String Parallel Rewriting: analysis of the structure of the derivations

P. Bourhis, D. Gallois, S. Tison

Meeting DELTA

26-28 Mars 2018
Contents

1 Motivation
2 Concepts in parallel rewriting
3 Main result : how to bound parallel complexity
4 Relation with Datalog
Word constraints
Word constraints
Word constraints
Word constraints
Word constraints
word constraint:
\[
b(x, x_1) a(x_1, x_2) b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1) c(z_1, z_2) a(z_2, y)
\]
Problem

\[ xL_y = \text{pair of nodes linked by a path labelled in } L \]
$xLy = \text{pair of nodes linked by a path labelled in } L$

$C \models xLy \sqsubseteq xL'y$?
Problem

\[ xLy = \text{pair of nodes linked by a path labelled in } L \]
\[ C \models xLy \sqsubseteq xL'y ? \]

Example:
\( xLy = \text{pair of nodes linked by a path labelled in } L \)

\[ C \models xLy \subseteq xL'y? \]

Example:
\[ C = \{ a(x, y) \rightarrow b(x, y) \} \]
Problem

\[ xL y = \text{pair of nodes linked by a path labelled in } L \]
\[ C \models xL y \sqsubseteq xL' y ? \]

Example:
\[ C = \{ a(x, y) \rightarrow b(x, y) \} \]
\[ C \models x(a + b)^* y \sqsubseteq xb^* y \]
Word constraints and RPQ

\[ b(x, x_1)a(x_1, x_2)b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1)c(z_1, z_2)a(z_2, y) \]
Word constraints and RPQ

\[ b(x, x_1) a(x_1, x_2) b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1) c(z_1, z_2) a(z_2, y) \]

\[ bab \rightarrow aca \]
Word constraints and RPQ

\[ b(x, x_1)a(x_1, x_2)b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1)c(z_1, z_2)a(z_2, y) \]

\[ bab \rightarrow aca \]

\[ \Rightarrow \text{Reduction to string rewriting system} \]
Word constraints and RPQ

\[ b(x, x_1)a(x_1, x_2)b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1)c(z_1, z_2)a(z_2, y) \]

\[ bab \rightarrow aca \]

⇒ Reduction to string rewriting system

**Theorem [GT03]**

\( C \) is a set of word constraints: \( C \models xLy \sqsubseteq xL'y \iff \ L \subseteq \text{Anc}_R(L') \)
Word constraints and RPQ

\[ b(x, x_1) a(x_1, x_2) b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1) c(z_1, z_2) a(z_2, y) \]

\[ bab \rightarrow aca \]

⇒ Reduction to string rewriting system

**Theorem [GT03]**

\( C \) is a set of word constraints: \( C \models xLy \sqsubseteq xL'y \iff L \subseteq \text{Anc}_R(L') \)

with \( \text{Anc}_R(L') = \{ x \mid \exists y \in L', x \rightarrow^* y \} \)
Word constraints and RPQ

\[ b(x, x_1) a(x_1, x_2) b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1) c(z_1, z_2) a(z_2, y) \]

\[ bab \rightarrow aca \]

⇒ Reduction to string rewriting system

**Theorem [GT03]**

\( C \) is a set of word constraints: \( C \models xLy \subseteq xL'y \iff L \subseteq \text{Anc}_R(L') \)

with \( \text{Anc}_R(L') = \{ x \mid \exists y \in L', x \xrightarrow{*} y \} \)

Compute efficiently \( \text{Anc}_R(L') \) by completion
Word constraints and RPQ

\[ b(x, x_1) a(x_1, x_2) b(x_2, y) \rightarrow \exists z_1, z_2, a(x, z_1) c(z_1, z_2) a(z_2, y) \]

\[ bab \rightarrow aca \]

⇒ Reduction to string rewriting system

**Theorem [GT03]**

C is a set of word constraints: \( C \models xLy \subseteq xL'y \iff L \subseteq \text{Anc}_R(L') \)
with \( \text{Anc}_R(L') = \{ x \mid \exists y \in L', x \longrightarrow^* y \} \)

