Chase Termination Beyond Polynomial Time

43rd Symposium on Principles of Database Systems (PODS'24)

Philipp Hanisch, Markus Krötzsch
TU Dresden, Knowledge-Based Systems Group
12th June 2024

Introduction

Tuple-generating dependencies (tgds): rules with existential quantifiers

Introduction

Tuple-generating dependencies (tgds): rules with existential quantifiers

Example

$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \to \exists V.\, \mathit{add}(e, S, V) \tag{1}$$

$$\mathit{add}(e, S, T) \to \mathit{add}(e, T, T) \tag{2}$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \to \mathit{add}(e, T, T) \tag{3}$$

$$\mathit{add}(e, S, T) \to \mathit{set}(T) \tag{4}$$

Introduction

Tuple-generating dependencies (tgds): rules with existential quantifiers

Example

$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \to \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \to \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \to \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \to \mathit{set}(T) \quad (4)$$

Chase: 'apply rules until nothing new follows'
$\to$ but might run forever

Motivation

Termination criteria are sufficient conditions for termination.

Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$

$\emptyset$

Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

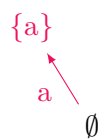
$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

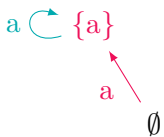
$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

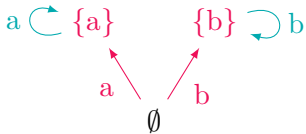
$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

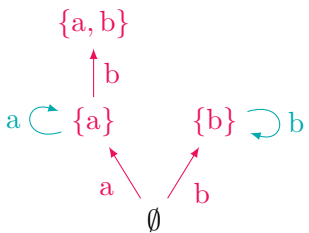
$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

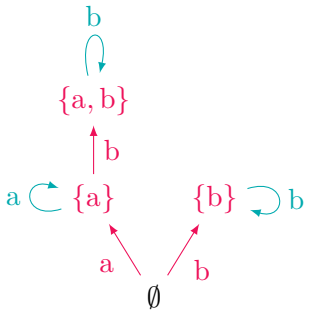
$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



Motivation

Termination criteria are sufficient conditions for termination.

All known criteria lead to decidable classes of tgds with chase termination in PTime, usually corresponding to Datalog queries [Zhang et al., AAAI'15; K. et al., ICDT'19].

Yet the set of all standard-chase terminating tgds captures all homomorphism-closed decidable queries, but is not semi-decidable [Bourgaux et al., KR'21].

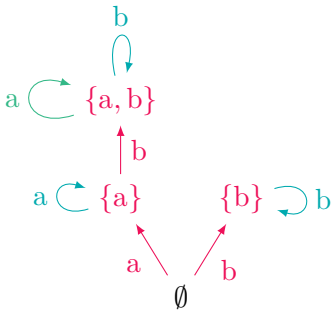
$$\mathcal{D} = \{\mathit{set}(\emptyset), \mathit{elem}(\mathrm{a}), \mathit{elem}(\mathrm{b}), \mathit{elem}(\mathrm{c})\}$$

