TECHNISCHE UNIVERSITÄT DRESDEN

THE POWER OF THE TERMINATING CHASE

Markus Krötzsch

Maximilian Marx

Sebastian Rudolph

TU Dresden

→ Download Paper

Lisbon, ICDT 2019 – Invited Tutorial

Fig. 1: The Chase

Part 1:

Tuple-Generating Dependencies

Part 1:

Existential Rules

Part 1:

Datalog$^+$

Part 1:

$$\forall \boldsymbol{x}, \boldsymbol{y}. \varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}. \psi[\boldsymbol{x}, \boldsymbol{z}]$$

Tuple-generating dependencies a.k.a. Existential rules a.k.a. Datalog$^+$

Definition: A rule is a formula of the form:

$$\forall \boldsymbol{x}, \boldsymbol{y}.\ \varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}.\ \psi[\boldsymbol{x}, \boldsymbol{z}]$$

Rule body: conjunction of atoms

- using variables from $\boldsymbol{x} \cup \boldsymbol{y}$
- possibly using constants

Rule head: conjunction of atoms

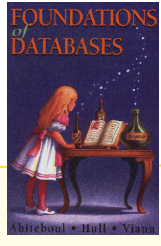
- using variables from $\boldsymbol{x} \cup \boldsymbol{z}$
- possibly using constants

Frontier: variables $\boldsymbol{x}$ used on both sides

Facts can be encoded as variable-free rules with empty body

Universal quantifiers are usually omitted

Example 1: Inclusion dependencies

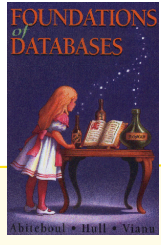


The following inclusion dependency from the Alice Book:

$$\text{Showings}[\text{Title}] \subseteq \text{Movies}[\text{Title}]$$

relates tables Showings[Theatre, Screen, Title, Snack] and Movies[Title, Director, Actor]

Example 1: Inclusion dependencies



The following inclusion dependency from the Alice Book:

$$\text{Showings}[\text{Title}] \subseteq \text{Movies}[\text{Title}]$$

relates tables Showings[Theatre, Screen, Title, Snack] and Movies[Title, Director, Actor]

This can be expressed by the rule:

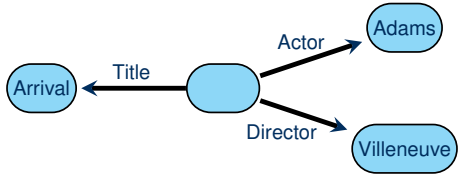
$$\text{Showings}(y_{\text{Theatre}}, y_{\text{Screen}}, x_{\text{Title}}, y_{\text{Snack}}) \rightarrow \exists z_{\text{Director}}, z_{\text{Actor}}.\text{Movies}(x_{\text{Title}}, z_{\text{Director}}, z_{\text{Actor}})$$

Example 2: Data exchange and data integration

Different databases often require different structures.

Example: The W3C RDB2RDF standard specifies how to translate relational databases into graph databases in RDF format

The tuple Movies(Arrival, Villeneuve, Adams), e.g., is translated to a graph of the form

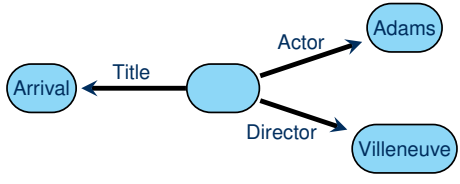


Example 2: Data exchange and data integration

Different databases often require different structures.

Example: The W3C RDB2RDF standard specifies how to translate relational databases into graph databases in RDF format

The tuple Movies(Arrival, Villeneuve, Adams), e.g., is translated to a graph of the form



This can be expressed by the rule:

$$\mathsf{Movies}(x_{\mathsf{Title}}, x_{\mathsf{Director}}, x_{\mathsf{Actor}}) \rightarrow \exists z.\, \mathsf{Title}(z, x_{\mathsf{Title}}) \wedge \mathsf{Director}(z, x_{\mathsf{Director}}) \wedge \mathsf{Actor}(z, x_{\mathsf{Actor}})$$

Example 3: Ontology-based Query Answering

Ontologies have been proposed as means to represent and exchange descriptive schema-level knowledge

Example: The W3C OWL Web Ontology Language is based on description logics (DLs). Many popular OWL/DL fragments can be translated into rules. The following ontology specifies some facts about parts of compound objects (corresponding DL syntax axiom in parenthesis):

$$\text{Bicycle}(x) \rightarrow \exists v.\text{hasPart}(x, v) \wedge \text{Wheel}(v) \qquad (\text{Bicycle} \sqsubseteq \exists \text{hasPart}.\text{Wheel})$$

$$\text{Wheel}(x) \rightarrow \exists w.\text{properPartOf}(x, w) \wedge \text{Bicycle}(w) \qquad (\text{Wheel} \sqsubseteq \exists \text{properPartOf}.\text{Bicycle})$$

$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y) \qquad (\text{properPartOf} \sqsubseteq \text{partOf})$$

$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x) \qquad (\text{hasPart} \sqsubseteq \text{partOf}^-)$$

$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x) \qquad (\text{partOf} \sqsubseteq \text{hasPart}^-)$$

Problems for Existential Rules

One of the main computational problems in these applications is the following:

Query answering under constraints

Input: A concrete database $\mathcal{D}$, a set of rules Σ, and a conjunctive query q
Problem: What are the certain answers of q over $\mathcal{D}$ and Σ? More formally:

- which substitutions σ from free variables in q to constants of $\mathcal{D}$
- satisfy the first-order entailment $\Sigma, \mathcal{D} \models q\sigma$?

Problems for Existential Rules

One of the main computational problems in these applications is the following:

Query answering under constraints

Input: A concrete database $\mathcal{D}$, a set of rules Σ, and a conjunctive query q
Problem: What are the certain answers of q over $\mathcal{D}$ and Σ? More formally:

- which substitutions σ from free variables in q to constants of $\mathcal{D}$
- satisfy the first-order entailment $\Sigma, \mathcal{D} \models q\sigma$?

The corresponding decision problem is as follows:

Query entailment under constraints

Input: A concrete database $\mathcal{D}$, a set of rules Σ, and a Boolean CQ q
Problem: Does $\Sigma, \mathcal{D} \models q$ hold?

Two perspectives on the use of rules

Two perspectives on the use of rules

Rules as “Ontologies”

- Logical theories encode knowledge
- Rules are exchanged and re-combined
- Modelling power related to combined complexity of reasoning

Two perspectives on the use of rules

Rules as “Ontologies”

- Logical theories encode knowledge
- Rules are exchanged and re-combined
- Modelling power related to combined complexity of reasoning

Rules as “Programs”

- Logical theories define computations
- Rules as declarative specifications
- Computational power related to data complexity

Two perspectives on the use of rules

Rules as “Ontologies”

- Logical theories encode knowledge
- Rules are exchanged and re-combined
- Modelling power related to combined complexity of reasoning

Requirements

- Standard exchange syntax
- Expressive power as modelling language (w.r.t. schema)
- Fast reasoners, robust to theory changes

Rules as “Programs”

- Logical theories define computations
- Rules as declarative specifications
- Computational power related to data complexity

Two perspectives on the use of rules

Rules as “Ontologies”

- Logical theories encode knowledge
- Rules are exchanged and re-combined
- Modelling power related to combined complexity of reasoning

Requirements

- Standard exchange syntax
- Expressive power as modelling language (w.r.t. schema)
- Fast reasoners, robust to theory changes

Rules as “Programs”

- Logical theories define computations
- Rules as declarative specifications
- Computational power related to data complexity

Requirements

- Appeal to human engineers
- Expressive power as query language (w.r.t. data)
- Fast reasoners, robust to database changes

Existential rules vs. logic programming

Note that query entailment under existential rules is inter-reducible to query (or fact) entailment for definite logic programs (Horn rules without $\exists$ but with function symbols).

Existential rules vs. logic programming

Note that query entailment under existential rules is inter-reducible to query (or fact) entailment for definite logic programs (Horn rules without $\exists$ but with function symbols).

"Existential rules → definite LP rules" Skolemisation: replace existentially quantified variables by function terms that apply fresh skolem functions to the frontier variables

Example: Skolemising the rule $\text{Wheel}(x) \rightarrow \exists w.\text{partOf}(x, w) \wedge \text{Bicycle}(w)$ yields $\text{Wheel}(x) \rightarrow \text{partOf}(x, f(x)) \wedge \text{Bicycle}(f(x))$, with f a skolem function.

Existential rules vs. logic programming

Note that query entailment under existential rules is inter-reducible to query (or fact) entailment for definite logic programs (Horn rules without $\exists$ but with function symbols).

"Existential rules → definite LP rules" Skolemisation: replace existentially quantified variables by function terms that apply fresh skolem functions to the frontier variables

Example: Skolemising the rule $\text{Wheel}(x) \rightarrow \exists w.\text{partOf}(x, w) \wedge \text{Bicycle}(w)$ yields $\text{Wheel}(x) \rightarrow \text{partOf}(x, f(x)) \wedge \text{Bicycle}(f(x))$, with f a skolem function.

"Definite LP rules → existential rules" Flatten function terms: for each n-ary function f, we introduce an $(n+1)$-ary predicate p_f, used to encode "$x = f(\boldsymbol{t})$" as $p_f(x, \boldsymbol{t})$

Example: The rule $R(x, y, f(x, y)) \rightarrow S(g(f(y, x)))$ is translated to $R(x, y, z) \wedge p_f(z, x, y) \rightarrow \exists v, w.p_f(w, y, x) \wedge p_g(v, w) \wedge S(w)$.

Reasoning for existential rules is difficult

Theorem: Query entailment under constraints is undecidable (but recursively enumerable). There is a fixed rule set Σ and BCQ q, such that $\{\mathcal{D} \mid \Sigma, \mathcal{D} \models q\}$ is undecidable.

Proof (sketch): Use a standard encoding of a Turing machine in logical rules, and apply it to a universal Turing machine. Existential quantifiers are used to create new memory cells and time points. □

Reasoning for existential rules is difficult

Theorem: Query entailment under constraints is undecidable (but recursively enumerable). There is a fixed rule set Σ and BCQ q, such that $\{\mathcal{D} \mid \Sigma, \mathcal{D} \models q\}$ is undecidable.

