24TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE BUENOS AIRES

IJCAI 15

Termination Analysis of Logic Programs

Sergio Greco and Cristian Molinaro

DIMES, University of Calabria, Italy

July 25th, 2015

Logic Program Termination Analysis

1. What kind of **Logic Programs**?

   1. Rules with function symbols.
   2. Existential rules.

Many applications in knowledge representation, logic programming, and databases: answer set programming, ontological query answering, data exchange, etc.

Logic Program Termination Analysis

1. What kind of **Logic Programs**?

   1. Rules with function symbols.
   2. Existential rules.

Many applications in knowledge representation, logic programming, and databases: answer set programming, ontological query answering, data exchange, etc.

2. **Termination Analysis**

   - The evaluation of such programs might not terminate.
   - Establishing termination is undecidable.

Logic Program Termination Analysis

1. What kind of **Logic Programs**?

1. Rules with function symbols.
2. Existential rules.

Many applications in knowledge representation, logic programming, and databases: answer set programming, ontological query answering, data exchange, etc.

2. **Termination Analysis**

- The evaluation of such programs might not terminate.
- Establishing termination is undecidable.
- **Termination Criteria:** sufficient conditions guaranteeing termination.

Outline

- Part I: Logic Programs with Function Symbols
  - Syntax and Semantics
  - Termination Criteria
- Part II: Existential Rules
  - The Chase and the Termination Problem
  - Termination Criteria
  - Adding EGDs

Part I

Logic Programs with Function Symbols

Context and Motivations

- **Function Symbols**
  - Make modeling easier and the resulting encodings more readable and concise.
  - Increase the expressive power.
  - Allow us to overcome the inability of handling infinite domains.

Context and Motivations

- **Function Symbols**
  - Make modeling easier and the resulting encodings more readable and concise.
  - Increase the expressive power.
  - Allow us to overcome the inability of handling infinite domains.

- **Problem:** Program evaluation might not terminate and it is undecidable whether the evaluation terminates.

Top-down Evaluation

- Apt, Bezem. Acyclic programs. ICLP (1990).
- Bol, Apt, Klop. An analysis of loop checking mechanism for logic programs. TCS (1991).
- Sagiv. A termination test for logic programs. ICLP (1991).
- Apt, Pedreschi. Reasoning about termination of pure Prolog programs. I&C (1993).
- De Schreye, Decorte. Termination of logic programs: The never-ending story. JLP (1994).
- Lindenstrauss, Sagiv. Automatic termination analysis of logic programs. ICLP (1997).
- Codish, Taboch. A semantic basis for the termination analysis of logic programs. JLP (1999).
- Ohlebusch. Termination of logic programs: Transformational methods revisited. AAECC (2001).
- Pedreschi, Ruggieri, Smaus. Classes of terminating logic programs. TPLP (2002).
- Bonatti. Reasoning with infinite stable models. AIJ (2004).
- Serebrenik, De Schreye. On termination of meta-programs. TPLP (2005).

Top-down Evaluation

- Bruynooghe, Codish, Gallagher, Genaim, Vanhoof. Termination analysis of logic programs through combination of type-based norms. ACM TOCL (2007).
- Nguyen, Giesl, Schneider-Kamp, De Schreye. Termination analysis of logic programs based on dependency graphs. LOPSTR (2007).
- Baselice, Bonatti, Criscuolo. On finitely recursive programs. TPLP (2009).
- Schneider-Kamp, Giesl, Nguyen. The dependency triple framework for termination of logic programs. LOPSTR (2009).
- Schneider-Kamp, Giesl, Serebrenik, Thiemann. Automated termination proofs for logic programs by term rewriting. ACM TOCL (2009).
- Nishida, Vidal. Termination of narrowing via termination of rewriting. Appl. Algebra Eng. Commun. Comput. (2010).
- Schneider-Kamp, Giesl, Stroder, Serebrenik, Thiemann. Automated termination analysis for logic programs with cut. TPLP (2010).
- Eiter, Simkus. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. ACM TOCL (2010).
- Voets, De Schreye. Non-termination analysis of logic programs with integer arithmetics. TPLP (2011).