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$

$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$

$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



Analysing termination

Chase: relational structure of nulls,
starting from a database

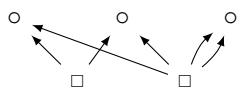
Analysing termination

Chase: relational structure of nulls,
starting from a database

□ □

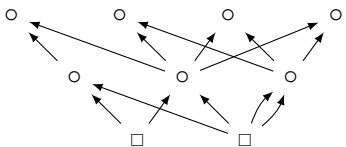
Analysing termination

Chase: relational structure of nulls,
starting from a database



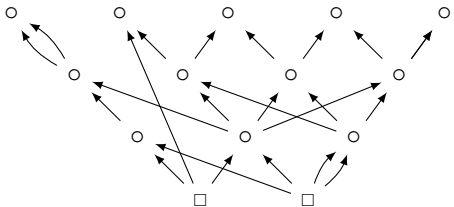
Analysing termination

Chase: relational structure of nulls,
starting from a database



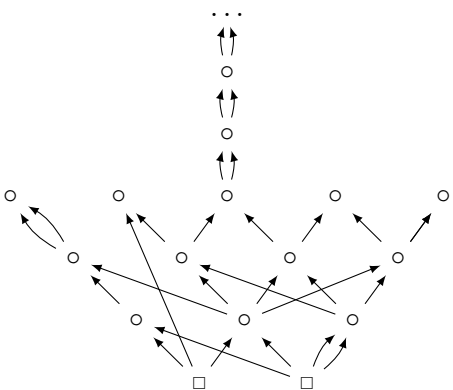
Analysing termination

Chase: relational structure of nulls,
starting from a database



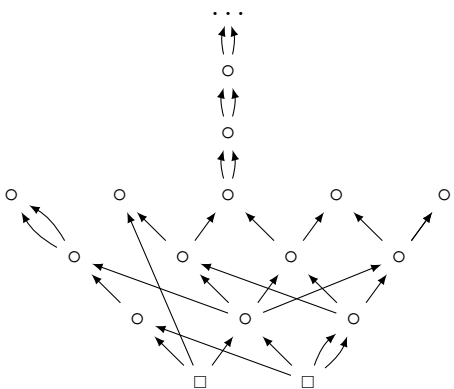
Analysing termination

Chase: relational structure of nulls,
starting from a database



Analysing termination

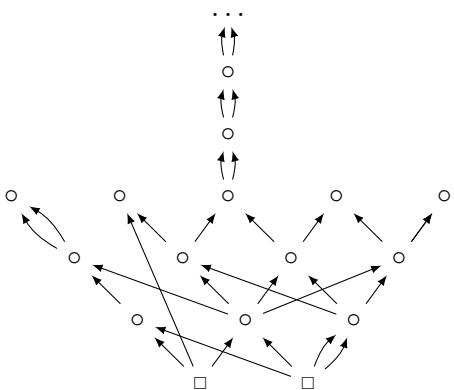
Chase: relational structure of nulls, starting from a database



Abstraction of the chase structure
$\rightarrow$ dependency graph

Analysing termination

Chase: relational structure of nulls, starting from a database

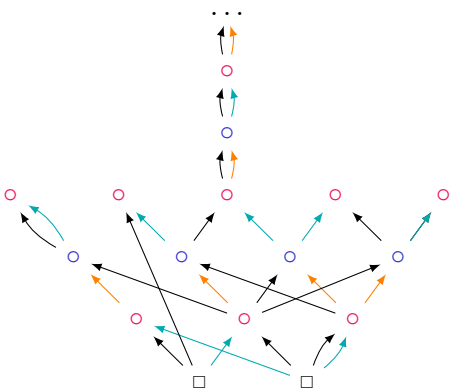


Abstraction of the chase structure
→ dependency graph

$$p(x_1, x_2) \to \exists v.\, q(x_2, v)$$
$$q(y_1, y_2) \to \exists w.\, p(y_2, w)$$
$$\ldots$$

Analysing termination

Chase: relational structure of nulls, starting from a database

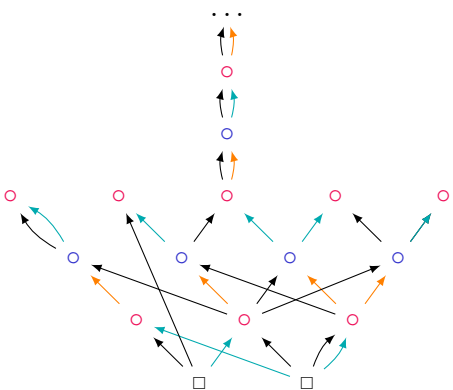


Abstraction of the chase structure
→ dependency graph

$$p(x_1, x_2) \to \exists v.\, q(x_2, v)$$
$$q(y_1, y_2) \to \exists w.\, p(y_2, w)$$
$$\ldots$$

