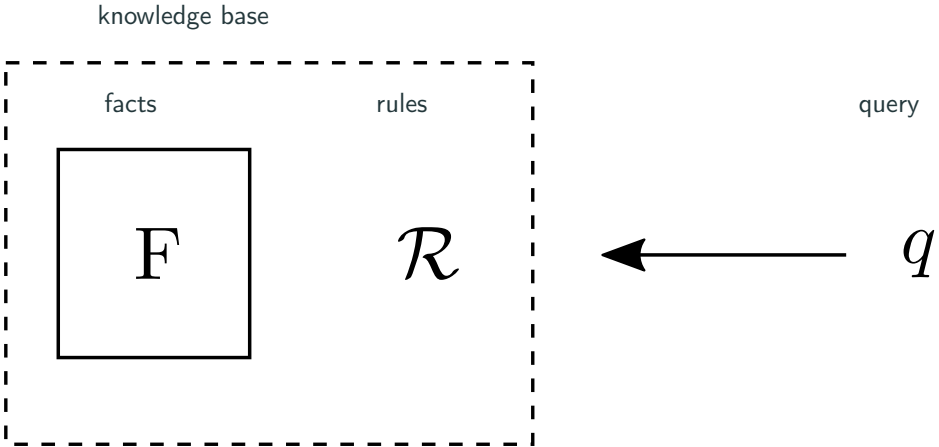
Rewriting the Infinite Chase

Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, Boris Motik

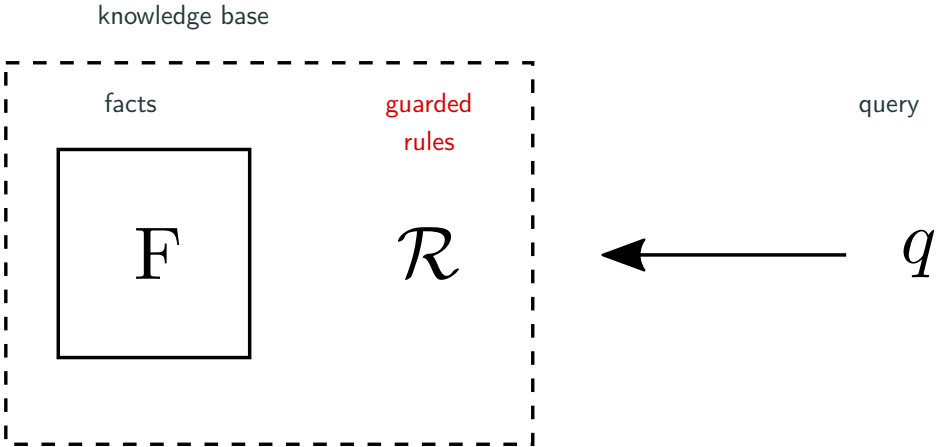
12/05/2023 - IRIF

Context

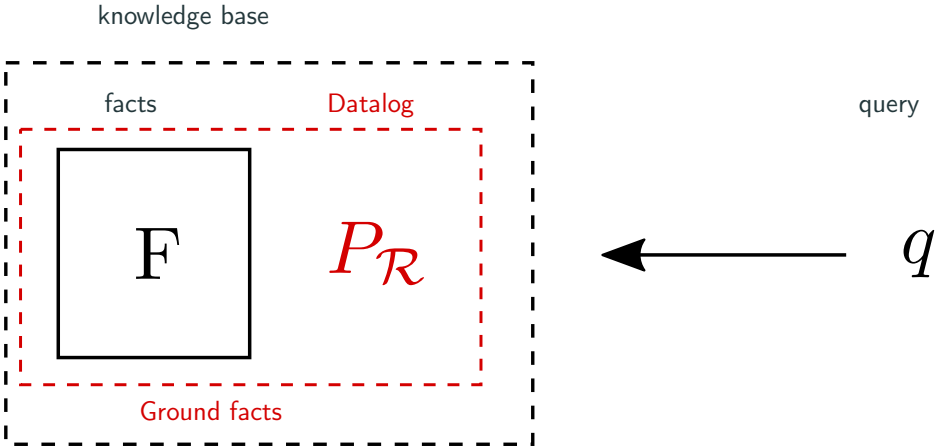
Ontology Mediated Query Answering



Context: Ontology Mediated Query Answering with Guarded Rules



Problem Statement: Datalog Rewriting



Contributions

Our contributions in the context of guarded rules rewriting are:

- a theoretical framework to understand and show completness of rewriting algorithms
- several concrete rewriting algorithms
- an empirical evaluation of our algorithms using an extensive benchmark

Guarded Rules

Definition

An **existential rule** is a FOL formula $r = \forall \bar{x} \forall \bar{y} B(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} H(\bar{x}, \bar{z})$

The rule r is **guarded** if there exists an atom $G \in B$ such that $\mathrm{var}(G) = \bar{x} \cup \bar{y}$.

We say that G is a **guard** of r.

A **Datalog rule** is a existential rule where $\bar{z} = \emptyset$.

Query answering with guarded rules

- 2EXP-TIME-c combined complexity
- PTIME-c data complexity
- No practical algorithm

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\mathrm{chase}_0(F, \mathcal{R}) = \underset{P}{a} \xrightarrow{R} b$$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\mathrm{chase}_1(F, \mathcal{R}) = \underset{P}{a} \xrightarrow{R} \underset{P}{b} \xrightarrow{R} n_0$$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\mathrm{chase}_2(F, \mathcal{R}) = \underset{P, S}{a} \xrightarrow{R} \underset{P}{b} \xrightarrow{R} \underset{P}{n_0} \xrightarrow{R} n_1$$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\mathrm{chase}_3(F, \mathcal{R}) = \underset{P, S}{a} \xrightarrow{R} \underset{P, S}{b} \xrightarrow{R} \underset{P}{n_0} \xrightarrow{R} \underset{P}{n_1} \xrightarrow{R} \cdots$$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\mathrm{chase}_\infty(F, \mathcal{R}) = a \xrightarrow[P,S]{R} b \xrightarrow[P,S]{R} n_0 \xrightarrow[P,S]{R} n_1 \xrightarrow[P,S]{R} \cdots$$

Query rewriting is not finite

$Q(x) \leftarrow S(x)$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\text{chase}(F, \mathcal{R}) = \underset{P,S}{a} \xrightarrow{R} \underset{P,S}{b} \xrightarrow{R} n_0 \xrightarrow{R} n_1 \xrightarrow{R} \cdots$$

Query rewriting is not finite

$Q(x) \leftarrow S(x)$

$Q_1(x) \leftarrow R(x, y) \wedge P(y)$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\text{chase}(F, \mathcal{R}) = \quad a \xrightarrow[P,S]{R} b \xrightarrow[P,S]{R} n_0 \xrightarrow{R} n_1 \xrightarrow{R} \cdots$$

Query rewriting is not finite

$Q(x) \leftarrow S(x)$

$Q_1(x) \leftarrow R(x, y) \wedge P(y)$

$Q_2(x) \leftarrow R(x, y) \wedge P(x_1) \wedge R(x_1, y)$

Motivating Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Chase is infinite

$$\text{chase}(F, \mathcal{R}) = a \xrightarrow[P,S]{R} b \xrightarrow[P,S]{R} n_0 \xrightarrow{R} n_1 \xrightarrow{R} \cdots$$

Query rewriting is not finite

$Q(x) \leftarrow S(x)$

$Q_1(x) \leftarrow R(x, y) \wedge P(y)$

$Q_2(x) \leftarrow R(x, y) \wedge P(x_1) \wedge R(x_1, y)$

$Q_3(x) \leftarrow R(x, y) \wedge P(x_2) \wedge R(x_2, x_1) \wedge R(x_1, y)$

$\ldots$

Datalog Rewriting of a Rule Set

Definition

Given $\mathcal{R}$ a set of guarded rules, $P_{\mathcal{R}}$ is a **Datalog rewriting** of $\mathcal{R}$ iff:

1. $P_{\mathcal{R}}$ is a finite set of Datalog rules,
2. for every F fact base and every A ground fact,

$$F, \mathcal{R} \models A \quad \text{iff} \quad F, P_{\mathcal{R}} \models A.$$

Existential-free conjunctive query answering

We can answer any conjunctive query Q without existential variable by evaluating Q on $\mathrm{chase}_\infty(F, P_{\mathcal{R}})$.