Bottom-up Evaluation

- Shmueli. Decidability and Expressiveness of Logic Queries. PODS (1987).
- Ramakrishnan, Bancilhon, Silberschatz. Safety of Recursive Horn Clauses With Infinite Relations. PODS (1987).
- Kifer, Ramakrishnan, Silberschatz. An Axiomatic Approach to Deciding Query Safety in Deductive Databases. PODS (1988).
- Krishnamurthy, Ramakrishnan, Shmueli. A framework for testing safety and effective computability. SIGMOD (1988), JCSS (1996).
- Chomicki. A decidable class of logic programs with function symbols. TR 1990.
- Chomicki, Imielinski. Finite representation of infinite query answers. TODS (1993).

Bottom-up Evaluation

- Syrjanen. Omega-restricted logic programs. LPNMR (2001).
- Gebser, Schaub, Thiele. Gringo : A new grounder for answer set programming. LPNMR (2007).
- Calimeri, Cozza, Ianni, Leone. Computable Functions in ASP: Theory and Implementation. ICLP (2008).
- Lierler, Lifschitz. One more decidable class of finitely ground programs. ICLP (2009).
- Greco, Spezzano, Trubitsyna. On the Termination of Logic Programs with Function Symbols. ICLP (2012)
- Calautti, Greco, Trubitsyna. Detecting decidable classes of finitely ground logic programs with function symbols. PPDP (2013).
- Greco, Molinaro, Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. TPLP (2013).
- Greco, Molinaro, Trubitsyna. Bounded Programs: A New Decidable Class of Logic Programs with Function Symbols. IJCAI (2013).
- Calautti, Greco, Molinaro, Trubitsyna. Checking Termination of Logic Programs with Function Symbols through Linear Constraints. RuleML (2014).
- Calautti, Greco, Spezzano, Trubitsyna. Checking Termination of Bottom-Up Evaluation of Logic Programs with Function Symbols. TPLP (2014).

Top-down vs. Bottom-up Evaluation

Example

$$\mathtt{p(X)} \leftarrow \mathtt{p(X)}.$$

- Non-terminating top-down evaluation.
- Completely harmless under bottom-up evaluation.

Top-down vs. Bottom-up Evaluation

Example

$$\mathtt{p(X)} \leftarrow \mathtt{p(X)}.$$

- Non-terminating top-down evaluation.
- Completely harmless under bottom-up evaluation.

We consider **bottom-up** evaluation.

Bottom-up Evaluation

Example

$$\texttt{len}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0}).$$
$$\texttt{len}(\texttt{Tail},\textbf{s}(\texttt{N})) \leftarrow \texttt{len}(\textbf{list}(\texttt{Head},\texttt{Tail}),\texttt{N}).$$

Bottom-up Evaluation

Example

$$\texttt{len}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0}).$$
$$\texttt{len}(\texttt{Tail},\mathbf{s}(\texttt{N})) \leftarrow \texttt{len}(\mathbf{list}(\texttt{Head},\texttt{Tail}),\texttt{N}).$$

Bottom-up evaluation:

$\texttt{len}([\texttt{b},\texttt{c}],\texttt{s}(\texttt{0})) \leftarrow \texttt{len}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0})$ yields $\texttt{len}([\texttt{b},\texttt{c}],\texttt{s}(\texttt{0}))$

Bottom-up Evaluation

Example

$$\texttt{len}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0}).$$
$$\texttt{len}(\texttt{Tail},\textbf{s}(\texttt{N})) \leftarrow \texttt{len}(\textbf{list}(\texttt{Head},\texttt{Tail}),\texttt{N}).$$

Bottom-up evaluation:

$\texttt{len}([\texttt{b},\texttt{c}],\texttt{s}(\texttt{0})) \leftarrow \texttt{len}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0})$ yields $\texttt{len}([\texttt{b},\texttt{c}],\texttt{s}(\texttt{0}))$

$\texttt{len}([\texttt{c}],\texttt{s}(\texttt{s}(\texttt{0}))) \leftarrow \texttt{len}([\texttt{b},\texttt{c}],\texttt{s}(\texttt{0}))$ yields $\texttt{len}([\texttt{c}],\texttt{s}(\texttt{s}(\texttt{0})))$