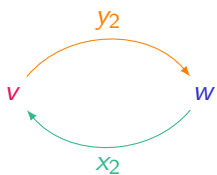
Analysing termination

Chase: relational structure of nulls, starting from a database



Abstraction of the chase structure
→ dependency graph

$$p(x_1, x_2) \to \exists v.\, q(x_2, v)$$
$$q(y_1, y_2) \to \exists w.\, p(y_2, w)$$
$$\dots$$



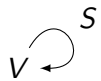
(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V) \quad (1)$$
$$add(e, S, T) \rightarrow add(e, T, T) \quad (2)$$
$$add(e, S, S) \wedge add(f, S, T) \rightarrow add(e, T, T) \quad (3)$$
$$add(e, S, T) \rightarrow set(T) \quad (4)$$



(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

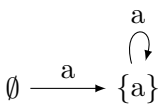
$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V) \quad (1)$$
$$add(e, S, T) \rightarrow add(e, T, T) \quad (2)$$
$$add(e, S, S) \wedge add(f, S, T) \rightarrow add(e, T, T) \quad (3)$$
$$add(e, S, T) \rightarrow set(T) \quad (4)$$

$\emptyset$

(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

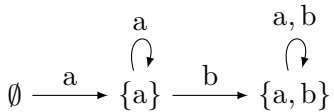
$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V) \quad (1)$$
$$add(e, S, T) \rightarrow add(e, T, T) \quad (2)$$
$$add(e, S, S) \wedge add(f, S, T) \rightarrow add(e, T, T) \quad (3)$$
$$add(e, S, T) \rightarrow set(T) \quad (4)$$



(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

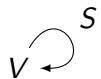
$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V) \quad (1)$$
$$add(e, S, T) \rightarrow add(e, T, T) \quad (2)$$
$$add(e, S, S) \wedge add(f, S, T) \rightarrow add(e, T, T) \quad (3)$$
$$add(e, S, T) \rightarrow set(T) \quad (4)$$

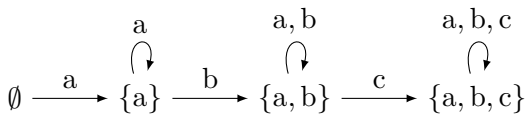


(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V) \quad (1)$$
$$add(e, S, T) \rightarrow add(e, T, T) \quad (2)$$
$$add(e, S, S) \wedge add(f, S, T) \rightarrow add(e, T, T) \quad (3)$$
$$add(e, S, T) \rightarrow set(T) \quad (4)$$

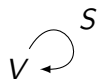


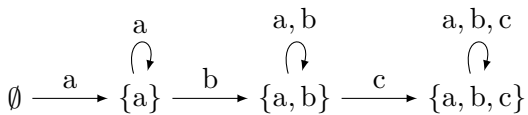


(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

$$\mathit{elem}(e) \wedge \mathit{set}(S) \rightarrow \exists V.\, \mathit{add}(e, S, V) \quad (1)$$
$$\mathit{add}(e, S, T) \rightarrow \mathit{add}(e, T, T) \quad (2)$$
$$\mathit{add}(e, S, S) \wedge \mathit{add}(f, S, T) \rightarrow \mathit{add}(e, T, T) \quad (3)$$
$$\mathit{add}(e, S, T) \rightarrow \mathit{set}(T) \quad (4)$$



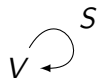


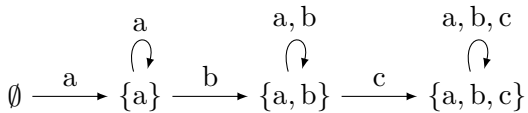
Tgd (2) 'blocks' application along a path for latest element

(A)cyclicity

Acyclic dependency graph $\rightarrow$ chase terminates
(joint acyclicity [K. and Rudolph, IJCAI'11])