Datalog Rewriting Example

Facts: $F = \{P(a), R(a, b)\}$
Query: $Q(x) \leftarrow S(x)$

Rules:

1. $R(x, y) \rightarrow \exists z\ R(y, z)$
2. $P(x) \wedge R(x, y) \rightarrow P(y)$
3. $R(x, y) \wedge P(y) \rightarrow S(x)$

Infinite chase

$$\mathrm{chase}_\infty(F, \mathcal{R}) = a \underset{P,S}{\xrightarrow{R}} b \underset{P,S}{\xrightarrow{R}} n_0 \underset{P,S}{\xrightarrow{R}} n_1 \underset{P,S}{\xrightarrow{R}} \cdots$$

A Datalog rewriting of $\mathcal{R}$ is the set $P_\mathcal{R}$ containing the rule 2, 3 and :

$$R(x, y) \wedge P(y) \rightarrow S(y)$$

$$\mathrm{chase}_\infty(F, P_\mathcal{R}) = a \underset{P,S}{\xrightarrow{R}} b$$

Outline

Tree-like Chase

Existential-based Datalog Rewriting

Skolem Datalog Rewriting

Implementation and Experiments

Tree-like Chase

Head Normal Form

Definition

A rule r is in **Head Normal Form (HNF)**, if r is Datalog or every atom in the head of r contains at least one existential variable.

We can always obtain from any rule a equivalent set of rules in HNF.

Example

The rule $A(x) \rightarrow \exists y\ B(x, y) \wedge C(y) \wedge G(x)$ becomes the following rules in HNF:

- $A(x) \rightarrow \exists y\ B(x, y) \wedge C(y)$
- $A(x) \rightarrow G(x)$

Remark

A rule will be either :

- a Datalog rule
- a non-full rule

Tree-like Chase

Definition

We say that A an atom **is guarded in** an atom set S if there exists $B \in S$ such that $\mathrm{terms}(A) \subseteq \mathrm{terms}(B)$.

Definition

A **chase tree** T is a directed tree where:

- each vertex contains a set of facts
- one vertex is recently updated

Definition

A **tree-like chase sequence** for a set of guarded rules $\mathcal{R}$ and a set of facts F is a sequence $T_0, \ldots, T_n$ of chase tree with:

- T_0 has a single vertex containing F
- T_{i+1} is obtained from T_i by a chase step using a rule in $\mathcal{R}$ or a propagation step

Example of Tree-like Chase

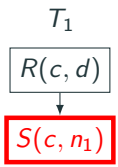
$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

T_0

$\boxed{R(c, d)}$

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

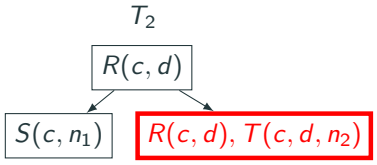


Chase step with a non-full rule

1. the rule is triggered by the facts in a parent vertex
2. a fresh child vertex is introduced containing the deduced facts

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

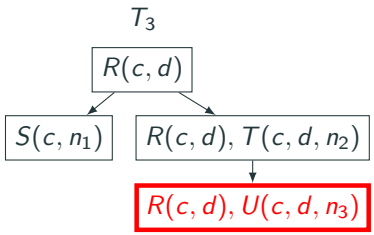


Chase step with a non-full rule

1. the rule is triggered by the facts in a parent vertex
2. a fresh child vertex is introduced containing the deduced facts
3. the child vertex inherits the parent facts that are guarded by the deduced facts

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

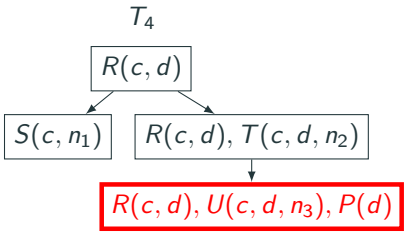


Chase step with a non-full rule

1. the rule is triggered by the facts in a parent vertex
2. a fresh child vertex is introduced containing the deduced facts
3. the child vertex inherits the parent facts that are guarded by the deduced facts

