Monadic Datalog Containment on Trees

André Frochaux[1] Martin Grohe[2] Nicole Schweikardt[1]

[1]Goethe-University Frankfurt am Main, Germany

[2]RWTH Aachen, Germany

AMW 2014

Universidad Nacional de Colombia

June 5th, 2014

Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

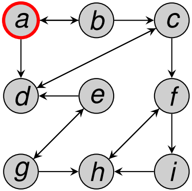
$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$\mathtt{Reach}(x) \leftarrow \mathtt{Red}(x)$
$\mathtt{Reach}(x) \leftarrow \mathtt{Reach}(y),\ \mathtt{E}(y,x)$

G:



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

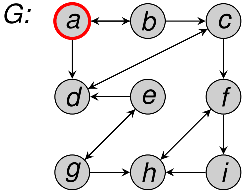
$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$\mathtt{Reach}(x) \leftarrow \mathtt{Red}(x)$
$\mathtt{Reach}(x) \leftarrow \mathtt{Reach}(y),\ \mathtt{E}(y,x)$

$$\mathcal{T}^0_{\mathcal{P}}(G) = \left\{ \begin{array}{c} Red(a),\ E(a,b),\ E(a,d), \\ E(b,a),\ \ldots\ ,\ E(i,h) \end{array} \right\}$$



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

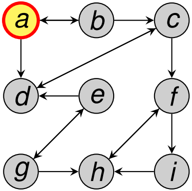
The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$$\mathtt{Reach(x)} \leftarrow \mathtt{Red(x)}$$
$$\mathtt{Reach(x)} \leftarrow \mathtt{Reach(y)},\ \mathtt{E(y,x)}$$

$$\mathcal{T}^1_{\mathcal{P}}(G) = \mathcal{T}^0_{\mathcal{P}}(G) \cup \{\ Reach(a)\ \}$$

G:



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

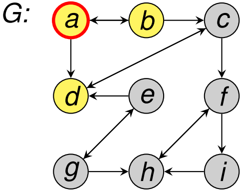
$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$$\mathtt{Reach}(x) \leftarrow \mathtt{Red}(x)$$
$$\mathtt{Reach}(x) \leftarrow \mathtt{Reach}(y),\ \mathtt{E}(y,x)$$

$$\mathcal{T}^2_{\mathcal{P}}(G) = \mathcal{T}^1_{\mathcal{P}}(G) \cup \{\ \mathit{Reach}(b),\ \mathit{Reach}(d)\ \}$$



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

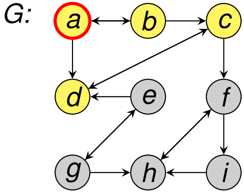
$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$$\mathtt{Reach(x)} \leftarrow \mathtt{Red(x)}$$
$$\mathtt{Reach(x)} \leftarrow \mathtt{Reach(y)},\ \mathtt{E(y,x)}$$

$$\mathcal{T}^3_{\mathcal{P}}(G) = \mathcal{T}^2_{\mathcal{P}}(G) \cup \{\ \mathit{Reach}(c)\ \}$$



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

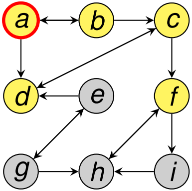
The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

```
Reach(x) ← Red(x)
Reach(x) ← Reach(y), E(y,x)
```

$$\mathcal{T}_{\mathcal{P}}^4(G) = \mathcal{T}_{\mathcal{P}}^3(G) \cup \{\ Reach(f)\ \}$$

G:



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

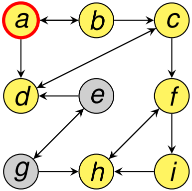
The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$$\mathtt{Reach}(x) \leftarrow \mathtt{Red}(x)$$
$$\mathtt{Reach}(x) \leftarrow \mathtt{Reach}(y),\ \mathtt{E}(y,x)$$

$$\mathcal{T}^5_{\mathcal{P}}(G) = \mathcal{T}^4_{\mathcal{P}}(G) \cup \{\ \mathit{Reach}(h),\ \mathit{Reach}(i)\ \}$$

G:



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

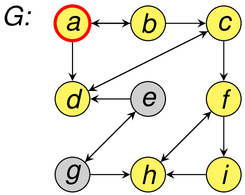
$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$\texttt{Reach(x)} \leftarrow \texttt{Red(x)}$
$\texttt{Reach(x)} \leftarrow \texttt{Reach(y), E(y,x)}$