Proof (sketch): Use a standard encoding of a Turing machine in logical rules, and apply it to a universal Turing machine. Existential quantifiers are used to create new memory cells and time points. □

This also implies that we cannot restrict to finite models.

Example: Consider a database $r(a, b)$ with constraints

$$r(x, y) \rightarrow \exists z.r(y, z)$$
$$r(x, y) \rightarrow t(x, y)$$
$$t(x, y) \wedge r(y, z) \rightarrow t(x, z)$$

The BCQ $\exists x.t(x, x)$ is not entailed by this theory, but it holds in all finite models.

Universal models

Certain answer semantics: What is true in all models?

But it is often enough to consider “most general models”:

Definition: A model $\mathcal{I}$ of a set of rules Σ is universal if it admits a homomorphism $h : \mathcal{I} \to \mathcal{J}$ to every model $\mathcal{J}$ of Σ.

Fact: The BCQs entailed by rule set Σ are exactly the BCQs that hold true on any of its universal models.

(The same works for all query languages whose models are closed under homomorphisms)

Decidable fragments

In the search for decidable fragments, several main principles have been explored:

- Finite models: there is a finite universal model
  - full dependencies (no $\exists$)
  - many acyclicity notions (more on this later)
- Tree-like models: there is universal model of bounded treewidth
  - Guarded rules
  - Frontier-guarded rules
- Rewritability: entailment can be reduced to first-order model checking
  - Linear tgds
  - Sticky rules

None of the general criteria are decidable, but the concrete conditions are.

Part 2:

The Chase

Applying a rule

Database $\mathcal{D}$

Rule $\rho = \varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}.\, \psi[\boldsymbol{x}, \boldsymbol{z}]$

Definition: Rule ρ is applicable to $\mathcal{D}$ if:

1. there is a function $h : \boldsymbol{x} \cup \boldsymbol{y} \rightarrow \text{adom}(\mathcal{D})$ such that $h(\varphi) \subseteq \mathcal{D}$ (a match)
2. there is no function $h' : \boldsymbol{x} \cup \boldsymbol{z} \rightarrow \text{adom}(\mathcal{D})$ with $h'(x) = h(x)$ for all $x \in \boldsymbol{x}$ and $h'(\psi) \subseteq \mathcal{D}$

The $\mathcal{D}'$ is the result of applying ρ to $\mathcal{D}$ under h if $\mathcal{D}' = \mathcal{D} \cup \hat{h}(\psi)$ and:

- $\hat{h}(x) = h(x)$ for all $x \in \boldsymbol{x}$
- $\hat{h}(z)$ is a fresh null for all $z \in \boldsymbol{z}$

The Chase(s)

A chase constructs a sequence of databases $\mathcal{D}_0 = \mathcal{D}, \mathcal{D}_1, \mathcal{D}_2, \ldots$ by applying rules.

The Standard Chase (a.k.a. restricted chase)

- Apply rules to matches in some order (strategy)

The Skolem Chase (a.k.a. semi-oblivious chase)

- Apply skolemised rules (in any order)

The Datalog-first Chase

- Apply rules to matches in some order that prioritises the application of rules without existential quantifiers

Other prominent chases: **oblivious chase** and **core chase**

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$
$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{hasPart}(n_2, n_3), \mathsf{Wheel}(n_3)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$
$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{hasPart}(n_2, n_3), \mathsf{Wheel}(n_3)\}$
$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{partOf}(n_1, n_2)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$
$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{hasPart}(n_2, n_3), \mathsf{Wheel}(n_3)\}$
$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{partOf}(n_1, n_2)\}$
$\mathcal{D}_6 = \mathcal{D}_5 \cup \{\mathsf{hasPart}(n_2, n_1)\}$

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \exists v.\text{hasPart}(x, v) \wedge \text{Wheel}(v)$$
$$\text{Wheel}(x) \rightarrow \exists w.\text{properPartOf}(x, w) \wedge \text{Bicycle}(w)$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\text{hasPart}(c, n_1), \text{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\text{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\text{properPartOf}(n_1, n_2), \text{Bicycle}(n_2)\}$
$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\text{hasPart}(n_2, n_3), \text{Wheel}(n_3)\}$
$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\text{partOf}(n_1, n_2)\}$
$\mathcal{D}_6 = \mathcal{D}_5 \cup \{\text{hasPart}(n_2, n_1)\}$
$\mathcal{D}_7 = \mathcal{D}_6 \cup \{\text{partOf}(n_3, n_2)\}$

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \exists v.\text{hasPart}(x, v) \wedge \text{Wheel}(v)$$
$$\text{Wheel}(x) \rightarrow \exists w.\text{properPartOf}(x, w) \wedge \text{Bicycle}(w)$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Applying the standard chase may yield:

$\mathcal{D}_1 = \mathcal{D} \cup \{\text{hasPart}(c, n_1), \text{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\text{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\text{properPartOf}(n_1, n_2), \text{Bicycle}(n_2)\}$
$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\text{hasPart}(n_2, n_3), \text{Wheel}(n_3)\}$
$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\text{partOf}(n_1, n_2)\}$
$\mathcal{D}_6 = \mathcal{D}_5 \cup \{\text{hasPart}(n_2, n_1)\}$
$\mathcal{D}_7 = \mathcal{D}_6 \cup \{\text{partOf}(n_3, n_2)\}$
$\mathcal{D}_8 = \mathcal{D}_7 \cup \{\text{partOf}(n_3, n_4), \text{Bicycle}(n_4)\}$

Will it terminate?

$$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$$

$$\begin{aligned}
\mathsf{Bicycle}(x) &\rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v) \\
\mathsf{Wheel}(x) &\rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w) \\
\mathsf{properPartOf}(x, y) &\rightarrow \mathsf{partOf}(x, y) \\
\mathsf{hasPart}(x, y) &\rightarrow \mathsf{partOf}(y, x) \\
\mathsf{partOf}(x, y) &\rightarrow \mathsf{hasPart}(y, x)
\end{aligned}$$

Applying the standard chase may yield:

$$\begin{aligned}
\mathcal{D}_1 &= \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\} \\
\mathcal{D}_2 &= \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\} \\
\mathcal{D}_3 &= \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\} \\
\mathcal{D}_4 &= \mathcal{D}_3 \cup \{\mathsf{hasPart}(n_2, n_3), \mathsf{Wheel}(n_3)\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{D}_5 &= \mathcal{D}_4 \cup \{\mathsf{partOf}(n_1, n_2)\} \\
\mathcal{D}_6 &= \mathcal{D}_5 \cup \{\mathsf{hasPart}(n_2, n_1)\} \\
\mathcal{D}_7 &= \mathcal{D}_6 \cup \{\mathsf{partOf}(n_3, n_2)\} \\
\mathcal{D}_8 &= \mathcal{D}_7 \cup \{\mathsf{partOf}(n_3, n_4), \mathsf{Bicycle}(n_4)\}
\end{aligned}$$

The chase can continue forever . . .

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the Datalog-first chase yields:

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the Datalog-first chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \exists v.\text{hasPart}(x, v) \wedge \text{Wheel}(v)$$
$$\text{Wheel}(x) \rightarrow \exists w.\text{properPartOf}(x, w) \wedge \text{Bicycle}(w)$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Applying the Datalog-first chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\text{hasPart}(c, n_1), \text{Wheel}(n_1)\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\text{partOf}(n_1, c)\}$

Will it terminate?

$$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$$

$$\begin{aligned}
\mathsf{Bicycle}(x) &\rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v) \\
\mathsf{Wheel}(x) &\rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w) \\
\mathsf{properPartOf}(x, y) &\rightarrow \mathsf{partOf}(x, y) \\
\mathsf{hasPart}(x, y) &\rightarrow \mathsf{partOf}(y, x) \\
\mathsf{partOf}(x, y) &\rightarrow \mathsf{hasPart}(y, x)
\end{aligned}$$

Applying the Datalog-first chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)$$
$$\mathsf{Wheel}(x) \rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the Datalog-first chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$
$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(n_1, n_2)\}$

Will it terminate?

$$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$$

$$\begin{aligned}
\mathsf{Bicycle}(x) &\rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v)\\
\mathsf{Wheel}(x) &\rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w)\\
\mathsf{properPartOf}(x, y) &\rightarrow \mathsf{partOf}(x, y)\\
\mathsf{hasPart}(x, y) &\rightarrow \mathsf{partOf}(y, x)\\
\mathsf{partOf}(x, y) &\rightarrow \mathsf{hasPart}(y, x)
\end{aligned}$$

Applying the Datalog-first chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$

$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(n_1, n_2)\}$

$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{hasPart}(n_2, n_1)\}$

Will it terminate?

$$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$$

$$\begin{aligned}
\mathsf{Bicycle}(x) &\rightarrow \exists v.\mathsf{hasPart}(x, v) \wedge \mathsf{Wheel}(v) \\
\mathsf{Wheel}(x) &\rightarrow \exists w.\mathsf{properPartOf}(x, w) \wedge \mathsf{Bicycle}(w) \\
\mathsf{properPartOf}(x, y) &\rightarrow \mathsf{partOf}(x, y) \\
\mathsf{hasPart}(x, y) &\rightarrow \mathsf{partOf}(y, x) \\
\mathsf{partOf}(x, y) &\rightarrow \mathsf{hasPart}(y, x)
\end{aligned}$$

Applying the Datalog-first chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, n_1), \mathsf{Wheel}(n_1)\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(n_1, c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(n_1, n_2), \mathsf{Bicycle}(n_2)\}$

$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(n_1, n_2)\}$

$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{hasPart}(n_2, n_1)\}$

No further rules are applicable. The chase terminates.

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \text{hasPart}(x, w(x)) \wedge \text{Wheel}(w(x))$$
$$\text{Wheel}(x) \rightarrow \text{properPartOf}(x, b(w)) \wedge \text{Bicycle}(b(w))$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \text{hasPart}(x, w(x)) \wedge \text{Wheel}(w(x))$$
$$\text{Wheel}(x) \rightarrow \text{properPartOf}(x, b(w)) \wedge \text{Bicycle}(b(w))$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Applying the skolem chase yields:

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \text{hasPart}(x, w(x)) \wedge \text{Wheel}(w(x))$$
$$\text{Wheel}(x) \rightarrow \text{properPartOf}(x, b(w)) \wedge \text{Bicycle}(b(w))$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\text{hasPart}(c, w(c)), \text{Wheel}(w(c))\}$

Will it terminate?