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

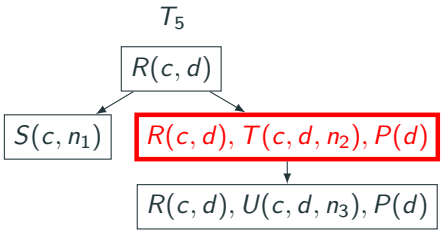


Chase step with a Datalog rule

- the rule is triggered by the facts in a vertex
- the deduced facts are added to the vertex

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

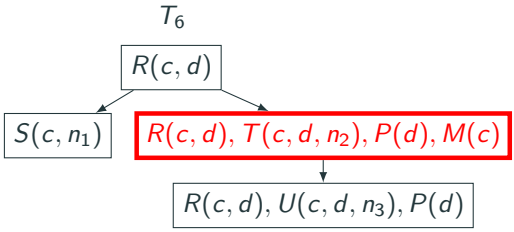


Propagation step

an atom A in a vertex v is copied in another vertex v', if A is guarded by the facts of v'

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

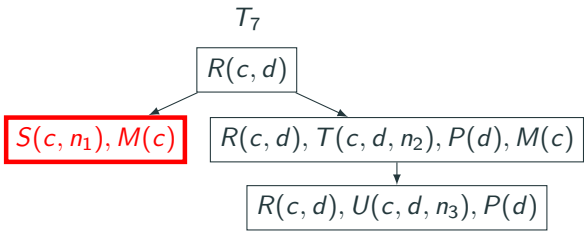


Chase step with a Datalog rule

- the rule is triggered by the facts in a vertex
- the deduced facts are added to the vertex

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$

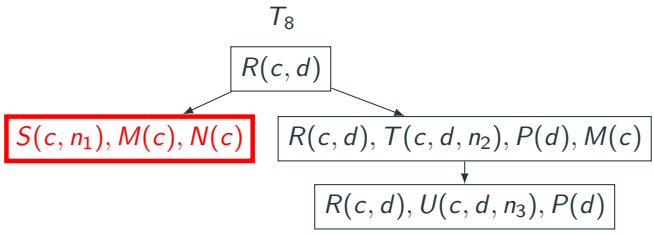


Propagation step

an atom A in a vertex v is copied in another vertex v', if A is guarded by the facts of v'

Example of Tree-like Chase

$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to N(x_1)$



Chase step with a Datalog rule

- the rule is triggered by the facts in a vertex
- the deduced facts are added to the vertex

Correction and Completeness of Tree-like Chase

Theorem

Given F a set of facts, $\mathcal{R}$ a guarded rule set, and Q a Boolean conjunctive query, $F, \mathcal{R} \models Q$ if and only if there exists $T_0, \ldots, T_n$ a tree-like chase sequence for $\mathcal{R}$ and F such that the facts in T_n implies Q.

The One-Pass Property

Definition

The tree-like chase sequence $T_0, \ldots, T_n$ is **one-pass** if:

- chase or propagate steps are applied to the recently-updated vertex,
- propagate steps only propagate a fact from a child to its parent,
- a chase step is applicable provided that no propagation step is applicable

The One-Pass Property

Definition

The tree-like chase sequence $T_0, \ldots, T_n$ is **one-pass** if:

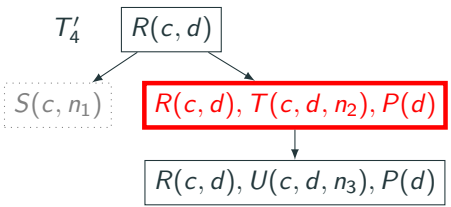
- chase or propagate steps are applied to the recently-updated vertex,
- propagate steps only propagate a fact from a child to its parent,
- a chase step is applicable provided that no propagation step is applicable

Theorem

*Given F a set of facts, $\mathcal{R}$ a guarded rule set, and Q a Boolean conjunctive query, $F, \mathcal{R} \models Q$ if and only if there exists $T_0, \ldots, T_n$ a **one-pass** tree-like chase sequence for $\mathcal{R}$ and F such that the facts in T_n implies Q.*

