String Parallel Rewriting : analysis of the structure of the derivations

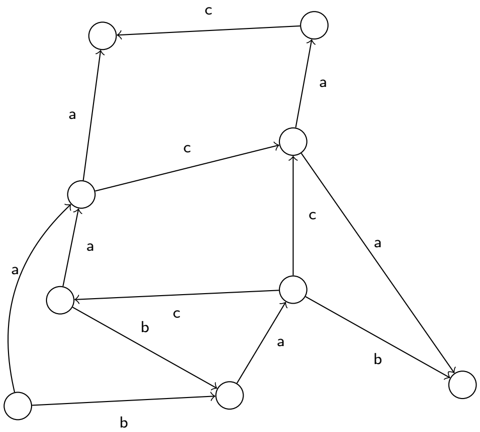
P. Bourhis, D. Gallois, S.Tison

Meeting DELTA

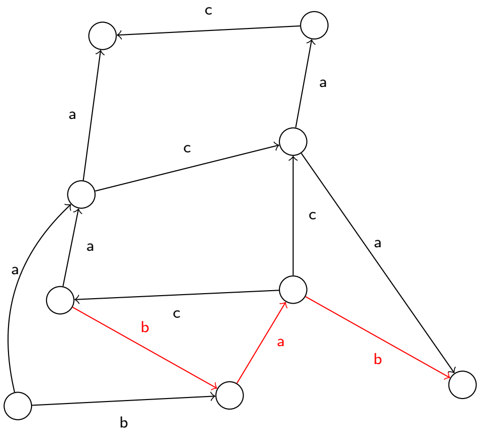
26-28 Mars 2018

Contents

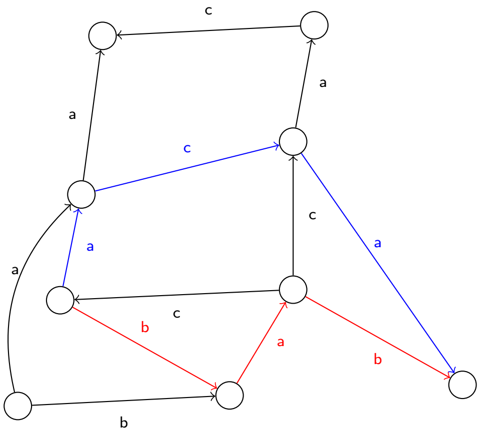
Word constraints



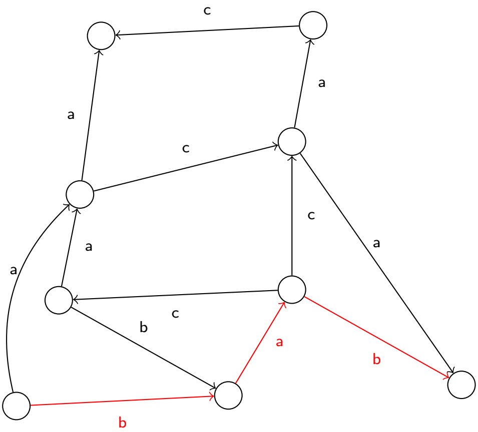
Word constraints



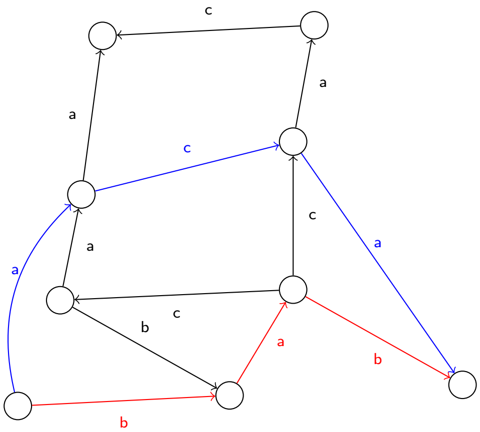
Word constraints



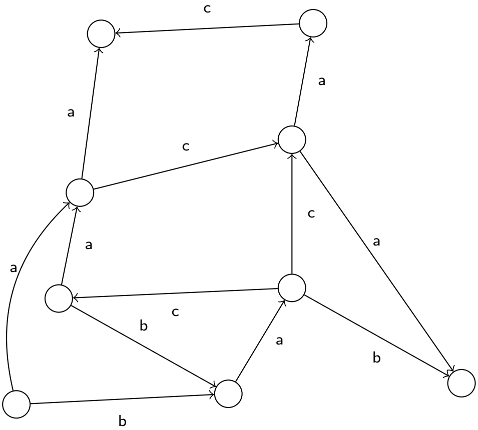
Word constraints



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

$xLy =$ pair of nodes linked by a path labelled in L

Problem

xLy = 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 :

