circ \phi \quad \text{composition} \\
\quad | \quad [\phi] \quad \text{test} \\
\quad | \quad \phi \lor \phi \quad \text{disjunction}
\]

- thanks to variables, you can define \( n \)-ary queries
- \( \text{Ans}(\phi, t) \): set of answers.
Query Evaluation

Query evaluation problem

- **Input**: a tree $t$, a formula $\phi(x_1, \ldots, x_n) \in \text{Comp}(L)$
- **Output**: $\text{Ans}(\phi, t)$
Query Evaluation

Query evaluation problem

- **Input**: a tree $t$, a formula $\phi(x_1, \ldots, x_n) \in \text{Comp}(L)$
- **Output**: $\text{Ans}(\phi, t)$

Polynomial-time query evaluation

- The number of $n$-tuples of nodes is exponential in $|t|$ and:
  $$|\text{Ans}(\phi, t)| << |t|^n$$
- one needs **polynomial-time** query evaluation:
  $$\text{poly}(|t|, |\phi|, |\text{Ans}(\phi, t)|)$$
Query Evaluation Algorithm for $\text{Comp}^{\text{nvs}}(L)$

Non-variable sharing fragment

- variable sharing: $q \circ x \circ q' \circ y \circ q'' \circ x$
- disallow variable sharing: $\phi_1 \circ \phi_2 \rightarrow \text{Var}(\phi_1) \cap \text{Var}(\phi_2) = \emptyset$
- $\text{Comp}^{\text{nvs}}(L) = \text{Comp}(L) + \text{non-variable sharing}$
- related to acyclicity of conjunctive queries (Yannakakis81)
Query Evaluation Algorithm for $\text{Comp}^{\text{nvs}}(L)$

Non-variable sharing fragment
- variable sharing: $q \circ x \circ q' \circ y \circ q'' \circ x$
- disallow variable sharing: $\phi_1 \circ \phi_2 \rightarrow \text{Var}(\phi_1) \cap \text{Var}(\phi_2) = \emptyset$
- $\text{Comp}^{\text{nvs}}(L) = \text{Comp}(L) + \text{non-variable sharing}$
- related to acyclicity of conjunctive queries (Yannakakis81)

Theorem

Query evaluation for $\text{Comp}^{\text{nvs}}(L)$ is in PTIME if query evaluation for $L$ is in PTIME.
Query Evaluation Algorithm for $\text{Comp}^{\text{nvs}}(L)$

Non-variable sharing fragment

- variable sharing: $q \circ x \circ q' \circ y \circ q'' \circ x$
- disallow variable sharing: $\phi_1 \circ \phi_2 \rightarrow \text{Var}(\phi_1) \cap \text{Var}(\phi_2) = \emptyset$
- $\text{Comp}^{\text{nvs}}(L) = \text{Comp}(L) + \text{non-variable sharing}$
- related to acyclicity of conjunctive queries (Yannakakis81)

Theorem

Query evaluation for $\text{Comp}^{\text{nvs}}(L)$ is in PTIME if query evaluation for $L$ is in PTIME.

Idea (Yannakakis81): process the formula recursively:

1. at each step, check if there is a solution $\rightarrow$ remain linear in $|\text{Ans}(\phi, t)|$
2. use memoization to avoid redundant calculus
Expressiveness

Two yardstick logics, FO and MSO

- MSO = FO + set quantification
- formulas $\psi(x_1, \ldots, x_n) \in FO (MSO)$ define $n$-ary queries
- $FO_n = n$-ary FO queries
- $MSO_n = n$-ary MSO queries
Exressiveness

Two yardstick logics, FO and MSO

- **MSO** = **FO** + set quantification
- formulas \( \psi(x_1, \ldots, x_n) \in \text{FO (MSO)} \) define \( n \)-ary queries
- **FO**\(_n\) = \( n \)-ary FO queries
- **MSO**\(_n\) = \( n \)-ary MSO queries

Theorem

\[
\begin{align*}
\text{FO}_n & = \text{Comp}^{\text{nvs}}(\text{FO}_2) \\
\text{MSO}_n & = \text{Comp}^{\text{nvs}}(\text{MSO}_2)
\end{align*}
\]

Remark

It uses folklore result from finite model theory based on the Shelah’s decomposition method. (Schwentick’00 or Marx’05 for instance).
Conditional XPath (Marx’04)

- extends CoreXPath with a “while” operator \((\text{axis} :: l[\text{test}])^+\)
- \(\text{CXPath} = FO_2\)
- query evaluation of a path expression \(p\) is in \(O(|p|.|t|)\)
$n$-ary XPath Extensions (I)

