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

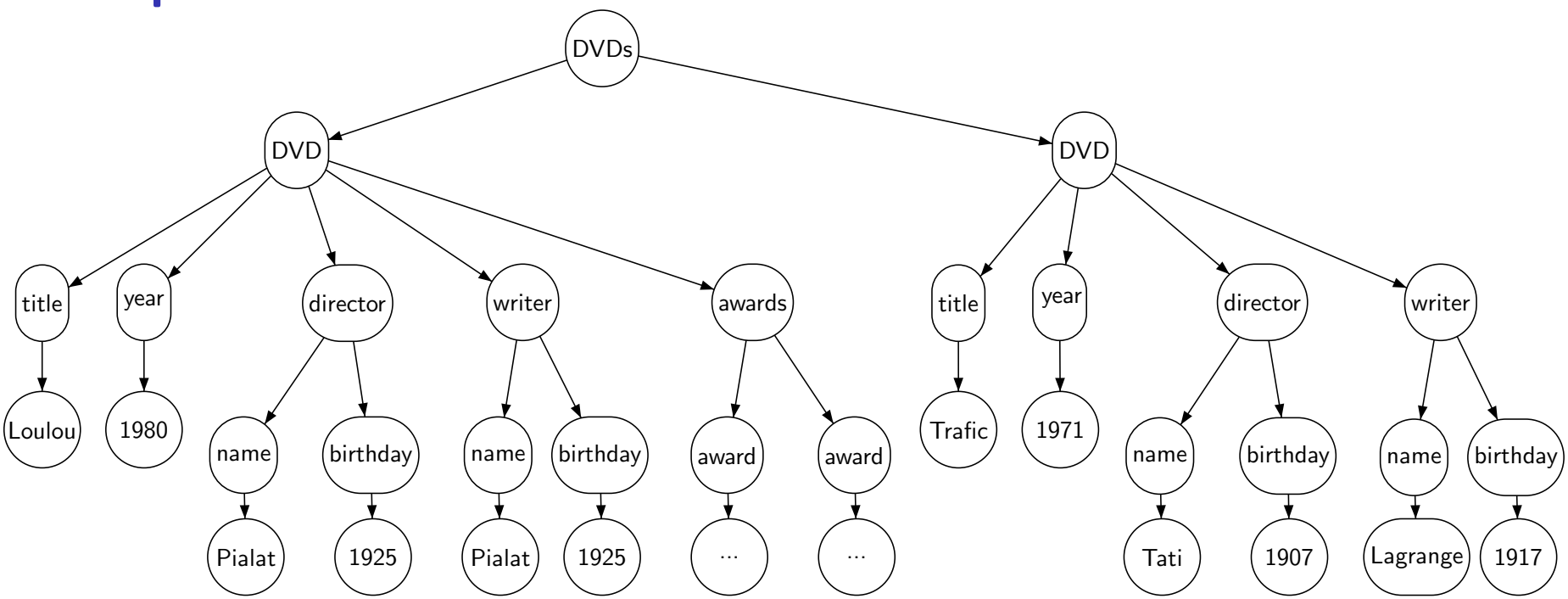
- markup language to represent tree-shaped data
- XML data big bang !
- standard for data exchange and data storage

eXtensible Markup Language

XML

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

Tree representation

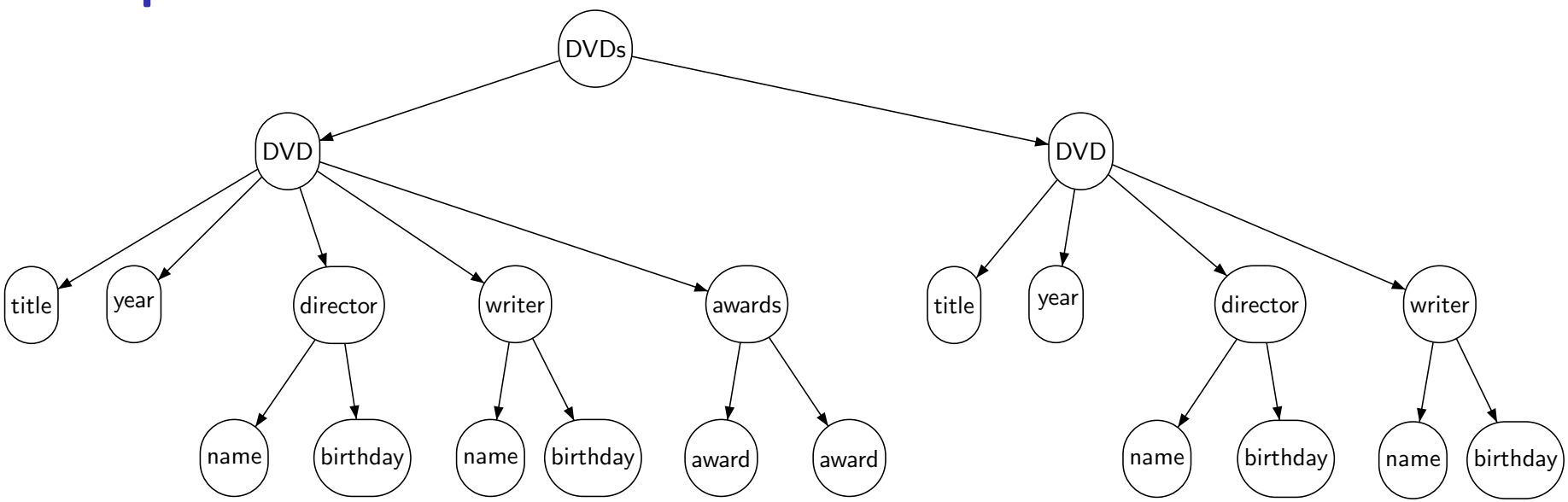


eXtensible Markup Language

XML

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

Tree representation

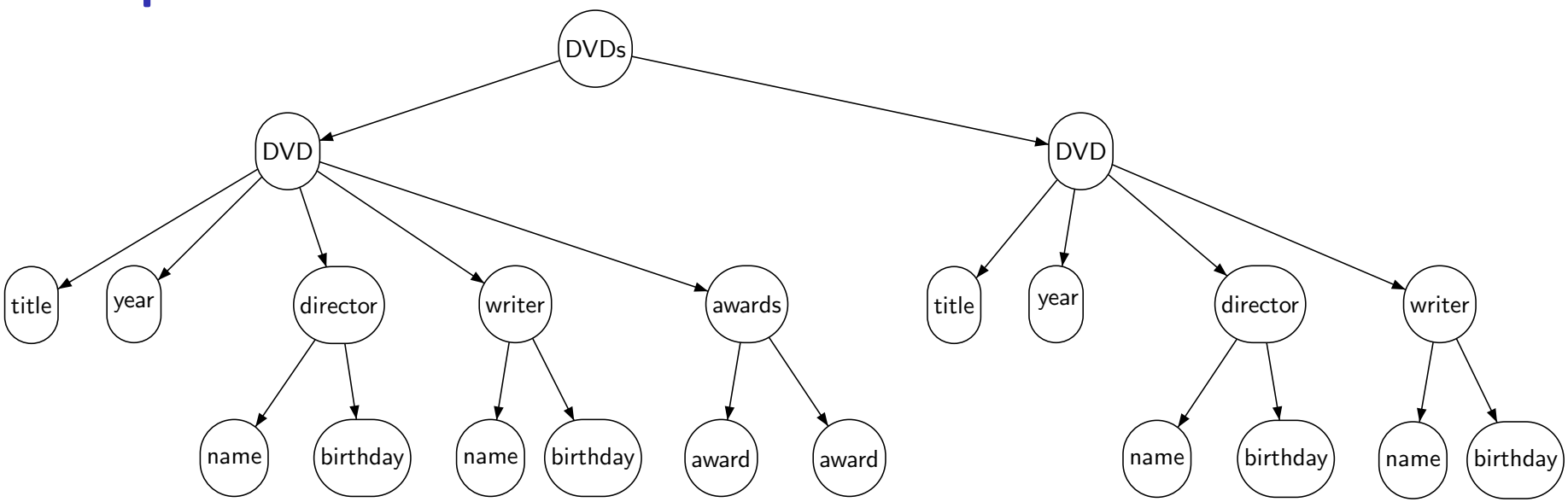


eXtensible Markup Language

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

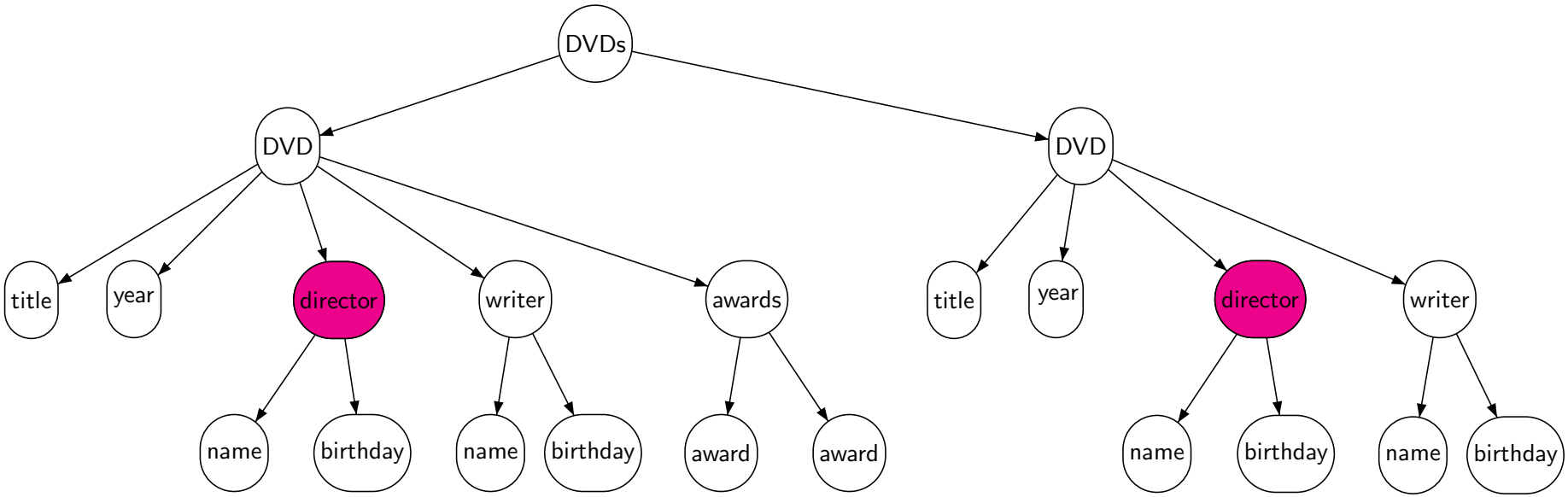
- access XML data, transform XML documents
- node selection in XML trees
- n-**ary** queries select **set of** n-**tuples** of nodes
  - $n = 1$: unary queries
  - $n = 2$: binary queries

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)

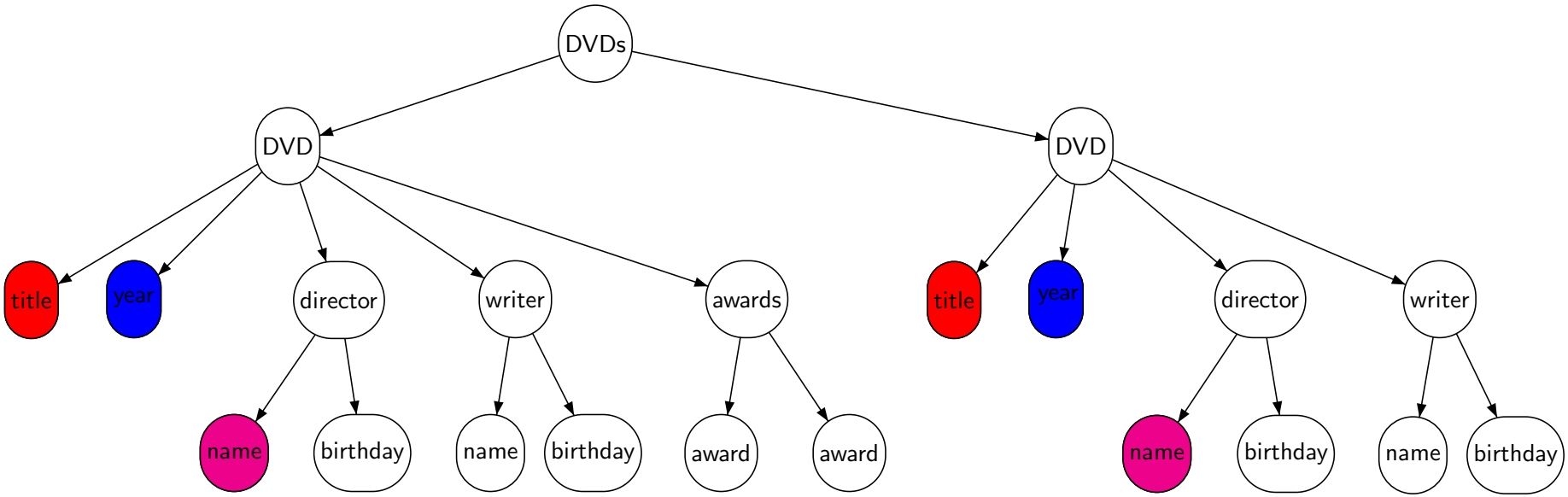


XML Queries

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 (title,year,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
  - μ-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
  - μ-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
   1. definitions
   2. expressiveness, query evaluation
   3. application to n-ary XPath logics
2. The Spatial Logic TQL
   1. Examples, Definition
   2. Expressiveness, Satisfiability
   3. Tree Automata with Global Constraints
3. Summary and Perspectives

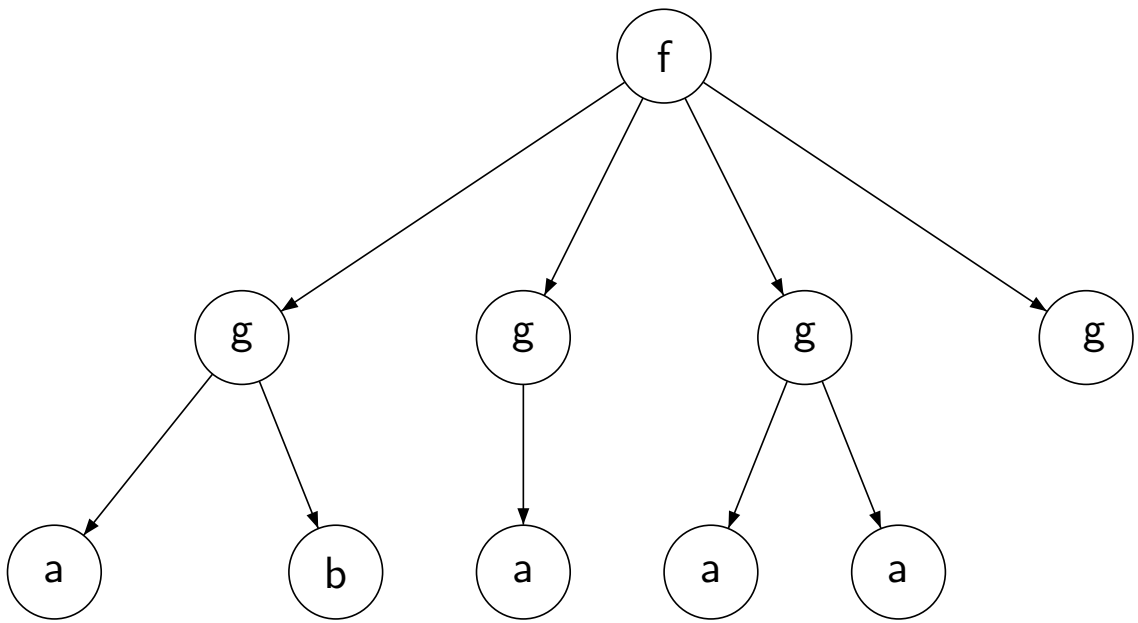
Outline

PART I: Composing Binary Queries

Trees and Queries

Trees

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



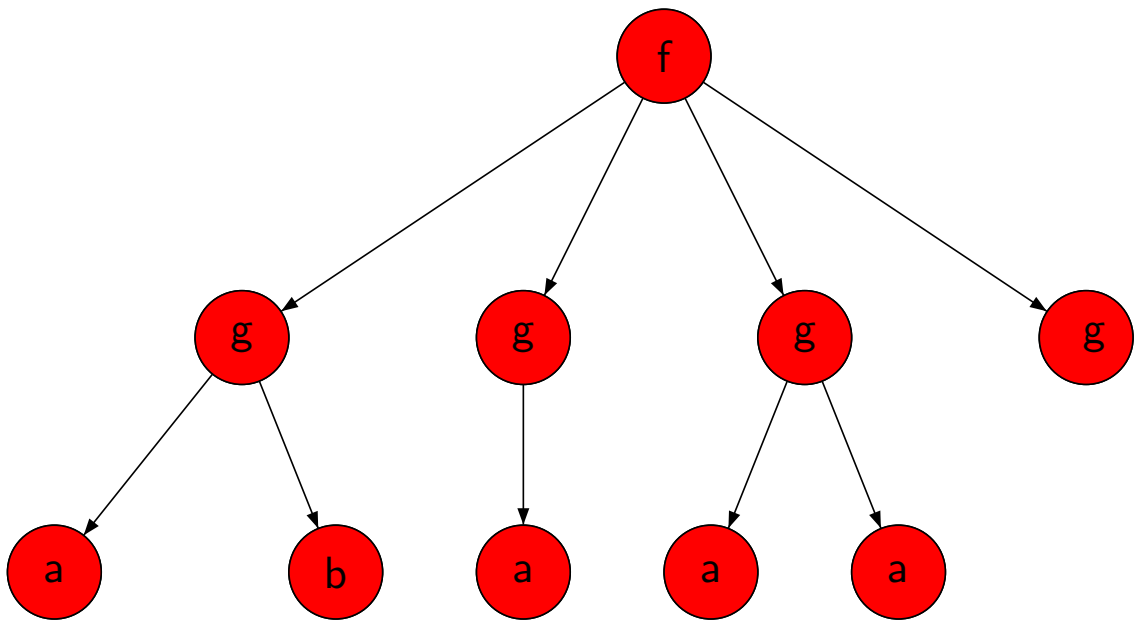
Queries

Trees and Queries

Trees

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

Unary Relations: $\text{nodes}(t)$



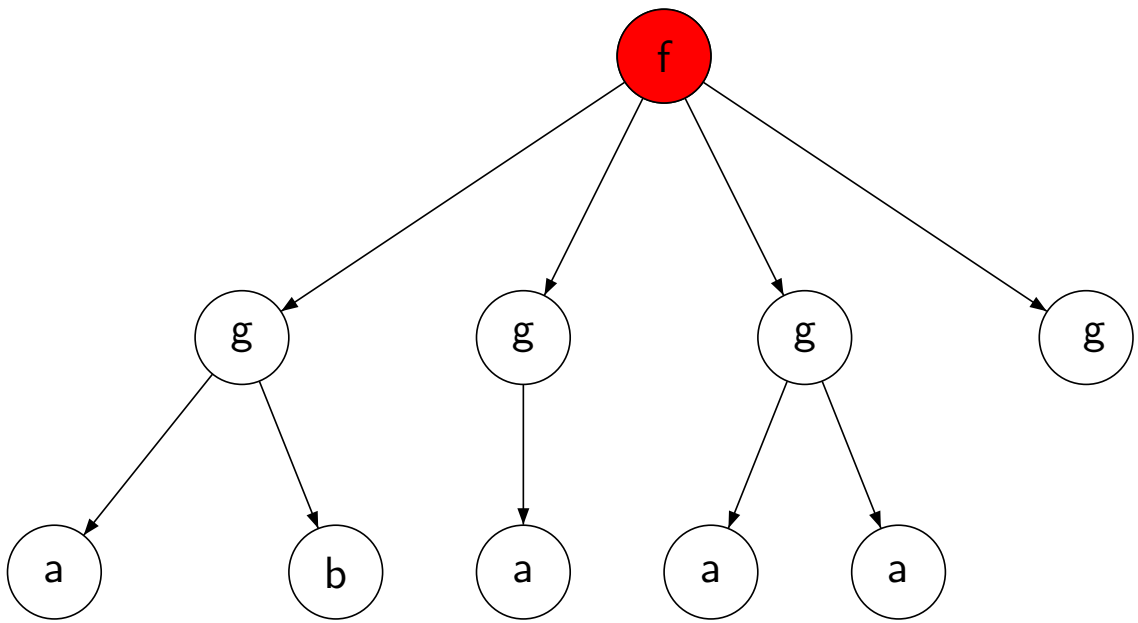
Queries

Trees and Queries

Trees

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

Unary Relations: nodes(t) , root(t)