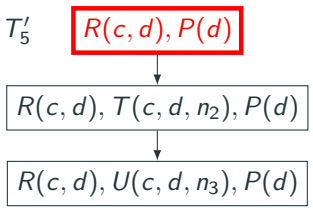
One-Pass Tree-like Chase Example

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$



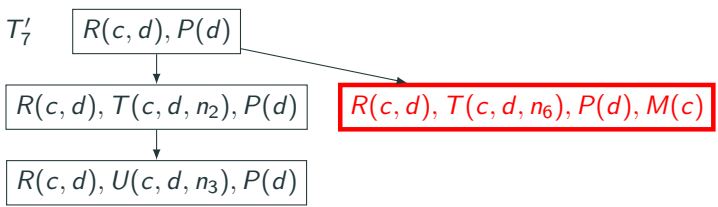
One-Pass Tree-like Chase Example

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$



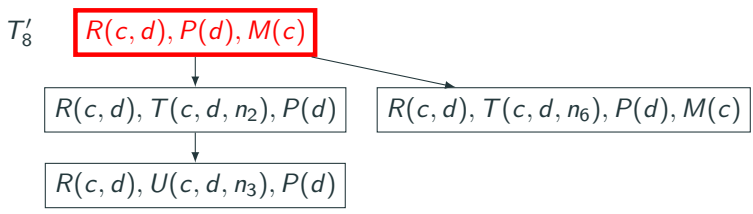
One-Pass Tree-like Chase Example

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$



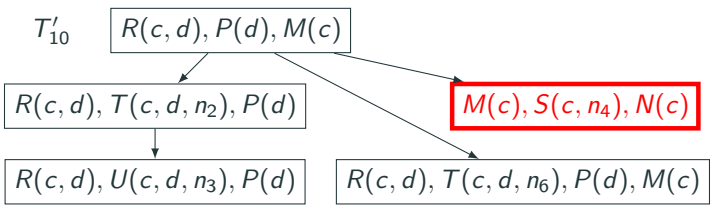
One-Pass Tree-like Chase Example

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$



One-Pass Tree-like Chase Example

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$



Loops and Datalog Rewriting

Definition

A **loop** at a vertex v is a subsequence $T_i, \ldots, T_j$ in a one-pass tree-like chase sequence, where:

- T_{i+1} is obtained by a chase step with a non-full rule,
- T_j is obtained by a propagation step that copies the atom A
- v is the recently updated vertex of both T_i and T_j

We call A the **output** of the loop.

Loops and Datalog Rewriting

Definition

A **loop** at a vertex v is a subsequence $T_i, \ldots, T_j$ in a one-pass tree-like chase sequence, where:

- T_{i+1} is obtained by a chase step with a non-full rule,
- T_j is obtained by a propagation step that copies the atom A
- v is the recently updated vertex of both T_i and T_j

We call A the **output** of the loop.

Theorem

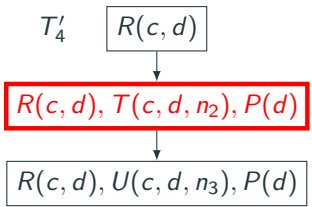
A Datalog rule set P is a Datalog rewriting of $\mathcal{R}$ a set of guarded rules if

- *P is a logical consequence of $\mathcal{R}$,*
- *each Datalog rule of $\mathcal{R}$ is a logical consequence of P,*
- *for each fact base F, a loop for F and $\mathcal{R}$ with the output A, we have $F, P \models A$.*

