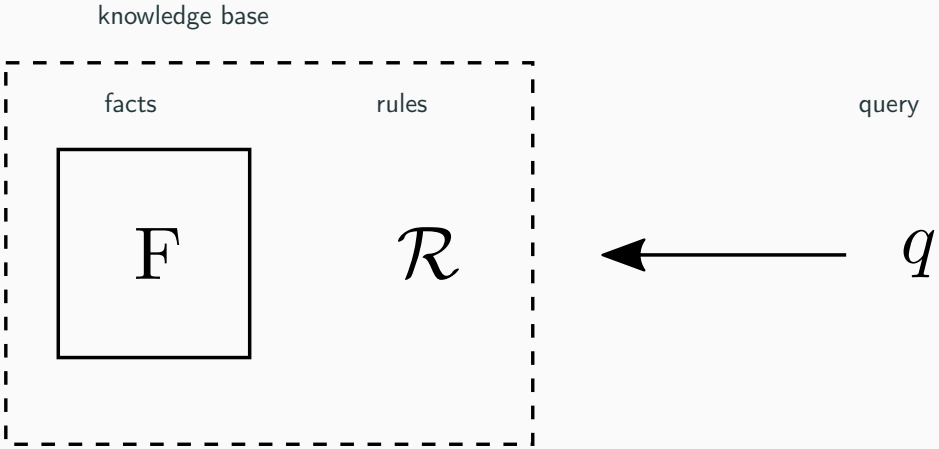
Rewriting the Infinite Chase

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

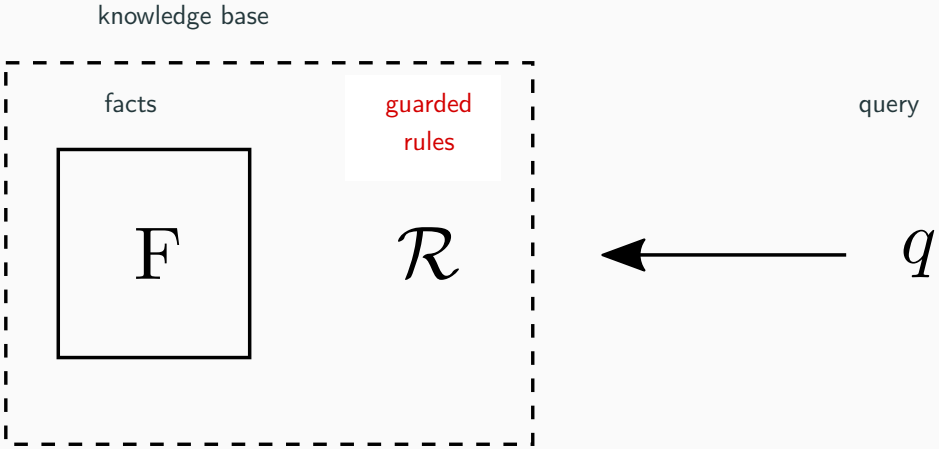
28/04/2022 - GraphIK

Context

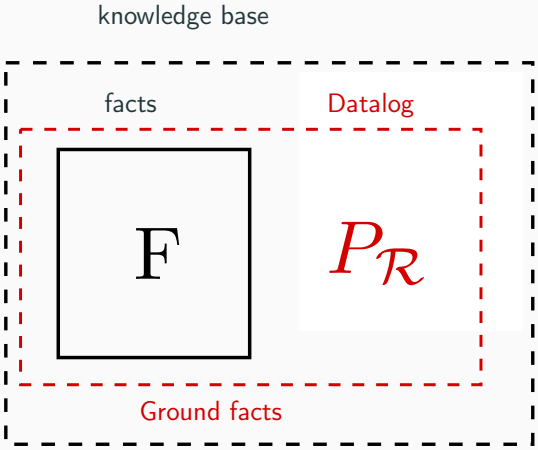
Ontology Mediated Query Answering



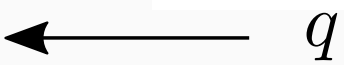
Context: Ontology Mediated Query Answering with Guarded Rules



Problem Statement: Datalog Rewriting



quantifier-free
conjunctive query



Datalog Rewriting of a Rule Set

Definition

An existential rule $r = \forall \bar{x} \forall \bar{y} B(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} H(\bar{x}, \bar{z})$ is **guarded** if there exists an atom $G \in B$ such that $\bar{x} \cup \bar{y} = \mathrm{var}(G)$. We say that G is a **guard** of r.