$\mathcal{T}^6_{\mathcal{P}}(G) = \mathcal{T}^5_{\mathcal{P}}(G) =: \mathcal{T}^{\omega}_{\mathcal{P}}(G)$



Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

$$h(x) \qquad \leftarrow \qquad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

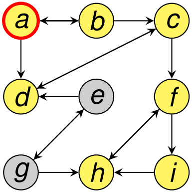
The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$\mathtt{Reach(x)} \leftarrow \mathtt{Red(x)}$
$\mathtt{Reach(x)} \leftarrow \mathtt{Reach(y),\ E(y,x)}$

$\mathcal{T}^6_{\mathcal{P}}(G) = \mathcal{T}^5_{\mathcal{P}}(G) =: \mathcal{T}^{\omega}_{\mathcal{P}}(G)$

G:



Query: $Q = (\mathcal{P}, P),\ \ P \in \mathrm{idb}(\mathcal{P}),\ \ Q(\mathcal{A}) := \{a \mid P(a) \in \mathcal{T}^{\omega}_{\mathcal{P}}(\mathcal{A})\}$

Monadic Datalog

A program $\mathcal{P}$ in *monadic Datalog* (for short: *mDatalog*) is a finite set of Datalog rules r of the form

$$h(x) \quad \leftarrow \quad b_1(\vec{x}_1),\ b_2(\vec{x}_2),\ \ldots,\ b_{n_r}(\vec{x}_{n_r}).$$

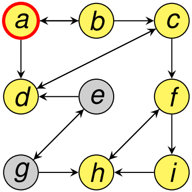
The semantics is defined by the *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$.

Example:

$\mathtt{Reach}(x) \leftarrow \mathtt{Red}(x)$
$\mathtt{Reach}(x) \leftarrow \mathtt{Reach}(y),\ \mathtt{E}(y,x)$

$\mathcal{T}_{\mathcal{P}}^{6}(G) = \mathcal{T}_{\mathcal{P}}^{5}(G) =: \mathcal{T}_{\mathcal{P}}^{\omega}(G)$

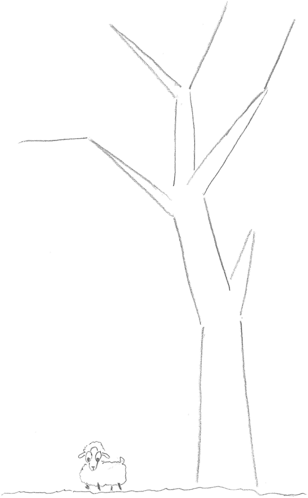
G:



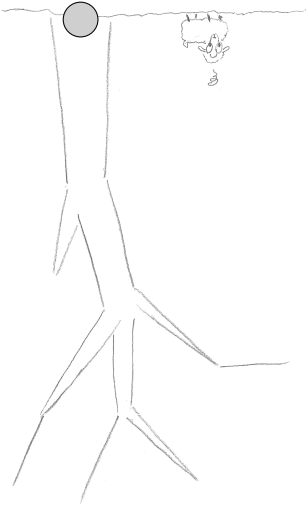
Query: $Q = (\mathcal{P}, P)$, $P \in \mathrm{idb}(\mathcal{P})$, $Q(\mathcal{A}) := \{a \mid P(a) \in \mathcal{T}_{\mathcal{P}}^{\omega}(\mathcal{A})\}$