$\mathcal{D} = \{\text{Bicycle}(c)\}$

$$\text{Bicycle}(x) \rightarrow \text{hasPart}(x, w(x)) \wedge \text{Wheel}(w(x))$$
$$\text{Wheel}(x) \rightarrow \text{properPartOf}(x, b(w)) \wedge \text{Bicycle}(b(w))$$
$$\text{properPartOf}(x, y) \rightarrow \text{partOf}(x, y)$$
$$\text{hasPart}(x, y) \rightarrow \text{partOf}(y, x)$$
$$\text{partOf}(x, y) \rightarrow \text{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\text{hasPart}(c, w(c)), \text{Wheel}(w(c))\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\text{partOf}(w(c), c)\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \mathsf{hasPart}(x, w(x)) \wedge \mathsf{Wheel}(w(x))$$
$$\mathsf{Wheel}(x) \rightarrow \mathsf{properPartOf}(x, b(w)) \wedge \mathsf{Bicycle}(b(w))$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, w(c)), \mathsf{Wheel}(w(c))\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(w(c), c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(w(c), b(w(c))),$
$\mathsf{Bicycle}(b(w(c)))\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \mathsf{hasPart}(x, w(x)) \wedge \mathsf{Wheel}(w(x))$$
$$\mathsf{Wheel}(x) \rightarrow \mathsf{properPartOf}(x, b(w)) \wedge \mathsf{Bicycle}(b(w))$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, w(c)), \mathsf{Wheel}(w(c))\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(w(c), c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(w(c), b(w(c))),$
$\mathsf{Bicycle}(b(w(c)))\}$

$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(w(c), b(w(c)))\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \mathsf{hasPart}(x, w(x)) \wedge \mathsf{Wheel}(w(x))$$
$$\mathsf{Wheel}(x) \rightarrow \mathsf{properPartOf}(x, b(w)) \wedge \mathsf{Bicycle}(b(w))$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, w(c)), \mathsf{Wheel}(w(c))\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(w(c), c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(w(c), b(w(c))), \mathsf{Bicycle}(b(w(c)))\}$

$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(w(c), b(w(c)))\}$

$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{hasPart}(b(w(c)), w(c))\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \mathsf{hasPart}(x, w(x)) \wedge \mathsf{Wheel}(w(x))$$
$$\mathsf{Wheel}(x) \rightarrow \mathsf{properPartOf}(x, b(w)) \wedge \mathsf{Bicycle}(b(w))$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, w(c)), \mathsf{Wheel}(w(c))\}$
$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(w(c), c)\}$
$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(w(c), b(w(c))), \mathsf{Bicycle}(b(w(c)))\}$

$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(w(c), b(w(c)))\}$
$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{hasPart}(b(w(c)), w(c))\}$
$\mathcal{D}_6 = \mathcal{D}_5 \cup \{\mathsf{hasPart}(b(w(c)), w(b(w(c)))), \mathsf{Wheel}(w(b(w(c))))\}$

Will it terminate?

$\mathcal{D} = \{\mathsf{Bicycle}(c)\}$

$$\mathsf{Bicycle}(x) \rightarrow \mathsf{hasPart}(x, w(x)) \wedge \mathsf{Wheel}(w(x))$$
$$\mathsf{Wheel}(x) \rightarrow \mathsf{properPartOf}(x, b(w)) \wedge \mathsf{Bicycle}(b(w))$$
$$\mathsf{properPartOf}(x, y) \rightarrow \mathsf{partOf}(x, y)$$
$$\mathsf{hasPart}(x, y) \rightarrow \mathsf{partOf}(y, x)$$
$$\mathsf{partOf}(x, y) \rightarrow \mathsf{hasPart}(y, x)$$

Applying the skolem chase yields:

$\mathcal{D}_1 = \mathcal{D} \cup \{\mathsf{hasPart}(c, w(c)), \mathsf{Wheel}(w(c))\}$

$\mathcal{D}_2 = \mathcal{D}_1 \cup \{\mathsf{partOf}(w(c), c)\}$

$\mathcal{D}_3 = \mathcal{D}_2 \cup \{\mathsf{properPartOf}(w(c), b(w(c))), \mathsf{Bicycle}(b(w(c)))\}$

$\mathcal{D}_4 = \mathcal{D}_3 \cup \{\mathsf{partOf}(w(c), b(w(c)))\}$

$\mathcal{D}_5 = \mathcal{D}_4 \cup \{\mathsf{hasPart}(b(w(c)), w(c))\}$

$\mathcal{D}_6 = \mathcal{D}_5 \cup \{\mathsf{hasPart}(b(w(c)), w(b(w(c)))), \mathsf{Wheel}(w(b(w(c))))\}$

The chase will certainly continue forever . . .

Chase termination

Some observations:

- Termination is strategy-dependent for standard and Datalog-first chase, but not for skolem chase
- Whenever skolem chase terminates, standard chase terminates for all strategies
- Whenever standard chase terminates (for some/all strategies), Datalog-first chase terminates (for all/some strategies)
- Termination always depends on the concrete database instance

Chase termination

Some observations:

- Termination is strategy-dependent for standard and Datalog-first chase, but not for skolem chase
- Whenever skolem chase terminates, standard chase terminates for all strategies
- Whenever standard chase terminates (for some/all strategies), Datalog-first chase terminates (for all/some strategies)
- Termination always depends on the concrete database instance

We can define rule classes based on their termination behaviour:

Termination on ...	instance $\mathcal{D}$	all instances
Skolem chase	$\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}}$	$\mathsf{CT}^{\mathsf{sk}}_{\forall}$
Standard chase (all strategies)	$\mathsf{CT}^{\mathsf{std}}_{\mathcal{D}\forall}$	$\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$
Datalog-first chase (all strategies)	$\mathsf{CT}^{\mathsf{dlf}}_{\mathcal{D}\forall}$	$\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$

The chase termination problem

Theorem (Gogacz & Marcinkowski, ICALP'14; Grahne & Onet, Fund.Inf.'18):
The classes $\mathsf{CT}^{\mathsf{x}}_{\mathcal{D}(\forall)}$ and $\mathsf{CT}^{\mathsf{x}}_{\forall(\forall)}$ are undecidable for all $\mathsf{x} \in \{\mathsf{sk}, \mathsf{std}, \mathsf{dlf}\}$.

The chase termination problem

Theorem (Gogacz & Marcinkowski, ICALP'14; Grahne & Onet, Fund.Inf.'18):
The classes $\mathsf{CT}^{\mathsf{x}}_{\mathcal{D}(\forall)}$ and $\mathsf{CT}^{\mathsf{x}}_{\forall(\forall)}$ are undecidable for all $\mathsf{x} \in \{\mathsf{sk}, \mathsf{std}, \mathsf{dlf}\}$.

The cases $\mathsf{CT}^{\mathsf{x}}_{\mathcal{D}(\forall)}$ are simple:

- Simulate a Turing machine in a standard encoding
- Halting reduces to chase termination

These cases are recursively enumerable (r.e.).

The chase termination problem

Theorem (Gogacz & Marcinkowski, ICALP'14; Grahne & Onet, Fund.Inf.'18):
The classes $\mathsf{CT}^{\mathsf{x}}_{\mathcal{D}(\forall)}$ and $\mathsf{CT}^{\mathsf{x}}_{\forall(\forall)}$ are undecidable for all $\mathsf{x} \in \{\mathsf{sk}, \mathsf{std}, \mathsf{dlf}\}$.

The cases $\mathsf{CT}^{\mathsf{x}}_{\mathcal{D}(\forall)}$ are simple:

- Simulate a Turing machine in a standard encoding
- Halting reduces to chase termination

These cases are recursively enumerable (r.e.).

Membership of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ in r.e. is also simple, due to the following result [Marnette, PODS'09]:

Proposition: $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ if and only if $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}^*}$, where $\mathcal{D}^*$ is the critical instance consisting of all atoms that can be stated over the signature using constants from Σ and an additional constant $*$.

Universal chase termination

Hardness of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is more tricky: how to simulate a Turing machine starting from $\mathcal{D}^*$?

- Every conjunctive query already matches
- It is difficult to apply rules in any orderly fashion

Solved by [Gogacz & Marcinkowski, ICALP'14] (showing r.e.-completeness)

Universal chase termination

Hardness of $\mathrm{CT}^{\mathsf{sk}}_{\forall}$ is more tricky: how to simulate a Turing machine starting from $\mathcal{D}^*$?

- Every conjunctive query already matches
- It is difficult to apply rules in any orderly fashion

Solved by [Gogacz & Marcinkowski, ICALP'14] (showing r.e.-completeness)

The case of $\mathrm{CT}^{\mathsf{std}}_{\forall\forall}$ (and with it $\mathrm{CT}^{\mathsf{dlf}}_{\forall\forall}$) is more difficult.

The critical instance is no longer relevant for all-instances termination:

Observation: Every rule set is in $\mathrm{CT}^{\mathsf{std}}_{\mathcal{D}^*}$.

Indeed, $\mathrm{CT}^{\mathsf{std}}_{\forall\forall}$ and $\mathrm{CT}^{\mathsf{dlf}}_{\forall\forall}$ are no longer r.e., although the exact degree of their undecidability remains open.

Decidable cases

Decidable cases

The (supposed) undecidability of chase termination has motivated significant research activities for finding sufficient termination criteria:

- omega-restrictedness [Syrjänen, LPNMR 2001]
- weak-acyclicity [Fagin et al., Theo. Comp. Sci. 2005]
- lambda restrictedness [Gebser, Schaub, Thiele, LPNMR 2007]
- finite domain [Calimeri et al. ICLP 2008]
- super-weak acyclicity [Marnette, PODS 2009]
- safety [Meier, Schmidt, & Lausen, Proc. VLDB 2009]
- argument restrictedness [Lierler & Lifschitz, ICLP 2009]
- joint acyclicity [MK & Rudolph, IJCAI 2011]
- acyclic graph of rule dependencies [Baget et al., Artif. Intell. 2011]
- Ω-acyclicity [Greco, Spezzano, & Trubitsyna, ICLP 2012]
- model faithful & model summarising ayclicity [Cuenca Grau et al., J. Artif. Intell. Res. 2013]

All of these criteria apply to $\mathrm{CT}^{\mathrm{sk}}_{\forall}$.