**Conditional XPath (Marx’04)**

- extends CoreXPath with a “while” operator \((\text{axis} :: l[\text{test}])^+)\)
- \(\text{CXPath} = FO_2\)
- query evaluation of a path expression \(p\) is in \(O(|p|.|t|)\)

**$n$-ary Conditional XPath**

- path expressions \(p\) + non-variable sharing
  \[
  p ::= \text{axis} :: l \mid p/p \mid p[\text{test}] \mid p \cup p \mid (\text{axis} :: l[\text{test}])^+
  \]
**n-ary XPath Extensions (I)**

**Conditional XPath (Marx’04)**

- extends CoreXPath with a “while” operator \((axis :: l[test])^+\)
- \(\text{CXPath} = FO_2\)
- query evaluation of a path expression \(p\) is in \(O(|p|.|t|)\)

**n-ary Conditional XPath**

- path expressions \(p\) + non-variable sharing
  \[ p ::= \text{axis} :: l | p/p | p[test] | p \cup p | (axis :: l[test])^+ | x \in \text{Var} \]
**n-ary XPath Extensions (I)**

**Conditional XPath (Marx’04)**

- extends CoreXPath with a “while” operator \((axis :: l[test])^+\)
- \(CXPath = FO_2\)
- query evaluation of a path expression \(p\) is in \(O(|p|.|t|)\)

**n-ary Conditional XPath**

- path expressions \(p +\) non-variable sharing
  \[p ::= axis :: l | p/p | p[test] | p \cup p | (axis :: l[test])^+ | x \in \text{Var}\]
- linear-time back and forth translations into \(\text{Comp}^{\text{nvs}}(CXPath)\)
- \(\rightarrow\) captures \(FO_n\)
- \(\rightarrow\) query evaluation in time \(O(|p|.|t|^2(1 + |\text{Ans}(p, t)|))\)

**Remark:** query evaluation of FO 0-ary queries is PSPACE-complete
$n$-ary XPath Extensions (II)

XPath 2.0

- extends XPath (1.0) with:
  - path intersection $p_1 \cap p_2$
  - path complement $\text{compl}(p)$
  - variables $x$
  - quantification $\text{for}\ x\ \text{in}\ p_1\ \text{return}\ p_2$

- captures $\text{FO}_n$ modulo linear-time

- $\text{CoreXPath2.0}$ formalized by ten Cate and Marx (07)
**n-ary XPath Extensions (II)**

**XPath 2.0**
- extends XPath (1.0) with:
  - path intersection \( p_1 \cap p_2 \)
  - path complement \( \text{compl}(p) \)
  - variables \( x \)
  - quantification \( \text{for } x \text{ in } p_1 \text{ return } p_2 \)
- captures \( \text{FO}_n \) modulo linear-time
- \( \text{CoreXPath2.0} \) formalized by ten Cate and Marx (07)

**Application of the composition language**
- to define a syntactic fragment of \( \text{CoreXPath2.0} \)
- \( \text{FO}_n \)-expressive
- with query evaluation problem in \( O(|p| \cdot |t|^3 + |p| \cdot |t|^2 \cdot |\text{Ans}(p, t)|) \)
Outline

PART II: The Spatial Logic TQL
TQL Examples

Example (Check if there is an awarded movie)

\[ DVDs[- \ | DVD[- | awards[-]] \ | -] \]
TQL Examples

Example (Check if there is an awarded movie)

\[ DVDs[ \_ | DVD[ \_ | awards[ \_ ] ] | \_ ] \]
TQL Examples

Example (Select all awarded movies)

\[ \phi(X) = DVDs[- | X] \wedge DVD[- | awards[-]] | -] \]

↓

tree variable
Example (Select all pairs of (director, writer))

\[
\phi(X, Y) = DVDs[- | DVD[-|year[-|X|Y|-]]]
\]
Example (Select all names of persons who are both director and writer)

\[ \phi(X) = DVDs[-|DVD[-|director[name[X]|-]|writer[name[X]|-]|-|]-] \]
Example (Select all director names who is not a writer)

\[ \phi(X) = DVDs[- | DVD[-|director[name[X]]-]|writer[name[\neg X]]-]|-]|-] \]
TQL Examples

Tree (dis)equality tests

- main difficulty of TQL satisfiability problem
- incomparable to FO fragments with data-value comparison (BojanczykDMSS06)
Hedge Algebra $H_\Lambda$

- $\Lambda$: countable set of labels
- hedge = ordered sequence of unranked trees
Hedge Algebra $H_{\Lambda}$