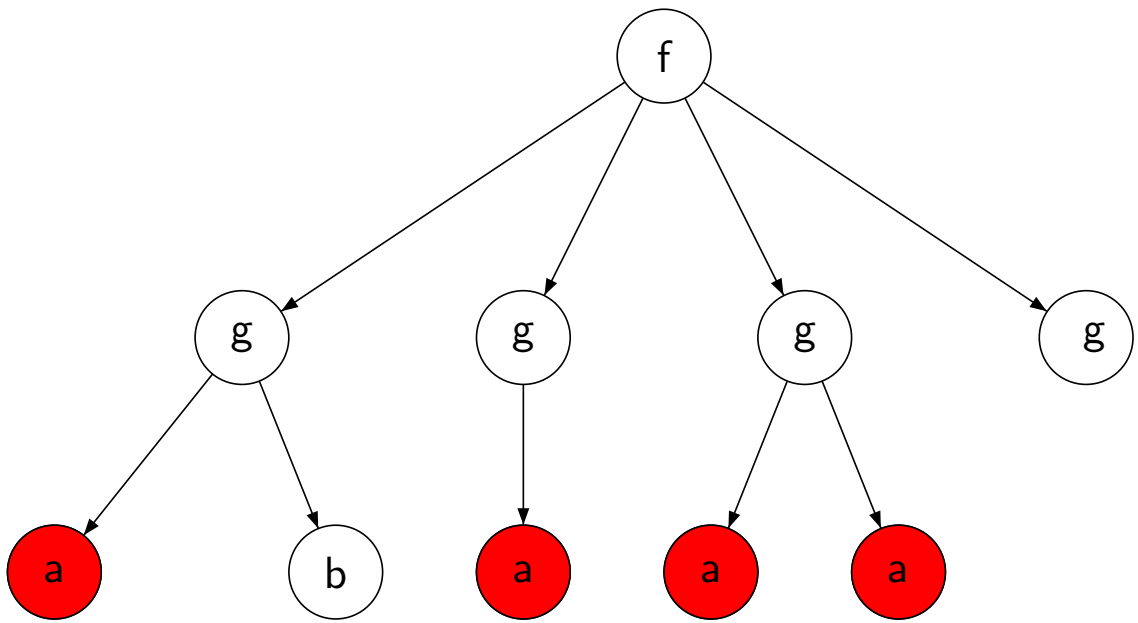
Queries

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

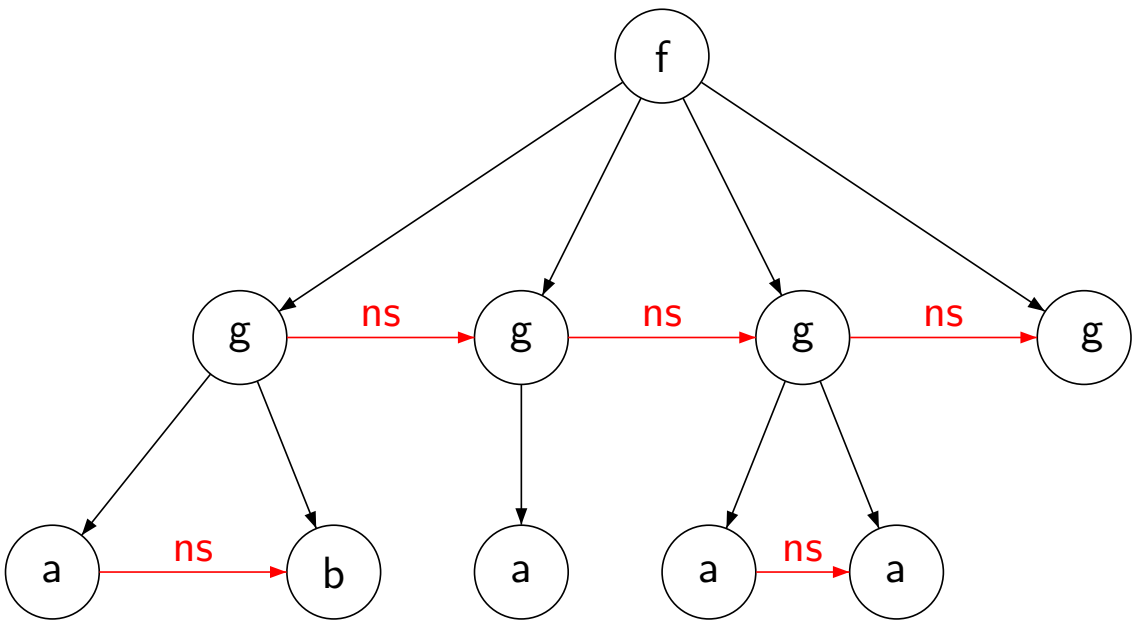
Trees and Queries

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: ns



Queries

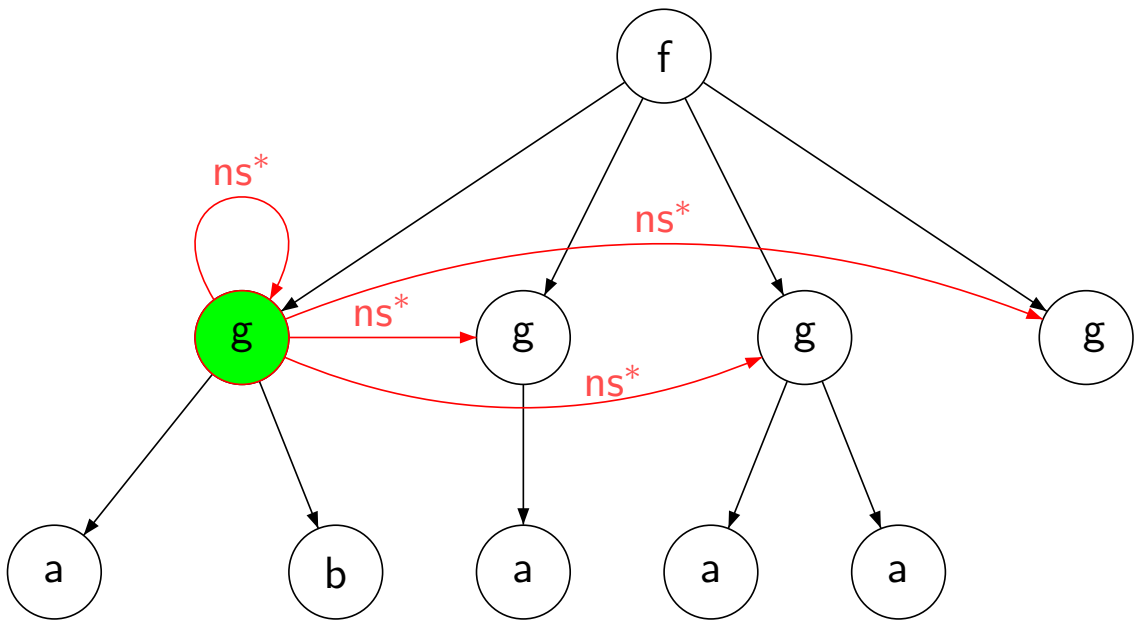
Trees and Queries

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: ns , ns^*



Queries

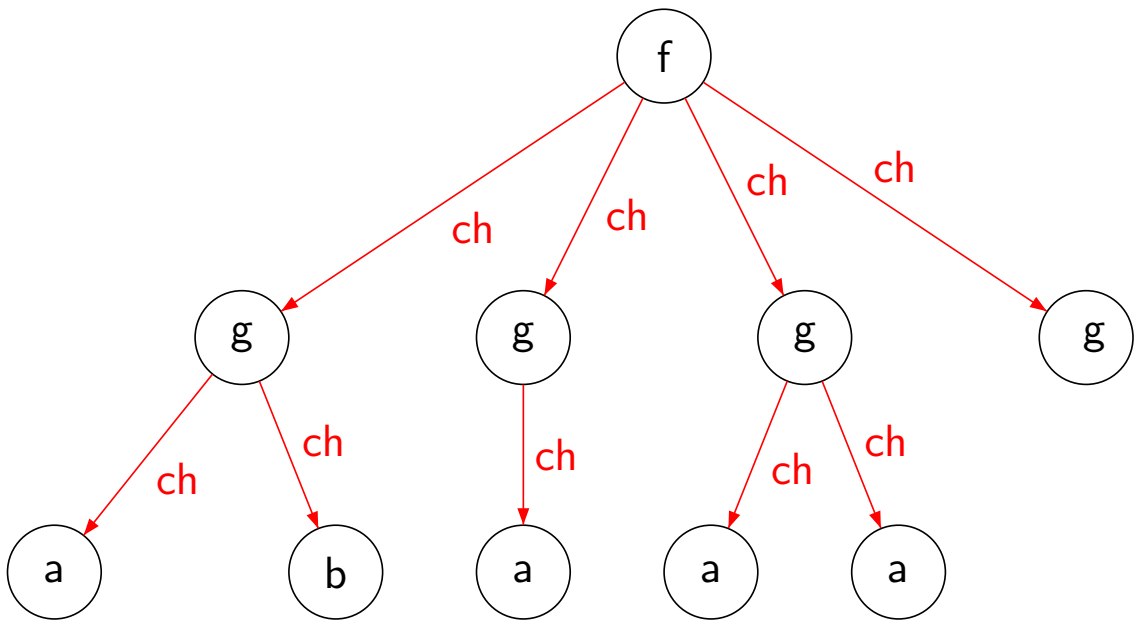
Trees and Queries

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: ns , ns^* , ch



Queries

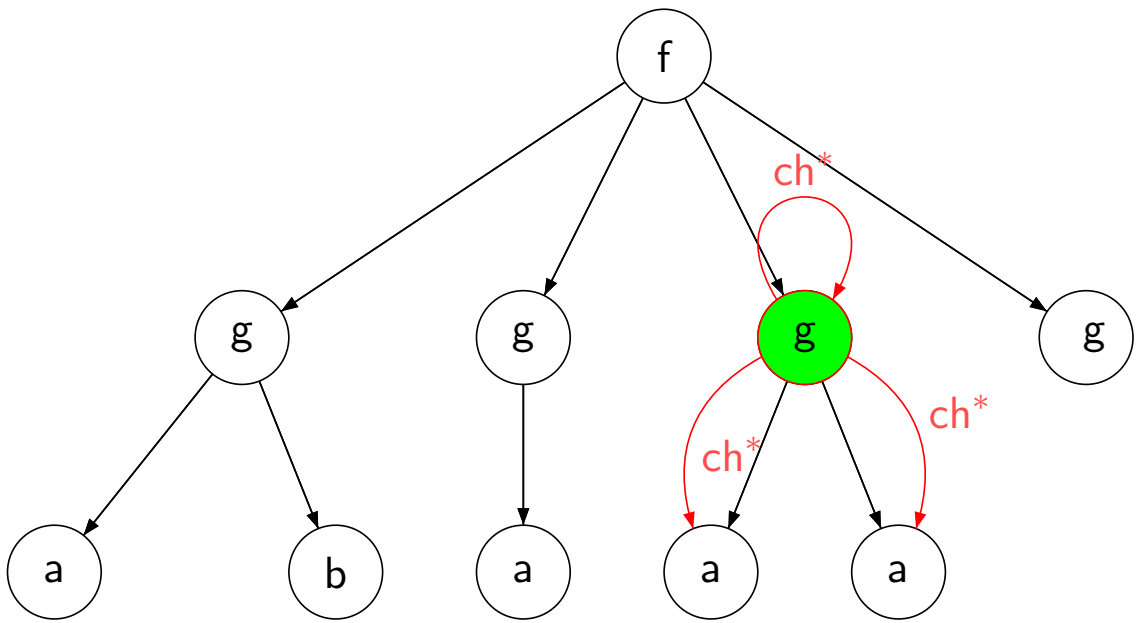
Trees and Queries

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: ns , ns^* , ch , 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: ns , ns^* , ch , ch^*

Queries

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

$$q(t) \subseteq \text{nodes}(t)^n$$

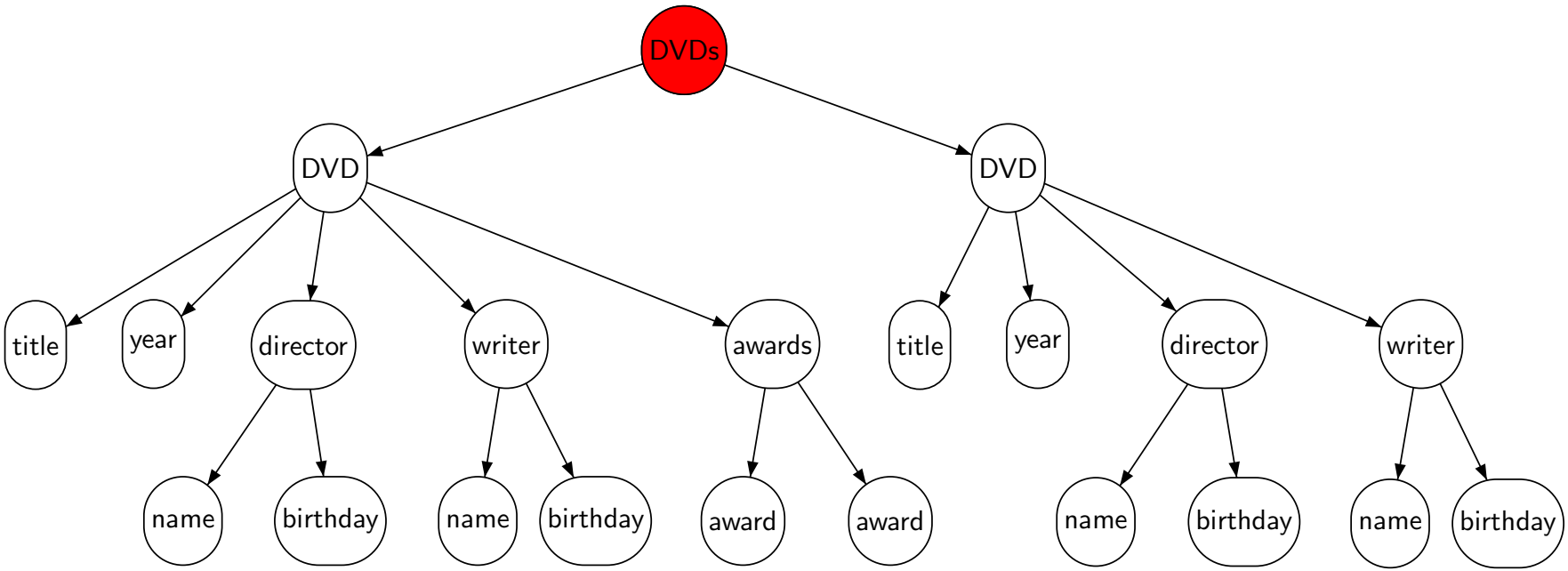
The navigational language XPath

- to navigate and select sets of nodes in XML trees
- by defining path expressions
- complex counting conditions
- *CoreXPath*: navigational core (GottlobKP02)

The navigational language XPath

Example: select all director names

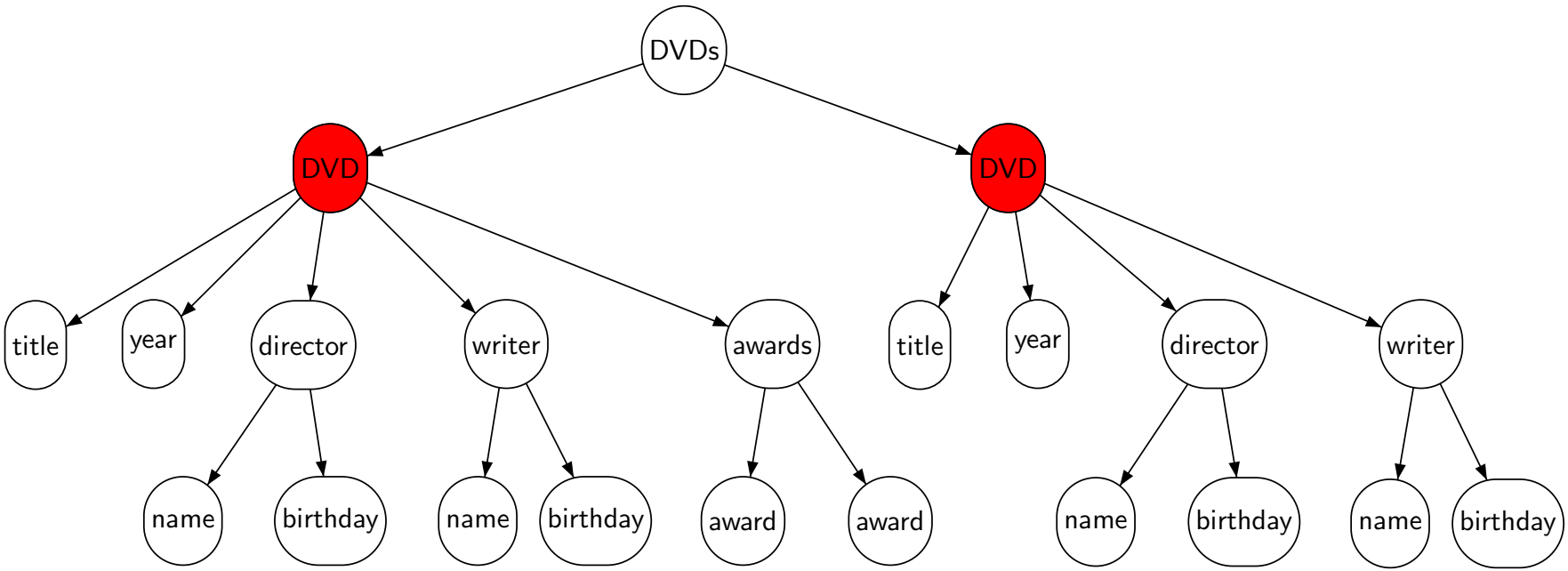
ch :: *DVD*/ch :: *director*/ch :: *name*



The navigational language XPath

Example: select all director names

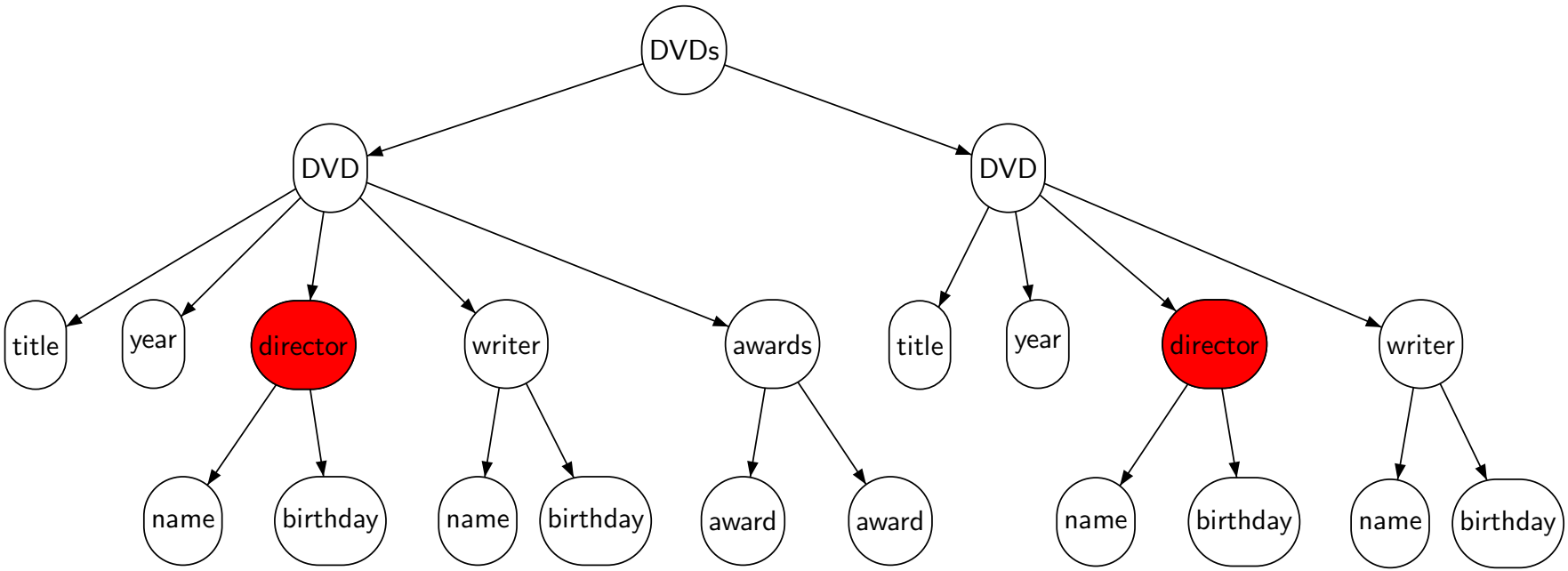
ch :: *DVD*/ch :: *director*/ch :: *name*



The navigational language XPath

Example: select all director names

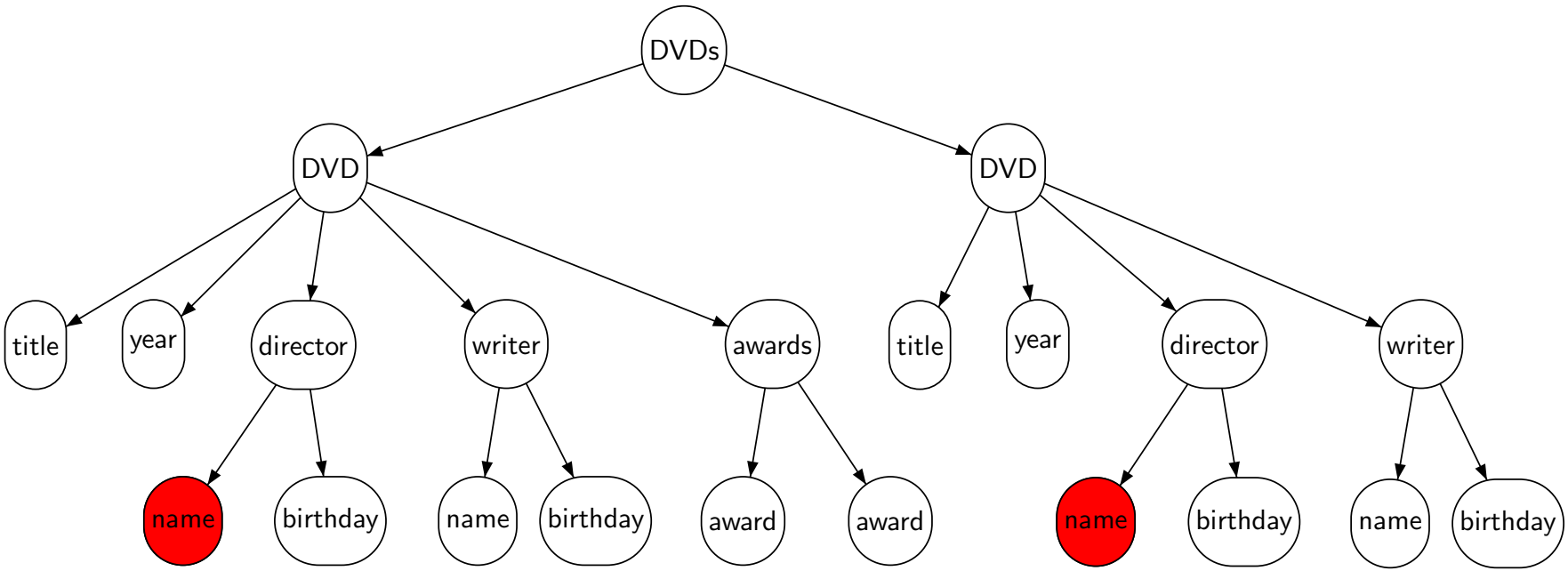
ch :: *DVD*/ch :: *director*/ch :: *name*



The navigational language XPath

Example: select all director names

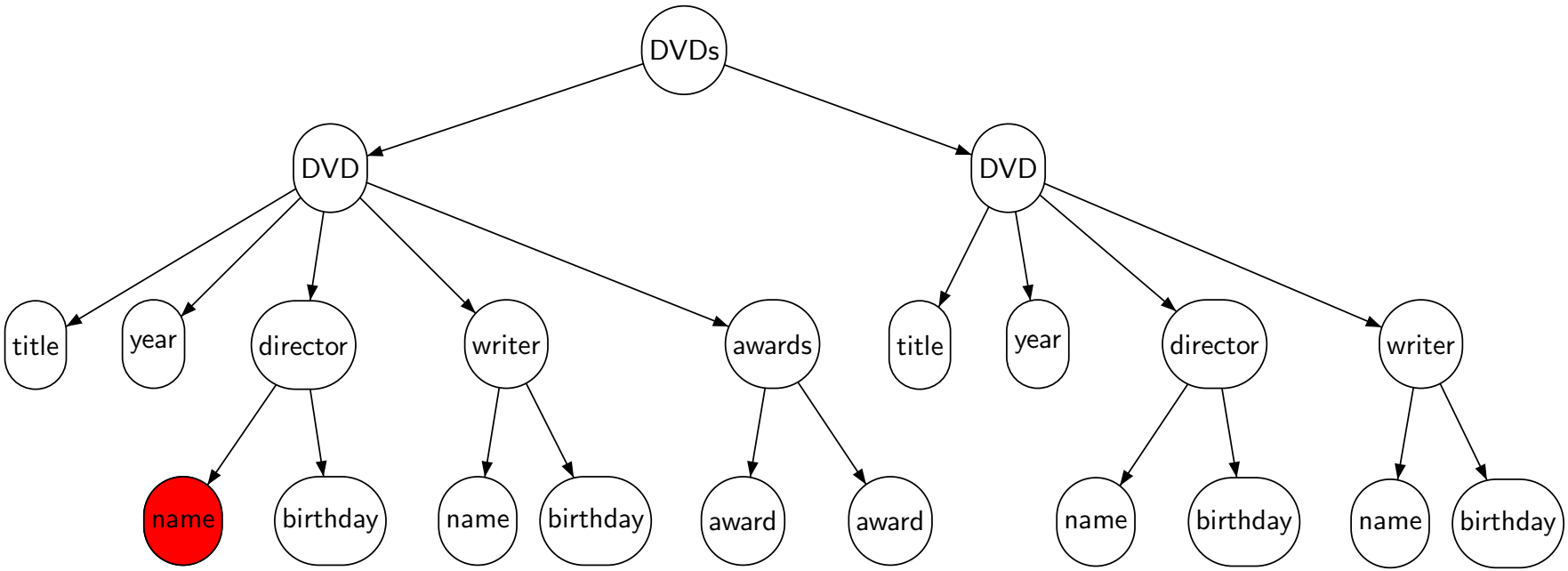
ch :: *DVD*/ch :: *director*/ch :: *name*



The navigational language XPath

Example: select all awarded director names

ch :: *DVD*[ch :: *awards*]/ch :: *director*/ch :: *name*



Expressions of CoreXPath and their semantics

Axis	$\mathsf{self}, \mathsf{ch}, \mathsf{ch}^+, \mathsf{ns}, \mathsf{ns}^+$
	$\mathsf{ch}^{-1}, (\mathsf{ch}^{-1})^+, \mathsf{ns}^{-1}, (\mathsf{ns}^{-1})^+$
Steps	$Axis::a$
	$Axis::*$
Composition	P_1/P_2
Union	$P_1 \cup P_2$
Tests	$P[\mathcal{T}]$
Path existence	P
Negation	not $\mathcal{T}$
Conjunction	$\mathcal{T}_1$ and $\mathcal{T}_2$

Expressions of CoreXPath and their semantics

Axis $\quad \mathsf{self}, \mathsf{ch}, \mathsf{ch}^+, \mathsf{ns}, \mathsf{ns}^+$
$\mathsf{ch}^{-1}, (\mathsf{ch}^{-1})^+, \mathsf{ns}^{-1}, (\mathsf{ns}^{-1})^+$