Chase variants in practice

Standard chase rule applications are harder than skolem chase rule applications:

- Skolem chase: guess match and verify absence of conclusions – NP
- Standard chase: guess match and verify non-entailment of conclusion – $\mathrm{NP}^{\mathrm{NP}}$ $(= \Sigma_p^2)$

Chase variants in practice

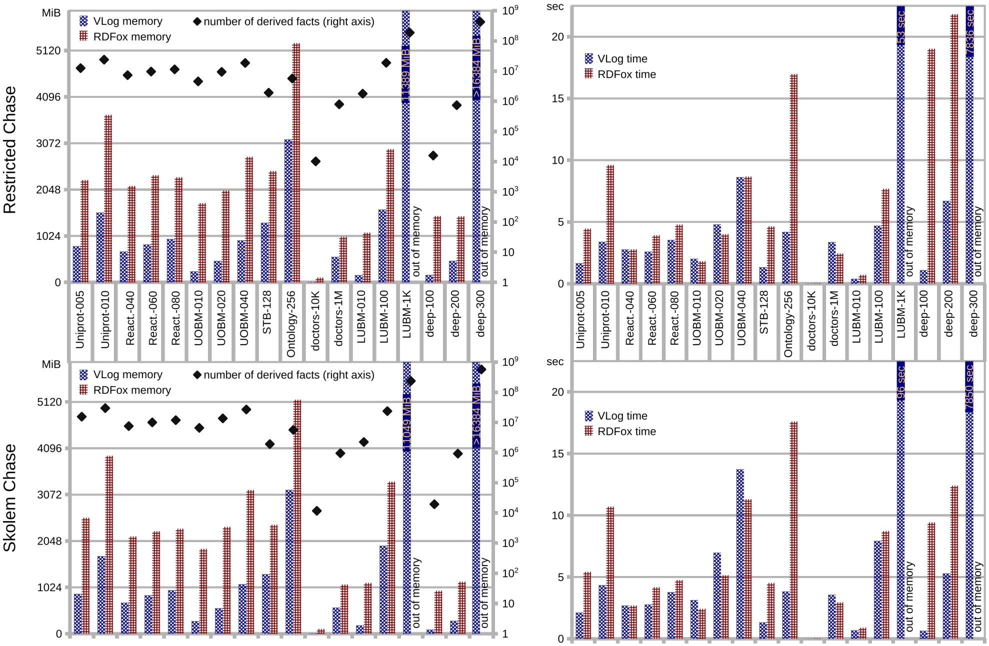
Standard chase rule applications are harder than skolem chase rule applications:

- Skolem chase: guess match and verify absence of conclusions – NP
- Standard chase: guess match and verify non-entailment of conclusion – NP^{NP} $(= \Sigma_p^2)$

Nevertheless, the standard chase is implemented by many existential rule engines:

- DEMo [Pichler & Savenkov, VLDB'09]
- RDFox [Motik et al., AAAI'14]
- Llunatic [Geerts et al., VLDB'14]
- Pegasus [Meier, VLDB'14]
- PDQ [Benedikt, Leblay, & Tsamoura, VLDB'14; VLDB'15]
- Graal [Baget et al., RuleML'15]
- VLog [Urbani, Jacobs, & MK, AAAI'16; Urbani et al., IJCAR'18]

See [Benedikt et al., PODS'17] and [Urbani et al., IJCAR'18] for recent benchmarks.


Restricted Chase
Skolem Chase
MiB
sec
VLog memory
RDFox memory
number of derived facts (right axis)
VLog time
RDFox time
0
1024
2048
3072
4096
5120
1
10
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
5
10
15
20
11389 MiB
>16384 MiB
11049 MiB
53 sec
7836 sec
96 sec
7850 sec
out of memory
Uniprot-005
Uniprot-010
React.-040
React.-060
React.-080
UOBM-010
UOBM-020
UOBM-040
STB-128
Ontology-256
doctors-10K
doctors-1M
LUBM-010
LUBM-100
LUBM-1K
deep-100
deep-200
deep-300

Part 3:

Expressivity

Expressive power

What is the expressive power of fragments of existential rules for which the chase terminates?

Expressive power

What is the expressive power of fragments of existential rules for which the chase terminates?

Follow-up question: what is “expressive power”?

Expressive power

What is the expressive power of fragments of existential rules for which the chase terminates?

Follow-up question: what is “expressive power”?

$\rightsquigarrow$ descriptive, not computational complexity

Definition: Consider a finite signature $\mathbf{R}^{\text{EDB}}$ of (extensional) database relations. An abstract query over $\mathbf{R}^{\text{EDB}}$ is a set $\mathfrak{D}$ of concrete databases over $\mathbf{R}^{\text{EDB}}$.

A set of rules Σ and BCQ q realise $\mathfrak{D}$ if, for every database $\mathcal{D}$ over $\mathbf{R}^{\text{EDB}}$,

$$\mathcal{D}, \Sigma \models q \text{ exactly if } \mathcal{D} \in \mathfrak{D}.$$

where Σ and q may use additional relations beyond $\mathbf{R}^{\text{EDB}}$.

$\rightsquigarrow$ Expressivity = abstract queries that can be realised (by a rule fragment)

Note: This is closer to the program view than to the ontology view.

A note on Datalog

Distinguishing extensional (EDB) and intensional (IDB) predicates is common for Datalog.

A note on Datalog

Distinguishing extensional (EDB) and intensional (IDB) predicates is common for Datalog.

Datalog as Second-Order Language

- EDB predicates = FO predicates; IDB prediates = SO variables
- Query answering: Second-order model checking
- Query containment et al.: undecidable

Datalog as First-Order Language

- EDB predicates = input predicates; IDB prediates = auxiliary/output predicates
- Query answering: first-order entailment
- Query containment et al.: decidable

A note on Datalog

Distinguishing extensional (EDB) and intensional (IDB) predicates is common for Datalog.

Datalog as Second-Order Language

- EDB predicates = FO predicates; IDB prediates = SO variables
- Query answering: Second-order model checking
- Query containment et al.: undecidable

Datalog as First-Order Language

- EDB predicates = input predicates; IDB prediates = auxiliary/output predicates
- Query answering: first-order entailment
- Query containment et al.: decidable

We only use EDB predicates to define expressivity. Everything here is first order.

Data complexity for $\mathsf{CT}^{\mathsf{sk}}_{\forall}$

Marnette [PODS 2009] showed the following general result:

Theorem: For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$.
The data complexity of BCQ entailment over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is PTime-complete.

Data complexity for $\mathsf{CT}^{\mathsf{sk}}_{\forall}$

Marnette [PODS 2009] showed the following general result:

Theorem: For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$.
The data complexity of BCQ entailment over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is PTime-complete.

Proof: There is a tuple-preserving mapping h from any database $\mathcal{D}$ to the critical instance $\mathcal{D}^*$:

- $h(c) = c$ for all constants in Σ
- $h(c) = *$ for all other constants

h extends to function terms by setting $h(f(\boldsymbol{c}) = f(h(\boldsymbol{c}))$.

Data complexity for $\mathsf{CT}^{\mathsf{sk}}_{\forall}$

Marnette [PODS 2009] showed the following general result:

Theorem: For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$.
The data complexity of BCQ entailment over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is PTime-complete.

Proof: There is a tuple-preserving mapping h from any database $\mathcal{D}$ to the critical instance $\mathcal{D}^*$:

- $h(c) = c$ for all constants in Σ
- $h(c) = *$ for all other constants

h extends to function terms by setting $h(f(\boldsymbol{c}) = f(h(\boldsymbol{c}))$.

This extended mapping satisfies: $r(\boldsymbol{t}) \in \mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ implies $r(h(\boldsymbol{t})) \in \mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$.

Data complexity for $\mathrm{CT}^{\mathrm{sk}}_{\forall}$

Marnette [PODS 2009] showed the following general result:

Theorem: For every $\Sigma \in \mathrm{CT}^{\mathrm{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$.
The data complexity of BCQ entailment over $\mathrm{CT}^{\mathrm{sk}}_{\forall}$ is PTime-complete.

Proof: There is a tuple-preserving mapping h from any database $\mathcal{D}$ to the critical instance $\mathcal{D}^*$:

- $h(c) = c$ for all constants in Σ
- $h(c) = *$ for all other constants

h extends to function terms by setting $h(f(\boldsymbol{c}) = f(h(\boldsymbol{c}))$.

This extended mapping satisfies: $r(\boldsymbol{t}) \in \mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D})$ implies $r(h(\boldsymbol{t})) \in \mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D}^*)$.

In particular: the depth and structure of function terms in $\mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D})$ is restricted to the depth and structure of terms in $\mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D}^*)$.

Data complexity for $\mathrm{CT}^{\mathrm{sk}}_{\forall}$

Marnette [PODS 2009] showed the following general result:

Theorem: For every $\Sigma \in \mathrm{CT}^{\mathrm{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$.
The data complexity of BCQ entailment over $\mathrm{CT}^{\mathrm{sk}}_{\forall}$ is PTime-complete.

Proof: There is a tuple-preserving mapping h from any database $\mathcal{D}$ to the critical instance $\mathcal{D}^*$:

- $h(c) = c$ for all constants in Σ
- $h(c) = *$ for all other constants

h extends to function terms by setting $h(f(\boldsymbol{c}) = f(h(\boldsymbol{c}))$.

This extended mapping satisfies: $r(\boldsymbol{t}) \in \mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D})$ implies $r(h(\boldsymbol{t})) \in \mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D}^*)$.

In particular: the depth and structure of function terms in $\mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D})$ is restricted to the depth and structure of terms in $\mathrm{chase}_{\mathrm{sk}}(\Sigma, \mathcal{D}^*)$.

The only data-dependent part are the additional constants in $\mathcal{D}$: the number of distinct terms and tuples is polynomial in this respect. □

From $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ to Datalog

The previous insight can be taken further
[MK & Rudolph IJCAI'11; Zhang, Zhang & You AAAI'15]

Theorem: For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and BCQ q, there is a set of Datalog rules Σ' and BCQ q' such that $\{\mathcal{D} \mid \mathcal{D}, \Sigma \models q\} = \{\mathcal{D} \mid \mathcal{D}, \Sigma' \models q'\}$.