Problem

xLy = pair of nodes linked by a path labelled in L

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

Example :

$C = \{a(x, y) \to b(x, y)\}$

Problem

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

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

$$b(x, x_1)a(x_1, x_2)b(x_2, y) \to \exists z_1, z_2, a(x, z_1)c(z_1, z_2)a(z_2, y)$$

$$bab \to aca$$

Word constraints and RPQ

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

$$bab \rightarrow aca$$

$\Rightarrow$ Reduction to string rewriting system

Word constraints and RPQ

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

$$bab \rightarrow aca$$

$\Rightarrow$ Reduction to string rewriting system

Theorem [GT03]

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

Word constraints and RPQ

$$b(x, x_1)a(x_1, x_2)b(x_2, y) \to \exists z_1, z_2, a(x, z_1)c(z_1, z_2)a(z_2, y)$$

$$bab \to aca$$

$\Rightarrow$ Reduction to string rewriting system

Theorem [GT03]

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

Word constraints and RPQ

$$b(x, x_1)a(x_1, x_2)b(x_2, y) \to \exists z_1, z_2, a(x, z_1)c(z_1, z_2)a(z_2, y)$$

$$bab \to aca$$

$\Rightarrow$ Reduction to string rewriting system

Theorem [GT03]

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

Compute efficiently $\mathrm{Anc}_R(L')$ by completion

Word constraints and RPQ

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

$$bab \rightarrow aca$$

$\Rightarrow$ Reduction to string rewriting system

Theorem [GT03]

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

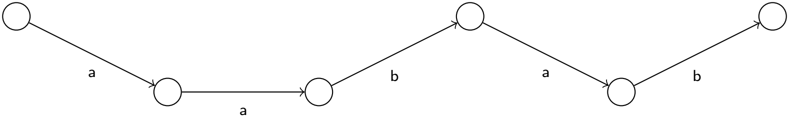
Compute efficiently $\mathrm{Anc}_R(L')$ by completion keeping worried by completion in database theory

Word constraints and completion

$$ab \to bc$$

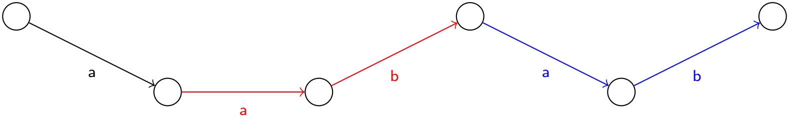
Word constraints and completion

$$ab \to bc$$



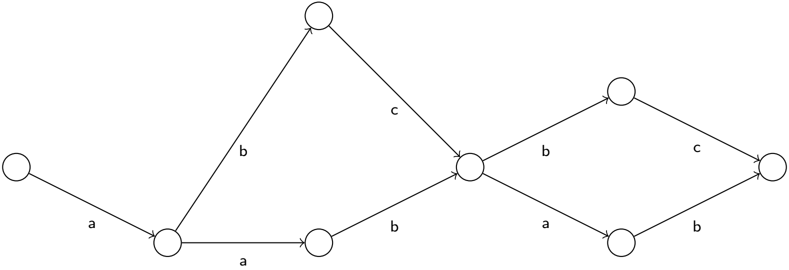
Word constraints and completion

$$ab \to bc$$



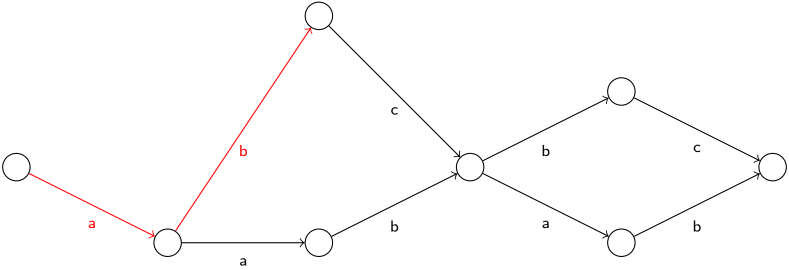
Word constraints and completion

$$ab \to bc$$



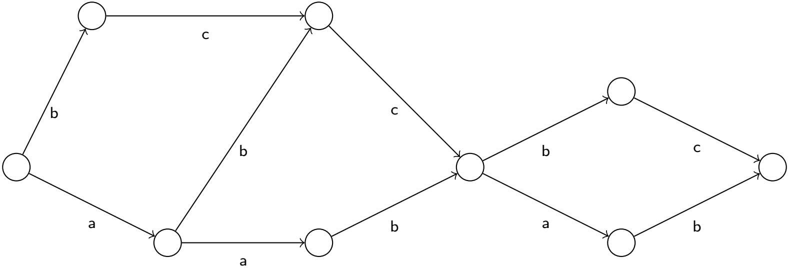
Word constraints and completion

$$ab \to bc$$



Word constraints and completion

$$ab \to bc$$



Contents

String parallel rewriting

Once upon a time...