Example of Datalog Rewriting based on the Loops

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$

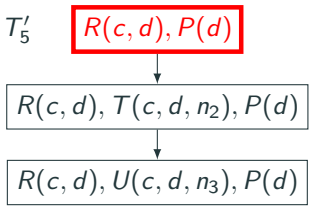
$T(x_1, x_2, x_3) \rightarrow P(x_2)$



Example of Datalog Rewriting based on the Loops

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$

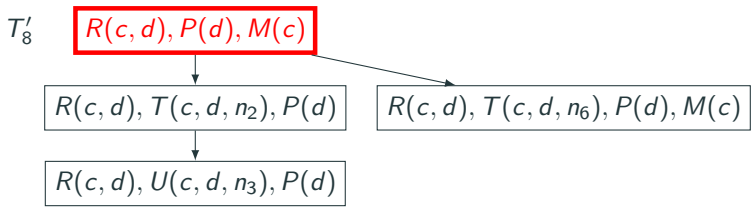
$T(x_1, x_2, x_3) \rightarrow P(x_2)$
$R(x_1, x_2) \rightarrow P(x_2)$



Example of Datalog Rewriting based on the Loops

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$

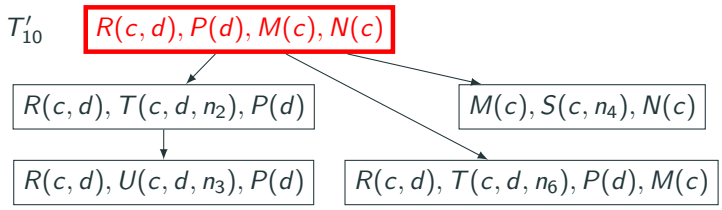
$T(x_1, x_2, x_3) \rightarrow P(x_2)$
$R(x_1, x_2) \rightarrow P(x_2)$
$R(x_1, x_2) \wedge P(x_2) \rightarrow M(x_1)$



Example of Datalog Rewriting based on the Loops

$R(x_1, x_2) \rightarrow \exists y\, S(x_1, y)$
$R(x_1, x_2) \rightarrow \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \rightarrow \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \rightarrow P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \rightarrow M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$

$T(x_1, x_2, x_3) \rightarrow P(x_2)$
$R(x_1, x_2) \rightarrow P(x_2)$
$R(x_1, x_2) \wedge P(x_2) \rightarrow M(x_1)$
$R(x_1, x_2) \wedge M(x_1) \rightarrow N(x_1)$



Existential-based Datalog Rewriting

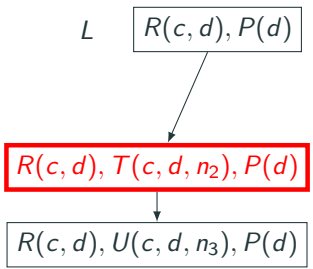
From Loops to Datalog Rewriting

1. $R(x_1, x_2) \rightarrow \exists y\ T(x_1, x_2, y)$
2. $T(x_1, x_2, x_3) \rightarrow \exists y\ U(x_1, x_2, y)$
3. $U(x_1, x_2, x_3) \rightarrow P(x_2)$

---

4. $T(x_1, x_2, x_3) \rightarrow P(x_2)$

Final state of a loop of output $P(d)$:



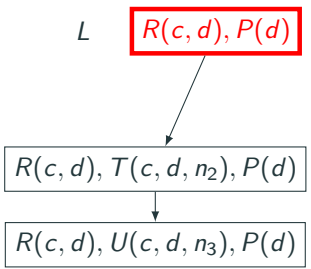
From Loops to Datalog Rewriting

1. $R(x_1, x_2) \rightarrow \exists y\ T(x_1, x_2, y)$
2. $T(x_1, x_2, x_3) \rightarrow \exists y\ U(x_1, x_2, y)$
3. $U(x_1, x_2, x_3) \rightarrow P(x_2)$

---

4. $T(x_1, x_2, x_3) \rightarrow P(x_2)$
5. $R(x_1, x_2) \rightarrow P(x_2)$

Final state of a loop of output $P(d)$:



The Existential-Based Rewriting Algorithm (ExbDR)

Definition

The **Existential-Based Datalog rewriting inference** takes

- a non-full rule τ
- a Datalog rule τ'
- a piece unifier u of the head of τ with the body of τ'

and returns the composition of τ with τ' with respect to u in HNF.

Definition

Given $\mathcal{R}$ a set of guarded rule in HNF, the **ExbDR algorithm**

1. applies the inference rule on the rules in $\mathcal{R}$ until it reaches a fixed point,
2. returns all Datalog rules.