$[\![.]\!]^t \subseteq \mathsf{nodes}(t) \times \mathsf{nodes}(t)$

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

Composition $\quad [\![P_1/P_2]\!]^t = [\![P_1]\!]^t \circ [\![P_2]\!]^t$

Union $\quad [\![P_1 \cup P_2]\!]^t = [\![P_1]\!]^t \cup [\![P_2]\!]^t$

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

$[\![.]\!]^t_{\text{test}} \subseteq \mathsf{nodes}(t)$

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

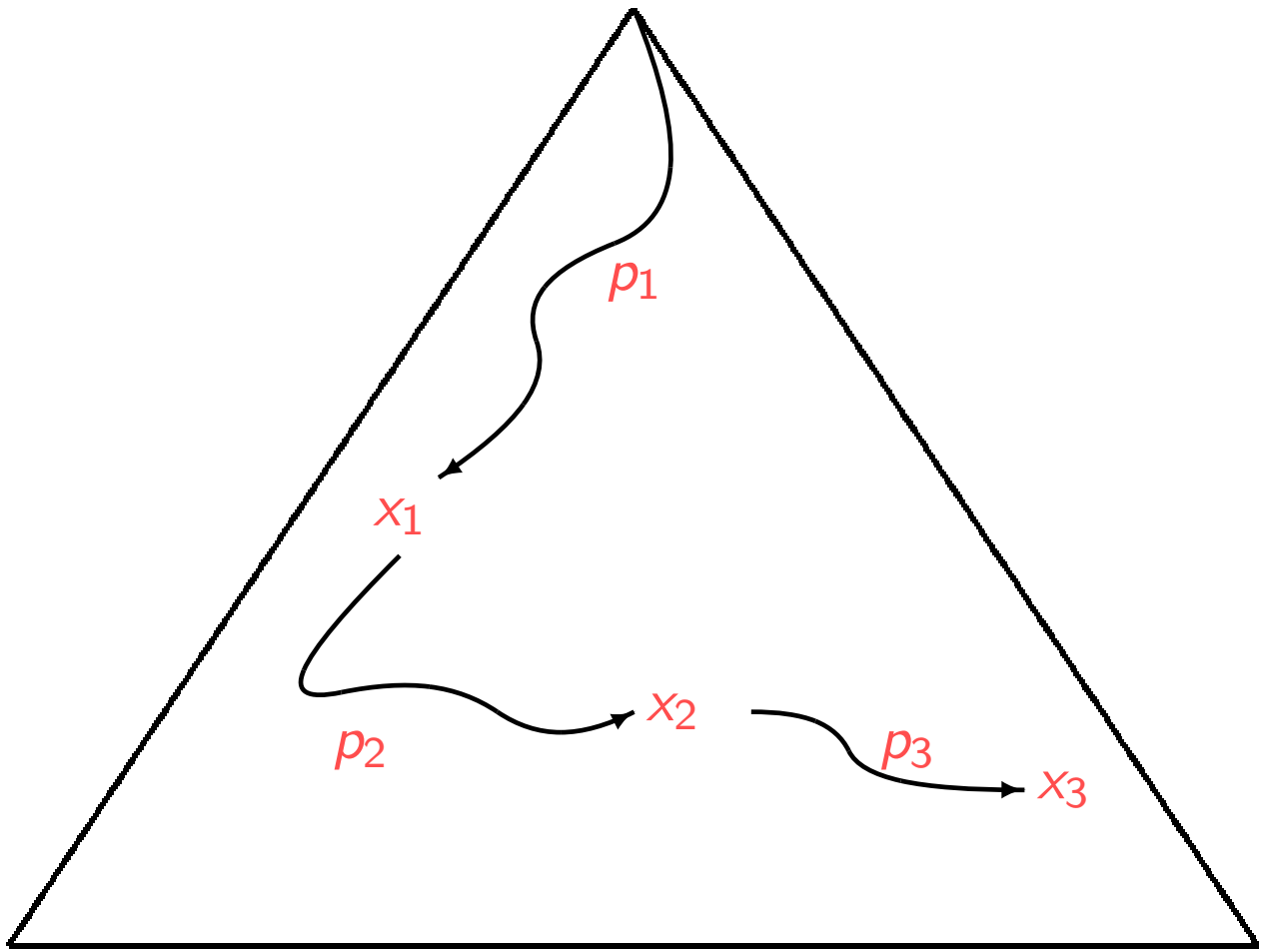
Negation $\quad [\![\text{not } \mathcal{T}]\!]^t_{\text{test}} = \mathsf{nodes}(t) - [\![\mathcal{T}]\!]^t_{\text{test}}$

Conjunction $\quad [\![\mathcal{T}_1 \text{ and } \mathcal{T}_2]\!]^t_{\text{test}} = [\![\mathcal{T}_1]\!]^t_{\text{test}} \cap [\![\mathcal{T}_2]\!]^t_{\text{test}}$

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

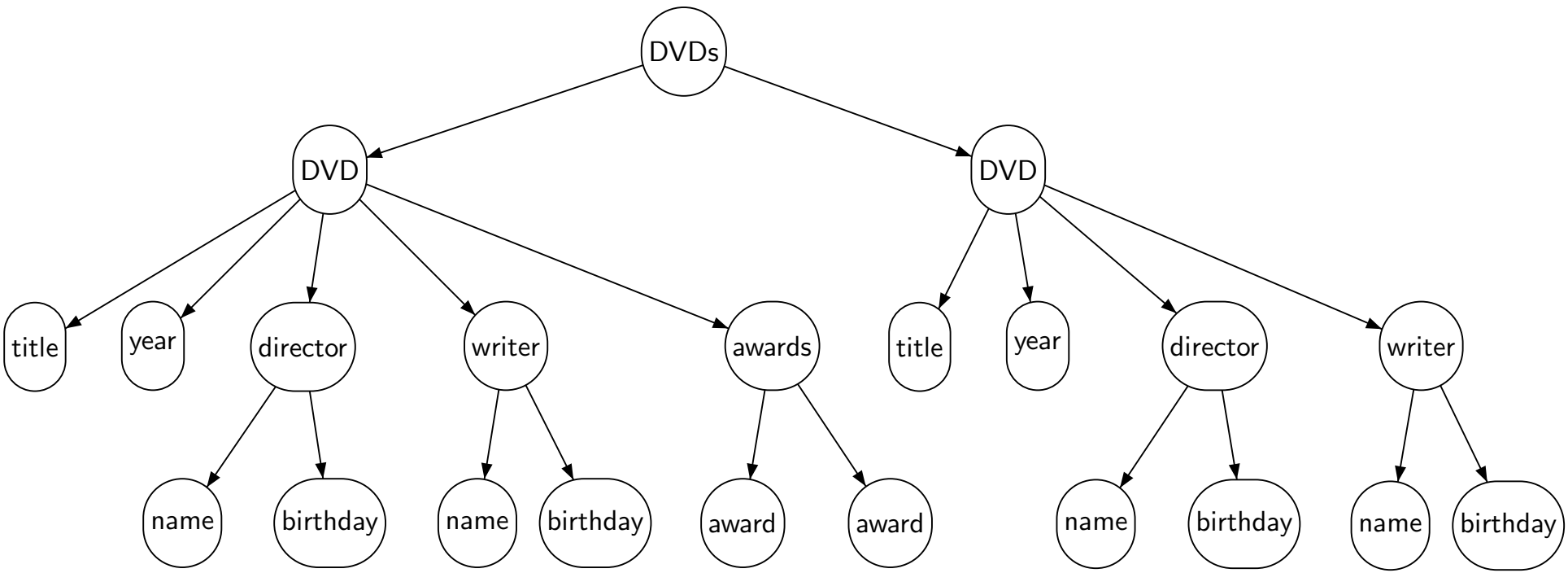


How to turn XPath into an n-ary query language?

- 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))

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

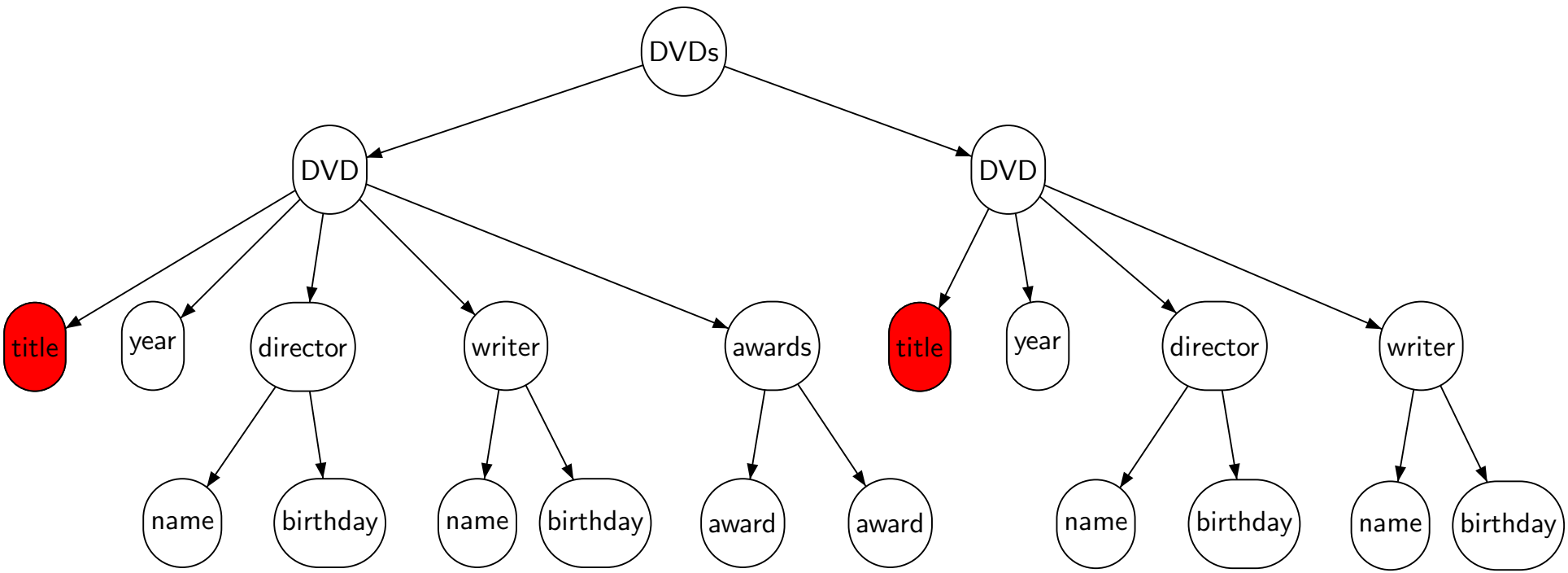


How to turn XPath into an n-ary query language?

- 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))

$\phi(x, y, z) \;=\; \mathsf{ch}^* :: \mathit{title}$

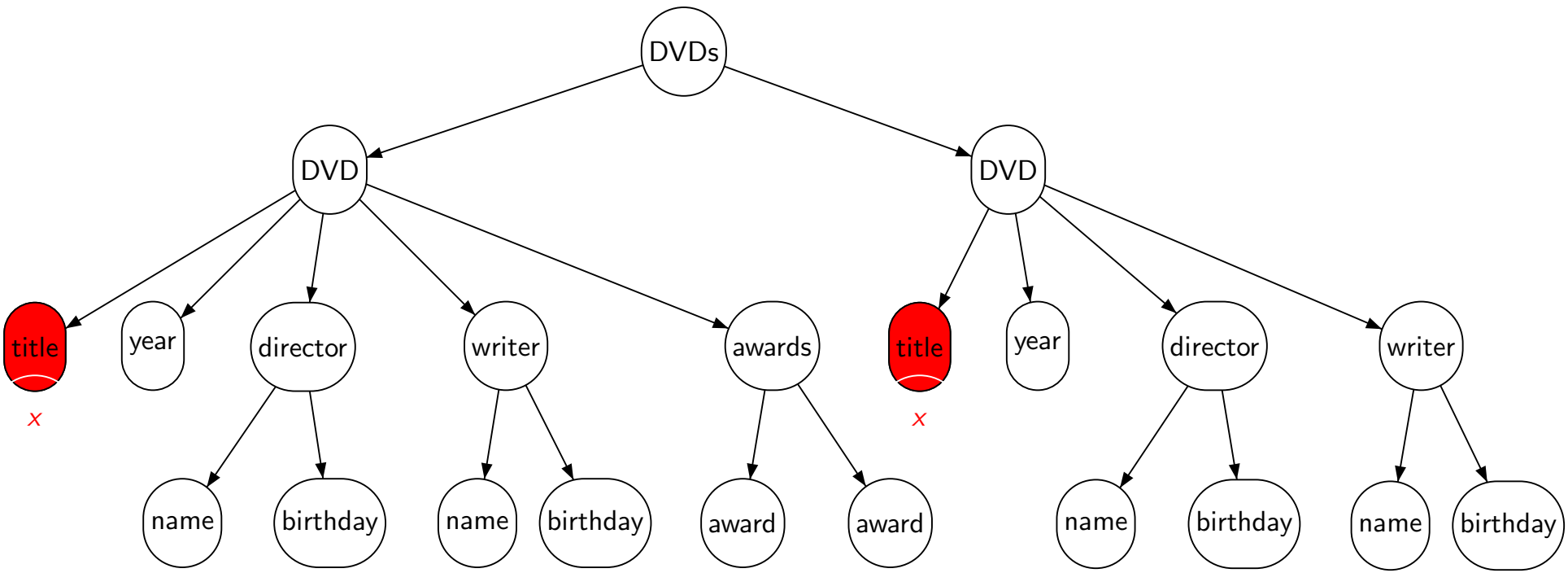


How to turn XPath into an n-ary query language?

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

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

$\phi(x, y, z) \;=\; \mathsf{ch}^* :: \mathit{title}/x$

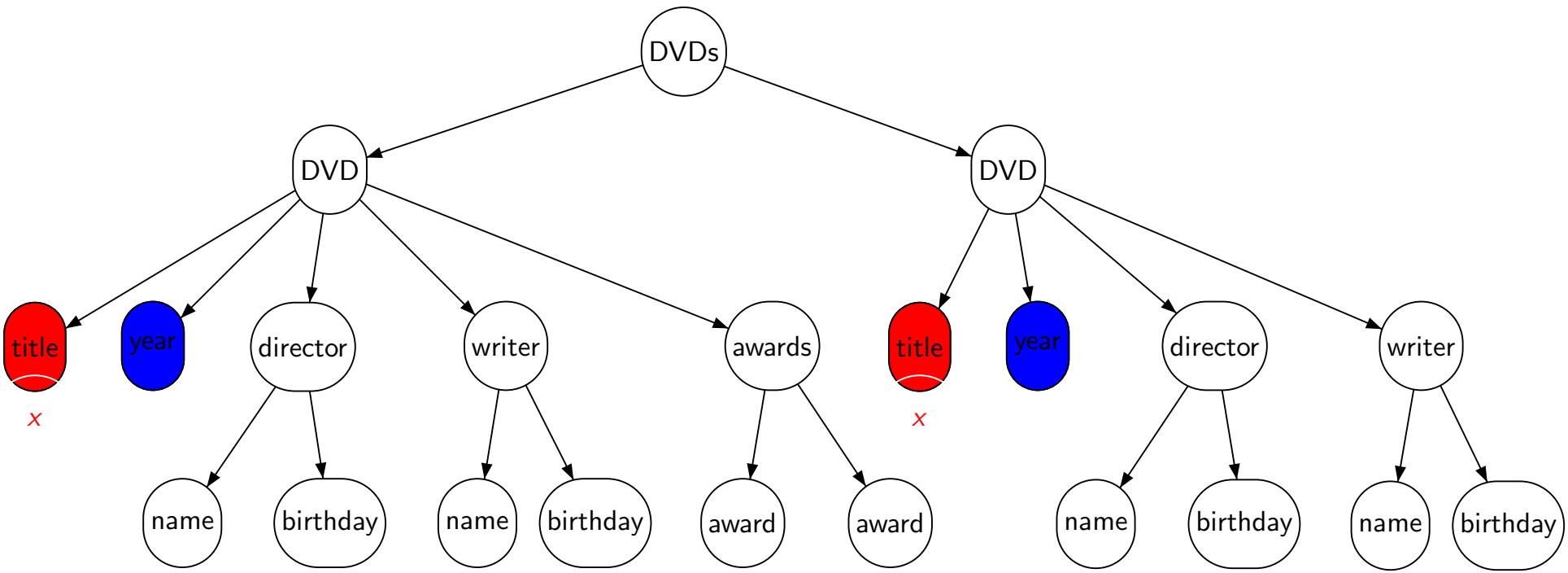


How to turn XPath into an n-ary query language?

- 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))

$\phi(x, y, z) \;=\; \mathsf{ch}^* :: \mathit{title}/x/\mathsf{ns} :: \mathit{year}$

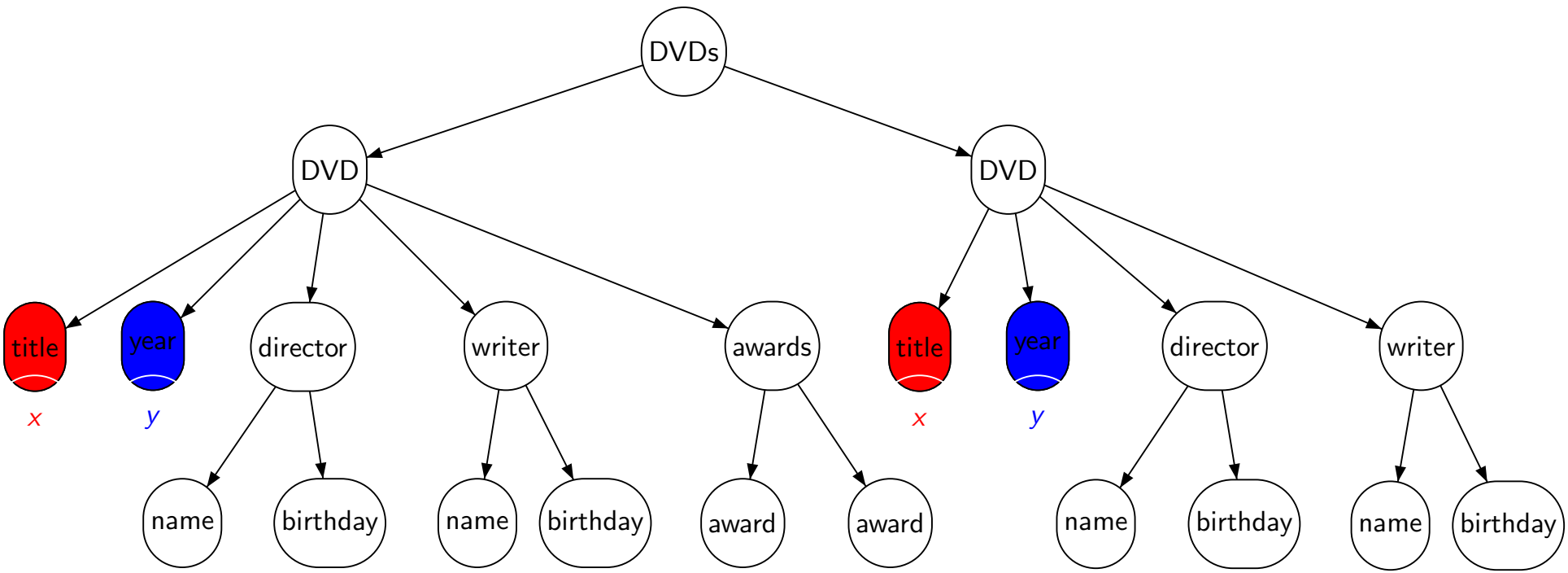


How to turn XPath into an n-ary query language?

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

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

$\phi(x, y, z) \;=\; \mathsf{ch}^* :: \mathit{title}/x/\mathsf{ns} :: \mathit{year}/y$

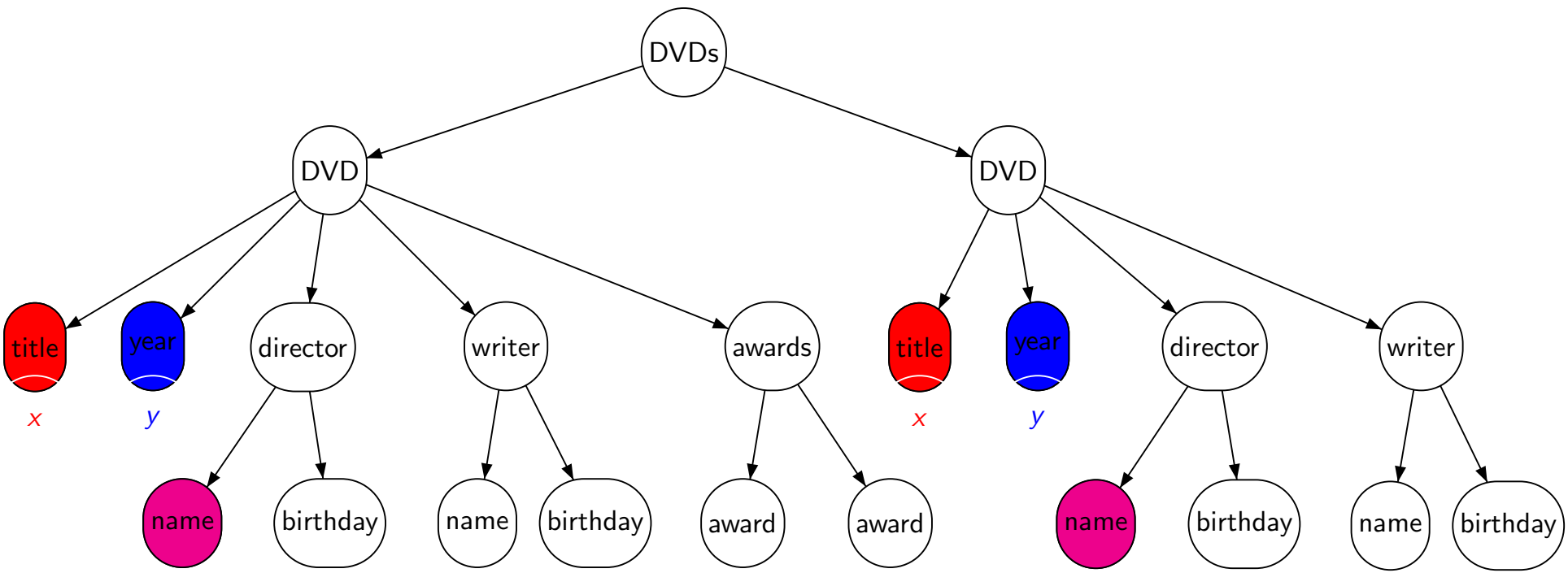


How to turn XPath into an n-ary query language?

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

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

$\phi(x, y, z) \;=\; \text{ch}^* :: \mathit{title}/x/\text{ns} :: \mathit{year}/\text{ns} :: \mathit{director}/\text{ch} :: \mathit{name}$

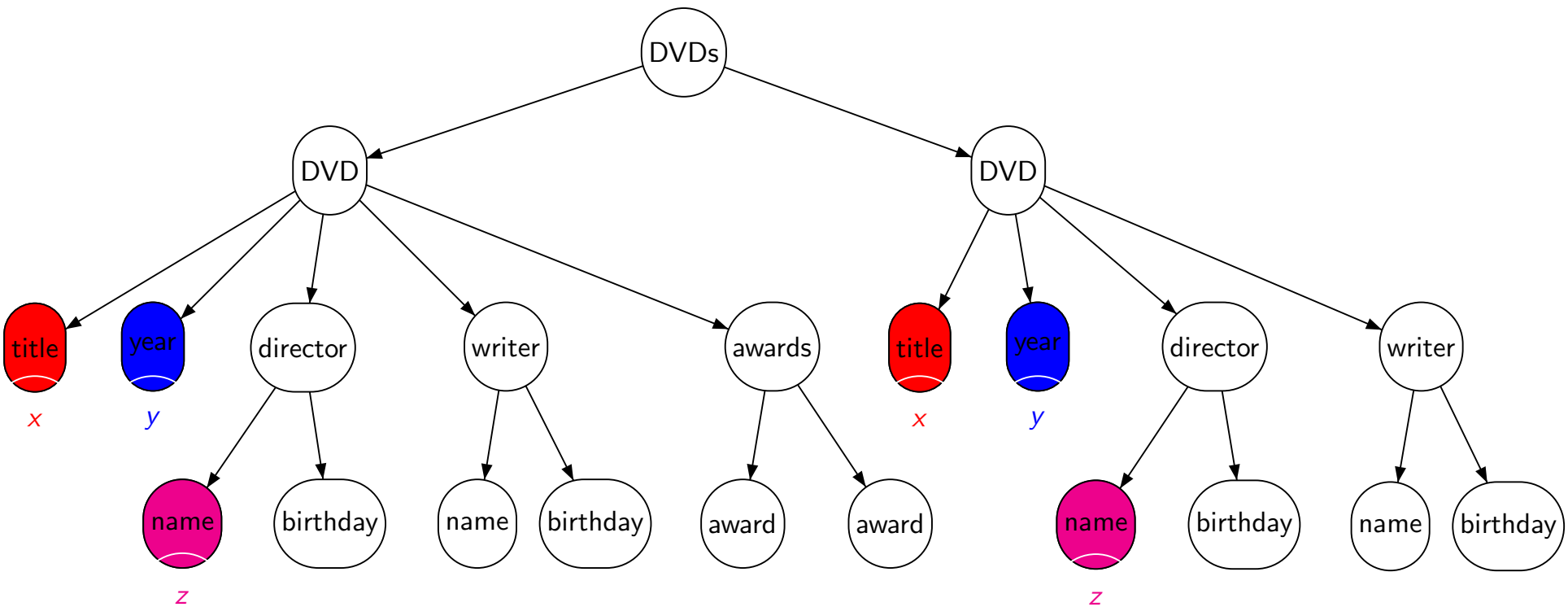


How to turn XPath into an n-ary query language?