- $\Lambda$: countable set of labels
- hedge = ordered sequence of unranked trees
- constant 0: empty hedge
Hedge Algebra $H_\Lambda$

- $\Lambda$: countable set of labels
- hedge = ordered sequence of unranked trees
- constant 0: empty hedge
- unary symbols $a \in \Lambda$:

$$a(\ldots) = \begin{array}{c} a \\ \ldots \end{array}$$
Hedge Algebra $H_\Lambda$

- $\Lambda$: countable set of labels
- hedge = ordered sequence of unranked trees
- constant $0$: empty hedge
- unary symbols $a \in \Lambda$:

\[ a(...) = \]

\[ a \]

\[ \ldots \]

binary symbol $|$
empty hedge  0
location  $\alpha[\phi]$  $\alpha \subseteq \Lambda$  (co)finite
concatenation  $\phi|\phi'$
TQL: Syntax and Semantics

- **empty hedge**: $0$
- **location**: $\alpha[\phi]$
- **concatenation**: $\phi|\phi'$
- **truth**
- **conjunction**: $\phi \land \phi'$
- **negation**: $\neg\phi$

$\alpha \subseteq \Lambda$ (co)finite
empty hedge
location
concatenation
truth
conjunction
negation
tree variable
recursion variable
least fixpoint

\[ \begin{align*}
0 & \quad \alpha[\phi] \\
\phi|\phi' & \\
\neg \phi & \\
\mu \xi. \phi & \\
\end{align*} \]

\( \alpha \subseteq \Lambda \) (co)finite
## TQL: Syntax and Semantics

- **semantics modulo** $\rho : \text{TreeVars} \rightarrow T_\Lambda$ and $\delta : \text{RecVars} \rightarrow 2^{H_\Lambda}$
- **set-based semantics**: $[[ \cdot ]]_{\rho,\delta} \subseteq H_\Lambda$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>empty hedge location</td>
</tr>
<tr>
<td>$\alpha[\phi]$</td>
<td>$\alpha \subseteq \Lambda$ (co)finite concatenation</td>
</tr>
<tr>
<td>$\phi</td>
<td>\phi'$</td>
</tr>
<tr>
<td>$\neg\phi$</td>
<td>conjunction</td>
</tr>
<tr>
<td>$\mu\xi.\phi$</td>
<td>negation</td>
</tr>
<tr>
<td>$X$</td>
<td>tree variable</td>
</tr>
<tr>
<td>$\xi$</td>
<td>recursion variable</td>
</tr>
<tr>
<td>$\mu\xi.\phi$</td>
<td>least fixpoint</td>
</tr>
</tbody>
</table>
TQL: Syntax and Semantics

- semantics modulo $\rho : \text{TreeVars} \rightarrow T_{\Lambda}$ and $\delta : \text{RecVars} \rightarrow 2^{H_{\Lambda}}$
- set-based semantics: $\llbracket \cdot \rrbracket_{\rho,\delta} \subseteq H_{\Lambda}$

- empty hedge $\llbracket 0 \rrbracket_{\rho,\delta} = \{ 0 \}$
- location $\llbracket \alpha[\phi] \rrbracket_{\rho,\delta} = \{ a(h) \mid h \in \llbracket \phi \rrbracket_{\rho,\delta}, a \in \alpha \}$
- concatenation $\llbracket \phi | \phi' \rrbracket_{\rho,\delta} = \{ h|h' \mid h \in \llbracket \phi \rrbracket_{\rho,\delta}, h' \in \llbracket \phi' \rrbracket_{\rho,\delta} \}$

- truth $\bot$
- conjunction $\phi \land \phi'$
- negation $\neg \phi$

- tree variable $X$
- recursion variable $\xi$
- least fixpoint $\mu \xi. \phi$
TQL: Syntax and Semantics

- semantics modulo $\rho : \text{TreeVars} \rightarrow T^\Lambda$ and $\delta : \text{RecVars} \rightarrow 2^{H^\Lambda}$
- set-based semantics: $\llbracket \cdot \rrbracket_{\rho,\delta} \subseteq H^\Lambda$

empty hedge $\llbracket [0] \rrbracket_{\rho,\delta} = \{0\}$

location $\llbracket \alpha[\phi] \rrbracket_{\rho,\delta} = \{a(h) \mid h \in \llbracket \phi \rrbracket_{\rho,\delta}, a \in \alpha\}$

concatenation $\llbracket \phi|\phi' \rrbracket_{\rho,\delta} = \{h|h' \mid h \in \llbracket \phi \rrbracket_{\rho,\delta}, h' \in \llbracket \phi' \rrbracket_{\rho,\delta}\}$

truth $\llbracket [\_] \rrbracket = H^\Lambda$

conjunction $\llbracket \phi \land \phi' \rrbracket = \llbracket \phi \rrbracket \cap \llbracket \phi' \rrbracket$

negation $\llbracket \neg \phi \rrbracket = H^\Lambda \setminus \llbracket \phi \rrbracket$