$Q = (\mathcal{P}, \mathtt{Reach})$, $Q(G) = \{a, b, c, d, f, h, i\}$

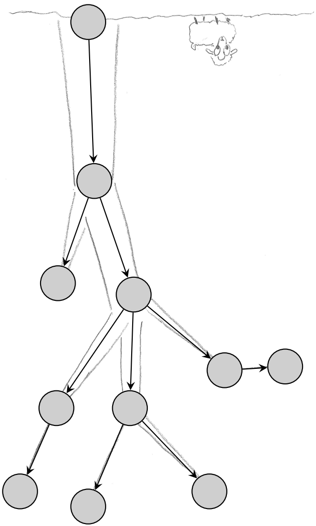
Σ-labeled Trees



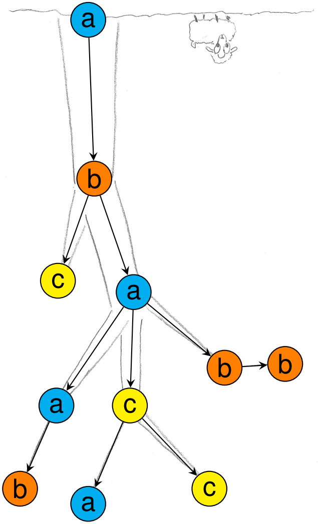
Σ-labeled Trees



Σ-labeled Trees

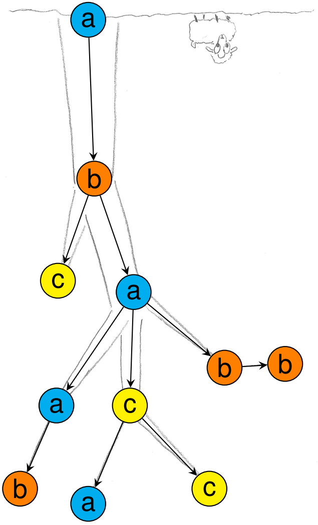


Σ-labeled Trees



Σ: finite, unranked alphabet

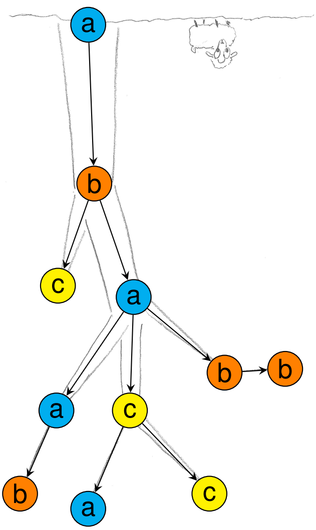
Σ-labeled Trees



Σ: finite, unranked alphabet

- **label**$_\alpha(x)$: node x carries label $\alpha \in \Sigma$

Σ-labeled Trees



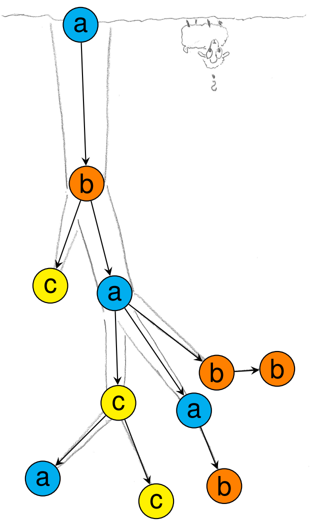
Σ: finite, unranked alphabet

- **label**$_\alpha(x)$: node x carries label $\alpha \in \Sigma$

Unordered trees

- **child**(x, y): y is child of x
- **root**(x): node x is the root
- **leaf**(x): node x is a leaf
- **desc**(x, y): y is descendant of x

Σ-labeled Trees



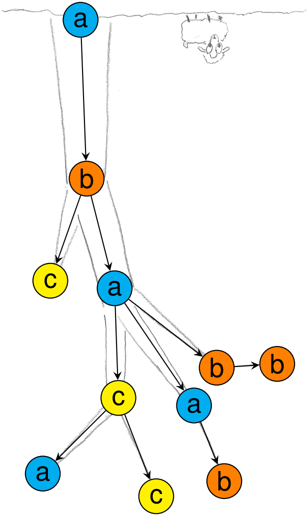
Σ: finite, unranked alphabet

- **label**$_\alpha(x)$: node x carries label $\alpha \in \Sigma$

Unordered trees

- **child**(x, y): y is child of x
- **root**(x): node x is the root
- **leaf**(x): node x is a leaf
- **desc**(x, y): y is descendant of x

Σ-labeled Trees



Σ: finite, unranked alphabet

- **label**$_\alpha(x)$: node x carries label $\alpha \in \Sigma$

Unordered trees

- **child**(x, y): y is child of x
- **root**(x): node x is the root
- **leaf**(x): node x is a leaf
- **desc**(x, y): y is descendant of x

Ordered trees

- **fc**(x, y): y is the first child of x
- **ns**(x, y): y is the next sibling of x
- **ls**(x): x is the last sibling
- **root**(x), **leaf**(x), **child**(x, y), **desc**(x, y)

Query Containment Problem (QCP)

Let τ be a schema for Σ-labeled trees.

Let Q_1 and Q_2 be queries in mDatalog(τ). Then we say:

$$Q_1 \subseteq Q_2, \quad \text{iff} \quad Q_1(T) \subseteq Q_2(T) \text{ for every } \Sigma\text{-labeled tree } T$$

QCP for unary queries in mDatalog(τ) on Σ-labeled trees		
Input:	Queries Q_1 and Q_2 in mDatalog(τ).	
Output:	`Yes`,	if $Q_1 \subseteq Q_2$,
	`No`,	otherwise.