From $\mathrm{CT}^{\mathrm{sk}}_{\forall}$ to Datalog

The previous insight can be taken further
[MK & Rudolph IJCAI'11; Zhang, Zhang & You AAAI'15]

Theorem: For every $\Sigma \in \mathrm{CT}^{\mathrm{sk}}_{\forall}$ and BCQ q, there is a set of Datalog rules Σ' and BCQ q' such that $\{\mathcal{D} \mid \mathcal{D}, \Sigma \models q\} = \{\mathcal{D} \mid \mathcal{D}, \Sigma' \models q'\}$.

Proof (idea): The terms in any skolem chase over Σ are bounded in size.

One can "flatten" such terms by increasing the arity of predicates, e.g.,

$$p(f(a,b)) \mapsto \hat{p}(f,a,b)$$

Arities must be large enough to accommodate all possible terms, but unused positions can be filled by a special constant $\square$, e.g.,

$$q(f(s(a,b),t(c,d))) \mapsto \hat{q}(f,s,a,b,t,c,d)$$
$$q(f(a,g(b))) \mapsto \hat{q}(f,a,\square,\square,g,b,\square)$$

It is easy to apply these replacements to rules and queries. $\square$

Discussion

Summary: Essentially all known chase termination criteria recognise fragments of existential rules that are basically syntactic simplifications of Datalog.

- Existential rules are usually more concise (flattening may incur exponential predicate arity)
- Combined complexity is accordingly higher (typically 2ExpTime-complete)
- But the expressive power is not more than Datalog

Discussion

Summary: Essentially all known chase termination criteria recognise fragments of existential rules that are basically syntactic simplifications of Datalog.

- Existential rules are usually more concise (flattening may incur exponential predicate arity)
- Combined complexity is accordingly higher (typically 2ExpTime-complete)
- But the expressive power is not more than Datalog

Thesis

Previous research on chase termination is best motivated from an ontological view, while not leading to significant advances for using rules as declarative programs/queries.

Beyond P

Surprisingly, this severe restriction in expressivity is specific to the skolem chase:

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ and a BCQ q that express a non-elementary Boolean query.

Beyond P

Surprisingly, this severe restriction in expressivity is specific to the skolem chase:

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ and a BCQ q that express a non-elementary Boolean query.

Proof: We reduce from the following non-elementary decision problem:

Input: A Turing machine $\mathcal{M}$ and a number k
Question: When started on the empty tape, does $\mathcal{M}$ halt in at most $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{k \text{ times}}$ steps?

Beyond P

Surprisingly, this severe restriction in expressivity is specific to the skolem chase:

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ and a BCQ q that express a non-elementary Boolean query.

Proof: We reduce from the following non-elementary decision problem:

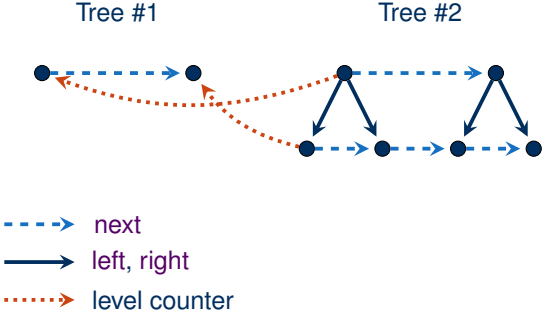
Input: A Turing machine $\mathcal{M}$ and a number k
Question: When started on the empty tape, does $\mathcal{M}$ halt in at most $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{k \text{ times}}$ steps?

Three ingredients are needed:

1. Rules that receive an input chain $\mathsf{first}(e_0), \mathsf{next}(e_0, e_1), \ldots, \mathsf{next}(e_{k-1}, e_k), \mathsf{last}(e_k)$ and construct a k-exponentially long chain (interesting)
2. Rules that simulate a Turing machine in time and space bounded by this chain (boring)
3. Rules that ensure termination even on malformed inputs (interesting)

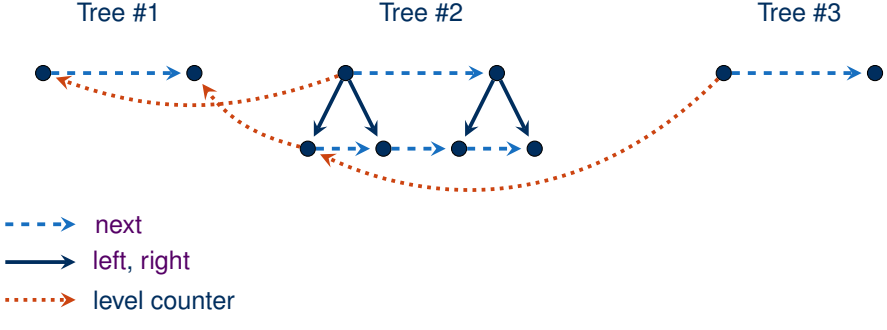
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



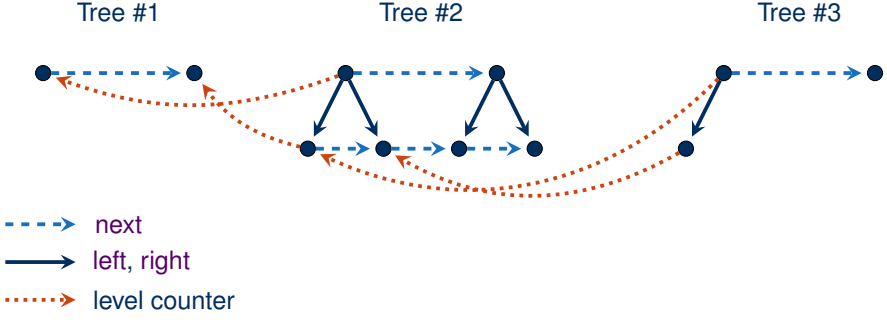
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



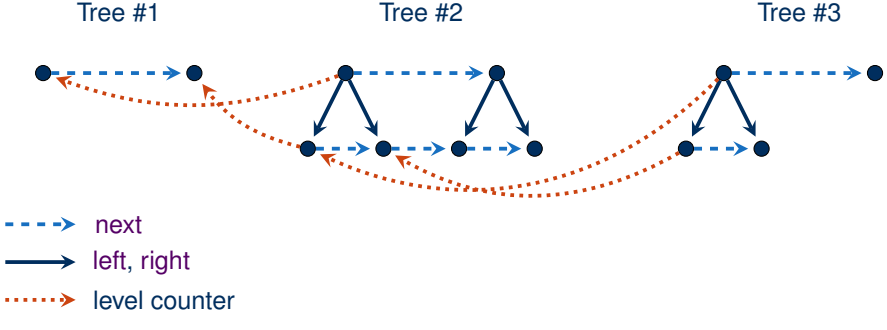
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



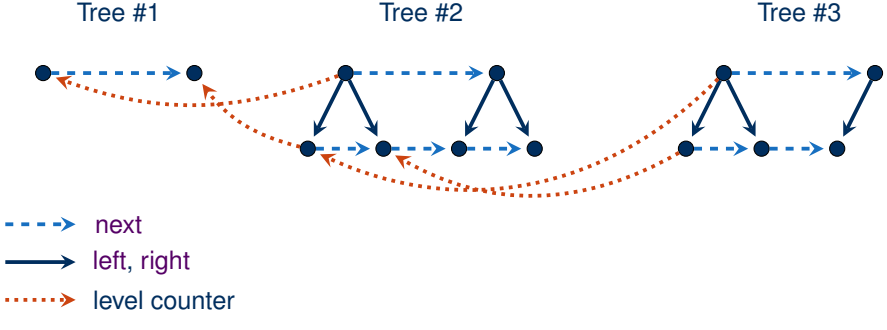
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



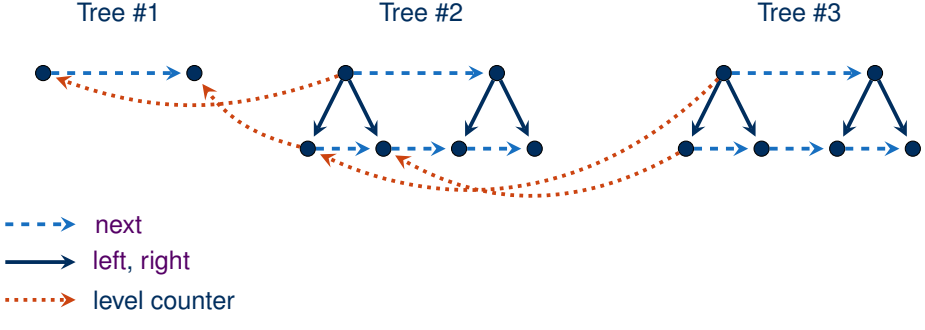
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



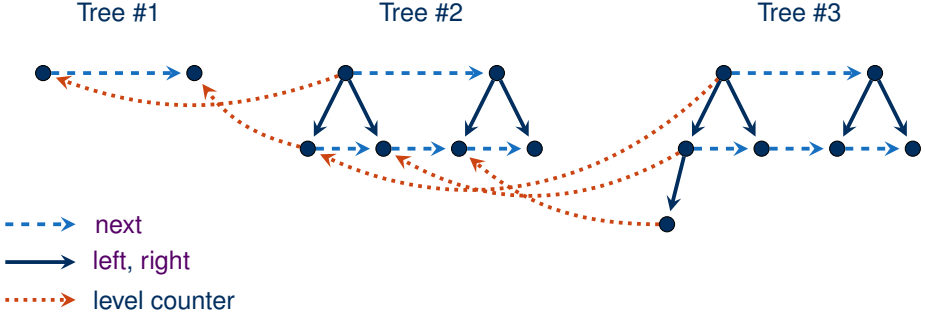
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



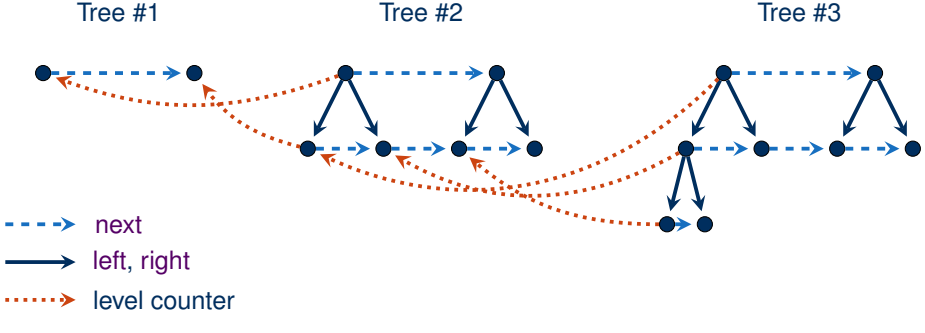
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