String parallel rewriting

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

String parallel rewriting

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

frog

String parallel rewriting

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

$$\boxed{\mathsf{f}}\,\mathsf{rog}$$

String parallel rewriting

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

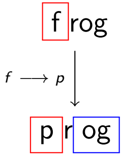
f rog

$f \longrightarrow p$ ↓

p rog

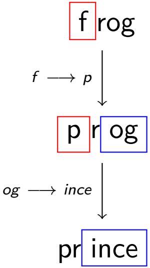
String parallel rewriting

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



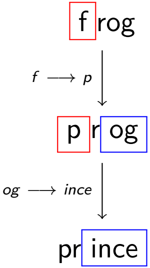
String parallel rewriting

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



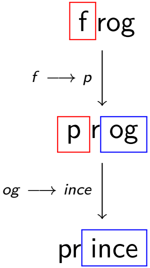
String parallel rewriting

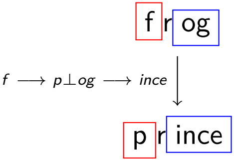
Once upon a time... $R = \{\mathrm{f} \to \mathrm{p}, \mathrm{og} \to \mathrm{ince}\}$



String parallel rewriting

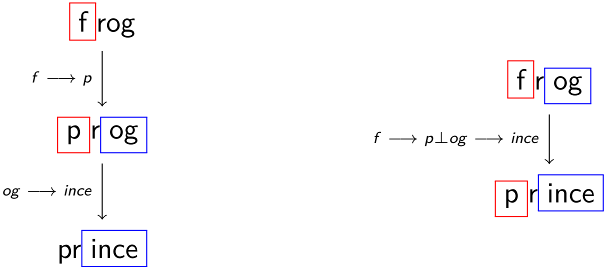
Once upon a time... $R = \{\mathsf{f} \to \mathsf{p}, \mathsf{og} \to \mathsf{ince}\}$





String parallel rewriting

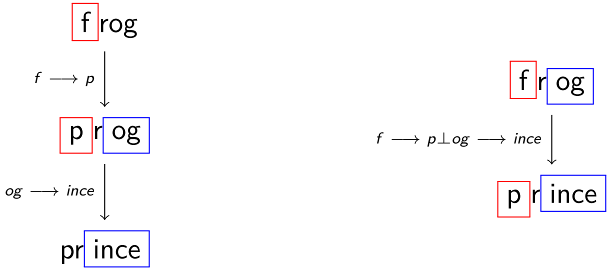
Once upon a time... $R = \{\mathrm{f} \to \mathrm{p}, \mathrm{og} \to \mathrm{ince}\}$



$$\mathrm{frog} \longrightarrow\!\!\!\!\!\!\!\ominus\;\; \mathrm{prince}$$

String parallel rewriting

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



$$\text{frog} \longrightarrow\!\!\!\!\!\!\!\ominus\;\; \text{prince}$$

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

srs parallely bounded

Definition

A srs is parallely bounded by k iff $\mathrel{-\!\!\ominus\!\!\rightarrow}^k = \mathrel{-\!\!\ominus\!\!\rightarrow}^*$

srs parallely bounded

Definition

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

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

srs parallely bounded

Definition

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

$R = \{a \to b, b \to a\}$ is paralelly bounded by 1

- R^{-1} is parallely-bounded

srs parallely bounded

Definition

A srs is parallely bounded by k iff $\mathrel{-\!\!\ominus\!\!\rightarrow}^k = \mathrel{-\!\!\ominus\!\!\rightarrow}^*$

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

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

srs parallely bounded

Definition

A srs is parallely bounded by k iff $\twoheadrightarrow\!\!\!\!\!\!\ominus^k = \twoheadrightarrow\!\!\!\!\!\!\ominus^*$

$R = \{a \to b, b \to a\}$ is paralelly bounded by 1

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

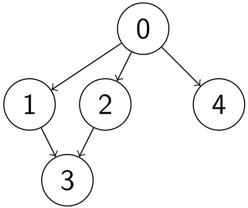
How to control the number of steps of completion ?