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V) \quad (1)$$
$$add(e, S, T) \rightarrow add(e, T, T) \quad (2)$$
$$add(e, S, S) \wedge add(f, S, T) \rightarrow add(e, T, T) \quad (3)$$
$$add(e, S, T) \rightarrow set(T) \quad (4)$$





Tgd (2) 'blocks' application along a path for latest element
Tgd (3) 'blocks' application along a path for previously added elements

Generalising this idea

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Generalising this idea

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Generalising this idea

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Generalising this idea

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Generalising this idea

$$elem(e) \land set(S) \rightarrow \exists V\,.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Goal: nulls should satisfy the tgd for all contexts that led to its creation

Generalising this idea

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Goal: nulls should satisfy the tgd for all contexts that led to its creation

Saturation conditions:

1. A null satisfies the context that was used to create it.
2. A null satisfies all contexts satisfied by its predecessor null.

Generalising this idea

$$elem(e) \wedge set(S) \rightarrow \exists V.\, add(e, S, V)$$

We identify three kinds of variables for the tgds we 'block':

- the existential variable that creates a new null
- the variable that may be matched with a previous null: the 'predecessor'
- the remaining variables: the 'context'

Goal: nulls should satisfy the tgd for all contexts that led to its creation

Saturation conditions:

1. A null satisfies the context that was used to create it.
2. A null satisfies all contexts satisfied by its predecessor null.

→ Datalog-entailment checks, based on tgds and dependency graph

Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

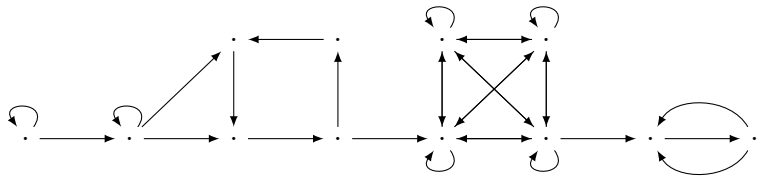
$\rightarrow$ tight complexity bounds based on how we break cycles

Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

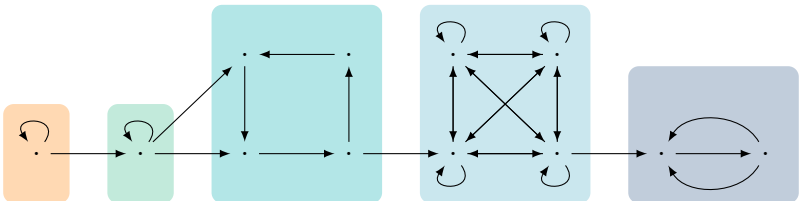


Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

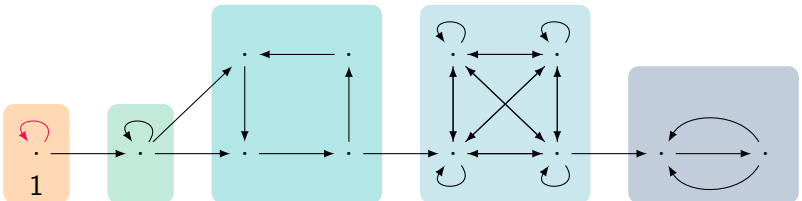


Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

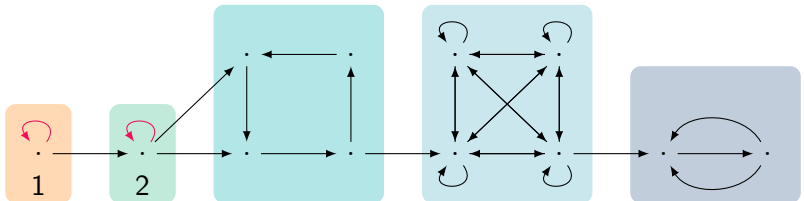


Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

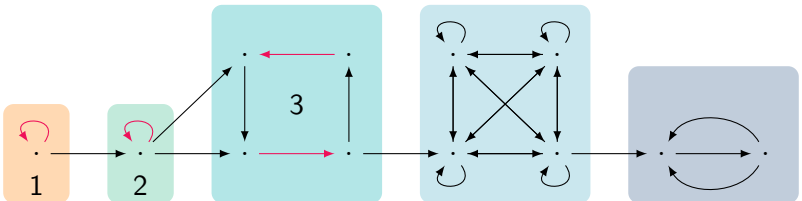


Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

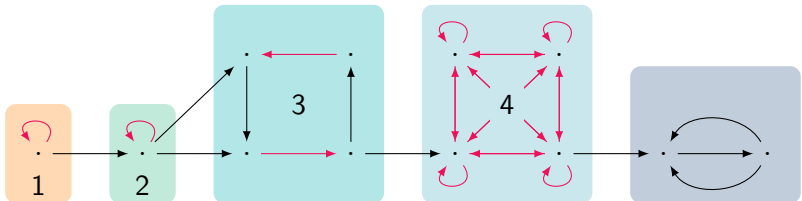


Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

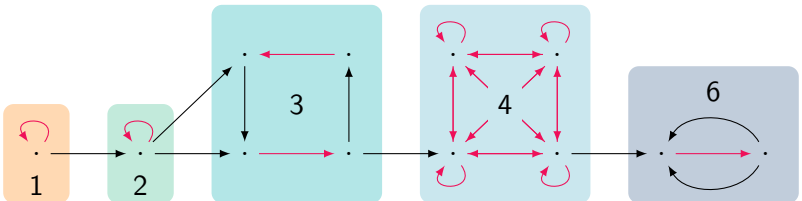


Breaking the cycles

Not all tgds need to be blocked:

- select ‘blocking’ tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

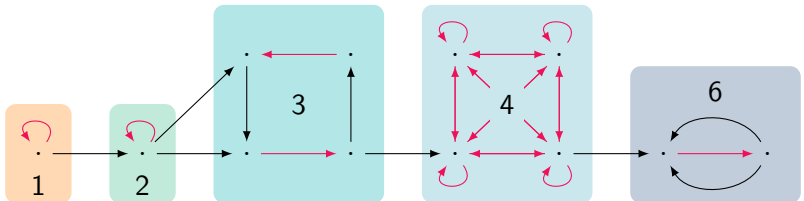


Breaking the cycles

Not all tgds need to be blocked:

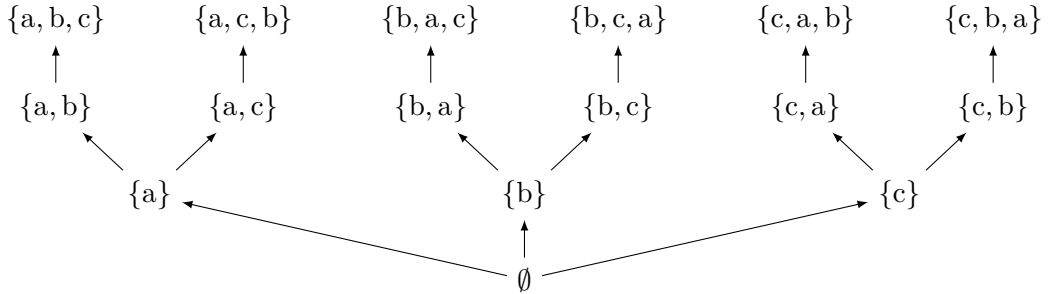
- select 'blocking' tgds to break cycles
- identify predecessor and context variables
- verify saturation criteria