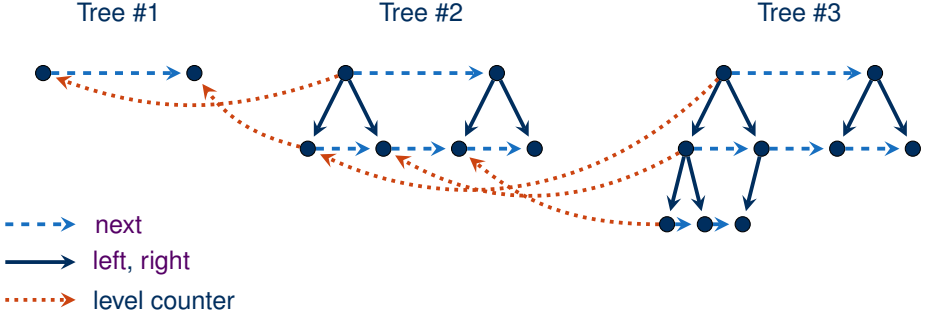
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



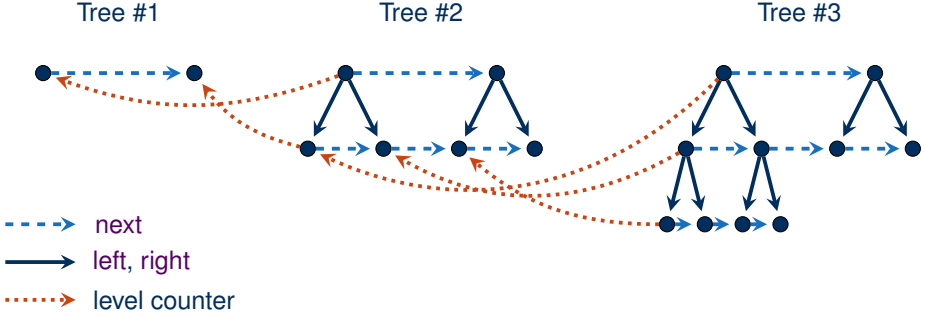
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



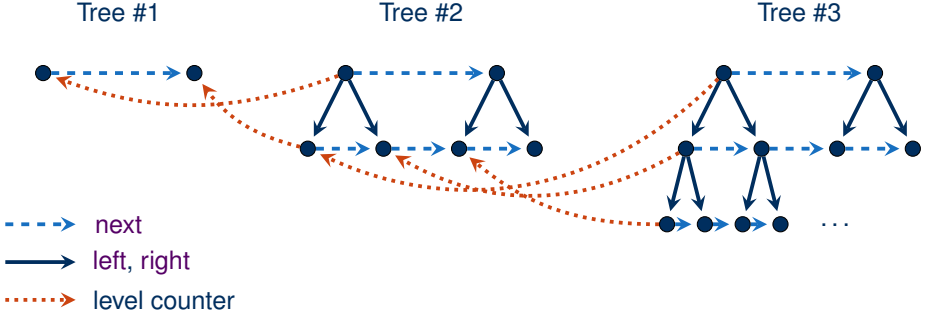
Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



Building a long chain

We construct a series of k full binary trees of depth $2, 2^2, 2^{2^2}, \ldots$ (we omit the roots)



Building a long chain

And here are the rules:

$$\mathsf{first}(v) \rightarrow \exists x.\mathsf{start}(x, x, v) \wedge \mathsf{end}(x)$$

$$\mathsf{start}(x, u, v) \wedge \mathsf{end}(u) \wedge \mathsf{next}(v, v') \rightarrow \exists y_1, y_2.\mathsf{start}(y_1, x, v') \wedge \mathsf{succ}(y_1, y_2) \wedge \mathsf{end}(y_2)$$

$$\begin{aligned}
\mathsf{start}(x, u, v) \wedge \mathsf{succ}(u, u') &\rightarrow \exists y.\mathsf{left}(x, y) \wedge \mathsf{start}(y, u', v) \\
\mathsf{left}(x, y) &\rightarrow \exists y'.\mathsf{right}(x, y') \wedge \mathsf{succ}(y, y') \\
\mathsf{right}(x, y) \wedge \mathsf{succ}(x, x') &\rightarrow \exists y'.\mathsf{left}(x', y') \wedge \mathsf{succ}(y, y') \\
\mathsf{end}(x) \wedge \mathsf{right}(x, y) &\rightarrow \mathsf{end}(y)
\end{aligned}$$

Ensuring termination

The chain construction works even in the skolem chase for inputs $\mathsf{first}(e_0), \mathsf{next}(e_0, e_1), \ldots, \mathsf{next}(e_{k-1}, e_k), \mathsf{last}(e_k)$.

Ensuring termination

The chain construction works even in the skolem chase for inputs $\mathsf{first}(e_0), \mathsf{next}(e_0, e_1), \ldots, \mathsf{next}(e_{k-1}, e_k), \mathsf{last}(e_k)$.

But every chase fails to terminate if the next-graph is cyclic.

Ensuring termination

The chain construction works even in the skolem chase for inputs $\mathsf{first}(e_0), \mathsf{next}(e_0, e_1), \ldots, \mathsf{next}(e_{k-1}, e_k), \mathsf{last}(e_k)$.

But every chase fails to terminate if the next-graph is cyclic.

We can ensure termination for Datalog-first chase using rules as follows:

$$\mathsf{next}(x, y) \rightarrow \mathsf{tnext}(x, y)$$
$$\mathsf{tnext}(x, y) \wedge \mathsf{next}(y, z) \rightarrow \mathsf{tnext}(x, z)$$

cycle detection

$$\mathsf{tnext}(x, x) \wedge \mathsf{adom}(u) \wedge \mathsf{adom}(v) \wedge \mathsf{adom}(w) \rightarrow \mathsf{start}(u, v, w)$$

flooding rule(s)

$$\ldots$$

- The rules are applied before applying existential rules (Datalog first)
- The flooding rules stop the introduction of new nulls
- Further rules are needed:
  also flood other atoms; extract adom from other atoms

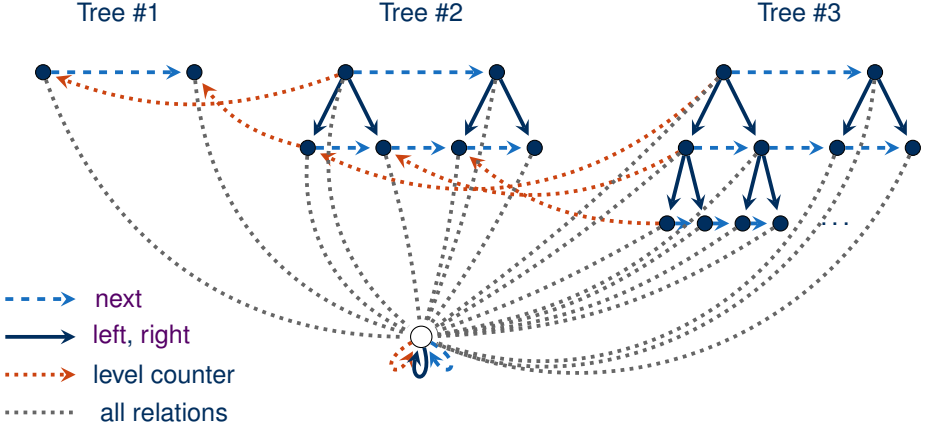
Beyond P in standard chase

Problem: We rely on the Datalog-first chase. With standard chase, cycles will eventually be found (fairness), but flooding rules might be applied too late to prevent new nulls

Beyond P in standard chase

Problem: We rely on the Datalog-first chase. With standard chase, cycles will eventually be found (fairness), but flooding rules might be applied too late to prevent new nulls

Solution: Create an emergency break that can be activated with just one rule application.



Beyond P in standard chase

Problem: We rely on the Datalog-first chase. With standard chase, cycles will eventually be found (fairness), but flooding rules might be applied too late to prevent new nulls

Solution: Create an emergency break that can be activated with just one rule application.

- Use a new unary predicate real, expected for all domain elements used so far, and require it for all elements in all rules
- Introduce a unique "break" element that is almost a critical instance, but not real:

$$\rightarrow \exists x.\mathsf{break}(x) \wedge \underbrace{\mathsf{start}(x,x,x) \wedge \mathsf{succ}(x,x) \wedge \mathsf{end}(x) \wedge \ldots}_{\text{all possible facts over } x\text{, except } \mathsf{real}(x)}$$

- Connect new nulls to this "break" element upon creation
  (doing this properly requires a rewriting to split start into several binary relations)
- To activate the emergency break, use a rule like

$$\mathsf{tnext}(x,x) \wedge \mathsf{break}(u) \rightarrow \mathsf{real}(u)$$

Beyond P in standard chase

Summary

- The emergency-break technique ensures termination by relying on fairness
- Hence, there are no runtime guarantees, but the rules are in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ and a BCQ q that express a non-elementary Boolean query.

Note: Very recent research found that fairness is never needed for termination when rules have a single head atom only [Gogacz et al. CoRR abs/1901.03897, 2019]

Part 3:

Expressivity in P

Back to P

High expressivity can be a plus, but many practical problems are in P anyway.

Can the terminating skolem chase solve every problem in P?

Back to P

High expressivity can be a plus, but many practical problems are in P anyway.

Can the terminating skolem chase solve every problem in P?

No, certainly not:

- Like Datalog, it does not capture P
- One would need to add input negation and linear order
- In particular, Datalog (and so $\mathrm{CT}^{\mathrm{sk}}_{\forall}$) can only express queries that are closed under homomorphism (i.e., monotone)

Back to P

High expressivity can be a plus, but many practical problems are in P anyway.

Can the terminating skolem chase solve every problem in P?

No, certainly not:

- Like Datalog, it does not capture P
- One would need to add input negation and linear order
- In particular, Datalog (and so $\mathrm{CT}^{\mathrm{sk}}_{\forall}$) can only express queries that are closed under homomorphism (i.e., monotone)

Can the terminating skolem chase express every homomorphism-closed query in P?

Back to P

High expressivity can be a plus, but many practical problems are in P anyway.

Can the terminating skolem chase solve every problem in P?