Compute efficiently \( \text{Anc}_R(L') \) by completion keeping worried by completion in database theory
Word constraints and completion

\[ ab \rightarrow bc \]
Word constraints and completion

$ab \rightarrow bc$
Word constraints and completion

(ab → bc)
Word constraints and completion

$ab \rightarrow bc$
Word constraints and completion

$ab \rightarrow bc$
Word constraints and completion

\[ ab \rightarrow bc \]
Contents

1 Motivation

2 Concepts in parallel rewriting

3 Main result: how to bound parallel complexity

4 Relation with Datalog
String parallel rewriting

Once upon a time...
String parallel rewriting

Once upon a time... \( R = \{ f \rightarrow p, \text{og} \rightarrow \text{ince} \} \)
Once upon a time... $R = \{ f \rightarrow p, \, og \rightarrow ince \}$

frog
String parallel rewriting

Once upon a time... $R = \{ f \rightarrow p, og \rightarrow ince \}$

frog
Once upon a time... \( R = \{ f \rightarrow p, \text{og} \rightarrow \text{ince} \} \)
Once upon a time... $R = \{ f \rightarrow p, \text{og} \rightarrow \text{ince} \}$
String parallel rewriting

Once upon a time... \( R = \{ f \rightarrow p, \ og \rightarrow \ ince \} \)
String parallel rewriting

Once upon a time... \( R = \{ f \rightarrow p, og \rightarrow ince \} \)
Once upon a time... $R = \{ f \rightarrow p, \text{og} \rightarrow \text{ince} \}$
String parallel rewriting

Once upon a time... $R = \{f \rightarrow p, \ og \rightarrow \ ince\}$

\[
\begin{align*}
\text{frog} & \quad \rightarrow \quad p \\
\text{frog} & \quad \rightarrow \quad \text{ince} \\
\text{ince} & \quad \rightarrow \quad \text{ince}
\end{align*}
\]
String parallel rewriting

Once upon a time... $R = \{ f \rightarrow p, \text{og} \rightarrow \text{ince} \}$

String specialization of synchronous/multi-step rewriting [BKdVT03] (or concurrent rewriting [GKM87, KV90]) on terms
Definition

A srs is parallely bounded by \( k \) iff \( \rightarrow^k = \rightarrow^* \)
srs parallely bounded

**Definition**

A srs is parallely bounded by $k$ iff $\rightarrow \circ \rightarrow^k = \rightarrow \circ \rightarrow^*$

$R = \{a \rightarrow b, b \rightarrow a\}$ is parallely bounded by 1
A srs is parallely bounded by $k$ iff $\Box \rightarrow^k = \Box \rightarrow^*$

$R = \{a \rightarrow b, b \rightarrow a\}$ is parallely bounded by 1

- $R^{-1}$ is parallely-bounded
Definition

A srs is parallely bounded by \( k \) iff \( \rightarrow \otimes^k = \rightarrow \otimes^* \)

\[ R = \{a \rightarrow b, b \rightarrow a\} \] is parallely bounded by 1

- \( R^{-1} \) is parallely-bounded
- \( R \) is a rational relation
A srs is parallely bounded by $k$ iff $\bigcirc \rightarrow^k = \bigcirc \rightarrow^*$

$R = \{a \rightarrow b, b \rightarrow a\}$ is parallely bounded by 1

- $R^{-1}$ is parallely-bounded
- $R$ is a rational relation

How to control the number of steps of completion?
Derivation graph

Understand interaction in a derivation by using a graph
\[
\sigma_1 = aaaaab \rightarrow_1 aaaab \rightarrow_2 aaaa \rightarrow_3 aaa \rightarrow_4 aa
\]
Derivation graph

$aaaaab \rightarrow_1 aaaab \rightarrow_2 aa \textbf{aa} \rightarrow_3 aa \textbf{a} \rightarrow_4 \textbf{a} \textbf{a}$
Derivation graph