Derivation graph

Understand interaction in a derivation by using a graph

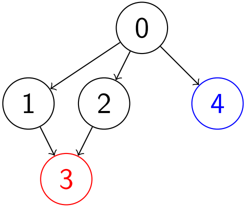
Derivation graph

$$\sigma_1 = aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaaa \longrightarrow_3 aaa \longrightarrow_4 aa$$



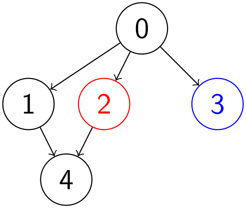
Derivation graph

$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aa\,\boxed{\text{aa}} \longrightarrow_3 \boxed{\text{aa}}\,\boxed{\text{a}} \longrightarrow_4 \boxed{\text{a}}\,a$$



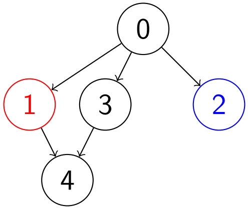
Derivation graph

$$aaaaab \longrightarrow_1 aaa\boxed{\mathrm{ab}} \longrightarrow_2 \boxed{\mathrm{aa}}\,a\boxed{\mathrm{a}} \longrightarrow_3 \boxed{\mathrm{a}}\,aa \longrightarrow_4 aa$$



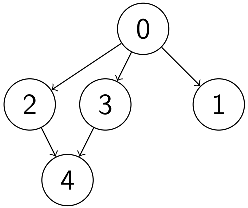
Derivation graph

$$a\,\boxed{aa}\,ab \longrightarrow_1 \boxed{aa}\,\boxed{a}\,ab \longrightarrow_2 \boxed{a}\,aaa \longrightarrow_3 aaa \longrightarrow_4 aa$$



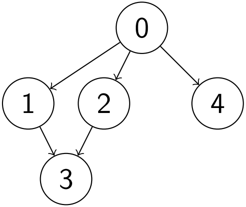
Derivation graph

$$\sigma_2 = aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaab \longrightarrow_3 aaa \longrightarrow_4 aa$$

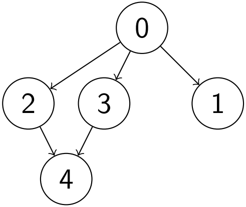


Derivation graph

$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaaa \longrightarrow_3 aaa \longrightarrow_4 aa$$

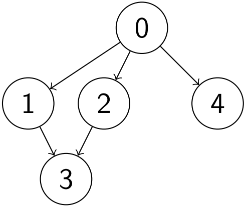


$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaab \longrightarrow_3 aaa \longrightarrow_4 aa$$

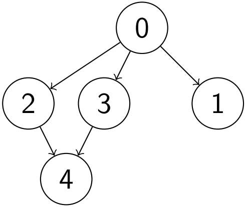


Derivation graph

$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaaa \longrightarrow_3 aaa \longrightarrow_4 aa$$



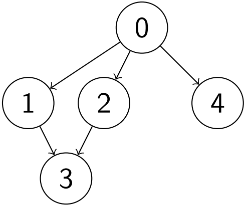
$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaab \longrightarrow_3 aaa \longrightarrow_4 aa$$



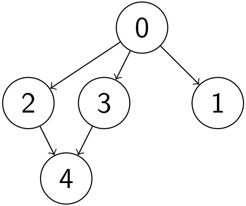
Causal equivalence [BKdVT03] : $\sigma_1 \sim \sigma_2$

Derivation graph

$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaaa \longrightarrow_3 aaa \longrightarrow_4 aa$$



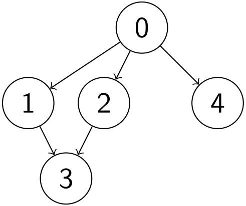
$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaab \longrightarrow_3 aaa \longrightarrow_4 aa$$



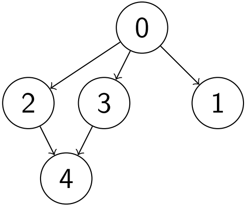
Causal equivalence [BKdVT03] : $\sigma_1 \sim \sigma_2 \Longrightarrow G(\sigma_1) \cong G(\sigma_2)$

Derivation graph

$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaaa \longrightarrow_3 aaa \longrightarrow_4 aa$$

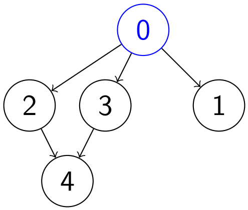


$$aaaaab \longrightarrow_1 aaaab \longrightarrow_2 aaab \longrightarrow_3 aaa \longrightarrow_4 aa$$