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

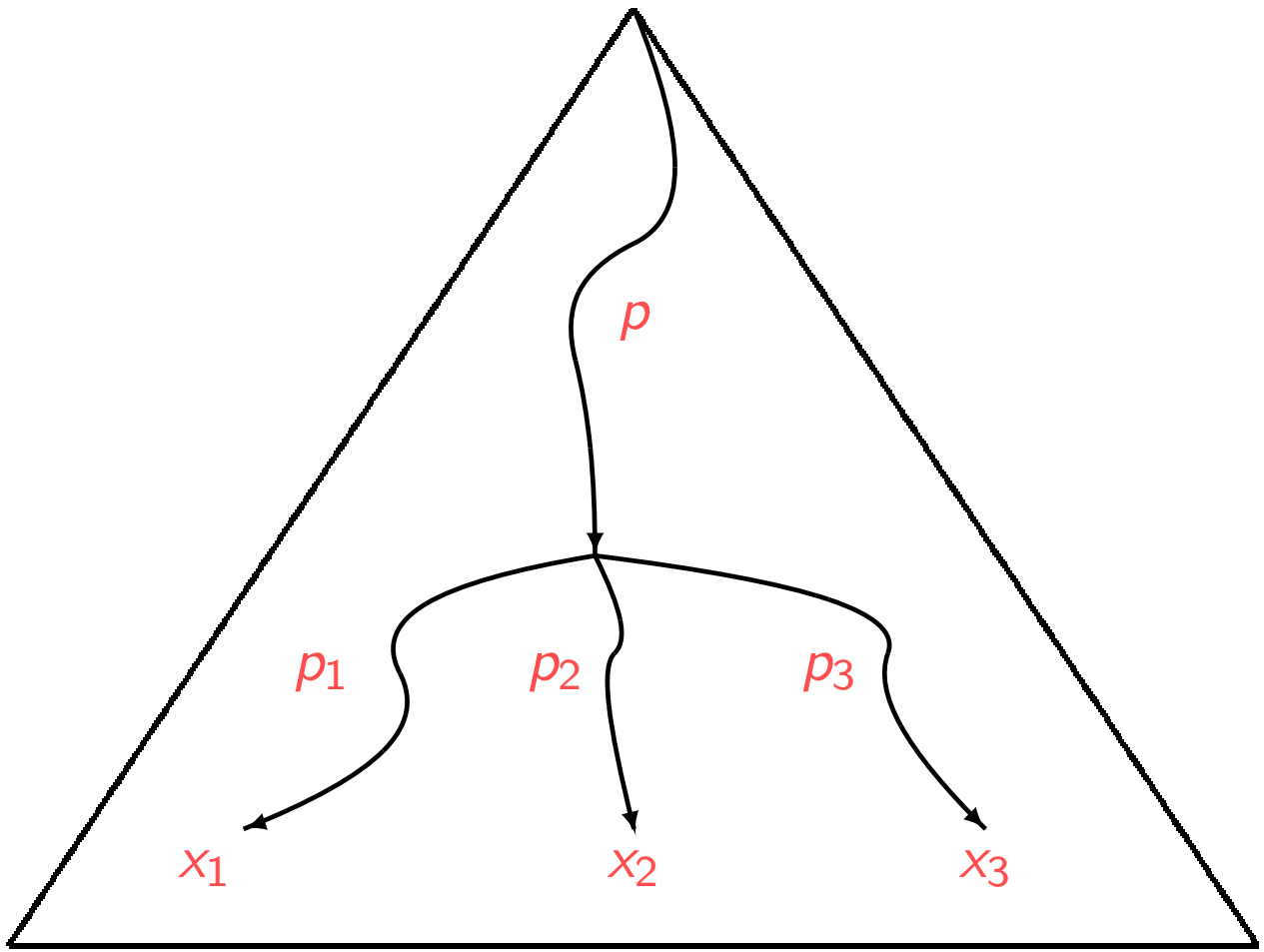
Example (All triples (title,year,director name))

$\phi(x, y, z) \;=\; \text{ch}^* :: \mathit{title}/x/\text{ns} :: \mathit{year}/\text{ns} :: \mathit{director}/\text{ch} :: \mathit{name}/z$



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

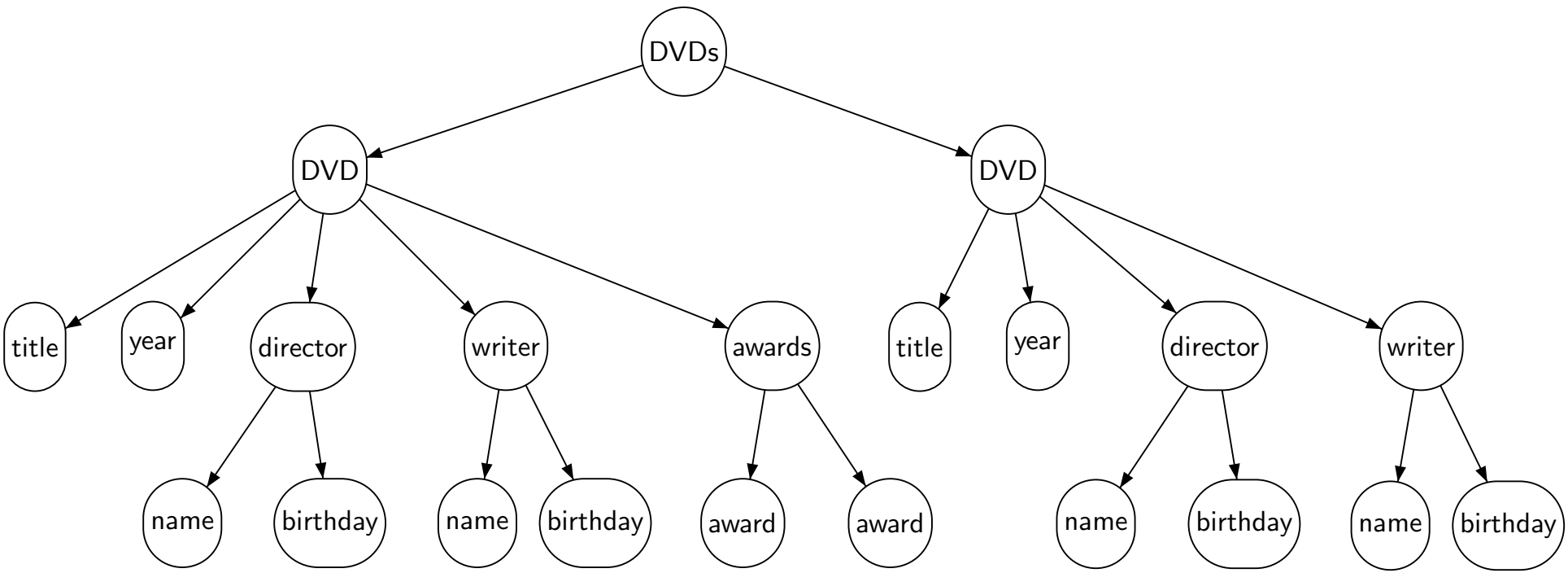


How to turn XPath into an n-ary query language?

- 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))

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

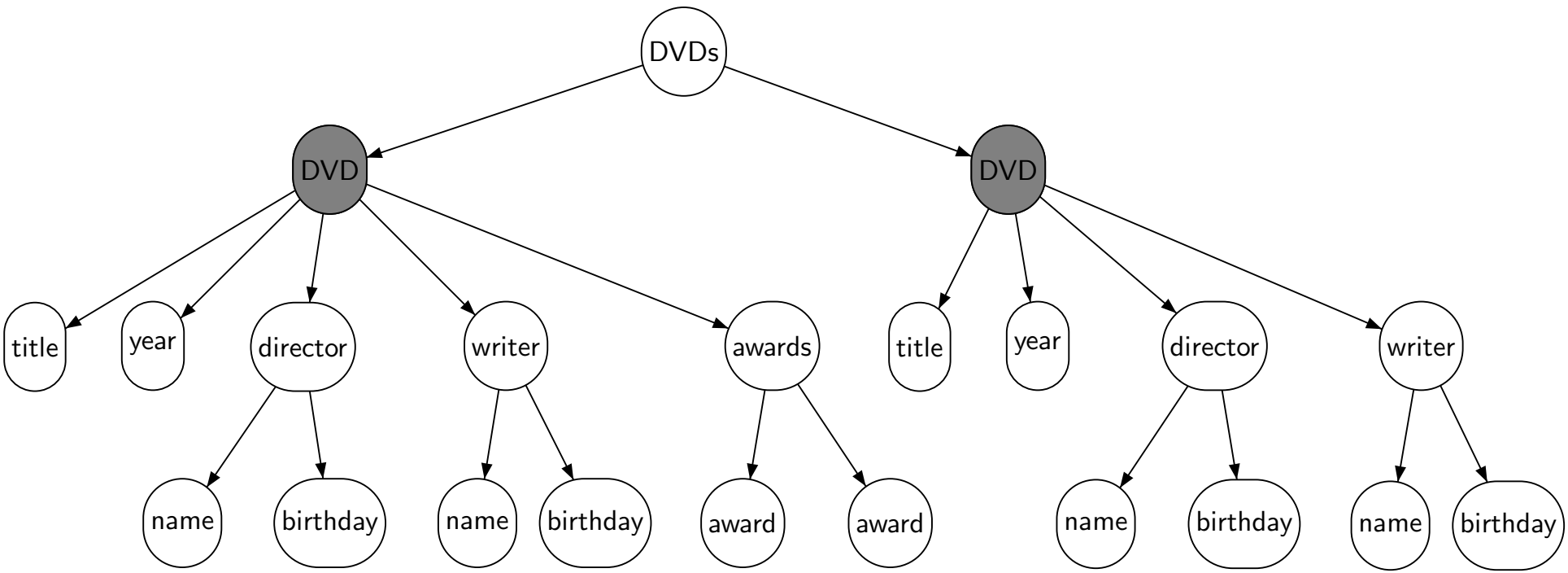


How to turn XPath into an n-ary query language?

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

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

$\phi(x, y, z) = \mathsf{ch} :: DVD$

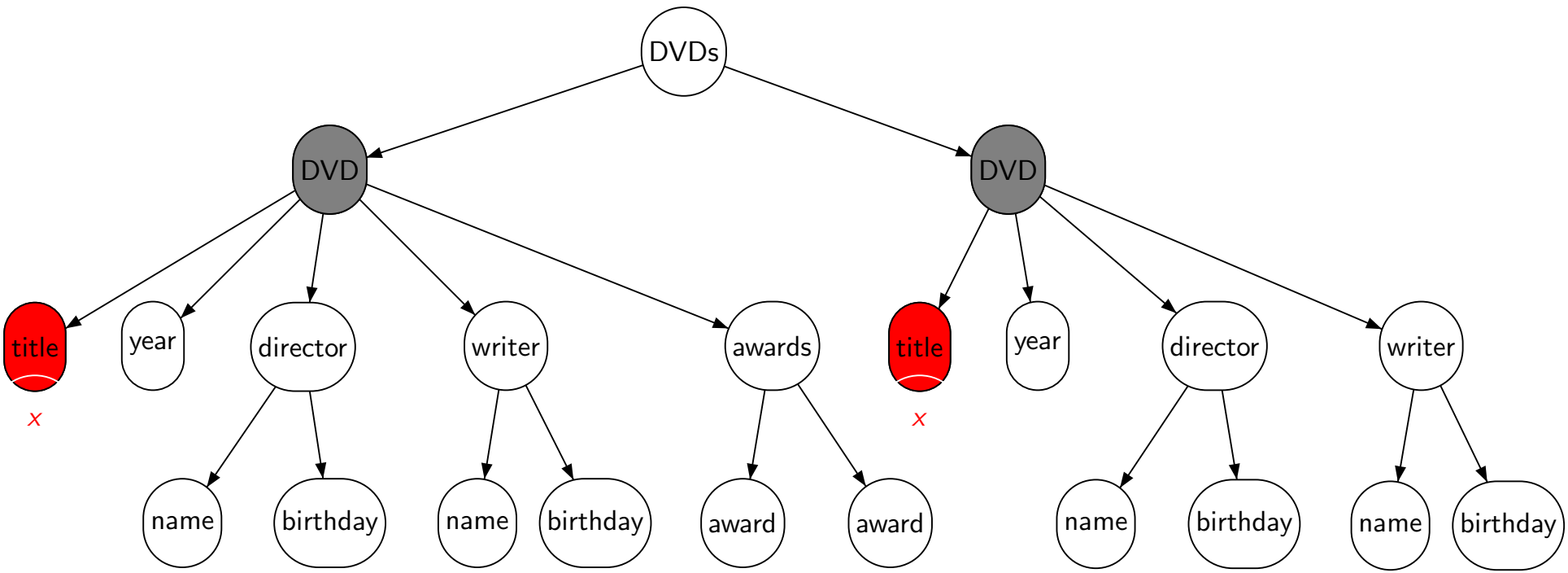


How to turn XPath into an n-ary query language?

- 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))

$\phi(x, y, z) = \mathsf{ch} :: DVD[\mathsf{ch} :: title/x]$

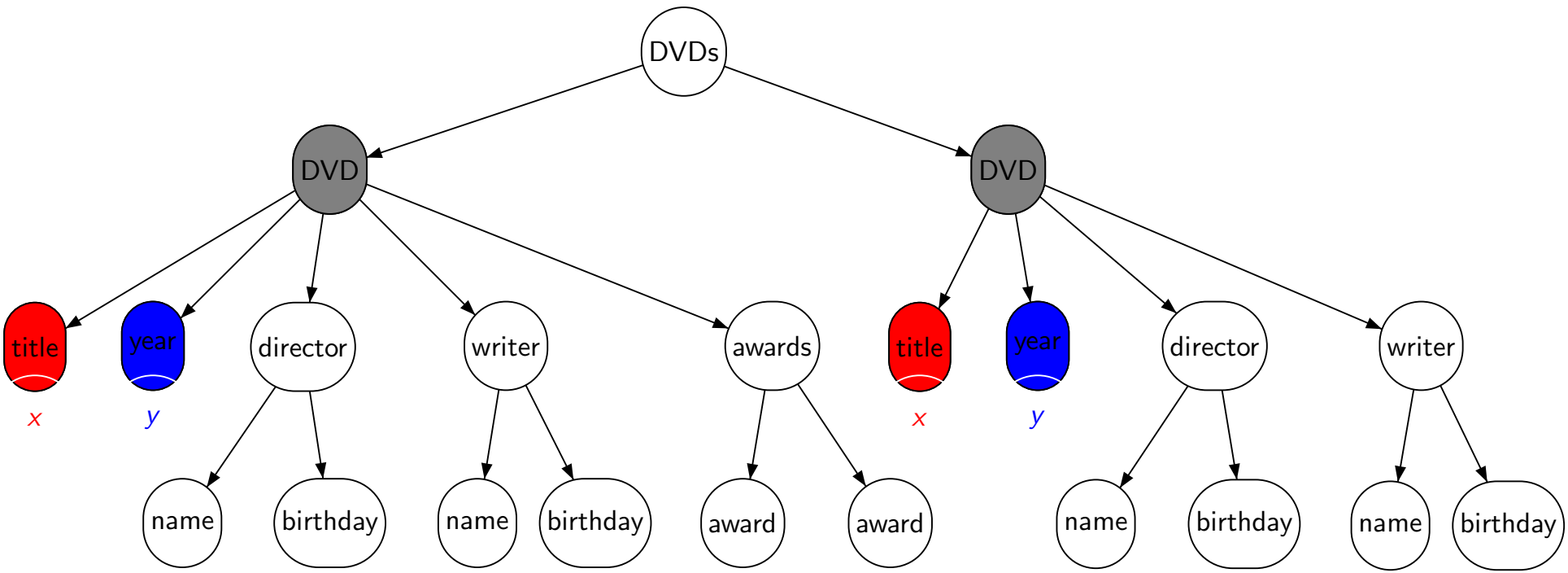


How to turn XPath into an n-ary query language?

- 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))

$\phi(x, y, z) = \mathrm{ch} :: DVD[\mathrm{ch} :: title/x][\mathrm{ch} :: year/y]$

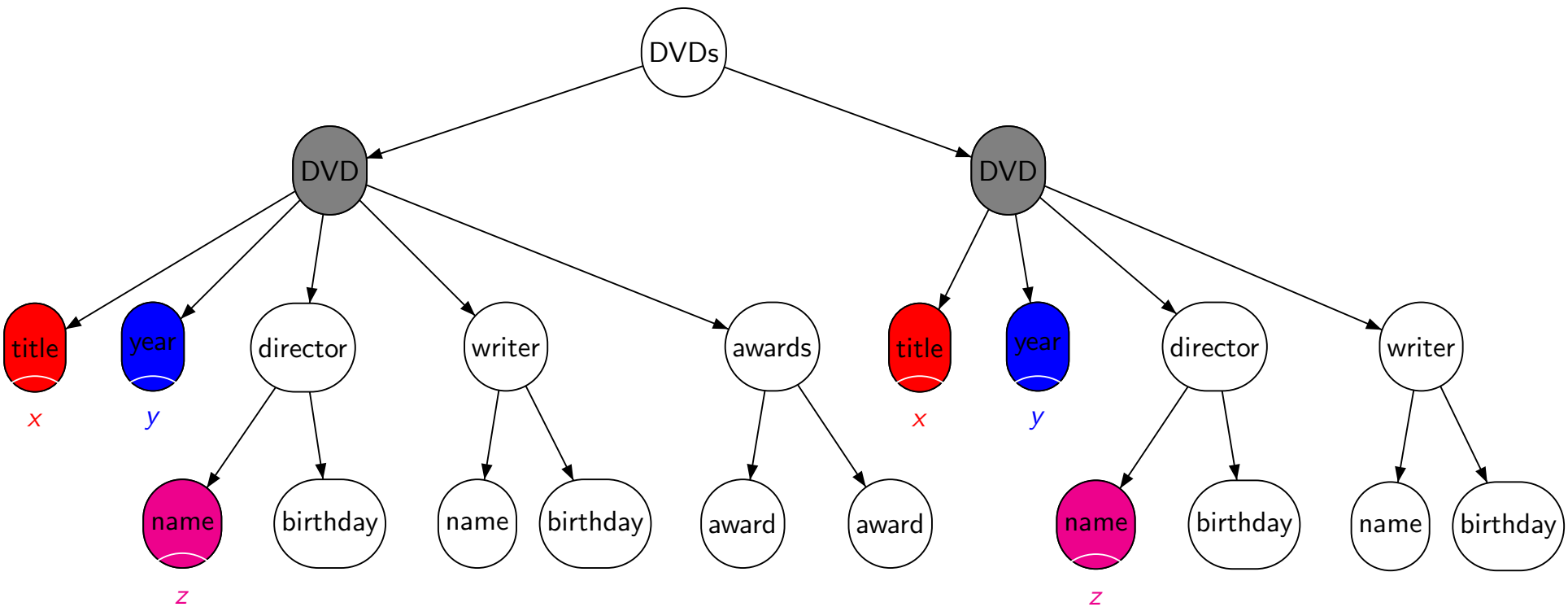


How to turn XPath into an n-ary query language?

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

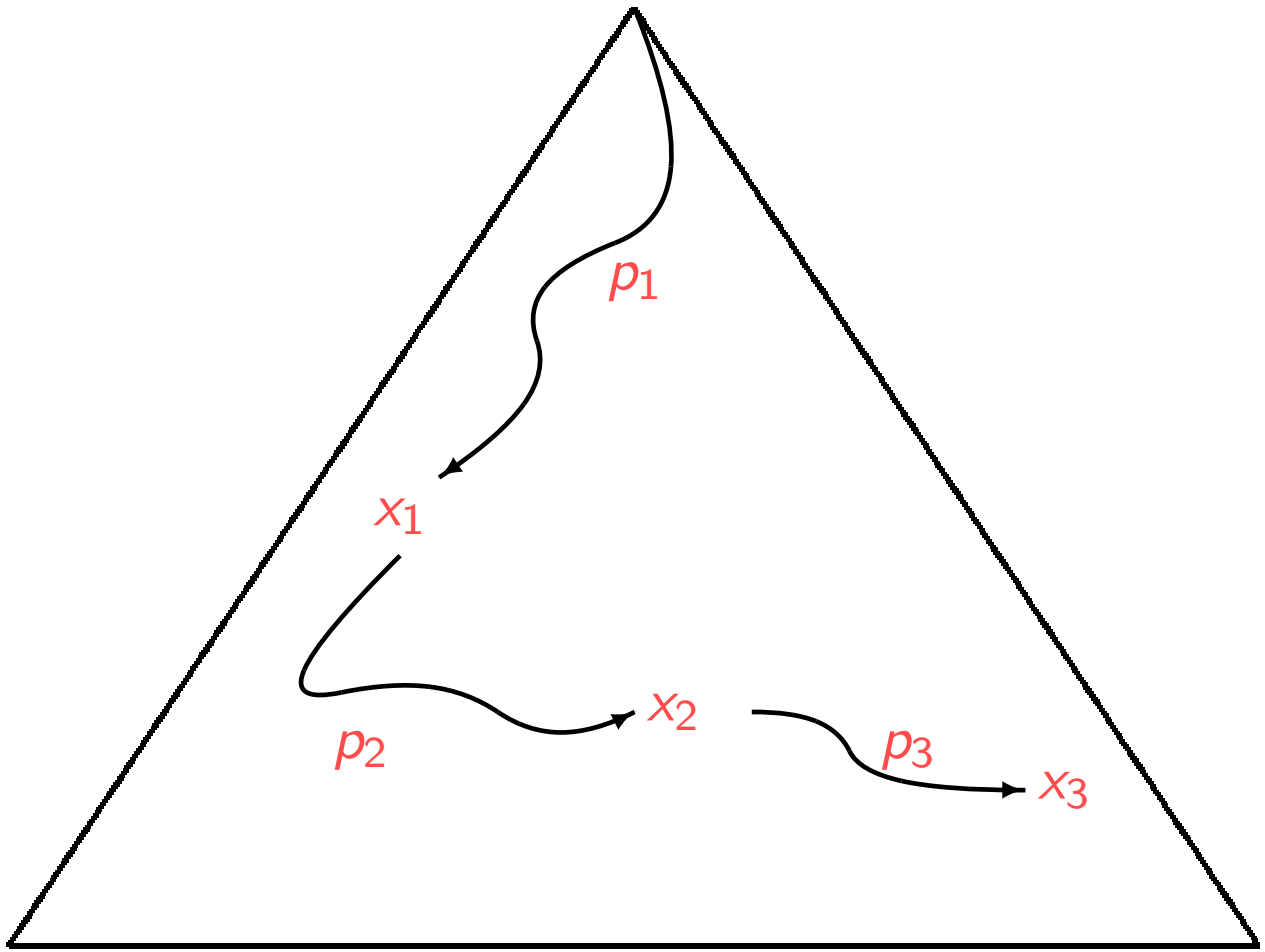
Example (All triples (title,year,director name))

$\phi(x, y, z) = \mathrm{ch} :: DVD[\mathrm{ch} :: title/x][\mathrm{ch} :: year/y][\mathrm{ch} :: director/\mathrm{ch} :: name/z]$



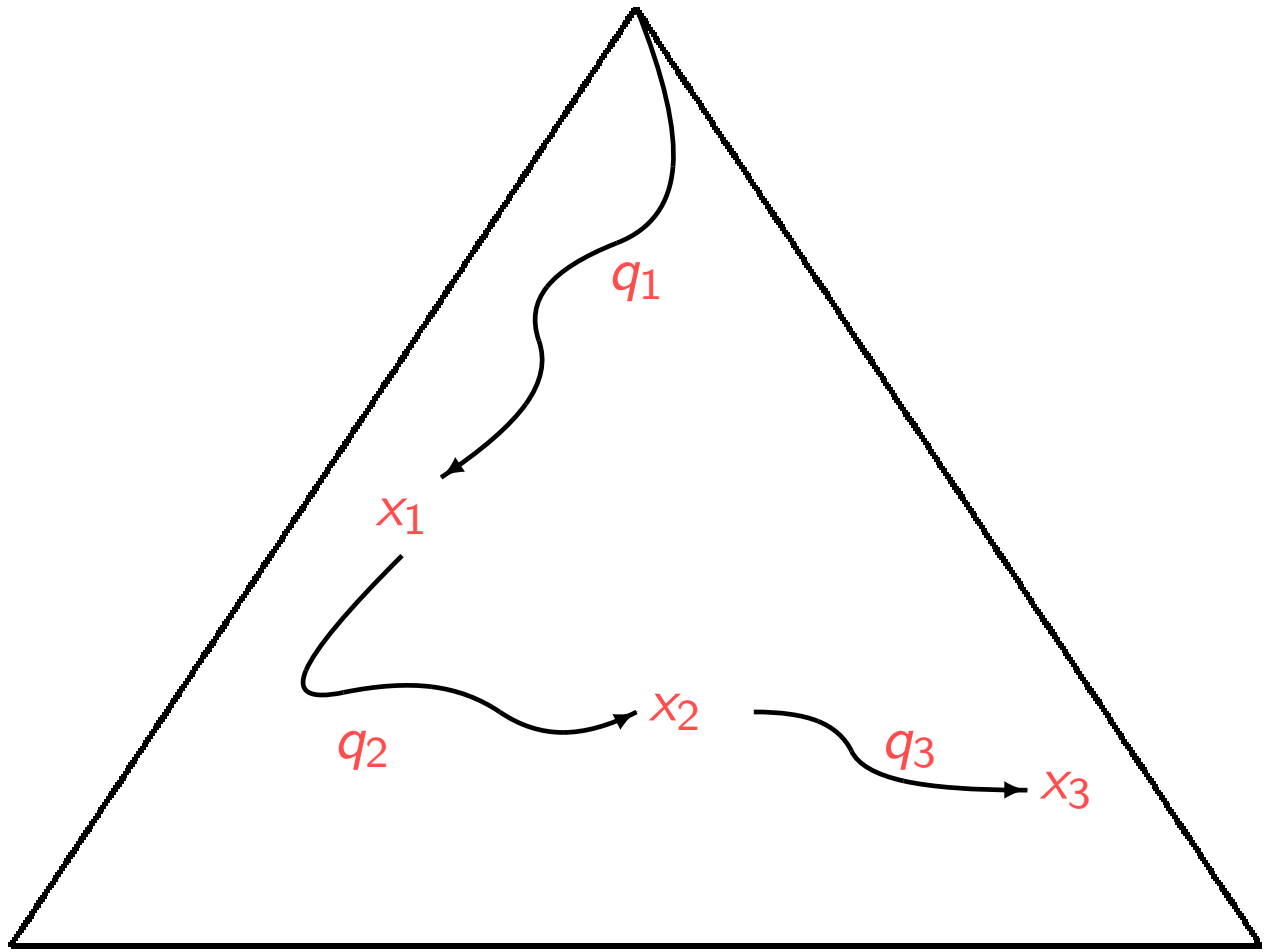
Idea of the composition language

- 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)

$$\underbrace{\mathsf{ch}^*::\mathit{title}}_{\downarrow\ q_1} \;/\; x \;/\; \underbrace{\mathsf{ns}::\mathit{year}}_{\downarrow\ q_2} \;/\; y \;/\; \underbrace{\mathsf{ns}::\mathit{director}/\mathsf{ch}::\mathit{name}}_{\downarrow\ q_3} \;/\; z$$

$$q_1 \circ x \circ q_2 \circ y \circ q_3 \circ z$$

where $q_1, q_2, q_3 \in \mathit{CoreXPath}$.