The output is a Datalog rewriting of $\mathcal{R}$.

Existential-Based Rewriting Example

Input rules:

$$R(u, v) \rightarrow \exists w \; R(v, w) \qquad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \qquad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \qquad (3)$$

Inferred rules:

Existential-Based Rewriting Example

Input rules:

$$R(u, v) \rightarrow \exists w \; R(v, w) \quad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

$$R(u, v) \wedge A(v) \rightarrow \exists w \; R(v, w) \wedge A(w) \quad (4)$$

Existential-Based Rewriting Example

Input rules:

$$R(u,v) \rightarrow \exists w\ R(v,w) \quad (1)$$

$$A(x) \wedge R(x,y) \rightarrow A(y) \quad (2)$$

$$R(x,y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

$$R(u,v) \wedge A(v) \rightarrow \exists w\ R(v,w) \wedge A(w) \quad (4)$$

$$R(u,v) \wedge A(v) \rightarrow C(v) \quad (5)$$

Existential-Based Rewriting Example

Input rules:

$$R(u, v) \rightarrow \exists w\ R(v, w) \quad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

$$R(u, v) \wedge A(v) \rightarrow \exists w\ R(v, w) \wedge A(w) \quad (4)$$

$$R(u, v) \wedge A(v) \rightarrow C(v) \quad (5)$$

The outputted Datalog rewriting is (2), (3) and (5).

Terminaison and Complexity of ExbDR

Terminaison

- The rules inferred by ExbDR are guarded,
- The number of variable in the body (resp. head) of the rules inferred by ExbDR is bounded by the maximum number of variable in the body (resp. head) of the input rules.

Complexity

The complexity of ExbDR is 2-EXPTIME and EXPTIME, if the arity is bounded.

Skolem Datalog Rewriting

Rule Skolemization

Definition

We skolemize a rule by

1. replacing every existential variable by a Skolem term built from a fresh function symbol and the body variables,
2. splitting the resulting rule into single-headed rules.

Example

The non-full rule $A(x) \to \exists y\ B(x, y), C(y), G(x)$ becomes the following Skolemized rules:

- $A(x) \to B(x, f(x))$,
- $A(x) \to C(f(x))$,
- $A(x) \to G(x)$.

The Skolem Datalog Rewriting Algorithm (SkDR)

Definition

The **Skolem rewriting inference** takes

- a rule τ which has Skolem-free body and contains Skolem terms in the head
- a rule τ' which either is Skolem-free or has Skolem-free head
- a mgu u of the head of τ with a guard of τ' or an atom containing Skolem terms

and returns the composition of τ with τ' with respect to u in HNF.

Definition

Given $\mathcal{R}$ a set of Skolemized guarded rule, the **algorithm SkDR**

1. applies the inference on the rules in $\mathcal{R}$ until it reaches a fixed point,
2. returns every Skolem-free rules.

The output is a Datalog rewriting of $\mathcal{R}$.

Skolem Datalog Rewriting Example

Input rules:

$$R(u, v) \rightarrow R(v, f(u, v)) \quad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

Skolem Datalog Rewriting Example

Input rules:

$$R(u, v) \rightarrow R(v, f(u, v)) \quad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

$$R(u, v) \wedge A(v) \rightarrow A(f(u, v)) \quad (4)$$

Skolem Datalog Rewriting Example

Input rules:

$$R(u, v) \rightarrow R(v, f(u, v)) \quad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

$$R(u, v) \wedge A(v) \rightarrow A(f(u, v)) \quad (4)$$

$$R(u, v) \wedge A(f(u, v)) \rightarrow C(v) \quad (5)$$

Skolem Datalog Rewriting Example

Input rules:

$$R(u, v) \rightarrow R(v, f(u, v)) \quad (1)$$

$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$

$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

Inferred rules:

$$R(u, v) \wedge A(v) \rightarrow A(f(u, v)) \quad (4)$$

$$R(u, v) \wedge A(f(u, v)) \rightarrow C(v) \quad (5)$$

$$R(u, v) \wedge A(v) \rightarrow C(v) \quad (6)$$

Skolem Datalog Rewriting Example

Input rules:

$$R(u, v) \to R(v, f(u, v)) \quad (1)$$
$$A(x) \wedge R(x, y) \to A(y) \quad (2)$$
$$R(x, y) \wedge A(y) \to C(x) \quad (3)$$