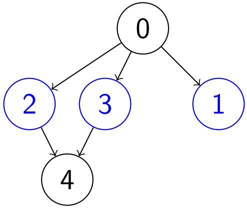


All topological sort of $G(\sigma)$ gives an equivalent derivation

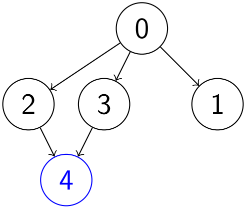
Maxmatch-bounded srs



Maxmatch-bounded srs

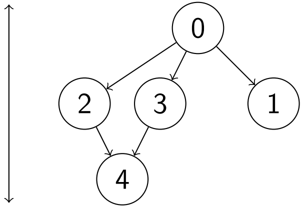


Maxmatch-bounded srs



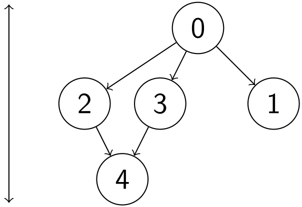
Maxmatch-bounded srs

Parallel complexity



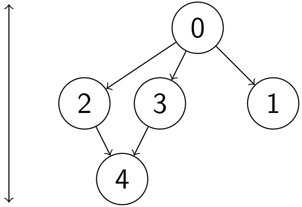
Maxmatch-bounded srs

Parallel complexity
$=$ depth of G



Maxmatch-bounded srs

Parallel complexity
$=$ depth of G



Definition

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

Properties and membership

R is maxmatch-bounded then :

Properties and membership

R is maxmatch-bounded then :

- R is terminating

Properties and membership

R is maxmatch-bounded then :

- R is terminating
- R is parallely bounded

Properties and membership

R is maxmatch-bounded then :

- R is terminating
- R is parallely bounded
- R^{-1} is maxmatch-bounded

Properties and membership

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 is a srs R is maxmatch-bounded by k is PSPACE-complete.

Contents

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 is a srs R is maxmatch-bounded by k is PSPACE-complete.

1. Guess a derivation of depth $k + 1$

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.

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 PSPACE-complete.

1. Guess a derivation of depth $k + 1$
2. Control the size of the derivation
3. Improve (space-)memory by "leftmost construction"

Conic derivation

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

Conic derivation

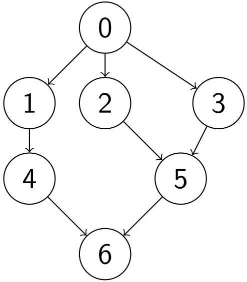
Goal : Find a model where the derivation has a bounded size
Conic = all nodes are linked to the last one
Cleaned = all letters will be used in the derivation

Conic derivation

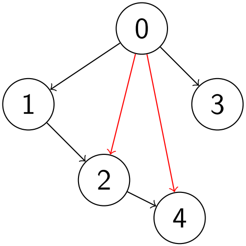
Goal : Find a model where the derivation has a bounded size

Conic = all nodes are linked to the last one

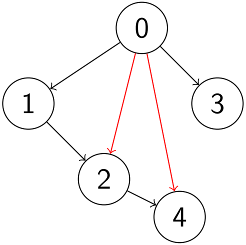
Cleaned = all letters will be used in the derivation



Laddered srs

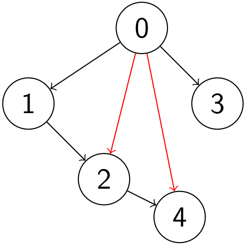


Laddered srs



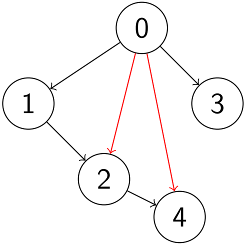
Laddered $=$

Laddered srs



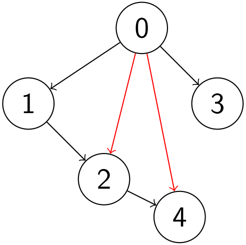
Laddered =
Letters are leveled and

Laddered srs



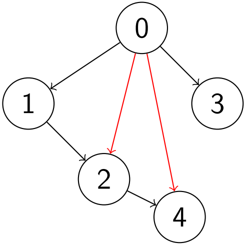
Laddered =
Letters are leveled and
if $u \to v$ then

Laddered srs



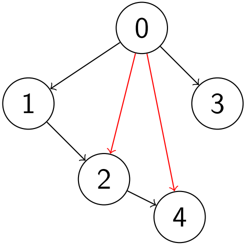
Laddered =
Letters are leveled and
if $u \to v$ then
$\forall a \in u, b \in v$

Laddered srs



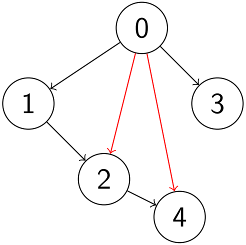
Laddered =
Letters are leveled and
if $u \to v$ then
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$

Laddered srs



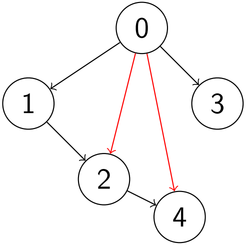
Laddered =
Letters are leveled and
if $u \to v$ then
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$
Σ_0