$aaaaab \xrightarrow{1} aaaaab \xrightarrow{2} aa\ a\ a \xrightarrow{3} a\ aa \xrightarrow{4} aa$
Derivation graph

\[
\begin{align*}
& \text{a}\framebox{aa} \text{ab} \longrightarrow_1 \text{aa}\framebox{a} \text{ab} \longrightarrow_2 \text{a} \framebox{aaa} \longrightarrow_3 \text{aaa} \longrightarrow_4 \text{aa}
\end{align*}
\]
\[ \sigma_2 = aaaaab \longrightarrow_1 aaaaab \longrightarrow_2 aaab \longrightarrow_3 aaa \longrightarrow_4 aa \]
Derivation graph

\[
\begin{align*}
\textit{aaaaab} & \rightarrow_1 1 \\
\textit{aaaab} & \rightarrow_2 \textit{aaaa} \\
& \rightarrow_3 \textit{aaa} \rightarrow_4 \textit{aa}
\end{align*}
\]
Causal equivalence [BKdVT03] : $\sigma_1 \sim \sigma_2$
Causal equivalence [BKdVT03] : \( \sigma_1 \sim \sigma_2 \implies G(\sigma_1) \cong G(\sigma_2) \)
Derivation graph

All topological sort of $G(\sigma)$ gives an equivalent derivation
Maxmatch-bounded srs
Maxmatch-bounded srs
Maxmatch-bounded srs
Maxmatch-bounded srs

Parallel complexity

Diagram:
- Node 0
  - Node 2
  - Node 3
  - Node 4
- Node 1
Maxmatch-bounded srs

Parallel complexity
= depth of $G$
Maxmatch-bounded srs

Definition

$R$ is $k$-maxmatch-bounded iff for any derivation $\sigma$, depth of $G(\sigma)$ is $\leq k$
Properties and membership

$R$ is maxmatch-bounded then:

For a given $k$, the problem of deciding if $R$ is maxmatch-bounded by $k$ is $\text{pspace}$-complete.
Properties and membership

$R$ is maxmatch-bounded then:

- $R$ is terminating
$R$ is maxmatch-bounded then:

- $R$ is terminating
- $R$ is parallely bounded
$R$ is maxmatch-bounded then:

- $R$ is terminating
- $R$ is parallely bounded
- $R^{-1}$ is maxmatch-bounded
Properties and membership

- If $R$ is maxmatch-bounded then:
  - $R$ is terminating
  - $R$ is parallely bounded
  - $R^{-1}$ is maxmatch-bounded

Main theorem

For a given $k$, the problem of deciding if a srs $R$ is maxmatch-bounded by $k$ is $\text{PSPACE}$-complete.
Contents

1 Motivation
2 Concepts in parallel rewriting
3 Main result : how to bound parallel complexity
4 Relation with Datalog
Idea of the proof

Membership theorem

For a given $k$, the problem of deciding is a srs $R$ is maxmatch-bounded by $k$ is PSPACE-complete.
Idea of the proof

Membership theorem
For a given $k$, the problem of deciding if a srs $R$ is maxmatch-bounded by $k$ is $\text{PSPACE}$-complete.

1. Guess a derivation of depth $k + 1$
Idea of the proof

Membership theorem
For a given $k$, the problem of deciding if a srs $R$ is maxmatch-bounded by $k$ is PSPACE-complete.

1. Guess a derivation of depth $k + 1$
2. Control the size of the derivation
Idea of the proof

Membership theorem

For a given $k$, the problem of deciding is a srs $R$ is maxmatch-bounded by $k$ is \texttt{PSPACE}-complete.