The composition language Comp(L)

Syntax of composition formulas Comp(L)

We start from L a binary query language, and Var a set of variables.

$$
\begin{array}{rcll}
\phi & := & q & q \in L \\
 & | & x & \textit{variable} \\
 & | & \phi \circ \phi & \textit{composition} \\
 & | & [\phi] & \textit{test} \\
 & | & \phi \vee \phi & \textit{disjunction}
\end{array}
$$

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

Query Evaluation

Query evaluation problem

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

Query Evaluation

Query evaluation problem

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

Polynomial-time query evaluation

- The number of n-tuples of nodes is exponential in $|t|$ and:

$$|\mathrm{Ans}(\phi, t)| \ << \ |t|^n$$

- one needs **polynomial-time** query evaluation:

$$poly(|t|, |\phi|, |\mathrm{Ans}(\phi, t)|)$$

Query Evaluation Algorithm for $\mathrm{Comp}^{\mathrm{nvs}}(L)$

Non-variable sharing fragment

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

Query Evaluation Algorithm for $\mathrm{Comp}^{\mathrm{nvs}}(L)$

Non-variable sharing fragment

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

Theorem

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

Query Evaluation Algorithm for $\mathrm{Comp}^{\mathrm{nvs}}(L)$

Non-variable sharing fragment

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

Theorem

Query evaluation for $\mathrm{Comp}^{\mathrm{nvs}}(L)$ 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 $|\mathrm{Ans}(\phi, t)|$
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

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

Theorem

$$\begin{aligned} FO_n &= Comp^{nvs}(FO_2) \\ MSO_n &= Comp^{nvs}(MSO_2) \end{aligned}$$

Remark

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

n-ary XPath Extensions (I)

Conditional XPath (Marx'04)

- extends *CoreXPath* with a "while" operator $(\text{axis} :: l[\text{test}])^+$
- 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
  $p ::= \text{axis} :: l \mid p/p \mid p[test] \mid p \cup p \mid (\text{axis} :: l[test])^+$

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
  $p ::= \text{axis} :: l \mid p/p \mid p[test] \mid p \cup p \mid (\text{axis} :: l[test])^+ \mid x \in \text{Var}$

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
  $p ::= \text{axis} :: l \mid p/p \mid p[test] \mid p \cup p \mid (\text{axis} :: l[test])^+ \mid x \in \text{Var}$
- linear-time back and forth translations into $\text{Comp}^{\text{nvs}}(CXPath)$
- $\longrightarrow$ captures FO_n
- $\longrightarrow$ 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	$\mathrm{compl}(p)$
variables	x
quantification	for x in p_1 return p_2

- captures FO_n modulo linear-time
- *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	$\mathrm{compl}(p)$
variables	x
quantification	for x in p_1 return p_2

- captures FO_n modulo linear-time
- *CoreXPath2.0* formalized by ten Cate and Marx (07)

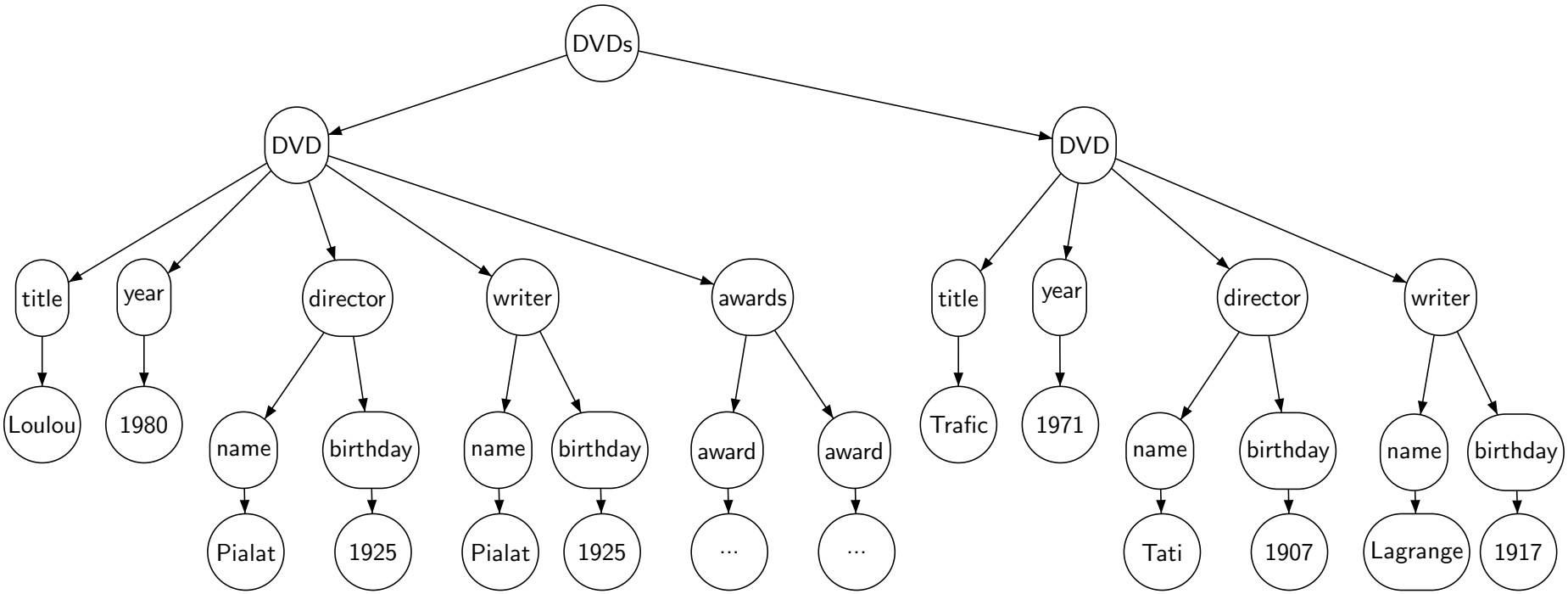
Application of the composition language

- to define a syntactic fragment of *CoreXPath2.0*
- FO_n-expressive
- with query evaluation problem in $O(|p|.|t|^3 + |p|.|t^2|.|\mathrm{Ans}(p,t)|)$

Outline

PART II: The Spatial Logic TQL

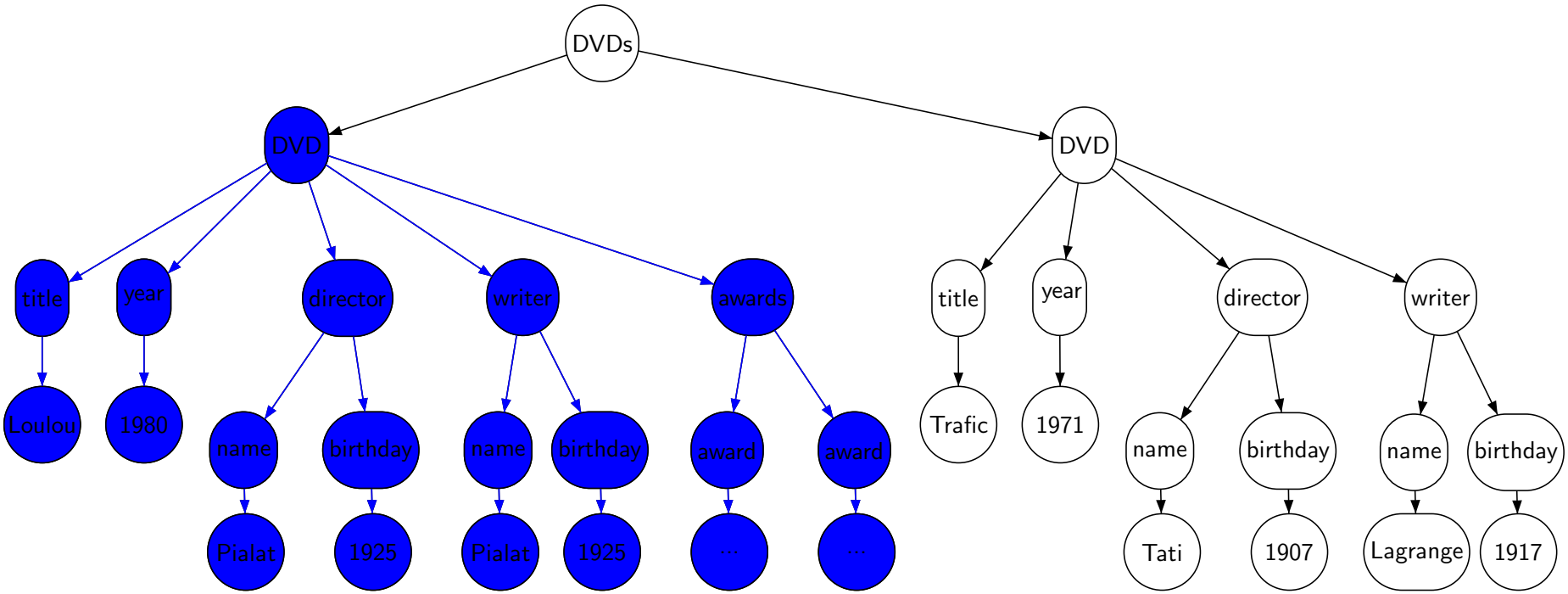
TQL Examples



Example (Check if there is an awarded movie)

$$DVDs[\_ \mid DVD[\_ \mid awards[\_]] \mid \_]$$

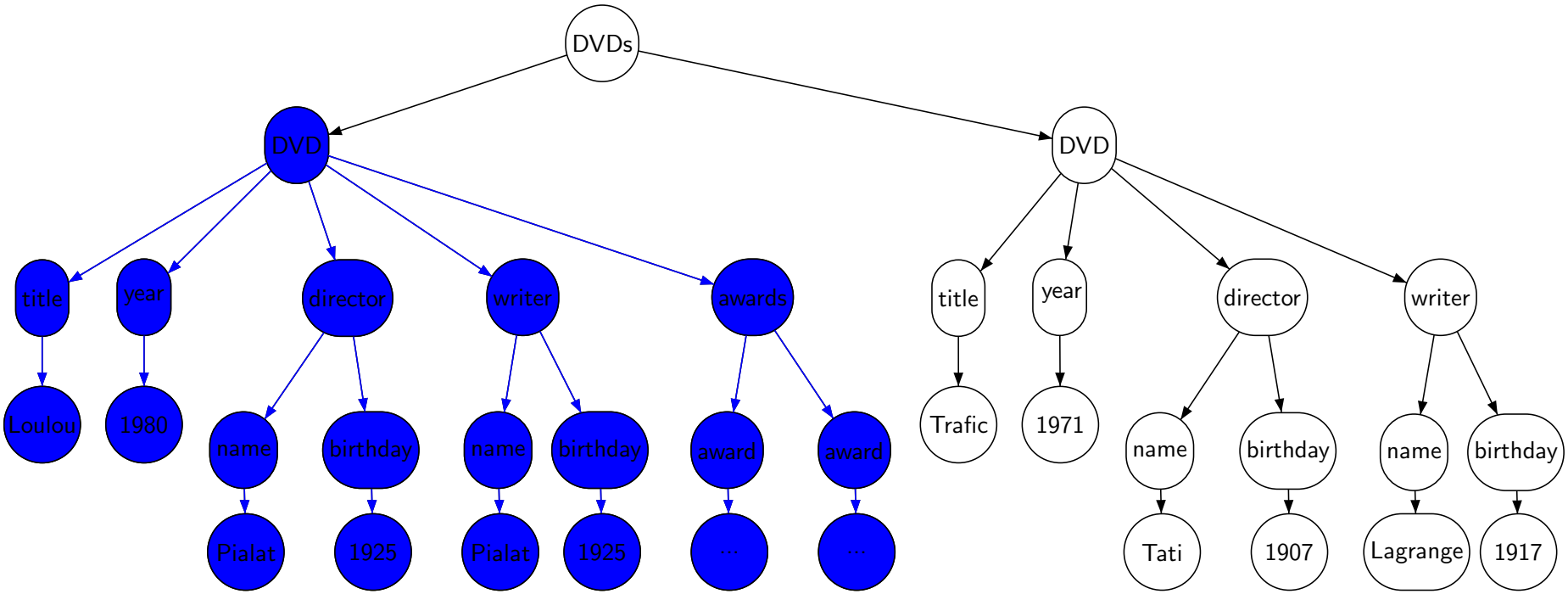
TQL Examples



Example (Check if there is an awarded movie)

$$DVDs[\_ \mid DVD[\_ \mid awards[\_]] \mid \_]$$

TQL Examples



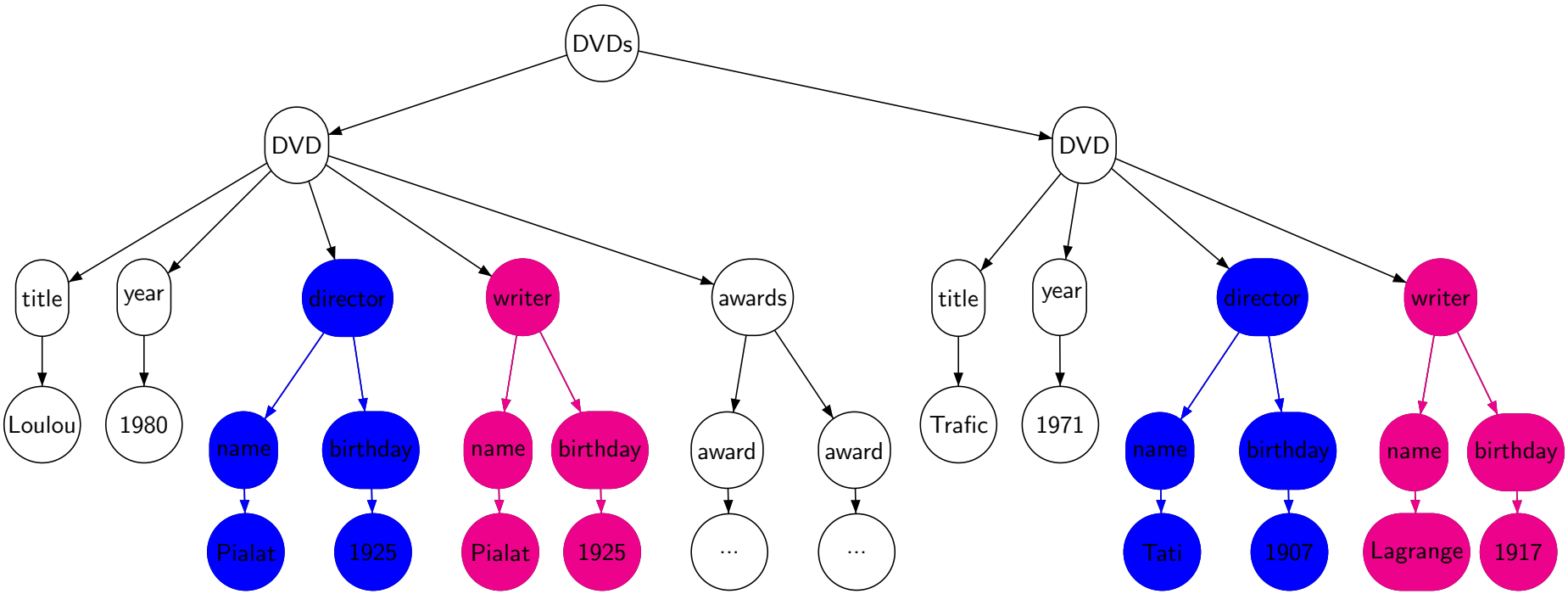
Example (Select all awarded movies)

$$\phi(X) \;=\; DVDs[\_ \mid X \wedge DVD[\_ \mid awards[\_]] \mid \_]$$

$\downarrow$

tree variable

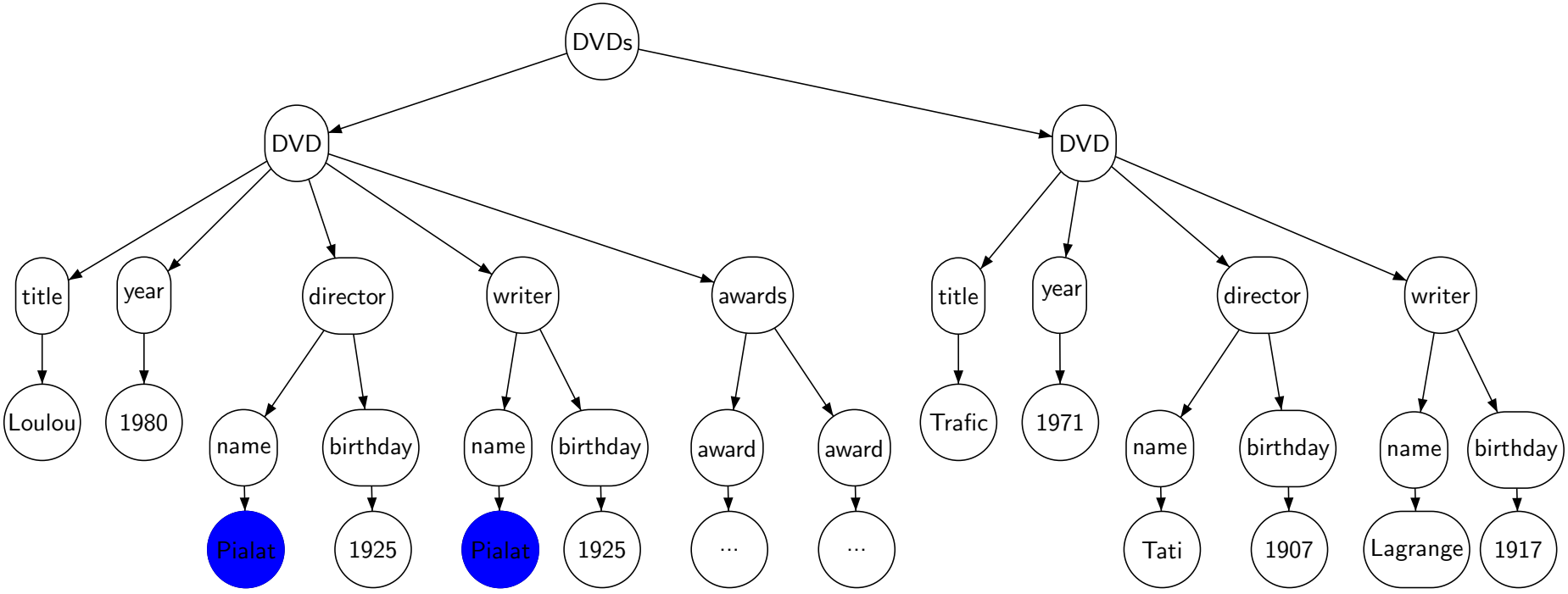
TQL Examples



Example (Select all pairs of (*director*, *writer*))

$$\phi(X, Y) \;=\; DVDs[\_ \mid DVD[\_|year[\_]|X|Y|\_]|\_]$$

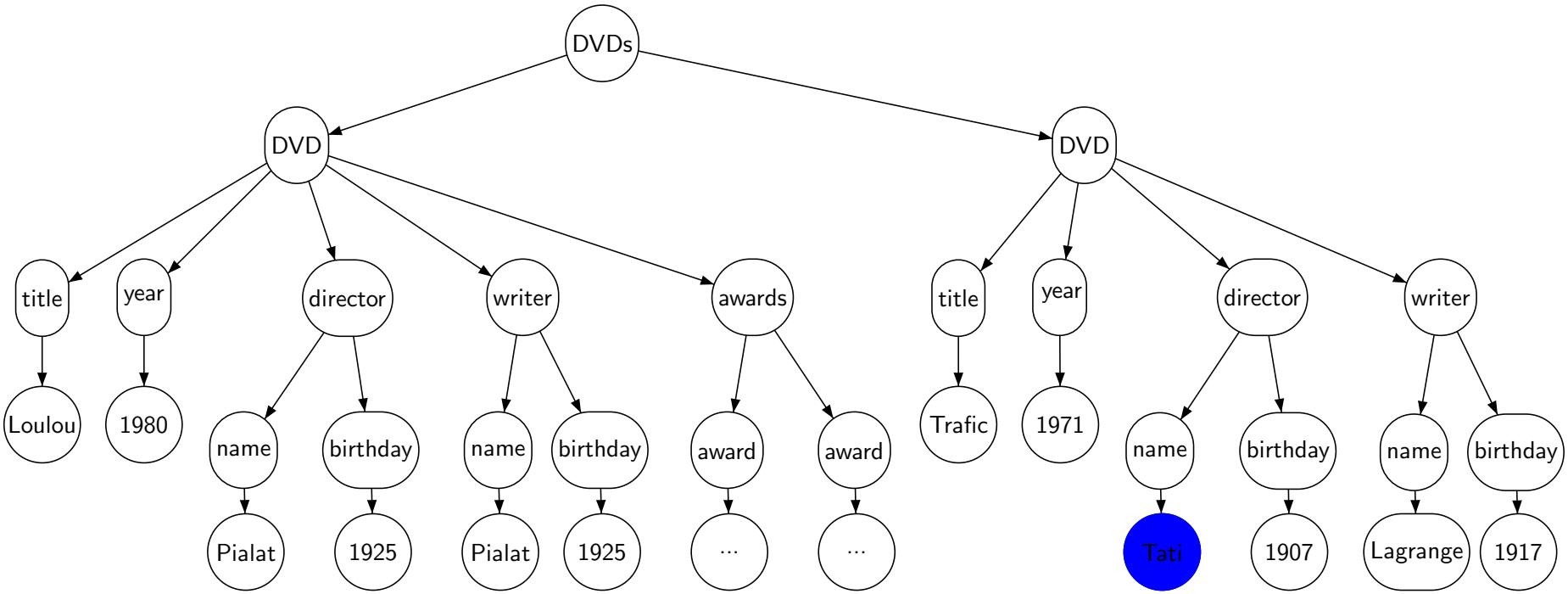
TQL Examples



Example (Select all names of persons who are both director and writer)

$$\phi(X) \;=\; DVDs[\_ \mid DVD[\_|director[name[X]|\_]|writer[name[X]|\_]|\_]|\_]$$