Bottom-up Evaluation

Example

$$\mathtt{len}([\mathtt{a},\mathtt{b},\mathtt{c}],\mathtt{0}).$$
$$\mathtt{len}(\mathtt{Tail},\mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head},\mathtt{Tail}),\mathtt{N}).$$

Bottom-up evaluation:

$\mathtt{len}([\mathtt{b},\mathtt{c}],\mathtt{s}(\mathtt{0}))$	$\leftarrow \mathtt{len}([\mathtt{a},\mathtt{b},\mathtt{c}],\mathtt{0})$	yields $\mathtt{len}([\mathtt{b},\mathtt{c}],\mathtt{s}(\mathtt{0}))$
$\mathtt{len}([\mathtt{c}],\mathtt{s}(\mathtt{s}(\mathtt{0})))$	$\leftarrow \mathtt{len}([\mathtt{b},\mathtt{c}],\mathtt{s}(\mathtt{0}))$	yields $\mathtt{len}([\mathtt{c}],\mathtt{s}(\mathtt{s}(\mathtt{0})))$
$\mathtt{len}([],\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))$	$\leftarrow \mathtt{len}([\mathtt{c}],\mathtt{s}(\mathtt{s}(\mathtt{0})))$	yields $\mathtt{len}([],\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))$

Bottom-up Evaluation

Example

$$\mathtt{len}([\mathtt{a},\mathtt{b},\mathtt{c}],\mathtt{0}).$$
$$\mathtt{len}(\mathtt{Tail},\mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head},\mathtt{Tail}),\mathtt{N}).$$

Bottom-up evaluation:

$\mathtt{len}([\mathtt{b},\mathtt{c}],\mathtt{s}(\mathtt{0}))$	$\leftarrow \mathtt{len}([\mathtt{a},\mathtt{b},\mathtt{c}],\mathtt{0})$	yields $\mathtt{len}([\mathtt{b},\mathtt{c}],\mathtt{s}(\mathtt{0}))$
$\mathtt{len}([\mathtt{c}],\mathtt{s}(\mathtt{s}(\mathtt{0})))$	$\leftarrow \mathtt{len}([\mathtt{b},\mathtt{c}],\mathtt{s}(\mathtt{0}))$	yields $\mathtt{len}([\mathtt{c}],\mathtt{s}(\mathtt{s}(\mathtt{0})))$
$\mathtt{len}([],\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))$	$\leftarrow \mathtt{len}([\mathtt{c}],\mathtt{s}(\mathtt{s}(\mathtt{0})))$	yields $\mathtt{len}([],\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))$

Fixpoint, the evaluation TERMINATES.

Bottom-up Evaluation

Example

$$\mathtt{nat}(0).$$
$$\mathtt{nat}(\mathbf{s}(\mathtt{X})) \leftarrow \mathtt{nat}(\mathtt{X}).$$

Bottom-up evaluation:

$\mathtt{nat}(\mathtt{s}(0)) \leftarrow \mathtt{nat}(0)$ yields $\mathtt{nat}(\mathtt{s}(0))$

Bottom-up Evaluation

Example

$$\mathtt{nat}(0).$$
$$\mathtt{nat}(\mathbf{s}(\mathtt{X})) \leftarrow \mathtt{nat}(\mathtt{X}).$$

Bottom-up evaluation:

$$\begin{array}{lcllc}
\mathtt{nat}(\mathtt{s}(0)) & \leftarrow & \mathtt{nat}(0) & \text{yields} & \mathtt{nat}(\mathtt{s}(0)) \\
\mathtt{nat}(\mathtt{s}(\mathtt{s}(0))) & \leftarrow & \mathtt{nat}(\mathtt{s}(0)) & \text{yields} & \mathtt{nat}(\mathtt{s}(\mathtt{s}(0)))
\end{array}$$