1. Guess a derivation of depth $k + 1$
2. Control the size of the derivation
3. Improve (space-)memory by "leftmost construction"
Goal: Find a model where the derivation has a bounded size
Conic derivation

Goal: Find a model where the derivation has a bounded size

Conic = all nodes are linked to the last one
Goal: Find a model where the derivation has a bounded size
Conic = all nodes are linked to the last one
Cleaned = all letters will be used in the derivation
Goal: Find a model where the derivation has a bounded size
Conic = all nodes are linked to the last one
Cleaned = all letters will be used in the derivation
Laddered srs

Definition:
A set of relations (SRS) is laddered if for all \( u \rightarrow v \),
\[
\forall a \in u, \quad b \in v, \quad f(b) = 1 + f(a)
\]

Example:
\( R = \{ a \rightarrow a + 1 \} \) is laddered by \( f(a) = t \).

Theorem:
\( R \in MB_{\max}(k) \) if and only if \( R_{Lad} \in MB_{\max}(2k+1) \).
Letters are leveled and if \( u \rightarrow v \) then \( \forall a \in u, b \in v \ f(b) = 1 + f(a) \).

Example: \( R = \{ a \rightarrow t, t \rightarrow a + 1 \} \) is laddered by \( f(a) = t \).

Theorem: \( R \in \text{MB}_{\text{max}}(k) \) if and only if \( R_{\text{Lad}} \in \text{MB}_{\text{max}}(2k + 1, \Sigma^*) \).
Laddered srs

Letters are leveled and

\[ u \rightarrow v \text{ then } \forall a \in u, b \in v \] \[ f(b) = 1 + f(a) \] \[ \sum^0_0, \sum^1_0, \sum^2_0, \ldots \]

Example: \[ R = \{ a \rightarrow a \} \] is laddered by \[ f(a) = t \]

Theorem \[ R \in \text{MB}_{\text{max}}(k) \iff R_{\text{Lad}} \in \text{MB}_{\text{max}}(2k+1, \Sigma^\ast_0) \]
Laddered $=$
Letters are leveled and
if $u \rightarrow v$ then

Example:

$\mathcal{R} = \{a \rightarrow a + 1\}$ is laddered by

$f(a) = \lambda$
Ladderedsrs

Ladderedsrs=Lettersareleveledand
ifu→vthen
∀a∈u,b∈v

Example:
R=\{a\text{at}a\rightarrow a\text{at}+1\}\is ladderedby
f(a) = t

Theorem
R∈MB\text{max}(k)iffR\text{Lad}∈MB\text{max}(2k+1,Σ∗0)}
Laddered srs

Letters are leveled and if \( u \rightarrow v \) then
\[
\forall a \in u, b \in v \Rightarrow f(b) = 1 + f(a)
\]

Example:
\[
R = \{a \rightarrow a + 1\}
\]
is laddered by
\[
f(a) = t
\]

Theorem
\[
R \in MB_{max}(k) \iff R_{Lad} \in MB_{max}(2k + 1, \Sigma^*)
\]
Laddered srs

Laddered =
Letters are leveled and
if $u \rightarrow v$ then
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$
$\sum_0$
Laddered srs

Laddered =
Letters are leveled and
if \( u \rightarrow v \) then
\[ \forall a \in u, b \in v \Rightarrow f(b) = 1 + f(a) \]
\[ \Sigma_0, \Sigma_1 \]
Laddered srs

Letters are leveled and if $u \rightarrow v$ then

$$\forall a \in u, b \in v \quad f(b) = 1 + f(a)$$

$$\Sigma_0, \Sigma_1, \Sigma_2...$$
Laddered srs

Letters are leveled and if \( u \rightarrow v \) then
\[
\forall a \in u, b \in v \quad f(b) = 1 + f(a)
\]
\[
\Sigma_0, \Sigma_1, \Sigma_2, \ldots
\]

Example:
\( R = \{a_t a_t \rightarrow a_{t+1}\} \) is laddered by \( f(a_t) = t \)
Laddered srs

Laddered =
Letters are leveled and
if \( u \to v \) then
\( \forall a \in u, b \in v \)
\( f(b) = 1 + f(a) \)
\( \Sigma_0, \Sigma_1, \Sigma_2... \)

Example:
\( R = \{a_t a_t \to a_{t+1}\} \) is laddered by \( f(a_t) = t \)