Laddered srs



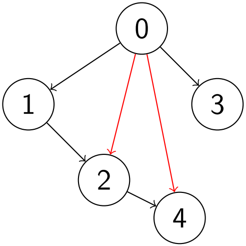
Laddered =
Letters are leveled and
if $u \to v$ then
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$
Σ_0 , Σ_1

Laddered srs



Laddered =
Letters are leveled and
if $u \to v$ then
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$
Σ_0 , Σ_1 , $\Sigma_2 \ldots$

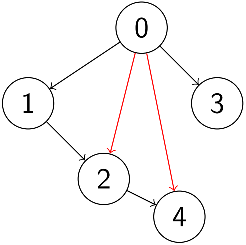
Laddered srs



Laddered =
Letters are leveled and
if $u \rightarrow v$ then
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$
Σ_0 , Σ_1 , Σ_2...

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
$\forall a \in u, b \in v$
$f(b) = 1 + f(a)$
Σ_0 , Σ_1 , Σ_2...

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

Theorem

$R \in \mathrm{MB}_{\max}(k)$ iff $R_{\mathrm{Lad}} \in \mathrm{MB}_{\max}(2k+1, \Sigma_0^*)$

Leftmost derivation

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"

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" i.e. forbid :

aa a

a *a* bb

bb *abb*

Leftmost derivation

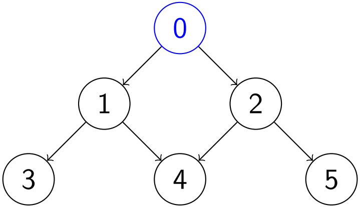
$$R = \{a \rightarrow bb, bab \rightarrow c, b \rightarrow f\}$$
$$aaa \longrightarrow^* fcf$$

Leftmost derivation

$$R = \{a \to bb, bab \to c, b \to f\}$$
$$aaa \longrightarrow^* fcf$$

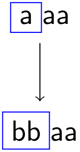
aaa

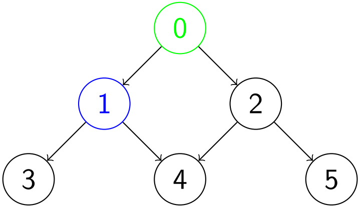
$\downarrow$



Leftmost derivation

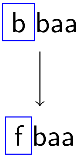
$$R = \{a \to bb, bab \to c, b \to f\}$$
$$aaa \longrightarrow^* fcf$$

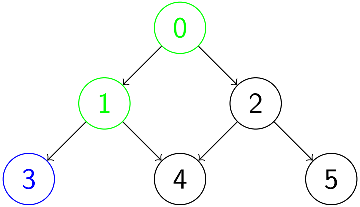




Leftmost derivation

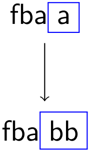
$$R = \{a \to bb, bab \to c, b \to f\}$$
$$aaa \longrightarrow^* fcf$$

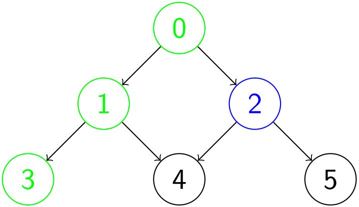




Leftmost derivation

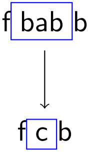
$$R = \{a \to bb, bab \to c, b \to f\}$$
$$aaa \longrightarrow^* fcf$$

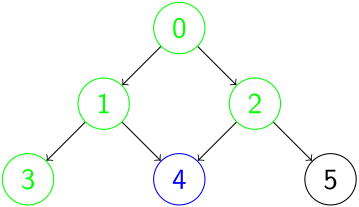




Leftmost derivation

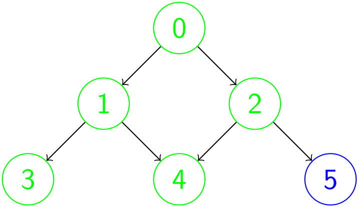
$$R = \{a \to bb, bab \to c, b \to f\}$$
$$aaa \longrightarrow^* fcf$$





Leftmost derivation

$$R = \{a \rightarrow bb, bab \rightarrow c, b \rightarrow f\}$$
$$aaa \longrightarrow^* fcf$$



PSPACE algorithm

For all derivation there is a derivation :

PSPACE algorithm

For all derivation there is a derivation :

- with the same depth

PSPACE algorithm

For all derivation there is a derivation :

- with the same depth
- cleaned

PSPACE algorithm

For all derivation there is a derivation :

- with the same depth
- cleaned
- conic

PSPACE algorithm

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 :

$$x_k = \overbrace{u_1^k u_2^k \ldots u_i^k}^{\text{no longer rewritten}} \quad v_i^k v_{i-1}^k \ldots v_1^k v_0^k$$