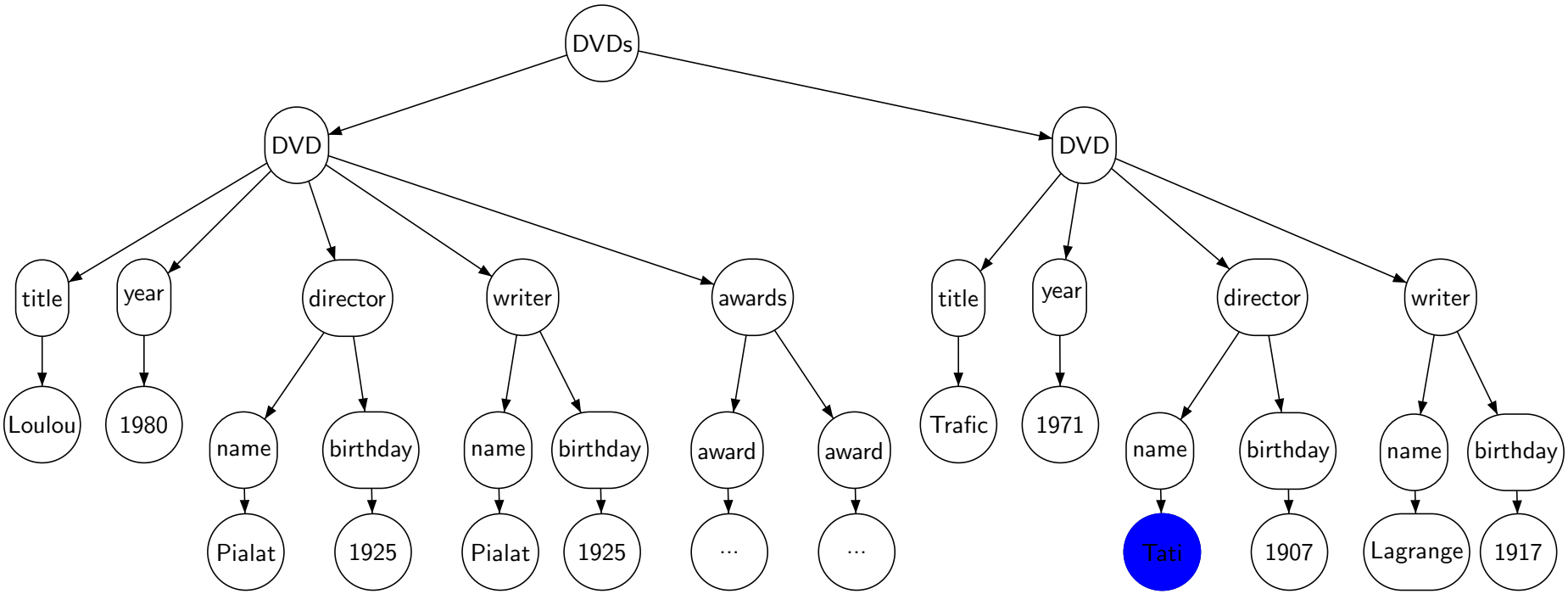
TQL Examples



Example (Select all director names who is not a writer)

$$\phi(X) \;=\; DVDs[\_ \mid DVD[\_|director[name[X]|\_]|writer[name[\neg X]|\_]|\_]|\_]$$

TQL Examples



Tree (dis)equality tests

- main difficulty of TQL satisfiability problem
- incomparable to FO fragments with data-value comparison (BojanczykDMSS06)

Hedge Algebra H_Λ

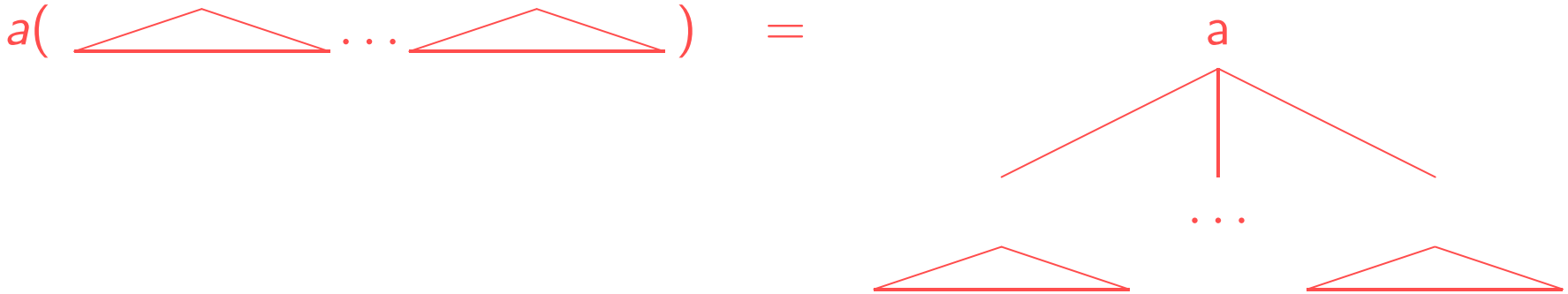
- Λ: countable set of labels
- hedge = ordered sequence of unranked trees

Hedge Algebra H_Λ

- Λ: countable set of labels
- hedge = ordered sequence of unranked trees
- constant 0: empty hedge

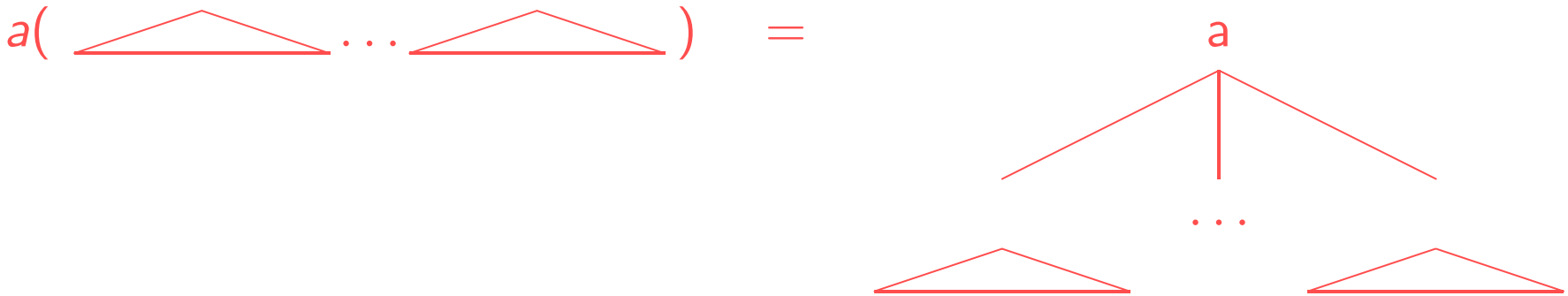
Hedge Algebra H_Λ

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



Hedge Algebra H_Λ

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



- binary symbol $|$

TQL: Syntax and Semantics

empty hedge	0		
location	$\alpha[\phi]$	$\alpha \subseteq \Lambda$	(co)finite
concatenation	$\phi \vert \phi'$		

TQL: Syntax and Semantics

empty hedge	0		
location	$\alpha[\phi]$	$\alpha \subseteq \Lambda$	(co)finite
concatenation	$\phi \vert \phi'$		
truth	$-$		
conjunction	$\phi \wedge \phi'$		
negation	$\neg\phi$		

TQL: Syntax and Semantics

empty hedge	0		
location	$\alpha[\phi]$	$\alpha \subseteq \Lambda$	(co)finite
concatenation	$\phi \vert \phi'$		
truth	$\_$		
conjunction	$\phi \wedge \phi'$		
negation	$\neg\phi$		
tree variable	X		
recursion variable	ξ		
least fixpoint	$\mu\xi.\phi$		

TQL: Syntax and Semantics

- semantics modulo $\rho : \mathit{TreeVars} \to T_\Lambda$ and $\delta : \mathit{RecVars} \to 2^{H_\Lambda}$
- set-based semantics: $[\![.]\!]_{\rho,\delta} \subseteq H_\Lambda$

empty hedge	0		
location	$\alpha[\phi]$	$\alpha \subseteq \Lambda$	(co)finite
concatenation	$\phi \mid \phi'$		
truth	$\_$		
conjunction	$\phi \wedge \phi'$		
negation	$\neg\phi$		
tree variable	X		
recursion variable	ξ		
least fixpoint	$\mu\xi.\phi$		

TQL: Syntax and Semantics

- semantics modulo $\rho : \mathit{TreeVars} \to T_\Lambda$ and $\delta : \mathit{RecVars} \to 2^{H_\Lambda}$
- set-based semantics: $[\![.]\!]_{\rho,\delta} \subseteq H_\Lambda$

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

truth	$\_$
conjunction	$\phi \wedge \phi'$
negation	$\neg\phi$

tree variable	X
recursion variable	ξ
least fixpoint	$\mu\xi.\phi$

TQL: Syntax and Semantics

- semantics modulo $\rho : \mathit{TreeVars} \to T_\Lambda$ and $\delta : \mathit{RecVars} \to 2^{H_\Lambda}$
- set-based semantics: $[\![.]\!]_{\rho,\delta} \subseteq H_\Lambda$

empty hedge	$[\![0]\!]_{\rho,\delta}$	$=$	$\{0\}$
location	$[\![\alpha[\phi]]\!]_{\rho,\delta}$	$=$	$\{a(h) \mid h \in [\![\phi]\!]_{\rho,\delta}, a \in \alpha\}$
concatenation	$[\![\phi \vert \phi']\!]_{\rho,\delta}$	$=$	$\{h \vert h' \mid h \in [\![\phi]\!]_{\rho,\delta}, h' \in [\![\phi']\!]_{\rho,\delta}\}$
truth	$[\![\_]\!]$	$=$	H_Λ
conjunction	$[\![\phi \wedge \phi']\!]$	$=$	$[\![\phi]\!] \cap [\![\phi']\!]$
negation	$[\![\neg\phi]\!]$	$=$	$H_\Lambda \setminus [\![\phi]\!]$
tree variable	X		
recursion variable	ξ		
least fixpoint	$\mu\xi.\phi$		

TQL: Syntax and Semantics

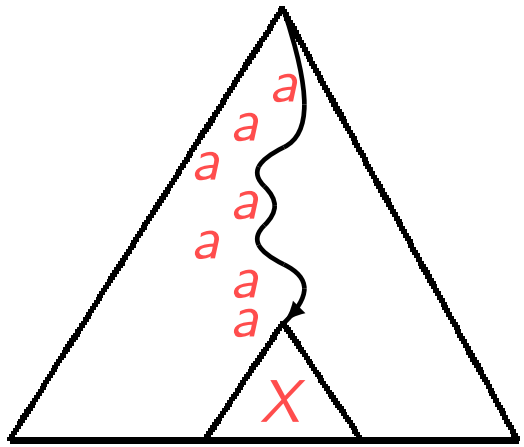
- semantics modulo $\rho : \mathit{TreeVars} \to T_\Lambda$ and $\delta : \mathit{RecVars} \to 2^{H_\Lambda}$
- set-based semantics: $[\![.]\!]_{\rho,\delta} \subseteq H_\Lambda$

empty hedge	$[\![0]\!]_{\rho,\delta}$	$=$	$\{0\}$
location	$[\![\alpha[\phi]]\!]_{\rho,\delta}$	$=$	$\{a(h) \mid h \in [\![\phi]\!]_{\rho,\delta}, a \in \alpha\}$
concatenation	$[\![\phi \vert \phi']\!]_{\rho,\delta}$	$=$	$\{h \vert h' \mid h \in [\![\phi]\!]_{\rho,\delta}, h' \in [\![\phi']\!]_{\rho,\delta}\}$
truth	$[\![\_]\!]$	$=$	H_Λ
conjunction	$[\![\phi \wedge \phi']\!]$	$=$	$[\![\phi]\!] \cap [\![\phi']\!]$
negation	$[\![\neg\phi]\!]$	$=$	$H_\Lambda \setminus [\![\phi]\!]$
tree variable	$[\![X]\!]_{\rho,\delta}$	$=$	$\{\rho(X)\}$
recursion variable	$[\![\xi]\!]_{\rho,\delta}$	$=$	$\delta(\xi)$
least fixpoint	$[\![\mu\xi.\phi]\!]_{\rho,\delta}$	$=$	$\bigcap\{S \subseteq H_\Lambda \mid [\![\phi]\!]_{\rho,\delta[\xi \mapsto S]} \subseteq S\}$

Examples with fixpoint

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

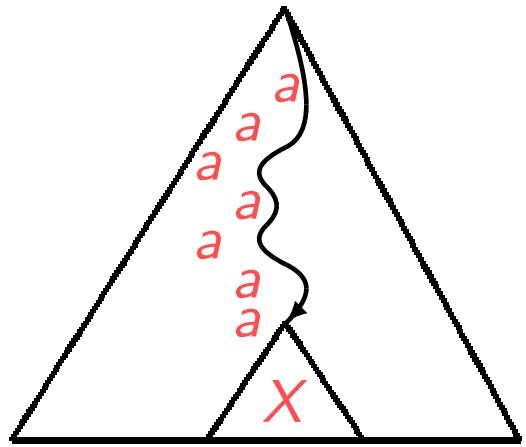
$$\phi(X) \;=\; \mu\xi.(\; a[\_|\xi|\_] \;\vee X)$$



Examples with fixpoint

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

$$\phi(X) \;=\; \mu\xi.(\; a[\_|\xi|\_] \;\vee X)$$



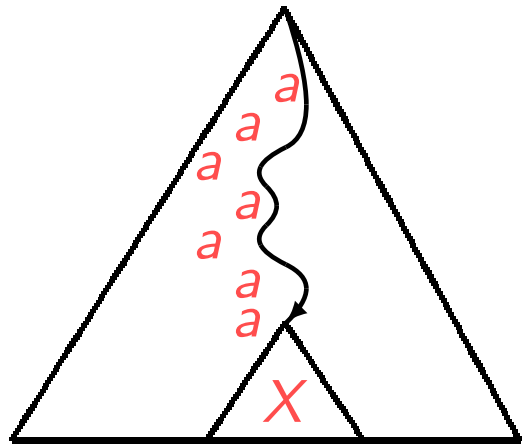
Example ($a^n b^n$)

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

Examples with fixpoint

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

$$\phi(X) \;=\; \mu\xi.(\; a[\_|\xi|\_] \;\vee X)$$



Example ($a^n b^n$)

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

- vertical recursion $\longrightarrow$ regular tree languages
- horizontal recursion $\longrightarrow$ context-free word languages

A Decidable Fragment: Bounded TQL

Satisfiability problem

Input: TQL formula ϕ **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 $\alpha[.]$

$$\mu\xi.(\ a[\_|\xi|\_]\ \vee X) \quad \rightarrow \quad \textbf{guarded}$$
$$\mu\xi.(a[0]|\xi|b[0]\ \vee\ 0) \quad \rightarrow \quad \textbf{not guarded}$$

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** ϕ^* 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** ϕ^* for horizontal recursion

$$[\![\phi^*]\!]_\rho \;=\; 0 \;\cup\; \bigcup_{i>0} \underbrace{[\![\phi]\!]_\rho | \cdots | [\![\phi]\!]_\rho}_{i \text{ times}}$$

A Decidable Fragment: Bounded TQL

Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

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

A Decidable Fragment: Bounded TQL

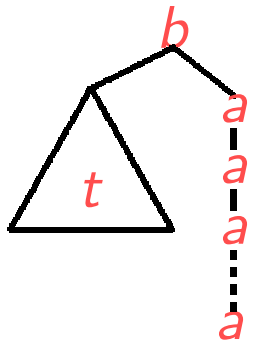
Proposition

Satisfiability of TQL formulas is undecidable.

Bounded TQL

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

$b[X \mid \mu\xi.(\neg X \wedge a[\xi] \ \vee \ 0)] \rightarrow$ **not bounded**



$(\neg X)^* \mid X \mid (\neg X)^* \rightarrow$ **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** ϕ^* 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** ϕ^* for horizontal recursion
- restriction negative variables: only a **bounded number of disequality tests** along the paths
- no **negative** occurences of Kleene star and $|$

Expressiveness and Satisfiability of Bounded TQL

Theorem

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

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*

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

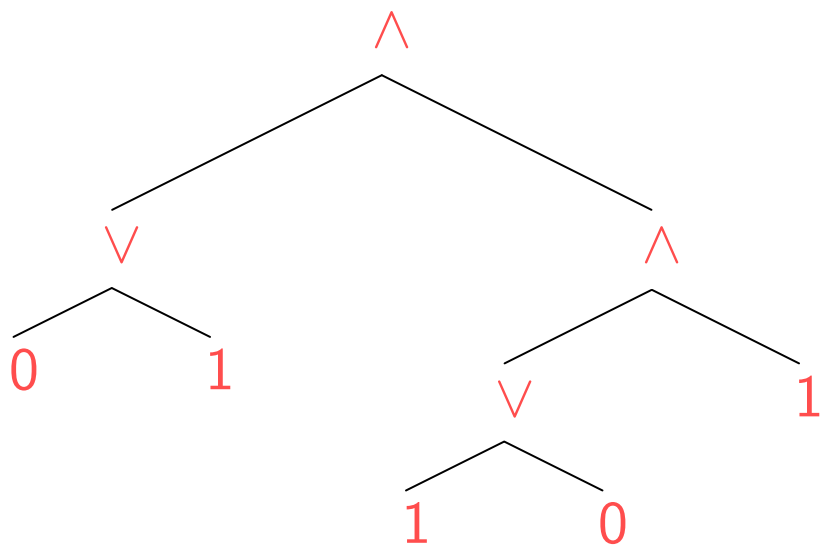
- Σ: finite alphabet
- Q: set of states
- $F \subseteq Q$: set of final states
- Δ: rules of the form $f(q_1, q_2) \longrightarrow q$ or $a \longrightarrow q$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

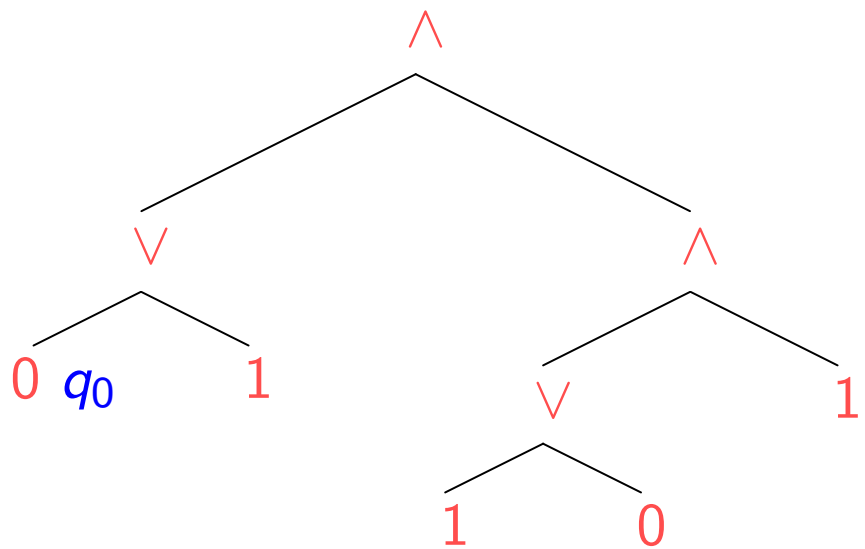
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

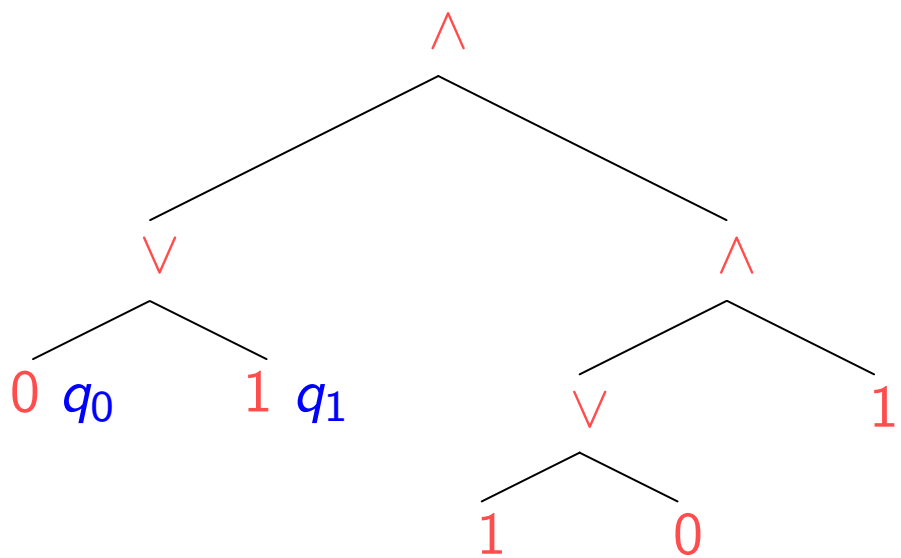
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

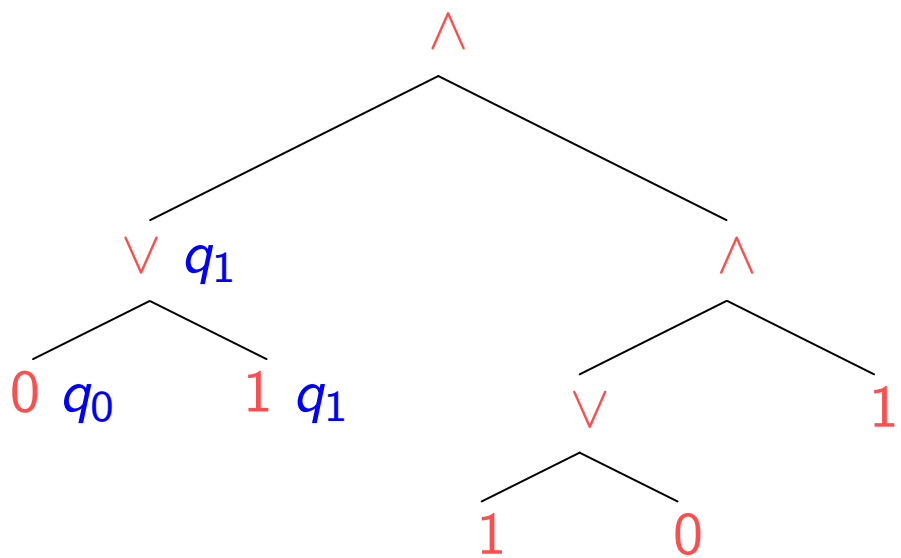
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

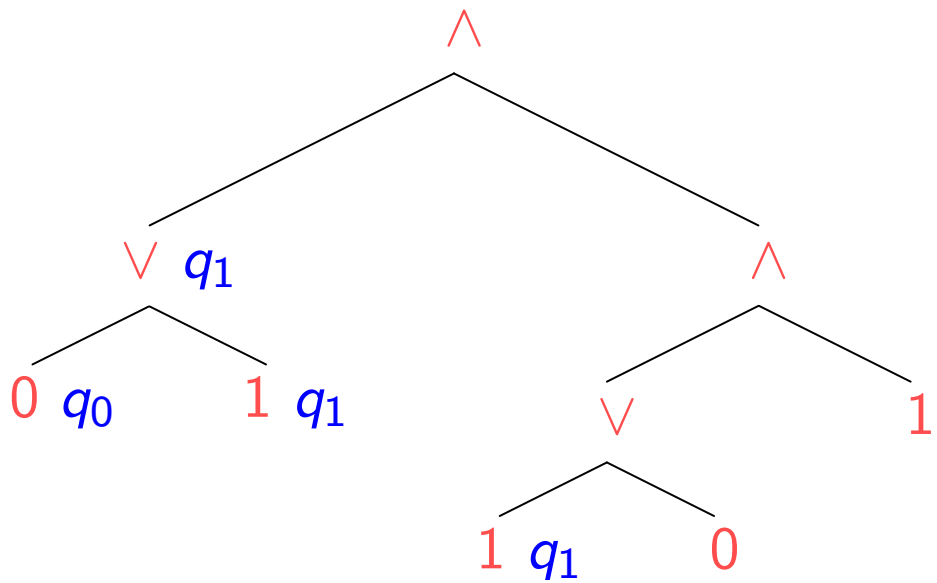
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

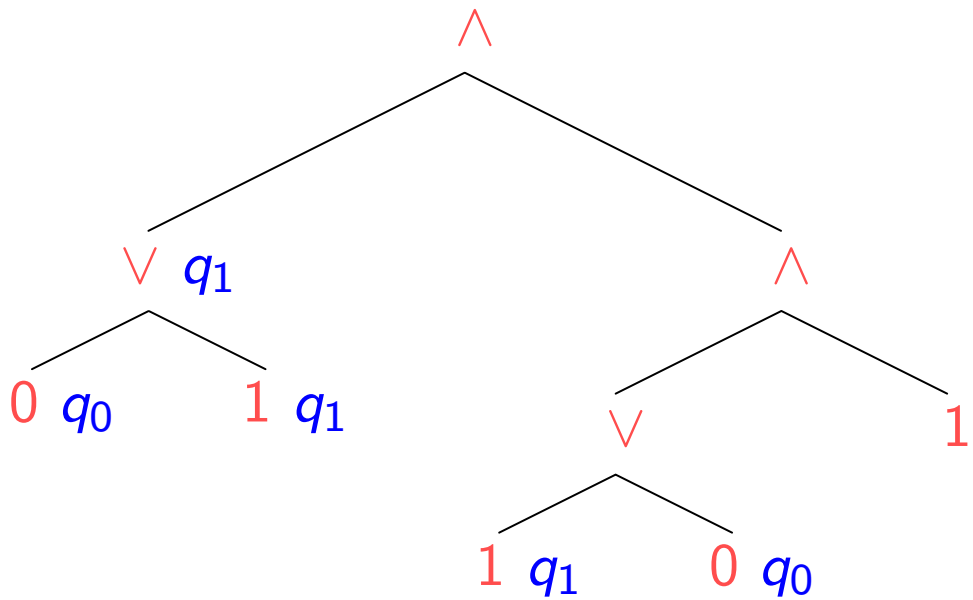
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