Theorem
\( R \in \text{MB}_{\text{max}}(k) \) iff \( R_{\text{Lad}} \in \text{MB}_{\text{max}}(2k + 1, \Sigma^*_0) \)
Idea: Control how to apply rewriting steps to keep a polynomial memory
Leftmost derivation

Idea: Control how to apply rewriting steps to keep a polynomial memory
Leftmost =
Leftmost derivation

Idea : Control how to apply rewriting steps to keep a polynomial memory

Leftmost = "Never rewrite a factor on the right to an other"
Idea: Control how to apply rewriting steps to keep a polynomial memory
Leftmost = "Never rewrite a factor on the right to an other" i.e. forbid:

\[
\begin{array}{c}
aa \\
 a \\
  a \\
  bb \\
 bb \\
 | a bb
\end{array}
\]
Leftmost derivation

\[ R = \{ a \rightarrow bb, \ bab \rightarrow c, \ b \rightarrow f \} \]

\[ \text{aaa} \xrightarrow{*} \text{fcf} \]
Leftmost derivation

\[ R = \{ a \rightarrow bb, \; bab \rightarrow c, \; b \rightarrow f \} \]

\[ aaa \xrightarrow{*} fcf \]
Leftmost derivation

\[ R = \{ a \rightarrow bb, bab \rightarrow c, b \rightarrow f \} \]

\[ aaa \rightarrow^* fcf \]
Leftmost derivation

\[ R = \{ a \rightarrow bb, \ bab \rightarrow c, \ b \rightarrow f \} \]

\[ aaa \xrightarrow{*} fcf \]
Leftmost derivation

\[ R = \{ a \rightarrow bb, \ bab \rightarrow c, b \rightarrow f \} \]

\[ aaa \rightarrow^\ast fcf \]
Leftmost derivation

\[ R = \{ a \rightarrow bb, \ bab \rightarrow c, \ b \rightarrow f \} \]

\[ aaa \rightarrow^* \ fcf \]
Leftmost derivation

\[ R = \{ a \rightarrow bb, \ bab \rightarrow c, \ b \rightarrow f \} \]

\[ aaa \rightarrow^* fcf \]
For all derivation there is a derivation:
For all derivation there is a derivation:

- with the same depth
For all derivations there is a derivation:
- with the same depth
- cleaned
For all derivation there is a derivation:
- with the same depth
- cleaned
- conic
For all derivation there is a derivation:

- with the same depth
- cleaned
- conic
- leftmost

on laddered systems:
PSPACE algorithm

For all derivation there is a derivation:

- with the same depth
- cleaned
- conic
- leftmost

on laddered systems:

\[ x_k = \begin{array}{c}
\text{no longer rewritten} \\
\left\{ u_1^k u_2^k \ldots u_i^k \right\} v_i^k v_{i-1}^k \ldots v_1^k v_0^k
\end{array} \]
PSPACE algorithm

For all derivation there is a derivation:

- with the same depth
- cleaned
- conic
- leftmost

on laddered systems:

\[
    x_k = \underbrace{u_1^k u_2^k \ldots u_i^k}_{\text{no longer rewritten}} v_i^k v_{i-1}^k \ldots v_1^k v_0^k
\]

for \( i \neq 0 : |v_i^k| < L + M \) with \( M = \max_{(l,r) \in R} |r| \)
For all derivation there is a derivation:

- with the same depth
- cleaned
- conic
- leftmost

on laddered systems:

\[ x_k = u_1^k u_2^k \ldots u_i^k v_i^k v_{i-1}^k \ldots v_1^k v_0^k \]

for \( i \neq 0 : \mid v_i^k \mid < L + M \) with \( M = \max_{(l,r) \in R} |r| \)
Contents

1. Motivation
2. Concepts in parallel rewriting
3. Main result: how to bound parallel complexity
4. Relation with Datalog
Control number of iterations in Datalog

**Definition**

$P$ is uniformly bounded by $k$ iff $\forall I, P^k(I) = P^{k+1}(I)$
Control number of iterations in Datalog