PSPACE algorithm

For all derivation there is a derivation :

- with the same depth
- cleaned
- conic
- leftmost

on laddered systems :

$$x_k = \overbrace{u_1^k u_2^k \ldots u_i^k}^{\text{no longer rewritten}} \quad v_i^k v_{i-1}^k \ldots v_1^k v_0^k$$

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

PSPACE algorithm

For all derivation there is a derivation :

- with the same depth
- cleaned
- conic
- leftmost

on laddered systems :

$$x_k = u_1^k u_2^k \dots u_i^k \underbrace{v_i^k v_{i-1}^k \dots v_1^k}_{\text{memory}} v_0^k$$

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

Contents

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]

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]

Chain Datalog case

Uniform boundedness is decidable for chain datalog [DG95]

Chain Datalog case

Uniform boundedness is decidable for chain datalog [DG95]

$$b(x, y) :- a_1(x, x_1) \wedge \cdots \wedge a_n(x_{n-1}, y)$$

Chain Datalog case

Uniform boundedness is decidable for chain datalog [DG95]

$$b(x, y) : -a_1(x, x_1) \wedge \cdots \wedge a_n(x_{n-1}, y)$$

$$a_1 a_2 ... a_n \rightarrow b \in R$$

Chain Datalog case

Uniform boundedness is decidable for chain datalog [DG95]

$$b(x, y) :- a_1(x, x_1) \wedge \cdots \wedge a_n(x_{n-1}, y)$$

$$a_1 a_2 \ldots a_n \rightarrow b \in R$$

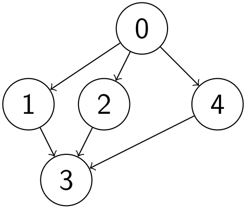
Datalog theorem

Let R be an inverse context free rewriting system. Let P_R be the corresponding Datalog program. Let k be an integer. R is paralelly bounded by k iff P_R is uniform-bounded by k

Chain Datalog case

Datalog theorem

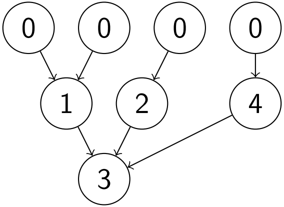
Let R be an inverse context free rewriting system. Let P_R be the corresponding Datalog program. Let k be an integer. R is paralelly bounded by k iff P_R is uniform-bounded by k



Chain Datalog case

Datalog theorem

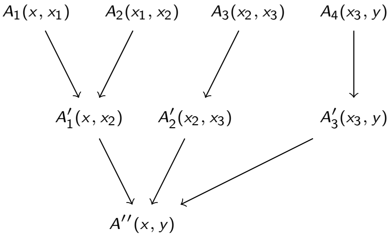
Let R be an inverse context free rewriting system. Let P_R be the corresponding Datalog program. Let k be an integer. R is paralelly bounded by k iff P_R is uniform-bounded by k



Chain Datalog case

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 paralelly bounded by k iff P_R is uniform-bounded by k



Generalisation to word constraints

We can extend notion of uniform boundedness to non-unary word constraints

Generalisation to word constraints

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

Generalisation to word constraints

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

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

Conclusion

Control parallel steps...

Conclusion

Control parallel steps...
$\rightarrow$ allow us to compute ancestors

Conclusion

Control parallel steps...
$\rightarrow$ allow us to compute ancestors
and help to decide $C \models xLy \sqsubseteq xL'y$

Conclusion

Control parallel steps...

$\rightarrow$ allow us to compute ancestors

and help to decide $C \models xLy \sqsubseteq xL'y$

$\rightarrow$ or understand chase completion in database theory

Conclusion

Control parallel steps...

$\rightarrow$ allow us to compute ancestors

and help to decide $C \models xLy \sqsubseteq xL'y$

$\rightarrow$ or understand chase completion in database theory

$$a \rightarrow aa \notin \mathrm{MB}_{\mathrm{max}}$$

Conclusion

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

$$a \to aa \notin \mathsf{MB}_{\mathsf{max}}$$

$$\models xa^{+}y \sqsubseteq xay$$

Ongoing & Future Work

Open question :

Ongoing & Future Work

Open question : Decide "$R \in \mathsf{MB}_{\mathsf{max}}$" ?

Ongoing & Future Work

Open question : Decide "$R \in \mathrm{MB}_{\mathrm{max}}$" ?

$\mathrm{MB}_{\mathrm{max}}$ with other srs classes

Ongoing & Future Work

Open question : Decide "$R \in \mathrm{MB}_{\mathrm{max}}$" ?

$\mathrm{MB}_{\mathrm{max}}$ with other srs classes

Link with tuple generating dependancies and Datalog

Ongoing & Future Work

Open question : Decide "$R \in \mathrm{MB}_{\mathrm{max}}$"?