Inferred rules:

$$R(u, v) \wedge A(v) \to A(f(u, v)) \quad (4)$$
$$R(u, v) \wedge A(f(u, v)) \to C(v) \quad (5)$$
$$R(u, v) \wedge A(v) \to C(v) \quad (6)$$

The outputted Datalog rewriting is (2), (3) and (6).

Comparison of SkDR and ExbDR

Proposition

There is a family of guarded rule set $(\mathcal{R}_n)_{n\in\mathbb{N}}$, where the number of inferred rules in ExbDR is exponentially larger than in SkDR.

Comparison of SkDR and ExbDR

Proposition

There is a family of guarded rule set $(\mathcal{R}_n)_{n\in\mathbb{N}}$, where the number of inferred rules in ExbDR is exponentially larger than in SkDR.

Proposition

There is a family of guarded rule set $(\mathcal{R}_n)_{n\in\mathbb{N}}$, where the number of inferred rules in SkDR is exponentially larger than in ExbDR.

Implementation and Experiments

Implementation Details

For each algorithm we introduce :

- pairwise **redundant rules deletion** based on an approximation of rule subsumption
- a **subsumption index** to find candidate rules subsuming or subsumed by a given rule
- a **unification index** to find candidate rules on which the rewriting inference can be applied with a given rule

Experiments Setting

Datasets

1. Description Logics ontologies from the Oxford ontology library: 428 guarded rule sets.
2. Blown up version of these ontologies with arity up to 10 instead of 2 and "satellite atoms".

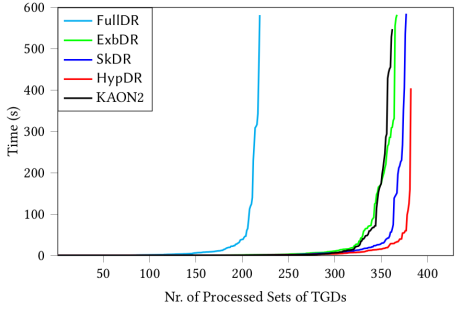
Inputs	#Non-full rules				#Datalog rules			
	Min	Max	Avg	Med	Min	Max	Avg	Med
428	1	172K	11K	789	2	157K	5K	283

Competitor

KAON2 for Description Logics only.

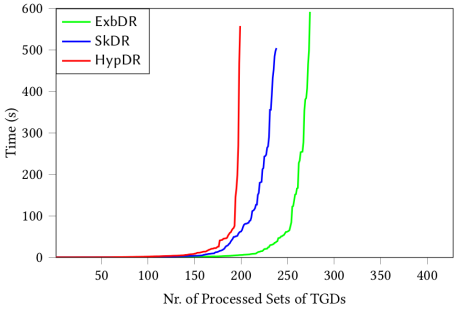
There is no competitor for general guarded rules.

Results for Rules Derived from Ontologies



X \ Y	time(Y)/time(X)>10 Exb	Sk	Hyper	KAON2	X and Y both fail Exb	Sk	Hyper	KAON2
Exb		19	0	19	61			
Sk	37		0	26	33	51		
Hyper	37	12		31	35	43	46	
KAON2	35	15	0		37	47	46	66

Results for Rules with Higher-Arity Relations



X \ Y	time(Y)/time(X)>10			X and Y both fail		
	Exb	Sk	Hyper	Exb	Sk	Hyper
Exb		61	87	154		
Sk	11		21	128	190	
Hyper	6	4		148	184	229

Conclusions

1. We studied a sufficient condition for a Datalog rule set to be a rewriting of a guarded rule set through the notion of loop in tree-like chase.
2. We proposed three Datalog rewriting algorithms and shows their differences
3. We implemented them with some optimizations and we conducted large experiments on them.

`http://www.cs.ox.ac.uk/people/maxime.buron/blog/gsat-exp/`