Definition

$P$ is uniformly bounded by $k$ iff $\forall I, P^k(I) = P^{k+1}(I)$

- Undecidable in general [AHV95]

Undecidable in arity 3 [HKMV95]

Still open in arity 2 [Mar99, GM14]
Control number of iterations in Datalog

**Definition**

$P$ is uniformly bounded by $k$ iff $\forall I, P^k(I) = P^{k+1}(I)$

- Undecidable in general [AHV95]
- Undecidable in arity 3 [HKMV95]
Control number of iterations in Datalog

Definition

\( P \) is uniformly bounded by \( k \) iff \( \forall I, P^k(I) = P^{k+1}(I) \)

- Undecidable in general [AHV95]
- Undecidable in arity 3 [HKMV95]
- Still open in arity 2 [Mar99, GM14]
Uniform boundedness is decidable for chain datalog [DG95]
Uniform boundedness is decidable for chain datalog [DG95]

\[ b(x, y) : -a_1(x, x_1) \land \cdots \land a_n(x_{n-1}, y) \]
Uniform boundedness is decidable for chain datalog [DG95]

\[ b(x, y) : \neg a_1(x, x_1) \land \cdots \land a_n(x_{n-1}, y) \land a_1a_2\ldots a_n \rightarrow b \in R \]
Uniform boundedness is decidable for chain datalog [DG95]

\[ b(x, y) : -a_1(x, x_1) \land \cdots \land a_n(x_{n-1}, y) \]
\[ a_1 a_2 \ldots a_n \rightarrow b \in R \]

**Datalog theorem**

Let \( R \) be an inverse context free rewriting system. Let \( P_R \) be the corresponding Datalog program. Let \( k \) be an integer. \( R \) is parallelly bounded by \( k \) iff \( P_R \) is uniform-bounded by \( k \)
Let $R$ be an inverse context free rewriting system. Let $P_R$ be the corresponding Datalog program. Let $k$ be an integer. $R$ is parallelly bounded by $k$ iff $P_R$ is uniform-bounded by $k$.
Let $R$ be an inverse context free rewriting system. Let $P_R$ be the corresponding Datalog program. Let $k$ be an integer. $R$ is parallelly bounded by $k$ iff $P_R$ is uniform-bounded by $k$. 

\[ \begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 2 & & \\
& & 4 \\
& & & 3 \\
\end{array} \]
Let $R$ be an inverse context free rewriting system. Let $P_R$ be the corresponding Datalog program. Let $k$ be an integer. $R$ is parallelly bounded by $k$ iff $P_R$ is uniform-bounded by $k$. 

\[
\begin{align*}
A_1(x, x_1) & \\ & \Downarrow \\
& A_1'(x, x_2) \\
& \Downarrow \\
& A''(x, y) \\
A_2(x_1, x_2) & \\ & \Downarrow \\
& A_2'(x_2, x_3) \\
& \Downarrow \\
& A_3(x_3, y) \\
A_3(x_2, x_3) & \\ & \Downarrow \\
& A_3'(x_3, y) \\
A_4(x_3, y) & \\ & \Downarrow \\
& A_4'(x_3, y)
\end{align*}
\]
Generalisation to word constraints

We can extend notion of uniform boundedness to non-unary word constraints
We can extend notion of uniform boundedness to non-unary word constraints (i.e. \( u \rightarrow v \in R \implies |v| \geq 2 \))
Generalisation to word constraints

We can extend notion of uniform boundedness to non-unary word constraints (i.e. $u \rightarrow v \in R \implies |v| \geq 2$)

$C_R$ is uniformly bounded by $k$ iff $R$ is $k$ maxmatch-bounded
Conclusion

Control parallel steps...
Control parallel steps...
→ allow us to compute ancestors
Control parallel steps...

→ allow us to compute ancestors

and help to decide $C \models xLy \sqsubseteq xL'y$
Control parallel steps...
→ allow us to compute ancestors
and help to decide $C \models xLy \sqsubseteq xL'y$
→ or understand chase completion in database theory
Control parallel steps...
→ allow us to compute ancestors and help to decide $C \models xLy \subseteq xL'y$
→ or understand chase completion in database theory

\[ a \rightarrow aa \notin MB_{\text{max}} \]
Control parallel steps...