Results

Previously known:

Containment of mDatalog over arbitrary finite structures:

Cosmadakis et al (1988):	EXPTIME-hard and in 2EXPTIME.
Benedikt et al (2012):	2EXPTIME-complete.

Results

Previously known:

Containment of mDatalog over arbitrary finite structures:

Cosmadakis et al (1988):	EXPTIME-hard and in 2EXPTIME.
Benedikt et al (2012):	2EXPTIME-complete.
Gottlob/Koch (2002):	mDatalog(τ_{GK}) on ordered Σ-labeled trees is EXPTIME-hard and decidable.

Results

Previously known:

Containment of mDatalog over arbitrary finite structures:

Cosmadakis et al (1988):	EXPTIME-hard and in 2EXPTIME.
Benedikt et al (2012):	2EXPTIME-complete.

τ_{GK} : **fc**, **ns**, **ls**, **root**, **leaf**, $(\textbf{label}_\alpha)_{\alpha\in\Sigma}$

Gottlob/Koch (2002):	mDatalog(τ_{GK}) on ordered Σ-labeled trees is EXPTIME-hard and decidable.

Results

Theorem:

τ_u : **child**, $(\textbf{label}_\alpha)_{\alpha\in\Sigma}$

The QCP for Boolean mDatalog(τ_u) *on unordered* Σ*-labeled trees is* EXPTIME*-hard.*

Corollary:

τ_o : **fc**, **ns**, $(\textbf{label}_\alpha)_{\alpha\in\Sigma}$

The QCP for Boolean mDatalog(τ_o) *on ordered* Σ*-labeled trees is* EXPTIME*-hard.*

Previously known:

Containment of mDatalog over arbitrary finite structures:

Cosmadakis et al (1988):	EXPTIME-hard and in 2EXPTIME.
Benedikt et al (2012):	2EXPTIME-complete.

τ_{GK} : **fc**, **ns**, **ls**, **root**, **leaf**, $(\textbf{label}_\alpha)_{\alpha\in\Sigma}$

Gottlob/Koch (2002):	mDatalog(τ_{GK}) on ordered Σ-labeled trees is EXPTIME-hard and decidable.

Results

Theorem: τ_u : **child**, $(\textbf{label}_\alpha)_{\alpha \in \Sigma}$

The QCP for Boolean mDatalog(τ_u) *on unordered* Σ*-labeled trees is* EXPTIME*-hard.*

Corollary: τ_o : **fc**, **ns**, $(\textbf{label}_\alpha)_{\alpha \in \Sigma}$

The QCP for Boolean mDatalog(τ_o) *on ordered* Σ*-labeled trees is* EXPTIME*-hard.*

Theorem: $\tau_{GK}^{\textbf{child}}$: **fc**, **ns**, **ls**, **child**, **root**, **leaf**, $(\textbf{label}_\alpha)_{\alpha \in \Sigma}$

The QCP for unary mDatalog($\tau_{GK}^{\textbf{child}}$) *on ordered* Σ*-labeled trees belongs to* EXPTIME*.*

Corollary: $\tau_u^{\textbf{root},\textbf{leaf}}$: **child**, **root**, **leaf**, $(\textbf{label}_\alpha)_{\alpha \in \Sigma}$

The QCP for unary mDatalog($\tau_u^{\textbf{root},\textbf{leaf}}$) *on unordered* Σ*-labeled trees belongs to* EXPTIME*.*

Sketching the Proof of the 2nd Theorem

Theorem:

The QCP for unary mDatalog($\tau_{GK}^{\textbf{child}}$) *on ordered* Σ*-labeled trees belongs to* EXPTIME*.*

Proof (sketch):

Given: unary Q_1 and Q_2 in mDatalog($\tau_{GK}^{\textbf{child}}$).
Question: decide, whether $Q_1 \subseteq Q_2$
Use the *automata-theoretic method*:

Sketching the Proof of the 2nd Theorem

Theorem:

The QCP for unary mDatalog($\tau_{GK}^{\mathbf{child}}$) *on ordered* Σ*-labeled trees belongs to* EXPTIME*.*

Proof (sketch):

Given: unary Q_1 and Q_2 in mDatalog($\tau_{GK}^{\mathbf{child}}$).
Question: decide, whether $Q_1 \subseteq Q_2$
Use the *automata-theoretic method*:

(1) $Q_1, Q_2 \rightsquigarrow$ *Boolean* queries Q_1', Q_2' on *binary* trees, such that

$$Q_1 \subseteq Q_2 \;\Leftrightarrow\; Q_1' \subseteq Q_2'$$

Sketching the Proof of the 2nd Theorem

Theorem:

The QCP for unary mDatalog($\tau_{GK}^{\textbf{child}}$) *on ordered* Σ*-labeled trees belongs to* EXPTIME*.*

Proof (sketch):

Given: unary Q_1 and Q_2 in mDatalog($\tau_{GK}^{\textbf{child}}$).
Question: decide, whether $Q_1 \subseteq Q_2$
Use the *automata-theoretic method*:

(1) $Q_1, Q_2 \rightsquigarrow$ *Boolean* queries Q_1', Q_2' on *binary* trees, such that

$$Q_1 \subseteq Q_2 \;\Leftrightarrow\; Q_1' \subseteq Q_2'$$

(2) $Q_1', Q_2' \rightsquigarrow$ tree automata $\mathtt{A}_1^{\textbf{yes}}$ and $\mathtt{A}_2^{\textbf{no}}$, such that

$\mathtt{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$ and $\mathtt{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$

Sketching the Proof of the 2nd Theorem

Theorem:

The QCP for unary $\mathrm{mDatalog}(\tau_{GK}^{\mathbf{child}})$ *on ordered* Σ*-labeled trees belongs to* EXPTIME.

Proof (sketch):

Given: unary Q_1 and Q_2 in $\mathrm{mDatalog}(\tau_{GK}^{\mathbf{child}})$.

Question: decide, whether $Q_1 \subseteq Q_2$

Use the *automata-theoretic method*:

(1) $Q_1, Q_2 \rightsquigarrow$ *Boolean* queries Q_1', Q_2' on *binary* trees, such that

$$Q_1 \subseteq Q_2 \;\Leftrightarrow\; Q_1' \subseteq Q_2'$$

(2) $Q_1', Q_2' \rightsquigarrow$ tree automata $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$, such that

$\mathtt{A}_1^{\mathbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \mathbf{yes}$ and $\mathtt{A}_2^{\mathbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \mathbf{no}$

(3) Construct the product automaton $\mathtt{B} : \mathcal{L}(\mathtt{B}) = \mathcal{L}(\mathtt{A}_1^{\mathbf{yes}}) \cap \mathcal{L}(\mathtt{A}_2^{\mathbf{no}})$.

Test: $\mathcal{L}(\mathtt{B}) = \varnothing$?

Note that $\mathcal{L}(\mathtt{B}) \neq \varnothing$ if, and only if, $Q_1 \not\subseteq Q_2$.

□

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathtt{A}_1^{\textbf{yes}}$: $\mathtt{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathtt{A}_2^{\textbf{no}}$: $\mathtt{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$.

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathtt{A}_1^{\textbf{yes}}$: $\mathtt{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathtt{A}_2^{\textbf{no}}$: $\mathtt{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$.

Translate Q_2' into equivalent MSO-sentence φ_2 of the form

$$\forall X_1 \cdots \forall X_n \, \exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathtt{A}_1^{\textbf{yes}}$: $\mathtt{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathtt{A}_2^{\textbf{no}}$: $\mathtt{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$.

Translate Q_2' into equivalent MSO-sentence φ_2 of the form

$$\forall X_1 \cdots \forall X_n \, \exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

$$\neg\varphi_2 \;\equiv\; \exists X_1 \cdots \exists X_n \, \neg\exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathbb{A}_1^{\textbf{yes}}$: $\mathbb{A}_1^{\textbf{yes}}$ accepts $T \iff Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathbb{A}_2^{\textbf{no}}$: $\mathbb{A}_2^{\textbf{no}}$ accepts $T \iff Q_2'(T) = \textbf{no}$.

Translate Q_2' into equivalent MSO-sentence φ_2 of the form

$$\forall X_1 \cdots \forall X_n \, \exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

$$\neg\varphi_2 \;\equiv\; \exists X_1 \cdots \exists X_n \, \neg\exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

If query in TMNF (cf., Gottlob, Koch: PODS 2002)

$\rightsquigarrow$ construction of $\mathbb{A}_2^{\textbf{no}}$ in 1-fold exponential time

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathbb{A}_1^{\textbf{yes}}$: $\mathbb{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathbb{A}_2^{\textbf{no}}$: $\mathbb{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$.