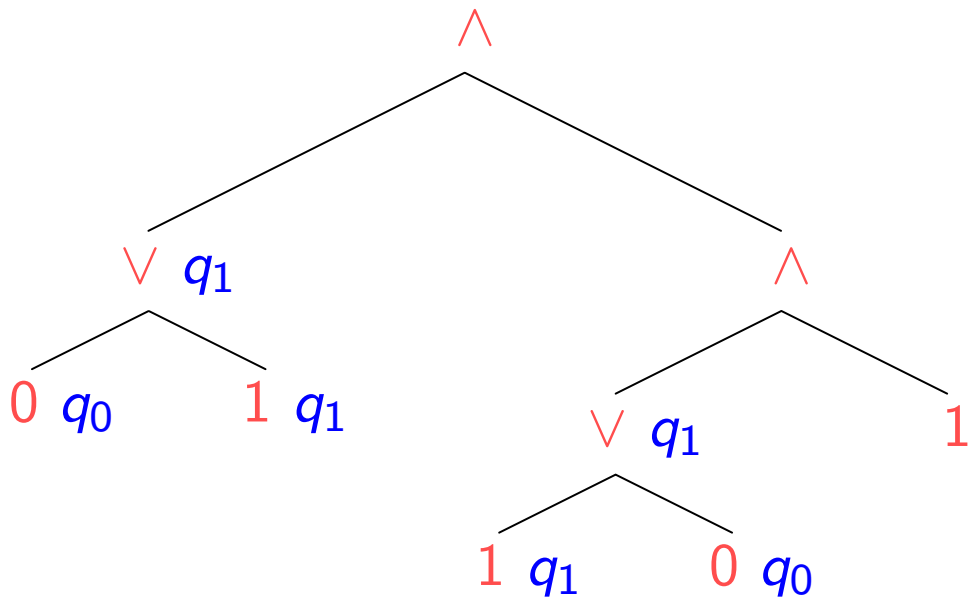
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

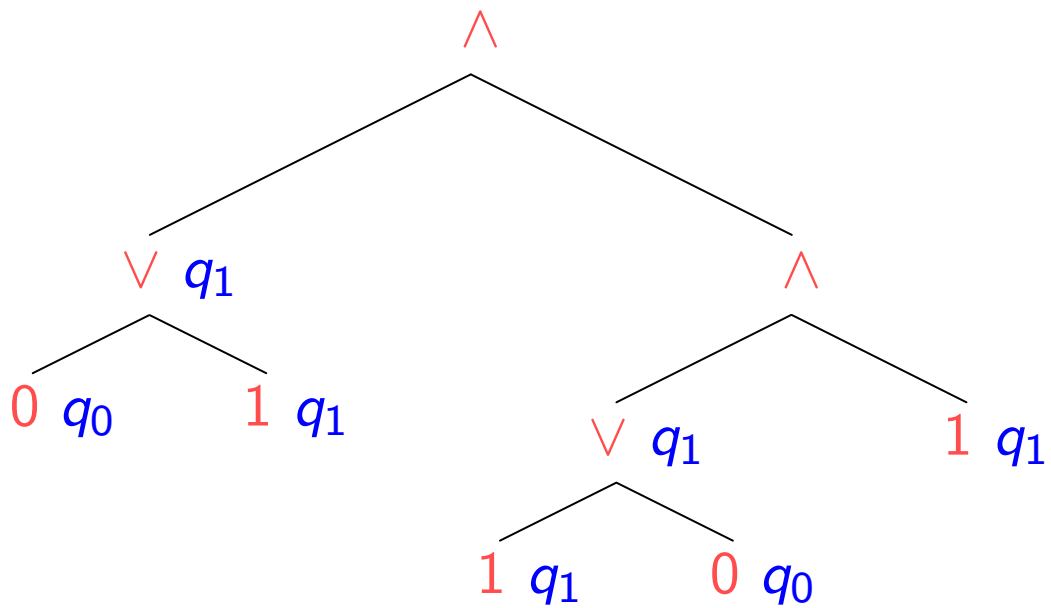
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

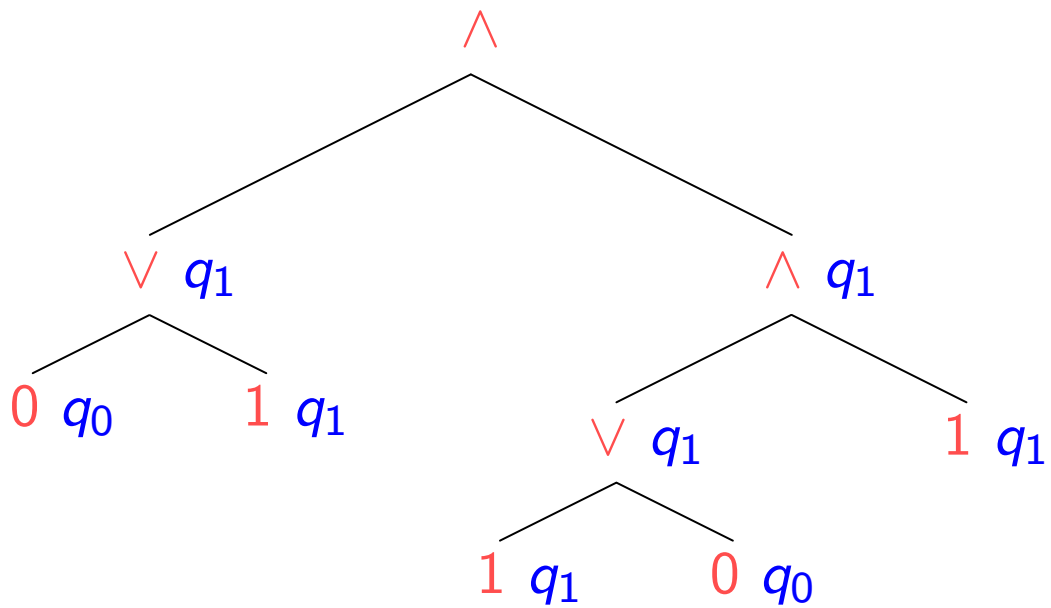
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

final states

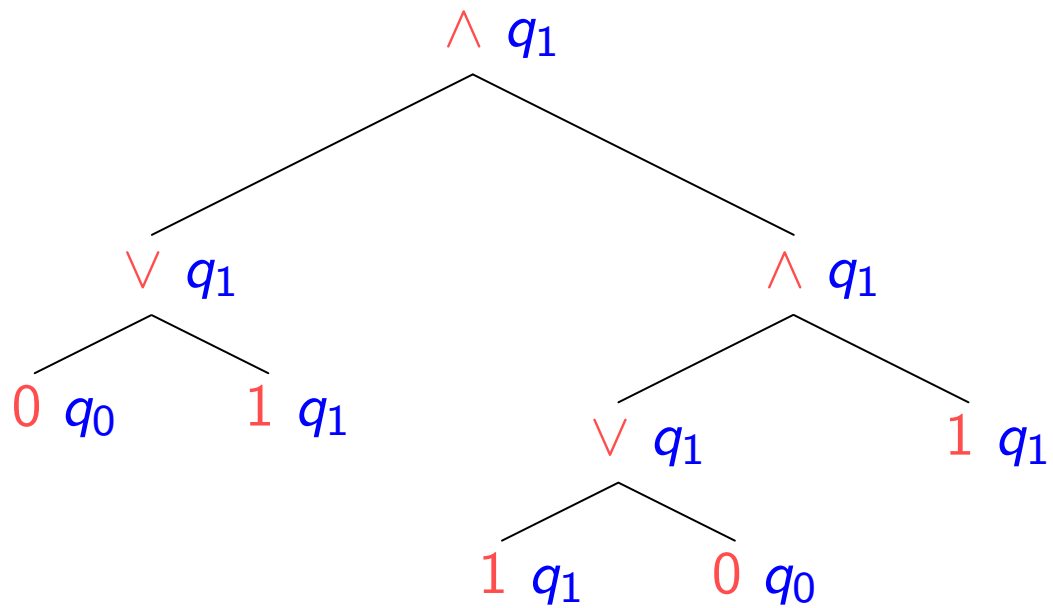
$F = \{q_1\}$

Bottom-up Tree Automata for Binary Trees

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

Example (variable-free satisfiable Boolean formulas)

a tree and a successful run



transitions

$0 \longrightarrow q_0 \quad 1 \longrightarrow q_1$

$\wedge(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \wedge b_2}$

$\vee(q_{b_1}, q_{b_2}) \longrightarrow q_{b_1 \vee b_2}$

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:

Σ alphabet
Q set of states
F set of final states
Δ set of rules

} tree automaton

Tree Automata with Global Equalities and Disequalities

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

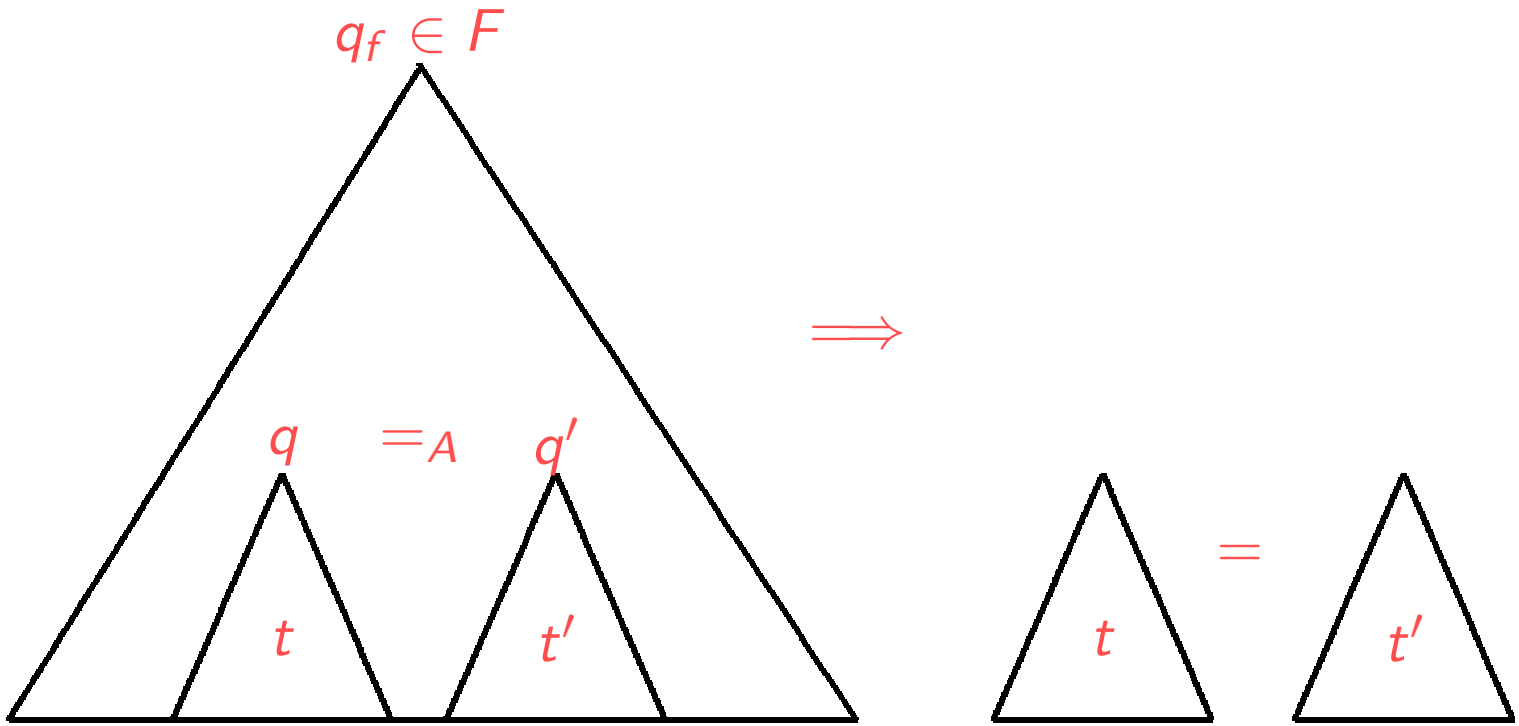
Σ alphabet
Q set of states
F set of final states
Δ set of rules

} tree automaton

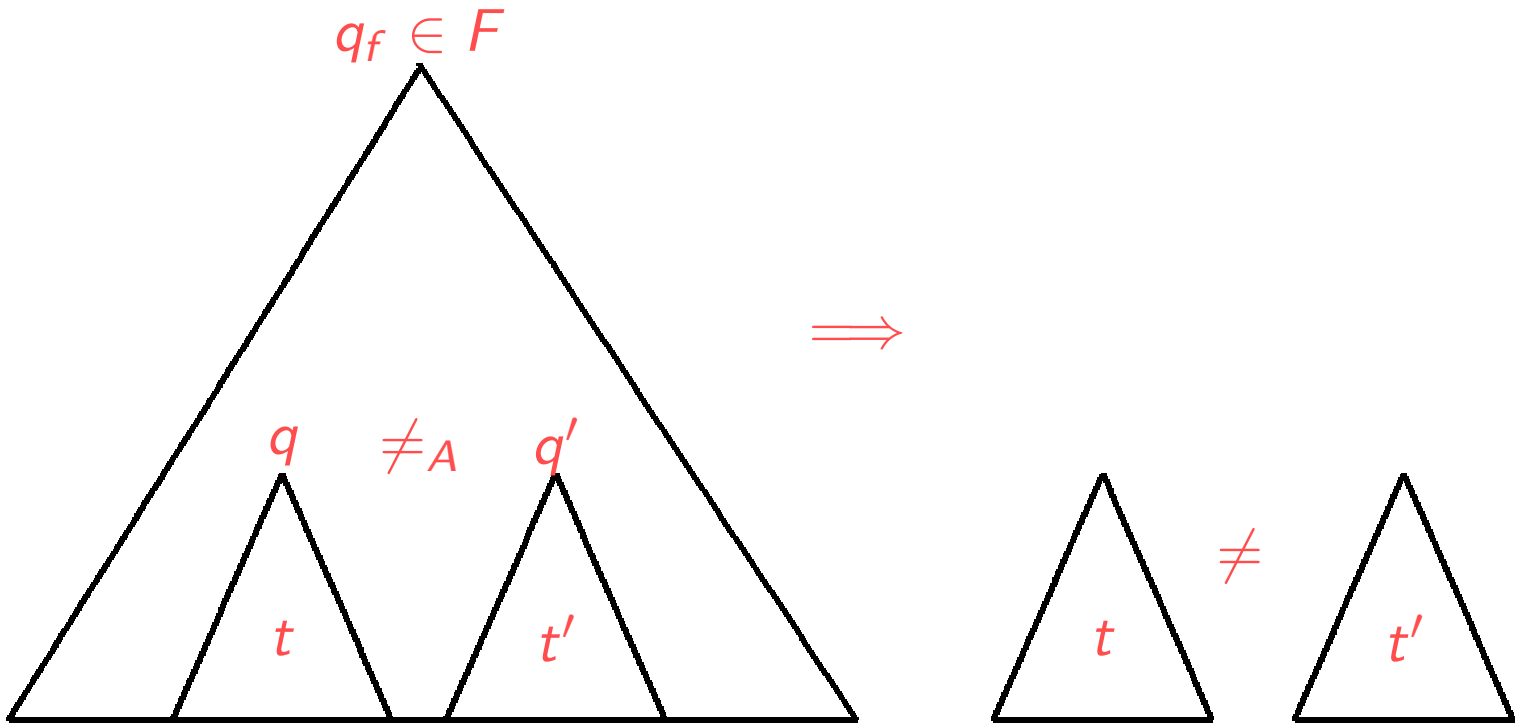
$=_A \subseteq Q^2$ reflexive and symmetric relation on a **subset** of Q

$\neq_A \subseteq Q^2$ irreflexive and symmetric relation

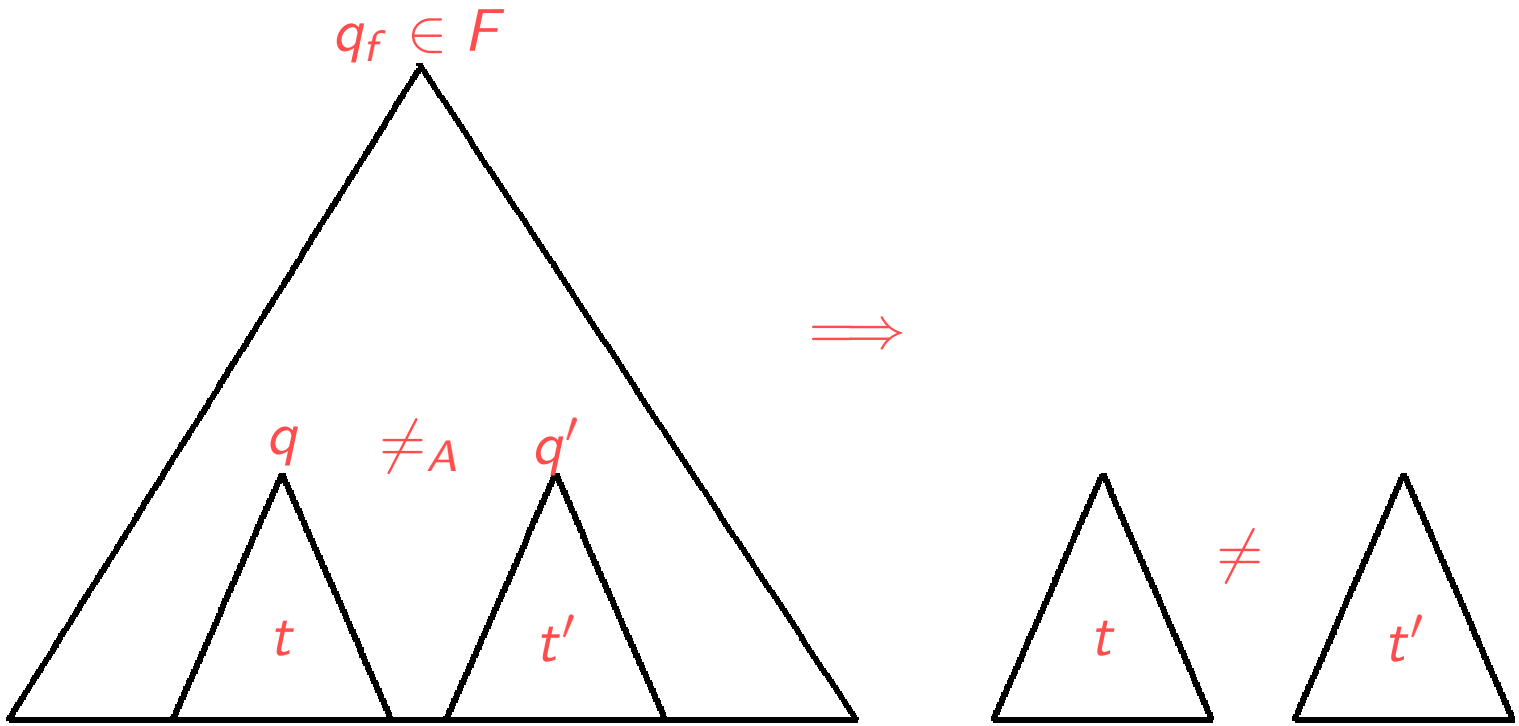
Successful Runs



Successful Runs

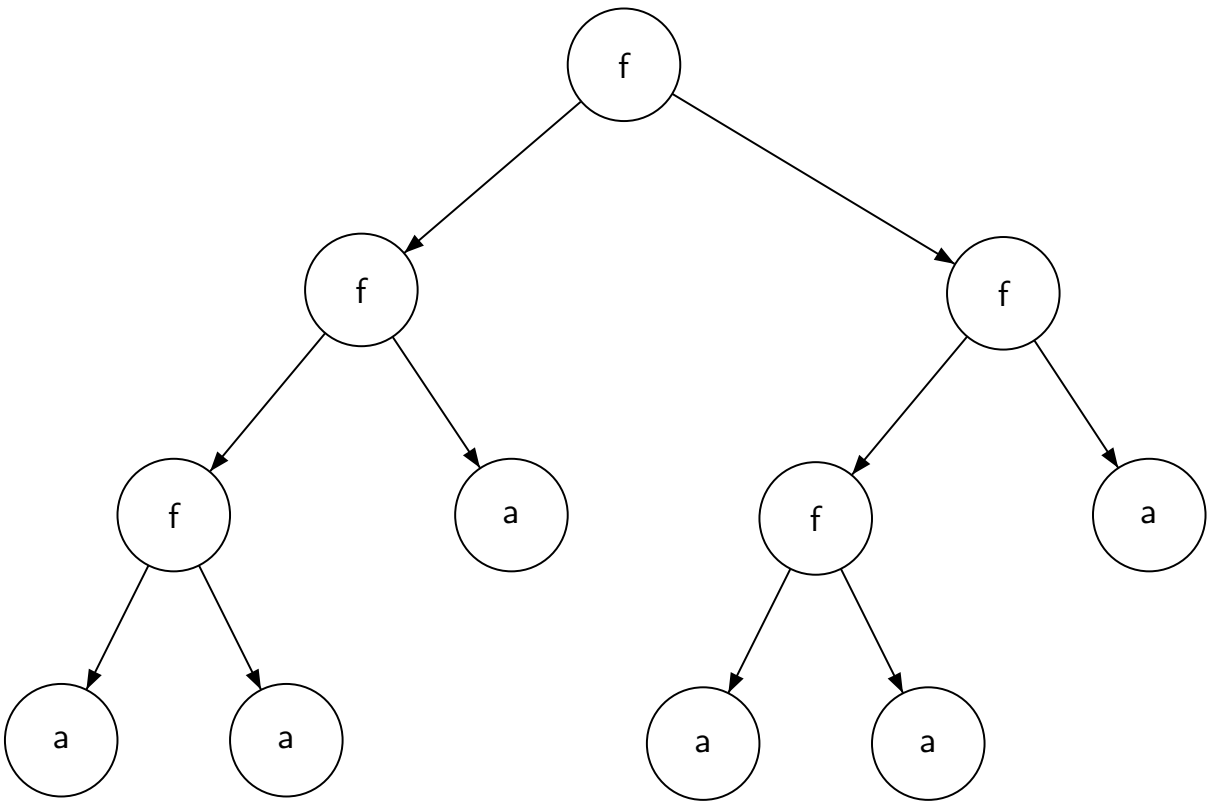


Successful Runs



- equalities and disequalities can be tested arbitrarily faraway
- different from usual **Automata with Constraints** where tests are **local** (BogaertT92, DauchetCC95, KariantoL07)

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

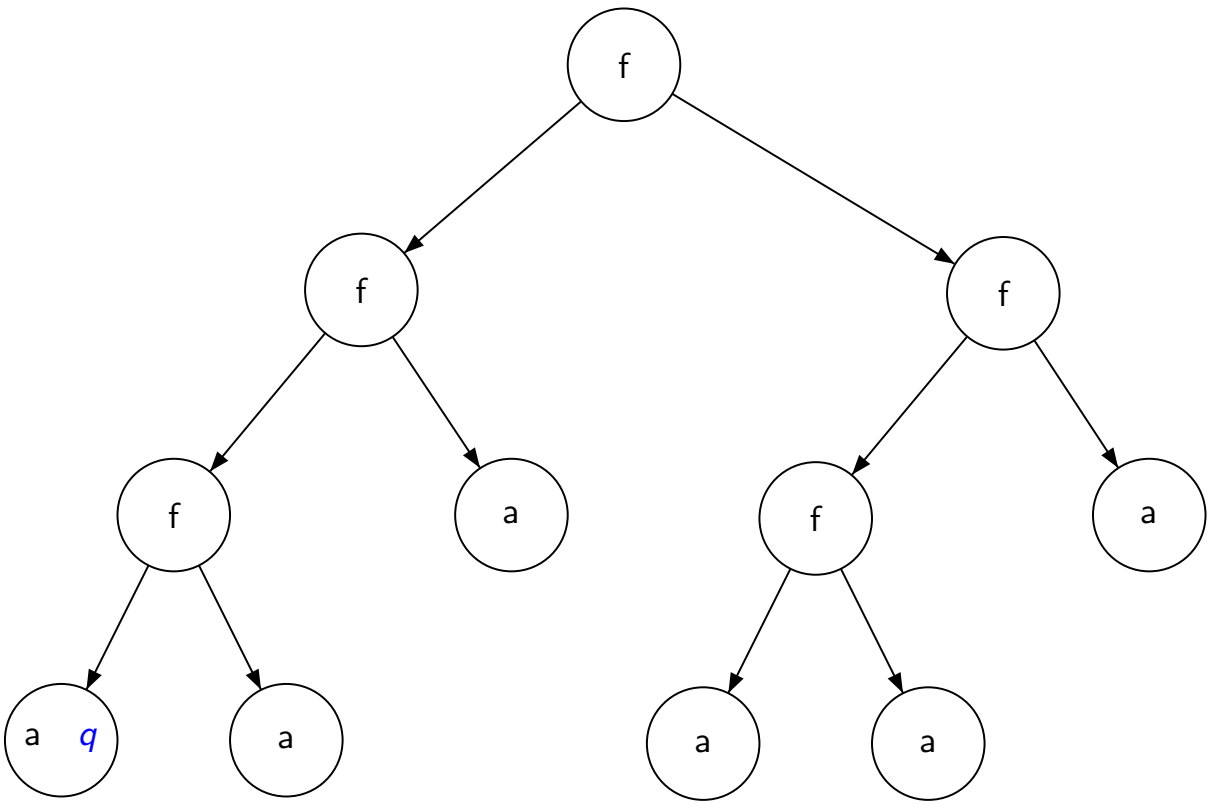


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 =_A q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