→ allow us to compute ancestors
and help to decide $C \models xLy \subseteq xL'y$

→ or understand chase completion in database theory

\[ a \rightarrow aa \notin MB_{\text{max}} \]

\[ \models xa^+y \subseteq xay \]
Open question:

\[ R \in MB_{\text{max}} \text{ with other srs classes} \]

Link with tuple generating dependencies and Datalog

More general rewriting system for RPQ optimization

\( a + \rightarrow a \)
Open question: Decide "$R \in MB_{\text{max}}"$?
Open question: Decide "$R \in MB_{\text{max}}$"?
$MB_{\text{max}}$ with other srs classes
Open question: Decide "$R \in MB_{\text{max}}"$?

$MB_{\text{max}}$ with other srs classes

Link with tuple generating dependacies and Datalog
Open question: Decide "$R \in MB_{max}\$"?
$MB_{max}$ with other srs classes
Link with tuple generating dependencies and Datalog
More general rewriting system for RPQ optimization
Open question: Decide "$R \in MB_{\text{max}}"$?

$MB_{\text{max}}$ with other srs classes

Link with tuple generating dependencies and Datalog

More general rewriting system for RPQ optimization

\((a^+ \rightarrow a)\)


Thank you!
\[ H(x) : \neg \alpha_1(x_1) \land \cdots \land \alpha_p(x_p) \]
$H(x) : \neg \alpha_1(x_1) \land \cdots \land \alpha_p(x_p)$

$\text{FoF}(x, y) \leftarrow \text{Friend}(x, y)$
$\text{FoF}(x, y) \leftarrow \text{FoF}(x, z) \land \text{FoF}(z, y)$
$$H(x) : -\alpha_1(x_1) \land \cdots \land \alpha_p(x_p)$$

$$\text{FoF}(x, y) \leftarrow \text{Friend}(x, y)$$
$$\text{FoF}(x, y) \leftarrow \text{FoF}(x, z) \land \text{FoF}(z, y)$$

$I = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos})$. 
Datalog

\[ H(x) : -\alpha_1(x_1) \land \cdots \land \alpha_p(x_p) \]

\[ \text{FoF}(x, y) \leftarrow \text{Friend}(x, y) \]
\[ \text{FoF}(x, y) \leftarrow \text{FoF}(x, z) \land \text{FoF}(z, y) \]

\[ I = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}). \]
\[ P(I) = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Bob}), \text{FoF}(\text{Bob}, \text{Carlos}). \]
Datalog

\[ H(x) : -\alpha_1(x_1) \land \cdots \land \alpha_p(x_p) \]

\[
\begin{align*}
\text{FoF}(x, y) & \leftarrow \text{Friend}(x, y) \\
\text{FoF}(x, y) & \leftarrow \text{FoF}(x, z) \land \text{FoF}(z, y)
\end{align*}
\]

\[ I = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}). \]

\[ P(I) = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Bob}), \text{FoF}(\text{Bob}, \text{Carlos}). \]

\[ P^2(I) = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Bob}), \text{FoF}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Carlos}). \]
$H(x) : -\alpha_1(x_1) \land \cdots \land \alpha_p(x_p)$

$\text{FoF}(x, y) \leftarrow \text{Friend}(x, y)$
$\text{FoF}(x, y) \leftarrow \text{FoF}(x, z) \land \text{FoF}(z, y)$

$I = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}).$

$P(I) = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Bob}), \text{FoF}(\text{Bob}, \text{Carlos}).$

$P^2(I) = \text{Friend}(\text{Alice}, \text{Bob}), \text{Friend}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Bob}), \text{FoF}(\text{Bob}, \text{Carlos}), \text{FoF}(\text{Alice}, \text{Carlos}).$

Finally, $P^2(I) = P^3(I)$