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,

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

Motivating Example

$$F = \{A(a), R(a, b)\}$$

$$R(u, v) \rightarrow \exists w\ R(v, w) \quad (1)$$
$$A(x) \wedge R(x, y) \rightarrow A(y) \quad (2)$$
$$R(x, y) \wedge A(y) \rightarrow C(x) \quad (3)$$

$$Q(x) \leftarrow C(x)$$

- The chase is infinite because of 1
- The rule set is not FUS because of 2
- We can not derive $C(b)$ without the non-full rule 1

A Datalog rewriting of these rules is the set containing 2, 3 and the rule:

$$R(x, y) \wedge A(y) \rightarrow C(y) \quad (4)$$

Outline

Tree-like Chase

Existential-based Datalog Rewriting

Skolem Datalog Rewriting

Hyper-resolution Datalog Rewriting

Implementation and Experiments

Tree-like Chase

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 $\mathrm{terms}(A) \subseteq \mathrm{terms}(B)$.

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 $\mathcal{R}$ **tree-like chase sequence** is a sequence of trees $T_0, \ldots, T_n$ such that:

- each vertex v in T_i is associated with a set of facts $F_i(v)$,

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 $\mathcal{R}$ **tree-like chase sequence** is a sequence of trees $T_0, \ldots, T_n$ such that:

- each vertex v in T_i is associated with a set of facts $F_i(v)$,
- T_{i+1} is obtained from T_i by applying either:
  - *chase step*, where r in $\mathcal{R}$ is triggered by π on $F_i(v)$,
    - if r is Datalog, then $F_{i+1}(v) = F_i(v) \cup \pi(\mathrm{head}(r))$,

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 $\mathcal{R}$ **tree-like chase sequence** is a sequence of trees $T_0, \ldots, T_n$ such that:

- each vertex v in T_i is associated with a set of facts $F_i(v)$,
- T_{i+1} is obtained from T_i by applying either:
  - *chase step*, where r in $\mathcal{R}$ is triggered by π on $F_i(v)$,
    - if r is Datalog, then $F_{i+1}(v) = F_i(v) \cup \pi(\mathrm{head}(r))$,
    - otherwise, we create v' a child of v in T_{i+1}, with $F_{i+1}(v') = \pi^{safe}(\mathrm{head}(r)) \cup \{a \in F_i(v) \mid a \text{ is guarded in } \pi^{safe}(\mathrm{head}(r))\}$

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 $\mathcal{R}$ **tree-like chase sequence** is a sequence of trees $T_0, \ldots, T_n$ such that:

- each vertex v in T_i is associated with a set of facts $F_i(v)$,
- T_{i+1} is obtained from T_i by applying either:
  - *chase step*, where r in $\mathcal{R}$ is triggered by π on $F_i(v)$,
    - if r is Datalog, then $F_{i+1}(v) = F_i(v) \cup \pi(\mathrm{head}(r))$,
    - otherwise, we create v' a child of v in T_{i+1}, with $F_{i+1}(v') = \pi^{safe}(\mathrm{head}(r)) \cup \{a \in F_i(v) \mid a \text{ is guarded in } \pi^{safe}(\mathrm{head}(r))\}$
  - *propagation step*, where if v and v' two vertices in T_i, and $a \in F_i(v)$ guarded in $F_i(v')$, then we define $F_{i+1}(v') = F_i(v') \cup \{a\}$.

Example of Tree-like chase

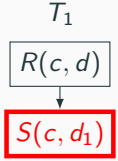
$R(x_1, x_2) \to \exists y\, S(x_1, y)$
$R(x_1, x_2) \to \exists y\, T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\, U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$
$T(x_1, x_2, x_3) \wedge P(x_2) \to M(x_1)$
$S(x_1, x_2) \wedge M(x_1) \to \exists y\, N(x_1, y)$