- tree variable $X$
- recursion variable $\xi$
- least fixpoint $\mu \xi. \phi$
TQL: Syntax and Semantics

- semantics modulo $\rho : \text{TreeVars} \rightarrow T_\Lambda$ and $\delta : \text{RecVars} \rightarrow 2^{H_\Lambda}$
- set-based semantics: $\llbracket . \rrbracket_{\rho,\delta} \subseteq H_\Lambda$

empty hedge $\llbracket 0 \rrbracket_{\rho,\delta} = \{0\}$
location $\llbracket \alpha[\phi] \rrbracket_{\rho,\delta} = \{a(h) \mid h \in \llbracket \phi \rrbracket_{\rho,\delta}, a \in \alpha\}$
concatenation $\llbracket \phi | \phi' \rrbracket_{\rho,\delta} = \{h|h'| \mid h \in \llbracket \phi \rrbracket_{\rho,\delta}, h' \in \llbracket \phi' \rrbracket_{\rho,\delta}\}$

truth $\llbracket \_ \rrbracket = H_\Lambda$
conjunction $\llbracket \phi \land \phi' \rrbracket = \llbracket \phi \rrbracket \cap \llbracket \phi' \rrbracket$

negation $\llbracket \neg \phi \rrbracket = H_\Lambda \setminus \llbracket \phi \rrbracket$

tree variable $\llbracket X \rrbracket_{\rho,\delta} = \{\rho(X)\}$
recursion variable $\llbracket \xi \rrbracket_{\rho,\delta} = \delta(\xi)$
least fixpoint $\llbracket \mu \xi . \phi \rrbracket_{\rho,\delta} = \cap\{S \subseteq H_\Lambda \mid \llbracket \phi \rrbracket_{\rho,\delta}[\xi \mapsto S] \subseteq S\}$
Examples with fixpoint

Example (Select all subtrees reachable from the root by following an 'a'-path)

\[ \phi(X) = \mu \xi. (a[-|\xi|-] \lor X) \]
Examples with fixpoint

Example (Select all subtrees reachable from the root by following an 'a'-path)

\[ \phi(X) = \mu \xi. (a \xi \lor X) \]

Example \((a^n b^n)\)

\[ \mu \xi. (a[0] \xi | b[0] \lor 0) \]
Examples with fixpoint

Example (Select all subtrees reachable from the root by following an 'a'-path)

$$\phi(X) = \mu \xi. (a[-|\xi|] \lor X)$$

Example ($a^n b^n$)

$$\mu \xi. (a[0]|\xi|b[0] \lor 0)$$

- vertical recursion $\rightarrow$ regular tree languages
- horizontal recursion $\rightarrow$ context-free word languages
A Decidable Fragment: Bounded TQL

Satisfiability problem

Input: TQL formula $\phi$  
Output: $\exists h \exists \rho \exists \delta, \ h \in [\phi]_{\rho, \delta}$?
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are guarded by some $\alpha[.]$
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$

\[
\mu \xi . (a[-|\xi|-] \lor X) \rightarrow \text{guarded}
\]
\[
\mu \xi . (a[0]|\xi|b[0] \lor 0) \rightarrow \text{not guarded}
\]
A Decidable Fragment: Bounded TQL

**Proposition**

*Satisfiability of TQL formulas is undecidable.*

**Bounded TQL**

- recursion variables are **guarded** by some $\alpha[.]$
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- add Kleene star $\phi^*$ for horizontal recursion
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- add Kleene star $\phi^*$ for horizontal recursion

\[
[\phi^*]_\rho = 0 \cup \bigcup_{i>0} [\phi]_\rho \cdots [\phi]_\rho
\]

i times
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- add Kleene star $\phi^*$ for horizontal recursion
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- add Kleene star $\phi^*$ for horizontal recursion
- restriction negative variables: only a **bounded number of disequality tests** along the paths
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- **add Kleene star** $\phi^*$ for horizontal recursion
- restriction negative variables: only a **bounded number of disequality tests** along the paths

$$b[X \mid \mu\xi. (\neg X \land a[\xi] \lor 0)] \rightarrow \text{not bounded}$$

$$(-X)^* \mid X \mid (-X)^* \rightarrow \text{bounded}$$
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- add Kleene star $\phi^*$ for horizontal recursion
- restriction negative variables: only a **bounded number of disequality tests** along the paths
A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