Translate Q_2' into equivalent MSO-sentence φ_2 of the form

$$\forall X_1 \cdots \forall X_n \, \exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

$$\neg\varphi_2 \equiv \exists X_1 \cdots \exists X_n \, \neg \exists z_1 \cdots \exists z_\ell \bigvee_{j=1}^{m} \xi_j$$

If query in TMNF (cf., Gottlob, Koch: PODS 2002)

$\rightsquigarrow$ construction of $\mathbb{A}_2^{\textbf{no}}$ in 1-fold exponential time

Problem:

By following this approach to construct $\mathbb{A}_1^{\textbf{yes}}$,
a second complementation leads to 2-fold exponential time.

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathtt{A}_1^{\textbf{yes}}$: $\mathtt{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathtt{A}_2^{\textbf{no}}$: $\mathtt{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$.

Key idea:

Boolean TMNF-query $Q_1' \rightsquigarrow$ two-way alternating tree automaton (2ATA) $\hat{\mathtt{A}}_1^{\textbf{yes}}$

(in polynomial time)

Sketch (cont.): Zoom into Step (2)

(2a) Construct tree automaton $\mathtt{A}_1^{\textbf{yes}}$: $\mathtt{A}_1^{\textbf{yes}}$ accepts $T \Leftrightarrow Q_1'(T) = \textbf{yes}$.

(2b) Construct tree automaton $\mathtt{A}_2^{\textbf{no}}$: $\mathtt{A}_2^{\textbf{no}}$ accepts $T \Leftrightarrow Q_2'(T) = \textbf{no}$.

Key idea:

Boolean TMNF-query $Q_1' \rightsquigarrow$ two-way alternating tree automaton (2ATA) $\hat{\mathtt{A}}_1^{\textbf{yes}}$

(in polynomial time)

$\hat{\mathtt{A}}_1^{\textbf{yes}} \rightsquigarrow$ tree automaton $\mathtt{A}_1^{\textbf{yes}}$

(in 1-fold exponential time)
(implicit in Vardi: ICALP'98 / Maneth, Friese, Seidl: 2010)

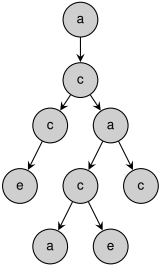
Sketch (cont.): Zoom into Step (2)

(2a)

(2b

Ke

Bo



$$\delta(X, c) = ((Y, \textit{stay}) \wedge (Z, \textit{stay})) \vee \ldots \vee ((\textit{is\_lc}, \textit{stay}) \wedge (Y, \textit{up}))$$

Sketch (cont.): Zoom into Step (2)

(2a)

(2b

K

B

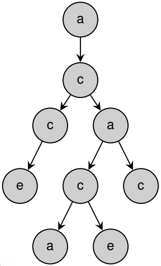
es

e)

$$X(x) \leftarrow \mathbf{lc}(y,x),\ Y(y)$$

$$\vdots$$

$$X(x) \leftarrow Y(x),\ Z(x)$$



e)

0)

$$\delta(X,c) = \big(((Y, \mathit{stay}) \wedge (Z, \mathit{stay})\big) \vee \ldots \vee \big((\mathit{is\_lc}, \mathit{stay}) \wedge (Y, \mathit{up})\big)$$

Final Remarks

Let τ be a schema for Σ-labeled trees and let **desc** $\in \tau$.

Theorem:

The QCP for unary mDatalog(τ) *on* Σ*-labeled trees can be solved in* 2*-fold exponential time.*

Final Remarks

Let τ be a schema for Σ-labeled trees and let **desc** $\in \tau$.

Theorem:

The QCP for unary mDatalog(τ) *on* Σ*-labeled trees can be solved in* 2*-fold exponential time.*

Current work:

- Close the gap between the EXPTIME lower and the 2EXPTIME upper bound for the case where the descendant-axis is involved

Final Remarks

Let τ be a schema for Σ-labeled trees and let **desc** $\in \tau$.

Theorem:

The QCP for unary mDatalog(τ) *on* Σ*-labeled trees can be solved in* 2*-fold exponential time.*

Current work:

- Close the gap between the EXPTIME lower and the 2EXPTIME upper bound for the case where the descendant-axis is involved
- Extend the results to related problems like Emptiness and Equivalence.

Final Remarks

Let τ be a schema for Σ-labeled trees and let **desc** $\in \tau$.

Theorem:

The QCP for unary mDatalog(τ) *on* Σ*-labeled trees can be solved in* 2*-fold exponential time.*

Current work:

- Close the gap between the EXPTIME lower and the 2EXPTIME upper bound for the case where the descendant-axis is involved
- Extend the results to related problems like Emptiness and Equivalence.

Thank you
for your attention!