No, certainly not:

- Like Datalog, it does not capture P
- One would need to add input negation and linear order
- In particular, Datalog (and so $\mathrm{CT}^{\mathrm{sk}}_{\forall}$) can only express queries that are closed under homomorphism (i.e., monotone)

Can the terminating skolem chase express every homomorphism-closed query in P?

No, but this is less obvious:

- Dawar & Kreutzer discovered a homomorphism closed, polynomial time query that cannot be expressed in Datalog [ICALP'08]

The Dawar & Kreutzer query

Input: A directed graph G with two distinguished vertices s and t
Question:
Is G cyclic, or is there a simple path from s to t of length $2^{2^{n^2}}$ for some $n \in \mathbb{N}$?

The Dawar & Kreutzer query

Input: A directed graph G with two distinguished vertices s and t
Question:
Is G cyclic, or is there a simple path from s to t of length $2^{2^{n^2}}$ for some $n \in \mathbb{N}$?

Definition: The **DK query** $\mathfrak{D}_{\mathsf{DK}}$ is the abstract Boolean query containing exactly those concrete databases that encode an instance of this decision problem using a binary relation edge and constant symbols s and t.

This query is:

- closed under homomorphisms
- solvable in polynomial time
- not expressible in Datalog, hence not expressible in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$

DK in Datalog-first chase

We find that Datalog-first is more powerful than skolem even for polynomial problems:

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ and BCQ q that realise the DK query. The Datalog-first chase on Σ is polynomial in the size of the input database.

DK in Datalog-first chase

We find that Datalog-first is more powerful than skolem even for polynomial problems:

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ and BCQ q that realise the DK query. The Datalog-first chase on Σ is polynomial in the size of the input database.

Proof: The rule set consists of three parts:

(A) Rules to create a "yardstick" to measure the length of single paths starting in s

(B) Rules to derive simple arithmetic relations on this yardstick

(C) Rules to stop the chase if there is a cycle

Our construction is inspired by a previous encoding by Rudolph & Thomazo [IJCAI'15].

(A) The Yardstick

$$\rightarrow \exists x.\mathsf{zero}(x) \qquad \text{start chain}$$

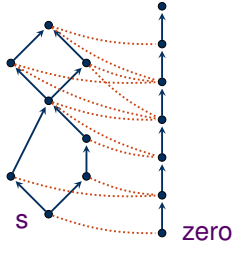
$$\mathsf{zero}(x) \rightarrow \mathsf{dist}(\mathsf{s}, x) \qquad \text{distance of s to s}$$

$$\mathsf{dist}(v, x) \rightarrow \exists x'.\mathsf{succ}(x, x') \qquad \text{extend chain}$$

$$\mathsf{dist}(v_1, x_1) \wedge \mathsf{edge}(v_1, v_2) \wedge \mathsf{succ}(x_1, x_2) \rightarrow \mathsf{dist}(v_2, x_2) \qquad \text{distance of } v_2 \text{ to s}$$

This creates a chain of succ relations, starting from a zero element

- The chain is only extended if its last elements are needed to measure some distance
- There will only be one chain for the whole graph



(B) The Arithmetic

$$\begin{aligned}
\mathsf{zero}(x) \wedge \mathsf{dist}(v, y) &\rightarrow \mathsf{add}(x, y, y) \wedge \mathsf{mul}(x, y, x) \\
\mathsf{add}(x, y, z) \wedge \mathsf{succ}(x, x') \wedge \mathsf{succ}(z, z') &\rightarrow \mathsf{add}(x', y, z') \\
\mathsf{mul}(x, y, z) \wedge \mathsf{succ}(x, x') \wedge \mathsf{add}(z, y, z') &\rightarrow \mathsf{mul}(x', y, z') \\
\mathsf{zero}(x) \wedge \mathsf{succ}(x, x') &\rightarrow \mathsf{exp}(x, x') \\
\mathsf{exp}(x, y) \wedge \mathsf{succ}(x, x') \wedge \mathsf{add}(y, y, y') &\rightarrow \mathsf{exp}(x', y') \\
\mathsf{mul}(x, x, y) \wedge \mathsf{exp}(y, y') \wedge \mathsf{exp}(y', z) \wedge \mathsf{dist}(\mathsf{t}, z) &\rightarrow \mathsf{goal}
\end{aligned}$$

The arithmetic is easy to implement:

- $\mathsf{add}(x, y, z)$ means "$x + y = z$"; $\mathsf{mul}(x, y, z)$ means "$x * y = z$"; $\mathsf{exp}(x, y)$ means "$2^x = y$"
- The last rule recognises the query condition

(C) The Cycle Stopper

$$\mathsf{edge}(v_1, v_2) \rightarrow \mathsf{path}(v_1, v_2)$$
$$\mathsf{edge}(v_1, v_2) \wedge \mathsf{path}(v_2, v_3) \rightarrow \mathsf{path}(v_1, v_3)$$
$$\mathsf{path}(v, v) \rightarrow \mathsf{goal}$$
$$\mathsf{dist}(v, x) \wedge \mathsf{goal} \rightarrow \mathsf{succ}(x, x)$$

Rules for cycle detection and ensuring termination

- The first three rules derive the goal on cycles
- The last rule will prevent new nulls to be created for extending the Yardstick

This completes the construction. It is easy to verify the claim. □

How about the standard chase?

The rules we used to realise the DK query are in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ but not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$:

- The flooding rule $\mathsf{dist}(v, x) \wedge \mathsf{goal} \rightarrow \mathsf{succ}(x, x)$ might be applied too late

How about the standard chase?

The rules we used to realise the DK query are in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ but not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$:

- The flooding rule $\mathsf{dist}(v, x) \wedge \mathsf{goal} \rightarrow \mathsf{succ}(x, x)$ might be applied too late

We can fix this by creating an emergency break as in the non-elementary computation:

- Add an almost-critical instance, connected to all new nulls
- Rely on fairness to activate the break if a cycle exists

How about the standard chase?

The rules we used to realise the DK query are in $\mathrm{CT}^{\mathsf{dlf}}_{\forall\forall}$ but not in $\mathrm{CT}^{\mathsf{std}}_{\forall\forall}$:

- The flooding rule $\mathsf{dist}(v, x) \wedge \mathsf{goal} \rightarrow \mathsf{succ}(x, x)$ might be applied too late

We can fix this by creating an emergency break as in the non-elementary computation:

- Add an almost-critical instance, connected to all new nulls
- Rely on fairness to activate the break if a cycle exists

Result: The DK query is in $\mathrm{CT}^{\mathsf{std}}_{\forall\forall}$, but our realisation does not yield any runtime bound

How about the standard chase?

The rules we used to realise the DK query are in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ but not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$:

- The flooding rule $\mathsf{dist}(v, x) \wedge \mathsf{goal} \rightarrow \mathsf{succ}(x, x)$ might be applied too late

We can fix this by creating an emergency break as in the non-elementary computation:

- Add an almost-critical instance, connected to all new nulls
- Rely on fairness to activate the break if a cycle exists

Result: The DK query is in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, but our realisation does not yield any runtime bound

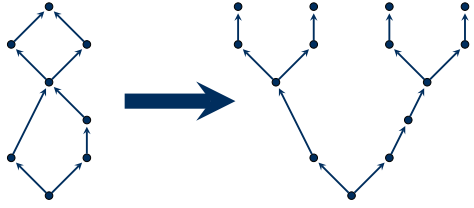
But we can do better:

Theorem: There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ and BCQ q that realise the DK query. The standard chase on Σ is at most exponential in the size of the input database.

DK in exponential standard chase

Basic idea:

- Replace the Yardstick by a tree that is an unravelled version of the graph:

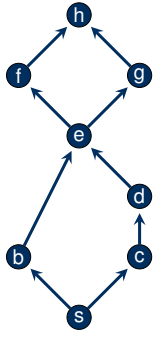


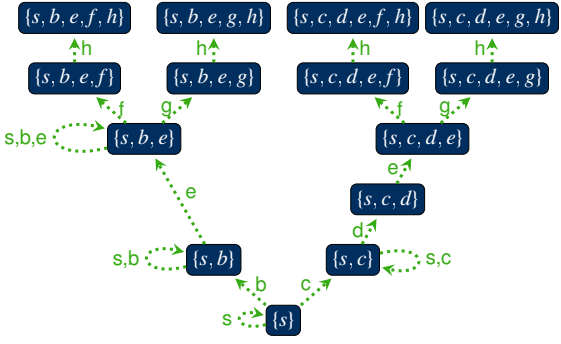
- It is then easy to use tree nodes as distances and to do arithmetic on the tree.

DK in exponential standard chase

Elements of the tree correspond to simple paths, considered as sets of vertices

- ins(v, x, y) means "$\{v\} \cup x = y$"; especially ins(v, x, x) means "$v \in x$"
- subset(v, w) means "$x \subset y$"

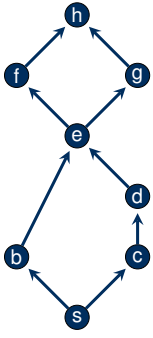


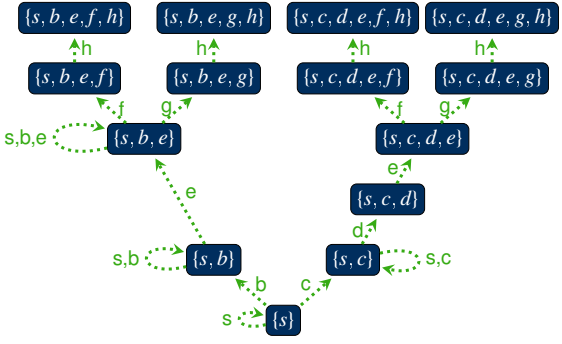


DK in exponential standard chase

Elements of the tree correspond to simple paths, considered as sets of vertices

- ins(v, x, y) means "$\{v\} \cup x = y$"; especially ins(v, x, x) means "$v \in x$"
- subset(v, w) means "$x \subset y$"





Termination trick:

- new sets are only created by adding elements that are not included yet
- relevant facts ins(v, x, x) derived in Datalog, before adding more elements

DK in exponential standard chase

And here are the rules for building the tree:

$$\rightarrow \exists x.\mathsf{zero}(x) \tag{1}$$

$$\mathsf{zero}(x) \rightarrow \mathsf{dist}(\mathsf{s}, x) \wedge \mathsf{ins}(\mathsf{s}, x, x) \wedge \mathsf{done}(x) \tag{2}$$

$$\mathsf{dist}(v_1, x_1) \wedge \mathsf{edge}(v_1, v_2) \wedge \mathsf{done}(x_1) \rightarrow \exists x_2.\mathsf{ins}(v_2, x_1, x_2) \wedge \mathsf{subset}(x_2, x_2) \tag{3}$$

$$\mathsf{subset}(x_1, x_2) \wedge \mathsf{ins}(v, x_0, x_1) \rightarrow \mathsf{ins}(v, x_2, x_2) \wedge \mathsf{subset}(x_0, x_2) \tag{4}$$

$$\mathsf{subset}(x_1, x_2) \wedge \mathsf{zero}(x_1) \rightarrow \mathsf{ins}(\mathsf{s}, x_2, x_2) \wedge \mathsf{done}(x_2) \tag{5}$$

$$\mathsf{dist}(v_1, x_1) \wedge \mathsf{edge}(v_1, v_2) \wedge \mathsf{ins}(v_2, x_1, x_2) \rightarrow \mathsf{dist}(v_2, x_2) \wedge \mathsf{succ}(x_1, x_2) \tag{6}$$

When a new set is created (3), we recursively copy elements from all subsets (4), and only consider the new set “done” when the empty set is reached (5).