- recursion variables are **guarded** by some $\alpha[.]$
- add Kleene star $\phi^*$ for horizontal recursion
- restriction negative variables: only a **bounded number of disequality tests** along the paths
- no **negative** occurrences of Kleene star and |
Theorem

1. Bounded TQL sentences capture MSO.
2. Satisfiability of bounded TQL is decidable (in 3NEXPTIME).
**Theorem**

1. **Bounded TQL sentences capture MSO.**
2. **Satisfiability of bounded TQL is decidable (in 3NEXPTIME).**
   - 2EXPTIME / EXPTIME-hard when no negated variables occur
   - EXPTIME for sentences
Expressiveness and Satisfiability of Bounded TQL

Theorem

1. Bounded TQL sentences capture MSO.
2. Satisfiability of bounded TQL is decidable (in 3NEXPTIME).
   - 2EXPTIME / EXPTIME-hard when no negated variables occur
   - EXPTIME for sentences

The proof is by reduction to emptiness of bounded TAGEDs.
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run transitions

<table>
<thead>
<tr>
<th>$\lor$</th>
<th>$\land$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$0$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

Final states

$F = \{ q_1 \}$
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run transitions

$$
\begin{array}{c}
\land \\
\lor \\
0 \quad q_0 \\
\lor \\
1 \\
\land \\
\land \\
1 \\
\lor \\
0 \\
\lor \\
1
\end{array}
$$

\[
\begin{align*}
0 & \rightarrow q_0 & 1 & \rightarrow q_1 \\
\land(q_{b_1}, q_{b_2}) & \rightarrow q_{b_1} \land q_{b_2} \\
\lor(q_{b_1}, q_{b_2}) & \rightarrow q_{b_1} \lor q_{b_2} \\
F & = \{ q_1 \}
\end{align*}
\]
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run:

```
∧  ∧
/ \ / \ / \ /
∨  0  1  ∧ 1
   q0  q1  ∧ 1
   0   1
```

Transitions:

- $0 \rightarrow q_0$
- $1 \rightarrow q_1$
- $\land(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \land b_2$
- $\lor(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \lor b_2$

Final states:

$F = \{q_1\}$
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run

Transitions

Final states

$F = \{q_1\}$
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run:

```
\begin{align*}
\land \quad \land \\
\lor q_1 \\
0 & q_0 & 1 & q_1 \\
\lor & & \lor \\
1 & q_1 & 0 \\
\end{align*}
```

Transitions:

```
0 \rightarrow q_0 \quad 1 \rightarrow q_1
\land(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \land q_{b_2}
\lor(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \lor q_{b_2}
```

Final states:

$F = \{q_1\}$
Bottom-up Tree Automata for Binary Trees

- \( \Sigma \): finite alphabet
- \( Q \): set of states
- \( F \subseteq Q \): set of final states
- \( \Delta \): rules of the form \( f(q_1, q_2) \rightarrow q \) or \( a \rightarrow q \)

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run

\[
\begin{array}{c}
\land \\
\lor q_1 \\
0 q_0 & 1 q_1 \\
\lor q_1 \\
1 q_1 & 0 q_0 \\
\land \\
0 \rightarrow q_0 & 1 \rightarrow q_1 \\
\land(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \land q_{b_2} \\
\lor(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \lor q_{b_2} \\
F = \{ q_1 \}
\end{array}
\]
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run

Transitions

- $0 \rightarrow q_0$
- $1 \rightarrow q_1$
- $\land(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \land q_{b_2}$
- $\lor(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \lor q_{b_2}$

Final states

$F = \{ q_1 \}$
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

A tree and a successful run:

```
∧
∨ q₁
0 q₀ 1 q₁
∧
∨ q₁
1 q₁ 0 q₀
```

Transitions:

- $0 \rightarrow q_0$
- $1 \rightarrow q_1$
- $\land(q_{b₁}, q_{b₂}) \rightarrow q_{b₁} \land b₂$
- $\lor(q_{b₁}, q_{b₂}) \rightarrow q_{b₁} \lor b₂$

Final states:

$F = \{ q₁ \}$
Bottom-up Tree Automata for Binary Trees

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run

 transitions

\[
\begin{align*}
0 & \rightarrow q_0 \\
1 & \rightarrow q_1 \\
\land(q_{b_1}, q_{b_2}) & \rightarrow q_{b_1} \land q_{b_2} \\
\lor(q_{b_1}, q_{b_2}) & \rightarrow q_{b_1} \lor q_{b_2}
\end{align*}
\]

final states

$F = \{ q_1 \}$
**Bottom-up Tree Automata for Binary Trees**

- $\Sigma$: finite alphabet
- $Q$: set of states
- $F \subseteq Q$: set of final states
- $\Delta$: rules of the form $f(q_1, q_2) \rightarrow q$ or $a \rightarrow q$