$\mathrm{MB}_{\mathrm{max}}$ with other srs classes

Link with tuple generating dependancies and Datalog

More general rewriting system for RPQ optimization

Ongoing & Future Work

Open question : Decide "$R \in \mathrm{MB}_{\mathrm{max}}$" ?
$\mathrm{MB}_{\mathrm{max}}$ with other srs classes
Link with tuple generating dependancies and Datalog
More general rewriting system for RPQ optimization
$(a^+ \to a)$

References I

- Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of databases*, Addison-Wesley, 1995.
- M. Bezem, J.W. Klop, R. de Vrijer, and Terese, *Term rewriting systems*, Cambridge Tracts in Theoretica, Cambridge University Press, 2003.
- Guozhu Dong and Seymour Ginsburg, *On decompositions of chain datalog programs into p (left-)linear 1-rule components*, The Journal of Logic Programming **23** (1995), no. 3, 203 – 236.
- Joseph Goguen, Claude Kirchner, and José Meseguer, *Concurrent term rewriting as a model of computation*, pp. 53–93, Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.
- Tomasz Gogacz and Jerzy Marcinkowski, *All–instances termination of chase is undecidable*, pp. 293–304, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

References II

- Gösta Grahne and Alex Thomo, *Query containment and rewriting using views for regular path queries under constraints*, Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA, 2003, pp. 111–122.
- Gerd G Hillebrand, Paris C Kanellakis, Harry G Mairson, and Moshe Y Vardi, *Undecidable boundedness problems for datalog programs*, The Journal of Logic Programming **25** (1995), no. 2, 163 – 190.
- Claude Kirchner and Patrick Viry, *Implementing parallel rewriting*, pp. 1–15, Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.
- Jerzy Marcinkowski, *Achilles, turtle, and undecidable boundedness problems for small DATALOG programs*, SIAM J. Comput. **29** (1999), no. 1, 231–257.

Thank you !

Datalog

$$H(x) :- \alpha_1(x_1) \wedge \cdots \wedge \alpha_p(x_p)$$

Datalog

$$H(x) :- \alpha_1(x_1) \wedge \cdots \wedge \alpha_p(x_p)$$

$$\mathrm{FoF}(x, y) \longleftarrow \mathrm{Friend}(x, y)$$
$$\mathrm{FoF}(x, y) \longleftarrow \mathrm{FoF}(x, z) \wedge \mathrm{FoF}(z, y)$$

Datalog

$$H(x) :- \alpha_1(x_1) \wedge \cdots \wedge \alpha_p(x_p)$$

$$\text{FoF}(x, y) \longleftarrow \text{Friend}(x, y)$$
$$\text{FoF}(x, y) \longleftarrow \text{FoF}(x, z) \wedge \text{FoF}(z, y)$$
$$I = \text{Friend(Alice,Bob), Friend(Bob,Carlos).}$$

Datalog

$$H(x) : -\alpha_1(x_1) \wedge \cdots \wedge \alpha_p(x_p)$$

$$\text{FoF}(x, y) \longleftarrow \text{Friend}(x, y)$$
$$\text{FoF}(x, y) \longleftarrow \text{FoF}(x, z) \wedge \text{FoF}(z, y)$$
$$I = \text{Friend(Alice,Bob), Friend(Bob,Carlos).}$$
$$P(I) = \text{Friend(Alice,Bob), Friend(Bob,Carlos), FoF(Alice,Bob),}$$
$$\text{FoF(Bob,Carlos).}$$

Datalog

$$H(x) :- \alpha_1(x_1) \wedge \cdots \wedge \alpha_p(x_p)$$

$$\text{FoF}(x, y) \longleftarrow \text{Friend}(x, y)$$
$$\text{FoF}(x, y) \longleftarrow \text{FoF}(x, z) \wedge \text{FoF}(z, y)$$
$$I = \text{Friend(Alice,Bob), Friend(Bob,Carlos).}$$
$$P(I) = \text{Friend(Alice,Bob), Friend(Bob,Carlos), FoF(Alice,Bob), FoF(Bob,Carlos).}$$
$$P^2(I) = \text{Friend(Alice,Bob), Friend(Bob,Carlos), FoF(Alice,Bob), FoF(Bob,Carlos), FoF(Alice,Carlos).}$$

Datalog

$$H(x) :- \alpha_1(x_1) \wedge \cdots \wedge \alpha_p(x_p)$$

$$\text{FoF}(x, y) \longleftarrow \text{Friend}(x, y)$$

$$\text{FoF}(x, y) \longleftarrow \text{FoF}(x, z) \wedge \text{FoF}(z, y)$$

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

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

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

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