(the rest of the proof is as before)

Discussion and Questions

What we have shown

Notation:

- We refine termination classes by adding a function class, e.g.,

  $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{poly})$ means "rules on which the standard chase terminates in polynomial time (for all strategies and instances)"

- For each rule language CT, we studied the class $[\![\mathsf{CT}]\!]$ of abstract queries that it can realise (using BCQs)

What we have shown

Notation:

- We refine termination classes by adding a function class, e.g.,

  $\mathrm{CT}^{\mathrm{std}}_{\forall\forall}(\mathrm{poly})$ means "rules on which the standard chase terminates in polynomial time (for all strategies and instances)"

- For each rule language CT, we studied the class $[\![\mathrm{CT}]\!]$ of abstract queries that it can realise (using BCQs)

Summary of results:

$$[\![\mathrm{Datalog}]\!] = [\![\mathrm{CT}^{\mathrm{sk}}_{\forall}]\!]$$
$$\shortparallel$$
$$[\![\mathrm{CT}^{\mathrm{sk}}_{\forall}(\mathrm{poly})]\!]$$

What we have shown

Notation:

- We refine termination classes by adding a function class, e.g.,

  $\mathrm{CT}^{\mathrm{std}}_{\forall\forall}(\mathrm{poly})$ means "rules on which the standard chase terminates in polynomial time (for all strategies and instances)"

- For each rule language CT, we studied the class $[\![\mathrm{CT}]\!]$ of abstract queries that it can realise (using BCQs)

Summary of results:

$$[\![\mathrm{Datalog}]\!] = [\![\mathrm{CT}^{\mathrm{sk}}_{\forall}]\!] \subset [\![\mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}(\mathrm{poly})]\!]$$

$$\| $$

$$[\![\mathrm{CT}^{\mathrm{sk}}_{\forall}(\mathrm{poly})]\!]$$

What we have shown

Notation:

- We refine termination classes by adding a function class, e.g.,

  $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{poly})$ means "rules on which the standard chase terminates in polynomial time (for all strategies and instances)"

- For each rule language CT, we studied the class $[\![\mathsf{CT}]\!]$ of abstract queries that it can realise (using BCQs)

Summary of results:

$$
\begin{array}{ccccccc}
[\![\mathsf{Datalog}]\!] = [\![\mathsf{CT}^{\mathsf{sk}}_{\forall}]\!] & \subset & [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\mathsf{poly})]\!] & \subset & [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\mathsf{exp})]\!] & \subset \ldots \subset & [\![\bigcup_k \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(k\text{-exp})]\!] \\
\| & & \cup\| & & \cup\| & & \cup\| \\
[\![\mathsf{CT}^{\mathsf{sk}}_{\forall}(\mathsf{poly})]\!] & \subseteq & [\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{poly})]\!] & \subset & [\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{exp})]\!] & \subset \ldots \subset & [\![\bigcup_k \mathsf{CT}^{\mathsf{std}}_{\forall\forall}(k\text{-exp})]\!]
\end{array}
$$

What we have shown

Notation:

- We refine termination classes by adding a function class, e.g.,

  $\mathrm{CT}^{\mathrm{std}}_{\forall\forall}(\mathrm{poly})$ means "rules on which the standard chase terminates in polynomial time (for all strategies and instances)"

- For each rule language CT, we studied the class $[\![\mathrm{CT}]\!]$ of abstract queries that it can realise (using BCQs)

Summary of results:

$$
\begin{array}{ccccccccc}
[\![\mathrm{Datalog}]\!] = [\![\mathrm{CT}^{\mathrm{sk}}_{\forall}]\!] & \subset & [\![\mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}(\mathrm{poly})]\!] & \subset & [\![\mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}(\mathrm{exp})]\!] & \subset \ldots \subset & [\![\bigcup_k \mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}(k\text{-exp})]\!] & \subset & [\![\mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}]\!] \\
\| & & \cup\| & & \cup\| & & \cup\| & & \cup\| \\
[\![\mathrm{CT}^{\mathrm{sk}}_{\forall}(\mathrm{poly})]\!] & \subseteq & [\![\mathrm{CT}^{\mathrm{std}}_{\forall\forall}(\mathrm{poly})]\!] & \subset & [\![\mathrm{CT}^{\mathrm{std}}_{\forall\forall}(\mathrm{exp})]\!] & \subset \ldots \subset & [\![\bigcup_k \mathrm{CT}^{\mathrm{std}}_{\forall\forall}(k\text{-exp})]\!] & \subset & [\![\mathrm{CT}^{\mathrm{std}}_{\forall\forall}]\!]
\end{array}
$$

What we have learned

What we have learned

Lesson 1: If we want robust chase termination, the skolem chase is much less expressive than the standard chase

- Differences in highly expressive as well as polynomial queries
- Using function symbols instead of $\exists$ might cost expressive power

What we have learned

Lesson 1: If we want robust chase termination, the skolem chase is much less expressive than the standard chase

- Differences in highly expressive as well as polynomial queries
- Using function symbols instead of $\exists$ might cost expressive power

Lesson 2: Our current rule reasoners are more powerful than we thought

- Any restricted chase implementation is sound and complete for $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, not just for decidable fragments thereof
- Datalog-first is often implemented already (as a natural heuristic)

What we have learned

Lesson 1: If we want robust chase termination, the skolem chase is much less expressive than the standard chase

- Differences in highly expressive as well as polynomial queries
- Using function symbols instead of $\exists$ might cost expressive power

Lesson 2: Our current rule reasoners are more powerful than we thought

- Any restricted chase implementation is sound and complete for $\mathrm{CT}^{\mathrm{std}}_{\forall\forall}$, not just for decidable fragments thereof
- Datalog-first is often implemented already (as a natural heuristic)

Lesson 3: A Datalog-first chase strategy might have worst-case runtime benefits

- Other ways of slightly regulating chase strategies might also work
- Even (truly) random strategies might lead to almost certain termination

Questions

Questions

- **Absolute expressibility** Does some chase capture all homomorphism-closed queries? If not, what does it capture?

Questions

- **Absolute expressibility** Does some chase capture all homomorphism-closed queries? If not, what does it capture?
- **Relative expressibility** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!]$? If not, can we rewrite rule sets? How about the core chase?

Questions

- **Absolute expressibility** Does some chase capture all homomorphism-closed queries? If not, what does it capture?
- **Relative expressibility** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!]$? If not, can we rewrite rule sets? How about the core chase?
- **Complexity relationships** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{poly})]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\mathsf{poly})]\!]$ strict? (conjecture: yes) Is the penalty always exponential? Can the Datalog-first chase always be worst-case optimal?

Questions

- **Absolute expressibility** Does some chase capture all homomorphism-closed queries? If not, what does it capture?
- **Relative expressibility** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!]$? If not, can we rewrite rule sets? How about the core chase?
- **Complexity relationships** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{poly})]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\mathsf{poly})]\!]$ strict? (conjecture: yes) Is the penalty always exponential? Can the Datalog-first chase always be worst-case optimal?
- **Decidable termination criteria** Which sufficient criteria can detect termination beyond skolem?

Questions

- **Absolute expressibility** Does some chase capture all homomorphism-closed queries? If not, what does it capture?
- **Relative expressibility** Is $[\![\mathrm{CT}^{\mathrm{std}}_{\forall\forall}]\!] \subset [\![\mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}]\!]$? If not, can we rewrite rule sets? How about the core chase?
- **Complexity relationships** Is $[\![\mathrm{CT}^{\mathrm{std}}_{\forall\forall}(\mathrm{poly})]\!] \subset [\![\mathrm{CT}^{\mathrm{dlf}}_{\forall\forall}(\mathrm{poly})]\!]$ strict? (conjecture: yes) Is the penalty always exponential? Can the Datalog-first chase always be worst-case optimal?
- **Decidable termination criteria** Which sufficient criteria can detect termination beyond skolem?
- **Termination on restricted database classes** Should we consider "not-quite-universal termination" that imposes requirements on the database? Which?

Questions

- **Absolute expressibility** Does some chase capture all homomorphism-closed queries? If not, what does it capture?
- **Relative expressibility** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!]$? If not, can we rewrite rule sets? How about the core chase?
- **Complexity relationships** Is $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\mathsf{poly})]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\mathsf{poly})]\!]$ strict? (conjecture: yes) Is the penalty always exponential? Can the Datalog-first chase always be worst-case optimal?
- **Decidable termination criteria** Which sufficient criteria can detect termination beyond skolem?
- **Termination on restricted database classes** Should we consider "not-quite-universal termination" that imposes requirements on the database? Which?
- **Practical applications** How to exploit beyond-skolem expressive power in practice?

TL;DR

One can still learn new things about the chase . . .

Lessons learnt

- Skolem chase is just Datalog in disguise – standard chase is more
- Existing rule reasoners could be used ways not considered yet
- Rule reasoners should offer some chase strategy control

. . . but there are many open questions


JOIN US IN DRESDEN
Several 4yr positions from 2019
Ph.D. students & postdocs
100% English & International
KR / Databases / AI
CONTACT
Markus Krötzsch markus.kroetzsch@tu-dresden.de