**Example (variable-free satisfiable Boolean formulas)**

A tree and a successful run:

```
and (q_1)

or (q_1)

0 q_0 1 q_1
```

Transitions:

- $0 \rightarrow q_0$
- $1 \rightarrow q_1$
- $\land(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \land q_{b_2}$
- $\lor(q_{b_1}, q_{b_2}) \rightarrow q_{b_1} \lor q_{b_2}$

Final states:

$F = \{q_1\}$
Tree Automata with Global Equalities and Disequalities

A tree automaton $A$ with global equalities and disequalities (TAGED) is given by:

$$\begin{align*}
\Sigma & \text{ alphabet} \\
Q & \text{ set of states} \\
F & \text{ set of final states} \\
\Delta & \text{ set of rules}
\end{align*}$$

\text{tree automaton}
Tree Automata with Global Equalities and Disequalities

A tree automaton $A$ with global equalities and disequalities (TAGED) is given by:

- $\Sigma$ alphabet
- $Q$ set of states
- $F$ set of final states
- $\Delta$ set of rules

$A = \subseteq Q^2$ reflexive and symmetric relation on a subset of $Q$
$\not= A \subseteq Q^2$ irreflexive and symmetric relation

E. Filiot
Logics for $n$-ary Queries in Trees
2008, October
Successful Runs

$q_f \in F$

$q = A q' \implies t t' = t t'$
Successful Runs

$q_f \in F$

$q \neq_A q' \Rightarrow t' \neq t'$
Successful Runs

- equalities and disequalities can be tested arbitrarily far away
- different from usual **Automata with Constraints** where tests are **local** (BogaertT92, DauchetCC95, KariantoL07)
Example: \( \{ f(t, t) \mid t \in T_\Sigma \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  
  \[
  a \rightarrow q \\
  f(q, q) \rightarrow q \\
  f(q, q) \rightarrow q_1 \\
  f(q, q) \rightarrow q_2 \\
  f(q_1, q_2) \rightarrow q_f 
  \]

- \( q_1 = q_2 \)
Example: \( \{ f(t, t) \mid t \in T_\Sigma \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  
  - \( a \rightarrow q \)
  - \( f(q, q) \rightarrow q \)
  - \( f(q, q) \rightarrow q_1 \)
  - \( f(q, q) \rightarrow q_2 \)
  - \( f(q_1, q_2) \rightarrow q_f \)

- \( q_1 = A q_2 \)
Example: \( \{ f(t, t) \mid t \in T_\Sigma \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  - \( a \rightarrow q \)
  - \( f(q, q) \rightarrow q \)
  - \( f(q, q) \rightarrow q_1 \)
  - \( f(q, q) \rightarrow q_2 \)
  - \( f(q_1, q_2) \rightarrow q_f \)
- \( q_1 = A q_2 \)
Example: \[ \{ f(t, t) \mid t \in T_\Sigma \} \]

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  
  \[
  a \rightarrow q \\
  f(q, q) \rightarrow q \\
  f(q, q) \rightarrow q_1 \\
  f(q, q) \rightarrow q_2 \\
  f(q_1, q_2) \rightarrow q_f \\
  \]

- \( q_1 = A \) \( q_2 \)
Example: \( \{ f(t, t) \mid t \in T_\Sigma \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  
  \[
  \begin{align*}
  a &\rightarrow q \\
  f(q, q) &\rightarrow q \\
  f(q, q) &\rightarrow q_1 \\
  f(q, q) &\rightarrow q_2 \\
  f(q_1, q_2) &\rightarrow q_f \\
  \end{align*}
  \]

- \( q_1 \equiv A q_2 \)
Example: \( \{ f(t, t) \mid t \in T_{\Sigma} \} \)

\[ \Sigma = \{ f, a \} \]

\[ Q = \{ q, q_f, q_1, q_2 \} \]

\[ F = \{ q_f \} \]

\[ \Delta = \]

\[ a \rightarrow q \]

\[ f(q, q) \rightarrow q \]

\[ f(q, q) \rightarrow q_1 \]

\[ f(q, q) \rightarrow q_2 \]

\[ f(q_1, q_2) \rightarrow q_f \]

\[ q_1 = A q_2 \]
Example: \( \{ f(t, t) \mid t \in T_\Sigma \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  
  \[
  \begin{align*}
  a & \rightarrow q \\
  f(q, q) & \rightarrow q \\
  f(q, q) & \rightarrow q_1 \\
  f(q, q) & \rightarrow q_2 \\
  f(q_1, q_2) & \rightarrow q_f \\
  \end{align*}
  \]