T_0

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

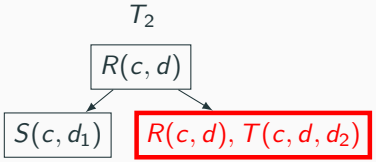
Example of Tree-like chase

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



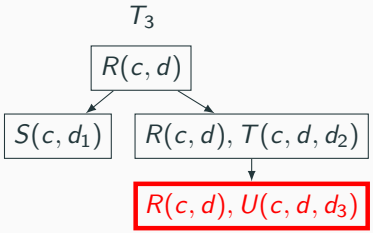
Example of Tree-like chase

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



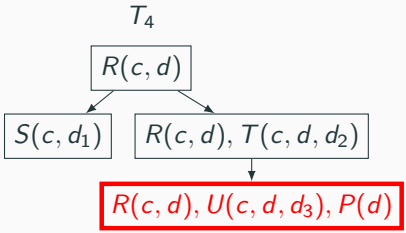
Example of Tree-like chase

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



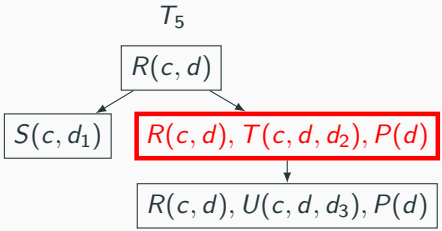
Example of Tree-like chase

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



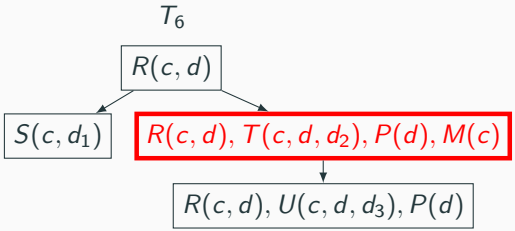
Example of Tree-like chase

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



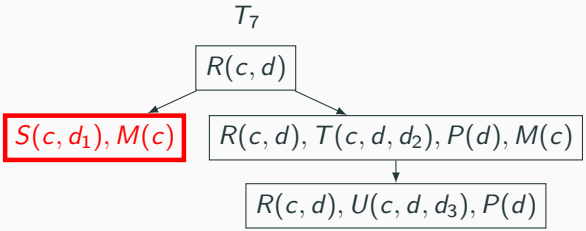
Example of Tree-like chase

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



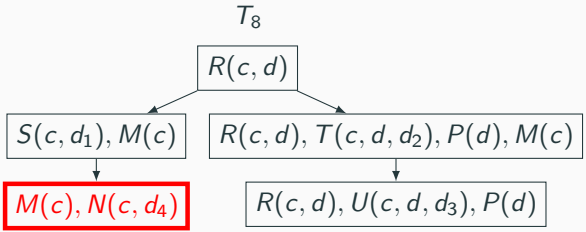
Example of Tree-like chase

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



Example of Tree-like chase

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



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 $\mathcal{R}$ tree-like chase sequence with:

- *T_0 contains only a root, whose the of set facts is F,*
- *$\bigcup_{v \in T_n} F_n(v) \models Q$ i.e. the facts in T_n implies Q.*

The One-Pass Property

Definition

The **recently-updated vertex** in T_i is the one that has being created or updated in moving from T_{i-1} to T_i.

Definition

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

- every chase or propagate is applied at the recently-updated vertex,
- every propagate step propagates a fact from a child to a parent.

The One-Pass Property

Definition

The **recently-updated vertex** in T_i is the one that has being created or updated in moving from T_{i-1} to T_i.

Definition

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

- every chase or propagate is applied at the recently-updated vertex,
- every propagate step propagates a fact from a child to a parent.

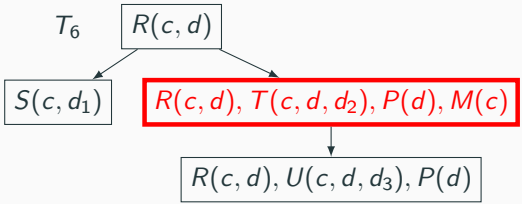
Theorem

*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 $\mathcal{R}$ **one-pass** tree-like chase sequence with:*

- *T_0 contains only a root, whose the of set facts is F,*
- $\bigcup_{v \in T_n} F_n(v) \models Q$.