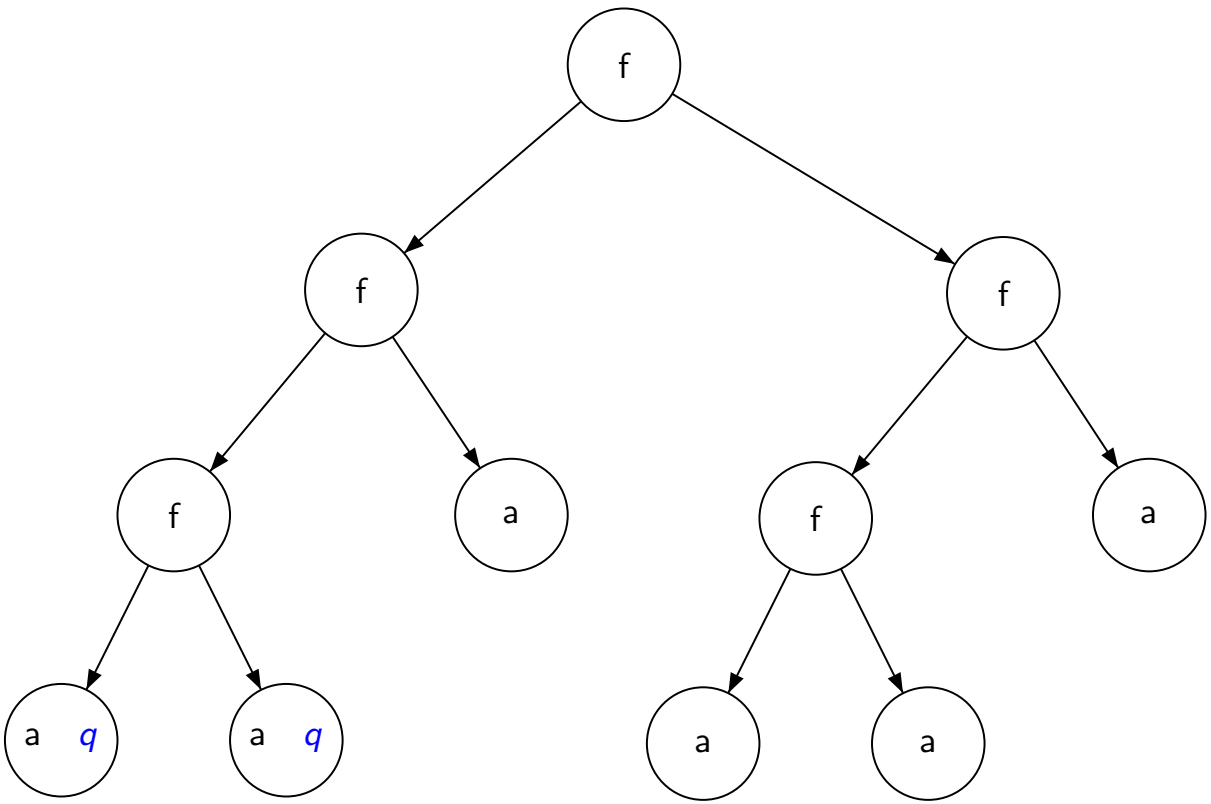


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 \ =_A \ q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

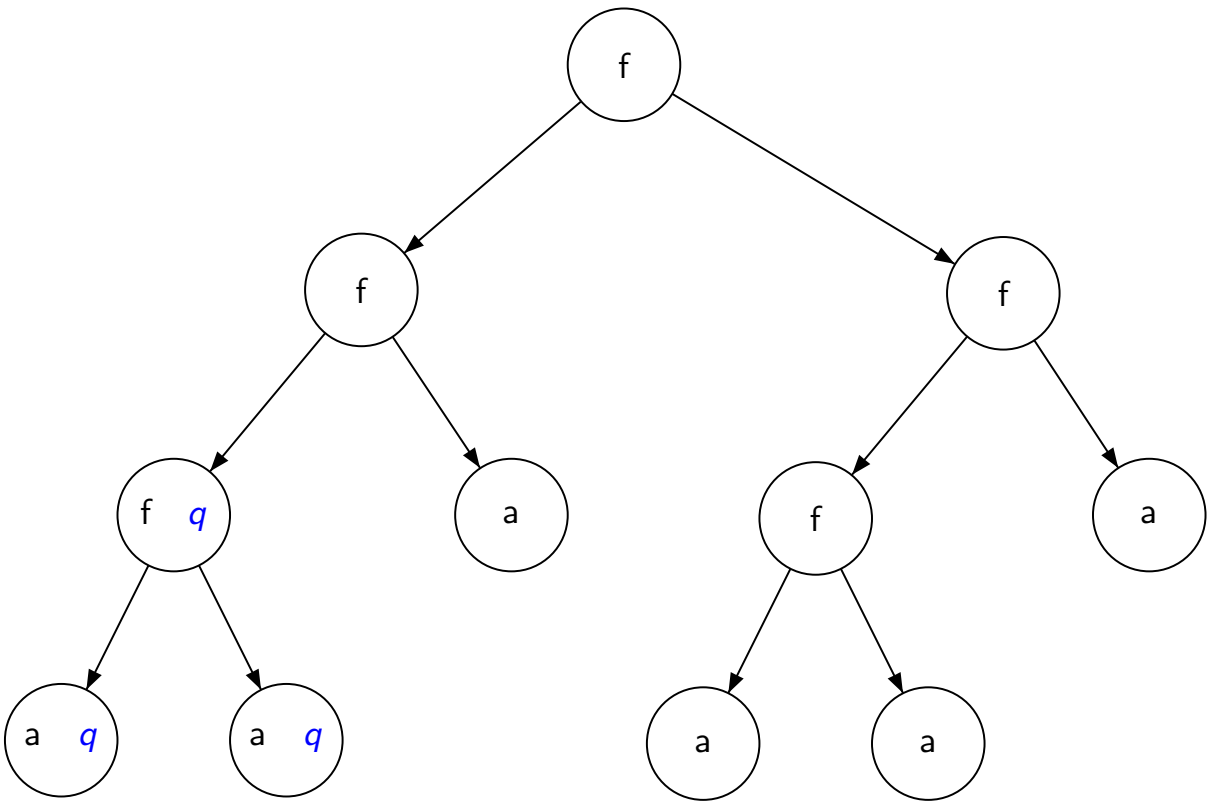


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 \ =_A \ q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

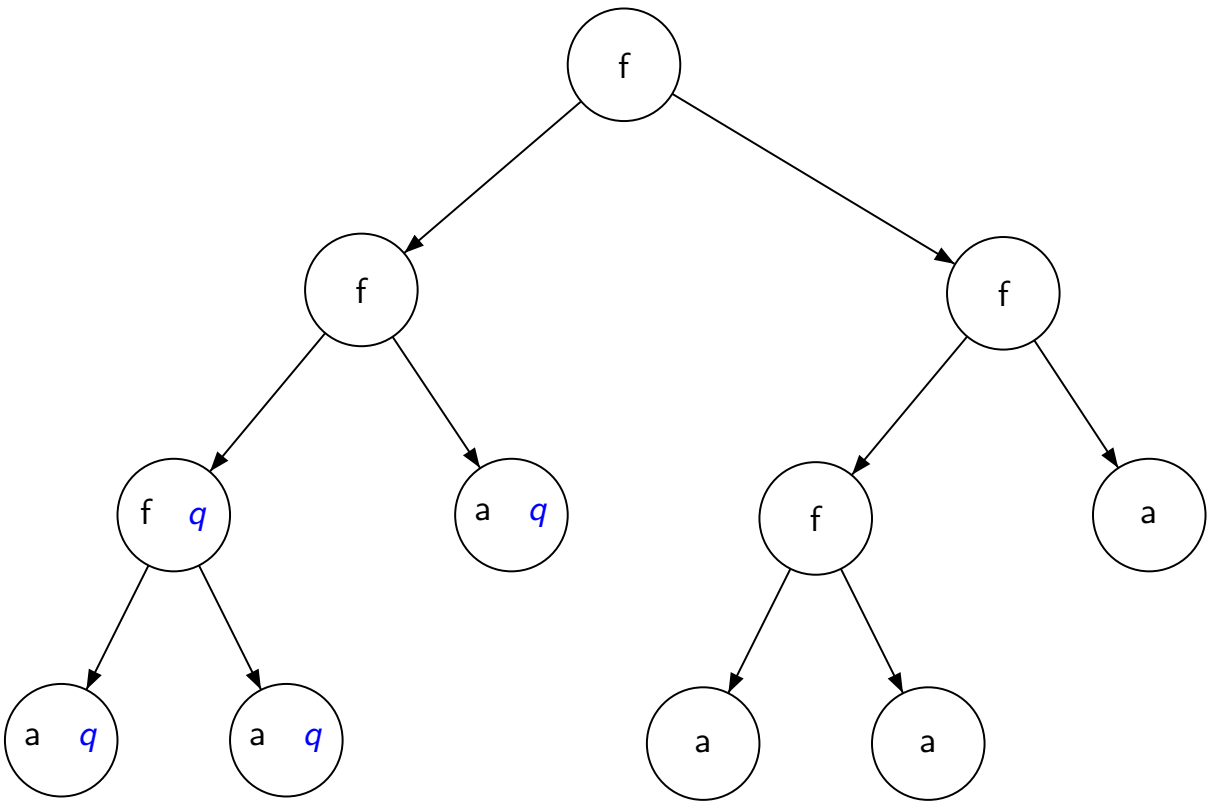


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 \ =_A \ q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

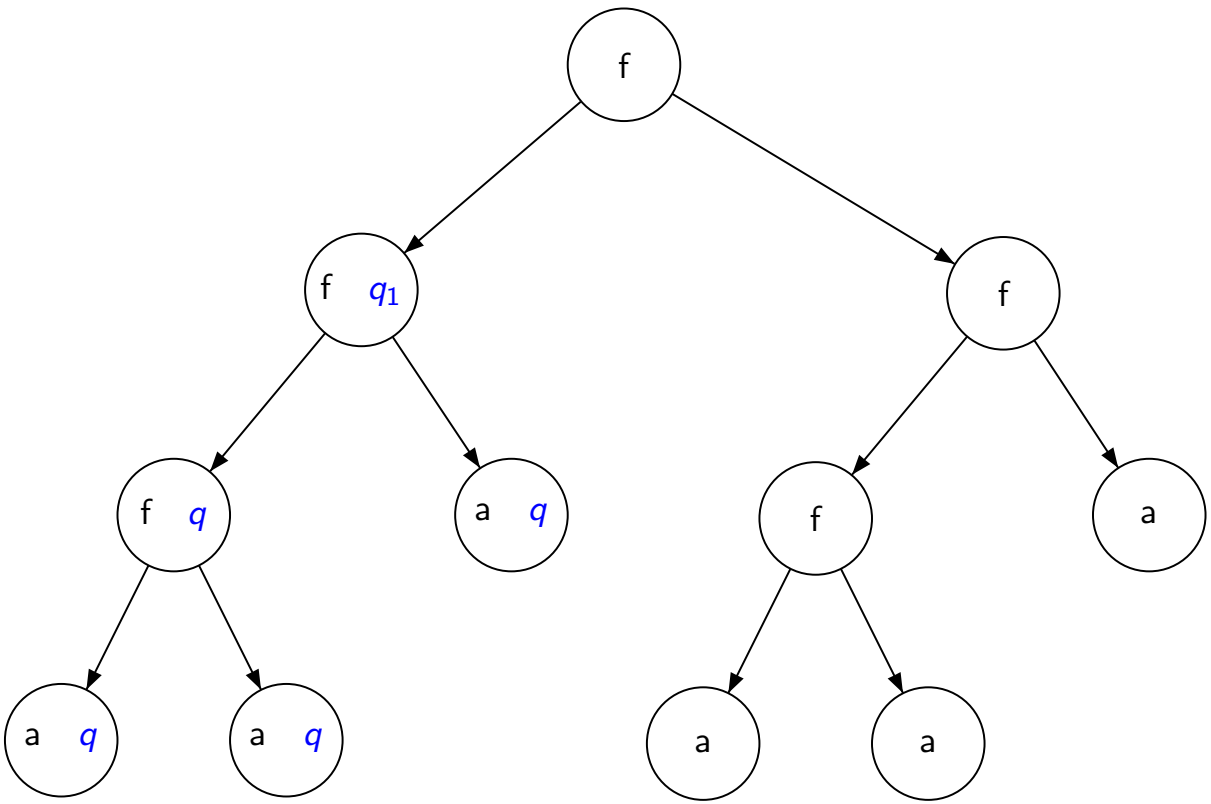


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 =_A q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

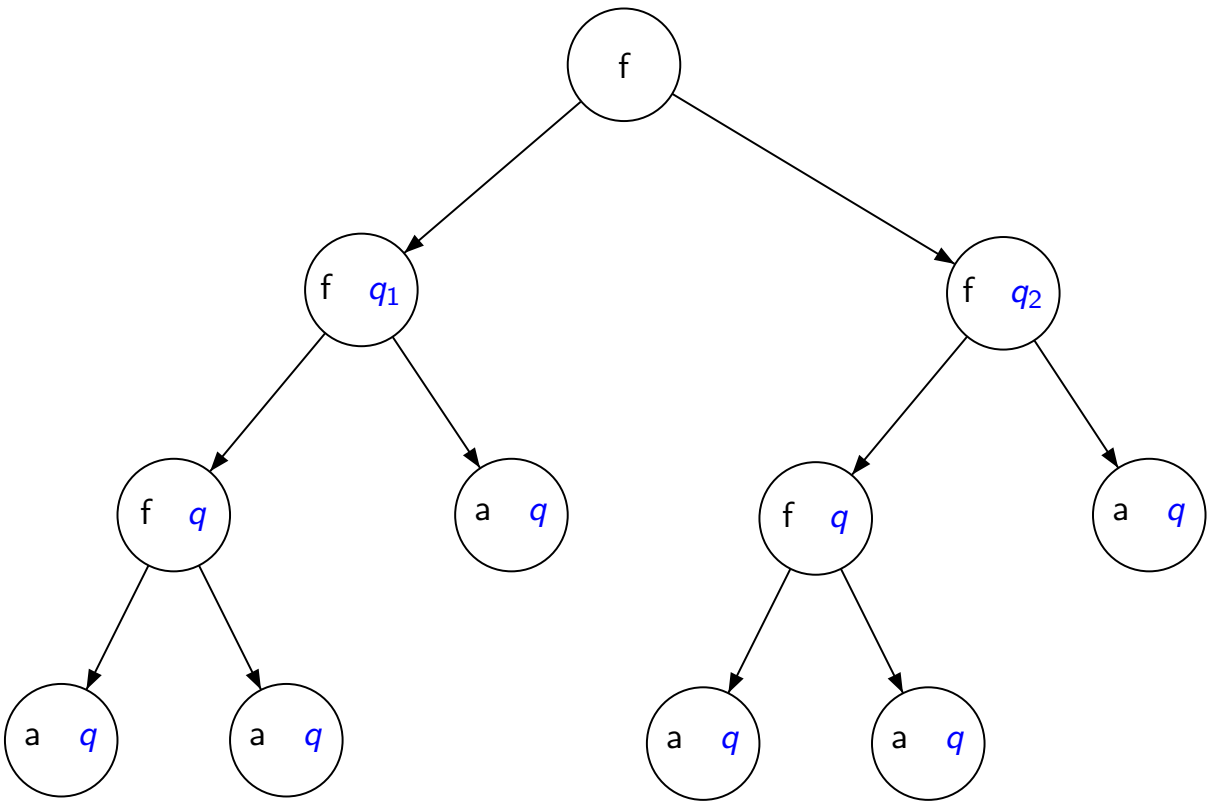


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 =_A q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

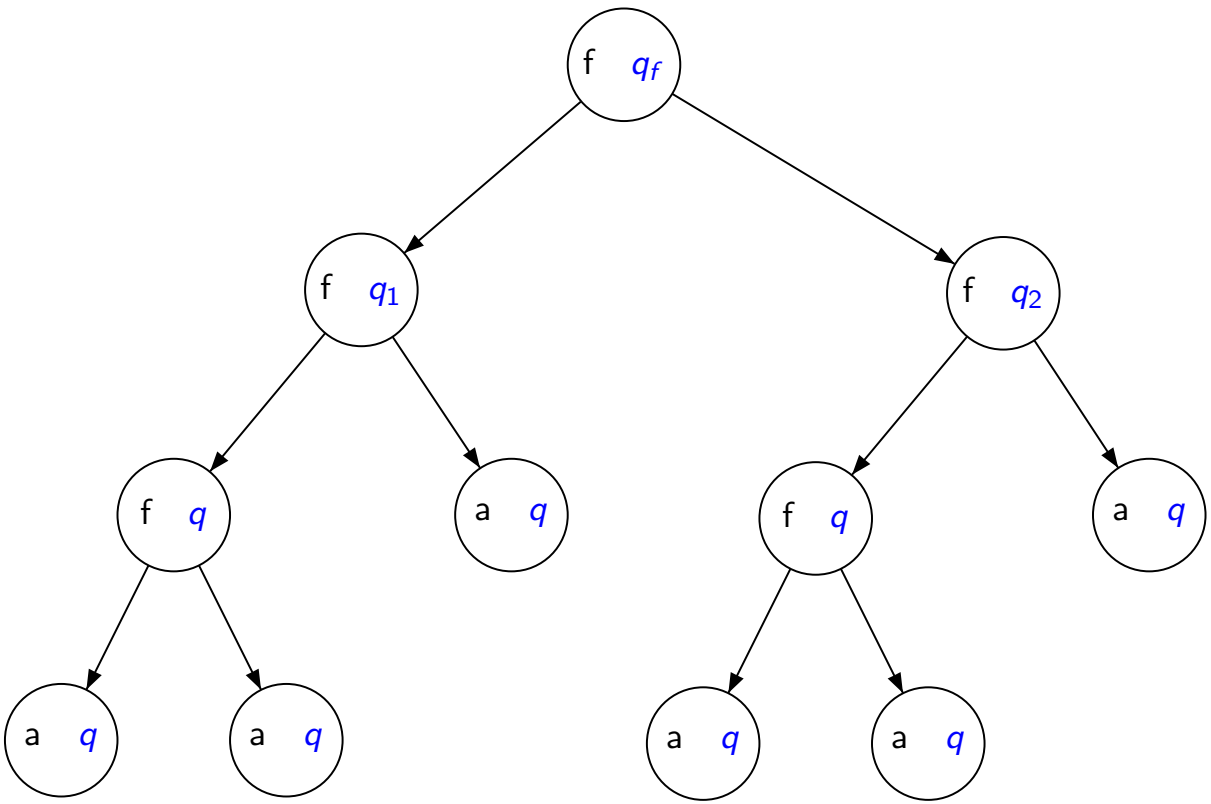


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 \ =_A \ q_2$

Example: $\{f(t,t) \mid t \in T_\Sigma\}$

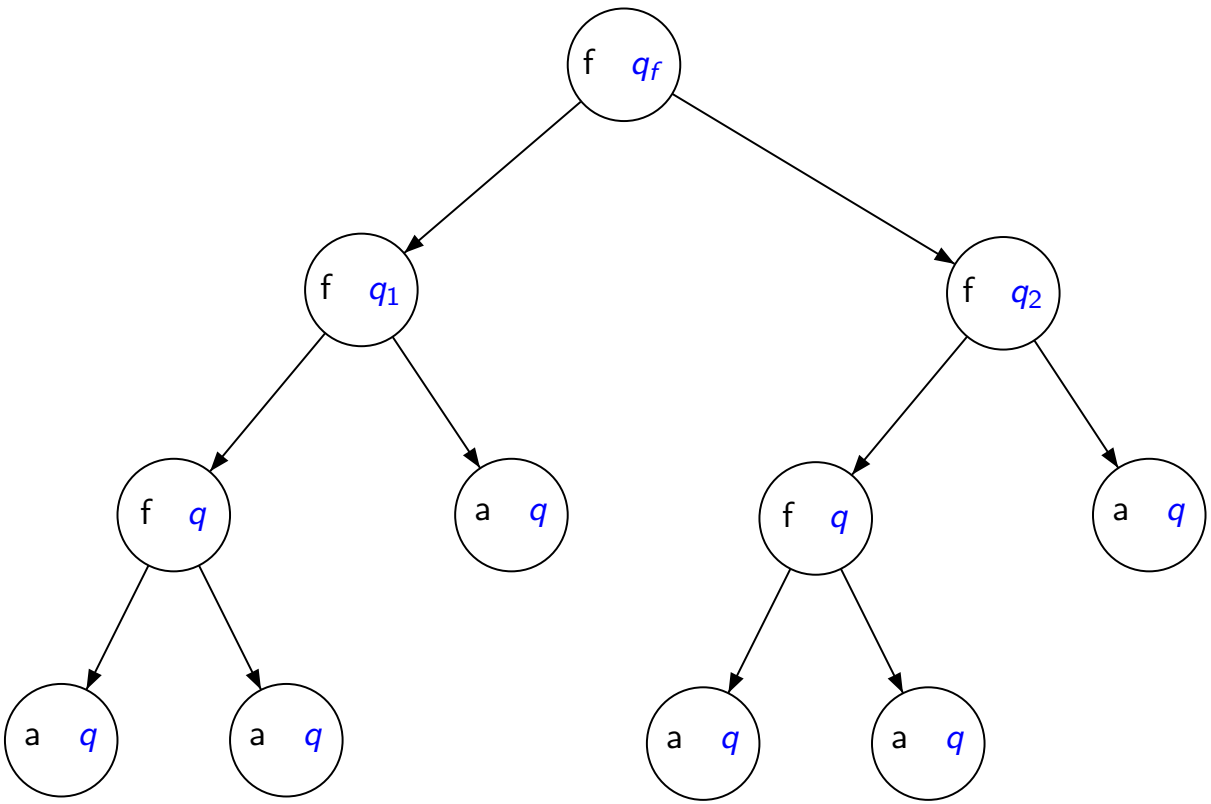


- $\Sigma = \{f, a\}$
- $Q = \{q, q_f, q_1, q_2\}$
- $F = \{q_f\}$
- $\Delta =$

$$a \longrightarrow q$$
$$f(q,q) \longrightarrow q$$
$$f(q,q) \longrightarrow q_1$$
$$f(q,q) \longrightarrow q_2$$
$$f(q_1,q_2) \longrightarrow q_f$$

- $q_1 =_A q_2$

Example: $\{f(t,s) \mid t,s \in T_\Sigma,\ t \neq s\}$



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

$$a \to q$$
$$f(q,q) \to q$$
$$f(q,q) \to q_1$$
$$f(q,q) \to q_2$$
$$f(q_1,q_2) \to q_f$$

- $q_1 \not\equiv_A q_2$

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**: $L(A) \neq \varnothing$?

Theorem

Emptiness is:

- *EXPTIME-complete for positive TAGED* ($\neq_A = \varnothing$)
- *decidable in NEXPTIME for negative TAGED* ($=_A = \varnothing$)
- *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 $\mathrm{dom}(\neq_A)$ 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 $\mathrm{dom}(\neq_A)$ 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 contraints 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;
- transitititions $\alpha(L) \longrightarrow q$ 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;
- transitititions $\alpha(L) \longrightarrow q$ where $L \subseteq Q^*$;
- lift all the results via a binary encoding of hedges.

Bounded TQL $\longrightarrow$ 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;
- transitiitions $\alpha(L) \rightarrow q$ 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
- $\{\ldots, X, \ldots\} =_A \{\ldots, X, \ldots\}$
- $\{\ldots, X, \ldots\} \neq_A \{\ldots, \neg X, \ldots\}$

Outline

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 $Comp(\mathsf{ch}, \mathsf{ch}^*, \mathsf{lab}_a)$
- streaming (GauwinCNT08), enumeration (collaboration with O.Gauwin, A.Durand, ANR Enum)

Some Perspectives

Query composition

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

TQL

- lower bounds (TQL + TAGED)
- guarded fragment

Some Perspectives

Query composition

- query answering algorithms specific to $Comp(\mathsf{ch}, \mathsf{ch}^*, \mathsf{lab}_a)$
- 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** $\exists \overline{x},\ \phi(\overline{x}) \wedge \neg\psi(\overline{x})$.

Some Perspectives

Query composition

- query answering algorithms specific to $Comp(\mathsf{ch}, \mathsf{ch}^*, \mathsf{lab}_a)$
- 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** $\exists \overline{x},\ \phi(\overline{x}) \wedge \neg\psi(\overline{x})$.
- emptiness of full TAGED
- application to security protocols (C.Vacher,F.Jacquemard, F.Klay)

Outline

Thank You