- \( q_1 = A \quad q_2 \)
Example: \( \{ f(t, t) | t \in T_\Sigma \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  - \( a \rightarrow q \)
  - \( f(q, q) \rightarrow q \)
  - \( f(q, q) \rightarrow q_1 \)
  - \( f(q, q) \rightarrow q_2 \)
  - \( f(q_1, q_2) \rightarrow q_f \)
- \( q_1 = A q_2 \)
Example: \( \{ f(t, s) \mid t, s \in T_\Sigma, \ t \neq s \} \)

- \( \Sigma = \{ f, a \} \)
- \( Q = \{ q, q_f, q_1, q_2 \} \)
- \( F = \{ q_f \} \)
- \( \Delta = \)
  - \( a \rightarrow q \)
  - \( f(q, q) \rightarrow q \)
  - \( f(q, q) \rightarrow q_1 \)
  - \( f(q, q) \rightarrow q_2 \)
  - \( f(q_1, q_2) \rightarrow q_f \)

- \( q_1 \neq A q_2 \)
Some properties of TAGEDs

Proposition

- TAGED-recognizable languages are closed by union and intersection, but not by complement;
- Membership is NP-complete;
- TAGED are not determinizable (counter-example \( \{ f(t, t) \mid t \in T_\Sigma \} \));
- Universality is undecidable.
Some properties of TAGEDs

Proposition

- TAGED-recognizable languages are closed by union and intersection, but not by complement;
- Membership is NP-complete;
- TAGED are not determinizable (counter-example \( \{ f(t, t) \mid t \in T_\Sigma \} \));
- Universality is undecidable.

Emptiness Problem

Input: a TAGED \( A \)  
Output: \( L(A) \neq \emptyset \)?

Theorem

Emptiness is:

- \( \text{EXPTIME-complete for positive TAGED (} \neq_A = \emptyset \) \)
- decidable in \( \text{NEXPTIME} \) for negative TAGED (\( =_A = \emptyset \) )
- decidable in linear-time for positive TAGED such that \( =_A \subseteq id_Q \)
Bounded TAGEDs

Definition
A bounded TAGED is a pair \((A, k)\) where \(A\) is a TAGED and \(k \in \mathbb{N}\) is a natural number.
Bounded TAGEDs

Definition
A bounded TAGED is a pair \((A, k)\) where \(A\) is a TAGED and \(k \in \mathbb{N}\) is a natural number.

Definition (Successful Runs)
Additional condition: along any branch, the number of states from \(\text{dom}(\neq_A)\) is smaller than \(k\).
Bounded TAGEDs

**Definition**
A bounded TAGED is a pair \((A, k)\) where \(A\) is a TAGED and \(k \in \mathbb{N}\) is a natural number.

**Definition (Successful Runs)**
Additional condition: along any branch, the number of states from \(\text{dom}(\neq A)\) is smaller than \(k\).

By using a pumping technique one can show that:

**Theorem**
*Emptiness of bounded TAGEDs is decidable in 2NEXPTIME.*
Emptiness of bounded TAGED: Sketch of Proof

**Idea:** if the automaton accepts a tree $t$ then $t$ is not too big (its size is bounded in $|A|$).
Emptiness of bounded TAGED: Sketch of Proof

**Idea:** if the automaton accepts a tree $t$ then $t$ is not too big (its size is bounded in $|A|$).

**Lemmata**

- $=_{A} \subseteq id_{Q}$ is always possible
- in a successful run, same (sub)run below same states of $=_{A}$
- pumping technique preserving the constraints induced by $=_{A}$
Emptiness of bounded TAGED: Sketch of Proof

**Idea:** if the automaton accepts a tree $t$ then $t$ is not too big (its size is bounded in $|A|$).

**Lemmata**

- $=_{A} \subseteq id_{Q}$ is always possible
- in a successful run, same (sub)run below same states of $=_{A}$
- pumping technique preserving the constraints induced by $=_{A}$

**Algorithm**

1. find a tree and a run satisfying the constraints from $=_{A}$ but maybe not from $\neq_{A}$
2. test whether (and its run) can be repaired (polynomial algorithm)
3. if the test fails, choose another tree.
Emptiness of bounded TAGED: Sketch of Proof

Idea: if the automaton accepts a tree $t$ then $t$ is not too big (its size is bounded in $|A|$).