→ tight complexity bounds based on how we break cycles

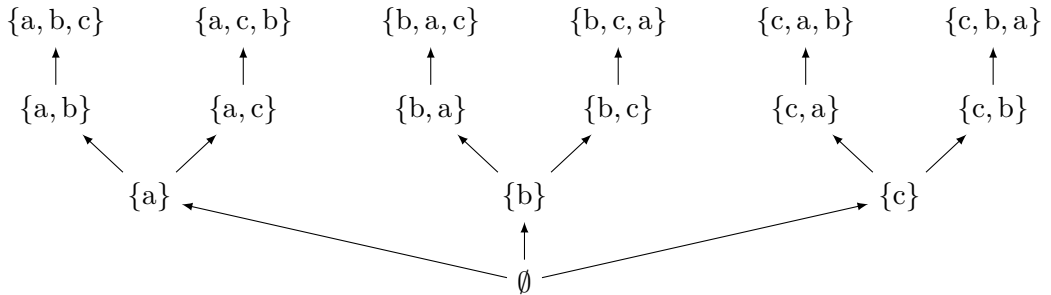


The saturation criteria define decidable classes of tgds with k-ExpTime-complete data complexity for every k.

Teaser

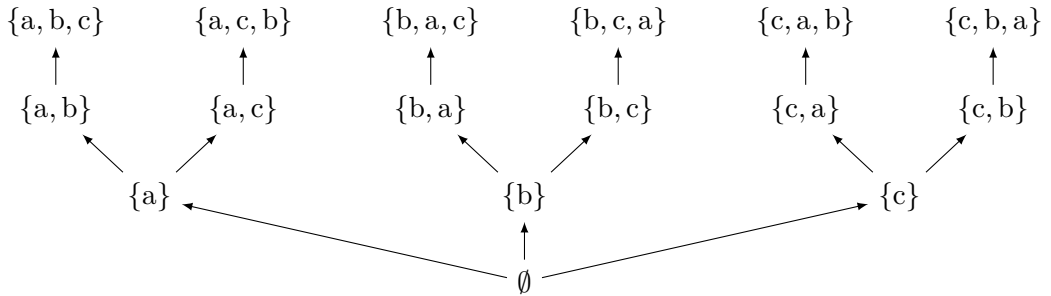


Teaser



There are exponentially many nulls, but they form a tree:
$\rightarrow$ query answering might be possible in PSPACE.

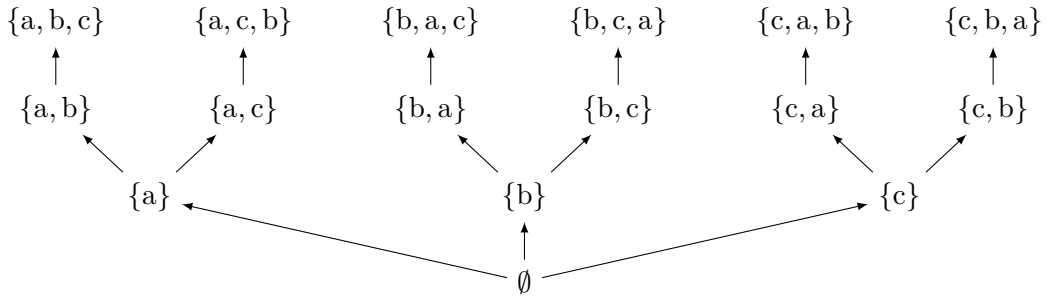
Teaser



There are exponentially many nulls, but they form a tree:
$\rightarrow$ query answering might be possible in PSPACE.

We define a syntactic condition and a chase that realises this complexity.

Teaser



There are exponentially many nulls, but they form a tree:
$\rightarrow$ query answering might be possible in PSPACE.

We define a syntactic condition and a chase that realises this complexity
and obtain decidable classes of tgds with PSPACE- and k-EXPSPACE-complete data complexity for all k.

Summary

Main results:

- decidable classes of tgds with PSPACE, k-EXPSPACE, k-EXPTIME data complexity for all k
- new methods for analysing the structure of the standard chase
- new chase procedures that are optimal for space-bound complexity classes

Open questions:

- refinement of criteria for further complexity classes
- capturing of queries in the covered complexity classes