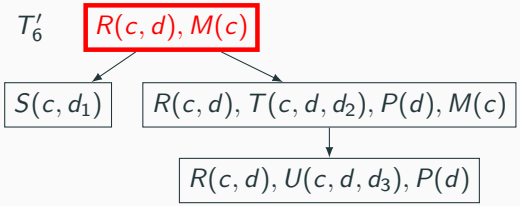
One-Pass Tree-like Chase Example

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



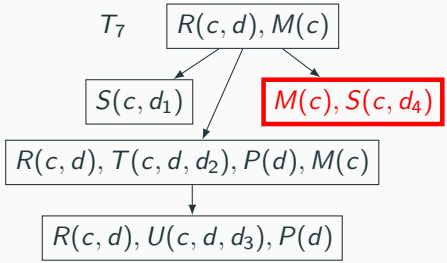
One-Pass Tree-like Chase Example

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



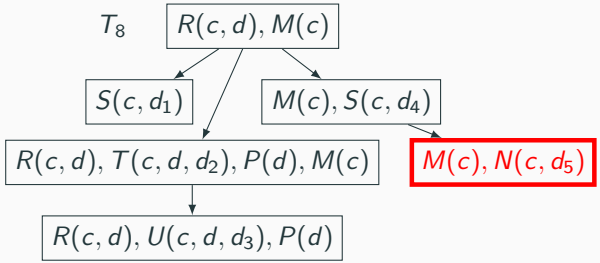
One-Pass Tree-like Chase Example

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



One-Pass Tree-like Chase Example

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



Loops and Datalog Rewriting

Definition

A $\mathcal{R}$ **loop** is $T_0, \ldots, T_n$ a $\mathcal{R}$ one-pass tree-like chase sequence, where:

- T_0 contains a single vertex r,
- the step from T_0 to T_1 creates is a chase step creating c a child of r,
- each chase step from T_1 through T_{n-1} impacts only the subtree of c,
- the final step propagates a fact A to r.

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

Loops and Datalog Rewriting

Definition

A $\mathcal{R}$ **loop** is $T_0, \ldots, T_n$ a $\mathcal{R}$ one-pass tree-like chase sequence, where:

- T_0 contains a single vertex r,
- the step from T_0 to T_1 creates is a chase step creating c a child of r,
- each chase step from T_1 through T_{n-1} impacts only the subtree of c,
- the final step propagates a fact A to r.

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

Theorem

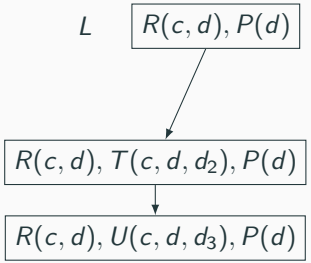
A Datalog rule set P is a Datalog rewriting of $\mathcal{R}$ a set of guarded rules if

- *P is a logical consequence of $\mathcal{R}$,*
- *for each fact base F, and each $\mathcal{R}$ loop with $F_0(r) = F$ and output A, we have $F, P \models A$.*

Loop Example

$R(x_1, x_2) \to \exists y\ T(x_1, x_2, y)$
$T(x_1, x_2, x_3) \to \exists y\ U(x_1, x_2, y)$
$U(x_1, x_2, x_3) \to P(x_2)$

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

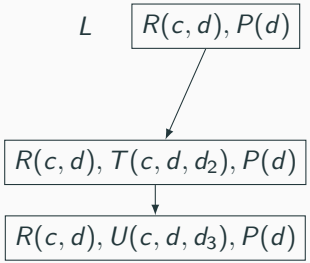


Existential-based Datalog Rewriting

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. $U(x_1, x_2, x_3) \rightarrow P(x_2)$

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



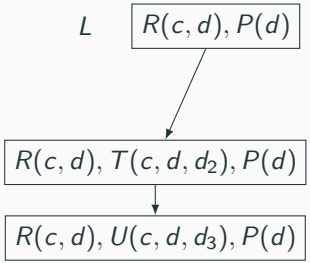
From Loops to Datalog Rewriting

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

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



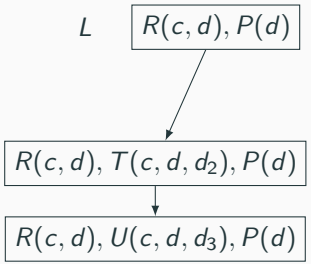
From Loops to Datalog Rewriting

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 $P(d)$:



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

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

The Existential-Based Rewriting Algorithm (ExbDR)

Definition

The Existential-Based Datalog rewriting inference rule takes a **non-full** rule τ and a **Datalog** rule τ' with:

- $\tau = \beta(\bar{x}) \rightarrow \exists \bar{y}\ \eta(\bar{x}, \bar{y})$,
- $\tau' = \beta'(\bar{z}) \rightarrow \eta'(\bar{z})$,

and a **piece unifier** (u, B', H') with a B' subset of β' (necessarily **containing a guard** of τ') and H' with a subset of η

and returns $u(\beta) \cup u(\beta' \setminus B') \rightarrow \exists \bar{y}\ u(\eta \cup \eta')$ 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 rule** takes two Skolemized rules:

1. $\tau = \beta \to H$ that contains only functional atom in its head,
2. $\tau' = \beta' \to H'$ and $B' \in \beta'$ such that either:
   - 2.1 τ' is Datalog and B' is a guard,
   - 2.2 B' is a functional atom

and u an unifier of B' and H,

and returns $u(\beta \cup (\beta' \setminus \{B'\})) \to u(H')$

Definition

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

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

Interest of SkDR compared to ExbDR

Proposition

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

Interest of SkDR compared to ExbDR

Proposition

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

Proof

For each $n \in \mathbb{N}$, let $\mathcal{R}_n$ contain the following rules.

$$A(x) \to \exists \vec{y}\ B(x,y)$$
$$B(x,y) \wedge C_i(x) \to D_i(y) \text{ for } 1 \leq i \leq n$$

ExDR infers, for $I \subseteq \{1..n\}$

$$A(x) \wedge \bigwedge_{i\in I} C_i(x) \to \exists y\ B(x,y) \wedge \bigwedge_{i\in I} D_i(y).$$

SkDR infers $A(x) \wedge C_i(x) \to D_i\big(f(x)\big)$ for $1 \leq i \leq n$.

Hyper-resolution Datalog Rewriting

The Hyper-resolution Datalog Rewriting Algorithm (HyperDR)

Definition

The **Hyper-resolution rewriting rule** takes guarded rules

$$\tau_1 = \beta_1 \to H_1 \quad \dots \quad \tau_n = \beta_n \to H_n \text{ and}$$
$$\tau' = A'_1 \wedge \dots \wedge A'_n \wedge \beta' \to H',$$

such that

- each rule τ_i with $1 \leq i \leq n$ contains only functional atom in its head, and
- rule τ' is Datalog,

and an MGU θ of $H_1, \dots, H_n$ and $A'_1, \dots, A'_n$ such that conjunction $\theta(\beta')$ is Datalog, and it derives

$$\theta(\beta_1) \wedge \dots \wedge \theta(\beta_n) \wedge \theta(\beta') \to \theta(H').$$

Hyper-resolution Datalog Rewriting on an 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:

Hyper-resolution Datalog Rewriting on an 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:

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

Hyper-resolution Datalog Rewriting on an 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:

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

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

Hyper-resolution Datalog Rewriting on an 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:

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

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

Implementation and Experiments

Unification index

- Every algorithm defines its own unification indexes.
- The unification indexes provide a fast method, given r a rule, to find C a set of the candidate rule such that the inference rule could be applied on r and r', for $r' \in C$.
- The indexes is based on the characteristics (predicate, function term and their position) of a guard atom, the head atoms or the body functional atoms following the algorithm.

Subsumption index

We use an subsumption index based on the atom predicates to help to find :

- *forward subsumption*: whether a new inferred rule is subsumed by one of the previously inferred rules,
- *backward subsumption*: whether a previously inferred rule is subsumed by a new one.

The subsumption we consider does not rely on homomorphism finding, but on atoms containment after an uniform variable renaming.

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”.

Competitor

KAON2 for Description Logics only.

There is no competitor for general guarded rules.

Results

Table 1: At the top, the table about ontologies, at the bottom, the table about higher-arity GTGDs. On the left, the number of inputs on which the algorithm on the row wins against the one on the column by one order of magnitude. On the right, the number of ontologies on which two algorithms reach the timeout simultaneously

	Exb	Sk	Hyper	KAON2	Exb	Sk	Hyper	KAON2
Exb	0	19	0	19	29			
Sk	37	0	0	26	1	19		
Hyper	37	12	0	31	3	11	14	
KAON2	35	15	0	0	5	15	14	34

	Exb	Sk	Hyper	Exb	Sk	Hyper
Exb	0	61	87	26		
Sk	11	0	21	0	62	
Hyper	6	4	0	20	56	101

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

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.

Thanks