Lemmata

- $=_A \subseteq id_Q$ is always possible
- in a successful run, same (sub)run below same states of $=_A$
- pumping technique preserving the constraints induced by $=_A$

Algorithm

1. find a tree and a run satisfying the constraints from $=_A$ but maybe not from $\neq_A$
2. test whether (and its run) can be repaired (polynomial algorithm)
3. if the test fails, choose another tree.

Termination

If the automaton accepts a tree, then it accepts a repairable tree satisfying the constraints from $=_A$ whose size is exponential in $|A|$ and $k$. 
TQL to TAGED

TAGED for hedges over an infinite alphabet

- extends hedge automata (Murata’99) with global tests;
- transitions $\alpha(L) \rightarrow q$ where $L \subseteq Q^*$;
- lift all the results via a binary encoding of hedges.
TQL to TAGED

TAGED for hedges over an infinite alphabet

- extends hedge automata (Murata’99) with global tests;
- transitions $\alpha(L) \rightarrow q$ where $L \subseteq Q^*$;
- lift all the results via a binary encoding of hedges.

Bounded TQL $\rightarrow$ Bounded TAGED

- new construction;
- two difficulties: variables and hedge operations;
TQL to TAGED

TAGED for hedges over an infinite alphabet

- extends hedge automata (Murata’99) with global tests;
- transitions $\alpha(L) \rightarrow q$ where $L \subseteq Q^*$;
- lift all the results via a binary encoding of hedges.

Bounded TQL $\rightarrow$ Bounded TAGED

- new construction;
- two difficulties: variables and hedge operations;
- states: sets of subformulas $\alpha[\phi], X, \neg X$;
- variables are added non-deterministically to the states;
- hedge operations are interpreted as operations on state languages

- $\{\ldots, X, \ldots\} \equiv_A \{\ldots, X, \ldots\}$
- $\{\ldots, X, \ldots\} \not\equiv_A \{\ldots, \neg X, \ldots\}$
Conclusion
Summary of the contributions

**Query composition (FNTT, PODS’07)**

- extends the navigational XPath paradigm to $n$-ary queries
- simple acyclicity notion
- FO-complete and polynomial $n$-ary XPath languages
Summary of the contributions

Query composition (FNTT, PODS’07)
- extends the navigational XPath paradigm to $n$-ary queries
- simple acyclicity notion
- FO-complete and polynomial $n$-ary XPath languages

TQL (FTT, CSL’07)
- tree pattern language for hedges
- decidable fragment with tree variables
- by reduction to TAGED (FTT, DLT’08)
- new automaton construction
Some Perspectives

Query composition

- query answering algorithms specific to $\text{Comp}(\text{ch}, \text{ch}^*, \text{lab}_a)$
- streaming (GauwinCNT08), enumeration (collaboration with O.Gauwin, A.Durand, ANR Enum)
Some Perspectives

Query composition
- query answering algorithms specific to $\text{Comp}(\text{ch}, \text{ch}^*, \text{lab}_a)$
- streaming (GauwinCNT08), enumeration (collaboration with O. Gauwin, A. Durand, ANR Enum)

TQL
- lower bounds ($\text{TQL} + \text{TAGED}$)
- guarded fragment
Some Perspectives

Query composition

- query answering algorithms specific to $\text{Comp}(\text{ch}, \text{ch}^*, \text{lab}_a)$
- streaming (GauwinCNT08), enumeration (collaboration with O.Gauwin, A.Durand, ANR Enum)

TQL

- lower bounds ($\text{TQL} + \text{TAGED}$)
- guarded fragment
- ... or at least, a decidable fragment closed by negation
- query inclusion $\forall \overline{x} \ (\phi(\overline{x}) \rightarrow \psi(\overline{x}))$ iff not $\exists \overline{x}, \phi(\overline{x}) \land \neg \psi(\overline{x})$. 
Some Perspectives

Query composition

- query answering algorithms specific to $\text{Comp}(\text{ch, ch}^*, \text{lab}_a)$
- streaming (GauwinCNT08), enumeration (collaboration with O.Gauwin, A.Durand, ANR Enum)

TQL

- lower bounds (TQL + TAGED)
- guarded fragment
- ... or at least, a decidable fragment closed by negation
- query inclusion $\forall \overline{x} \ (\phi(\overline{x}) \rightarrow \psi(\overline{x}))$ iff not $\exists \overline{x}, \ \phi(\overline{x}) \land \neg \psi(\overline{x})$.
- emptiness of full TAGED
- application to security protocols (C.Vacher, F.Jacquemard, F.Klay)
Thank You