Bottom-up Evaluation

Example

$$\mathtt{nat}(0).$$
$$\mathtt{nat}(\mathbf{s}(X)) \leftarrow \mathtt{nat}(X).$$

Bottom-up evaluation:

$$\begin{array}{lcllll}
\mathtt{nat}(s(0)) & \leftarrow & \mathtt{nat}(0) & \text{yields} & \mathtt{nat}(s(0)) \\
\mathtt{nat}(s(s(0))) & \leftarrow & \mathtt{nat}(s(0)) & \text{yields} & \mathtt{nat}(s(s(0))) \\
\mathtt{nat}(s(s(s(0)))) & \leftarrow & \mathtt{nat}(s(s(0))) & \text{yields} & \mathtt{nat}(s(s(s(0)))) \\
 & \vdots & & &
\end{array}$$

The evaluation does NOT terminate.

Termination Criteria

- (Decidable) Sufficient conditions guaranteeing the bottom-up evaluation termination.
- The use of function symbols is restricted.

"Terminating" Programs

We say that a program P is ***terminating*** iff **the evaluation of $P \cup D$ terminates for every finite set of facts D**.

Termination Criteria

- (Decidable) Sufficient conditions guaranteeing the bottom-up evaluation termination.
- The use of function symbols is restricted.

"Terminating" Programs

We say that a program P is ***terminating*** iff **the evaluation of $P \cup D$ terminates for every finite set of facts D**.

Termination Criteria

Define a decidable condition $\boldsymbol{C}$ such that for every program $\boldsymbol{P}$

$$\boldsymbol{P} \text{ satisfies } \boldsymbol{C} \quad \Rightarrow \quad \boldsymbol{P} \text{ is terminating.}$$

Termination Criteria

- ω-*restricted* programs [Syr01]
- λ-*restricted* programs [GST07]
- *Finite domain* programs [CCIL08]
- *Argument-restricted* programs [LL09]
- *Safe* and Γ-*acyclic* programs [CGST14]
- *Mapping-restricted* programs [CGT13]
- *Bounded programs* [GMT13b]
- *Rule-* and *cycle-bounded* programs [CGMT14]
- *Program Adornment* technique [GMT13a]

Termination Criteria

- ω-*restricted* programs [Syr01]
- λ-*restricted* programs [GST07]
- *Finite domain* programs [CCIL08]
- *Argument-restricted* programs [LL09]
- *Safe* and Γ-*acyclic* programs [CGST14]
- *Mapping-restricted* programs [CGT13]
- *Bounded programs* [GMT13b]
- *Rule-* and *cycle-bounded* programs [CGMT14]
- *Program Adornment* technique [GMT13a]
- *Size-restricted* programs, IJCAI 2015, talk on Wed 29th afternoon!

Syntax: Datalog with Function Symbols

Definition

We are given (pairwise disjoint) sets of ***constants***, ***variables***, ***function symbols*** (with arity > 0), and ***predicates*** (with arity ≥ 0).

Syntax: Datalog with Function Symbols

Definition

We are given (pairwise disjoint) sets of ***constants***, ***variables***, ***function symbols*** (with arity > 0), and ***predicates*** (with arity ≥ 0).

- A ***term*** is either a constant, a variable, or of the form $f(t_1, \ldots, t_m)$, where f is a function symbol of arity m and the t_i's are terms.

Syntax: Datalog with Function Symbols

Definition

We are given (pairwise disjoint) sets of ***constants***, ***variables***, ***function symbols*** (with arity > 0), and ***predicates*** (with arity ≥ 0).

- A ***term*** is either a constant, a variable, or of the form $f(t_1, \ldots, t_m)$, where f is a function symbol of arity m and the t_i's are terms.
- An ***atom*** is of the form $p(t_1, \ldots, t_n)$, where p is a predicate of arity n and the t_i's are terms.

Syntax: Datalog with Function Symbols

Definition

We are given (pairwise disjoint) sets of ***constants***, ***variables***, ***function symbols*** (with arity > 0), and ***predicates*** (with arity ≥ 0).

- A ***term*** is either a constant, a variable, or of the form $f(t_1, \ldots, t_m)$, where f is a function symbol of arity m and the t_i's are terms.
- An ***atom*** is of the form $p(t_1, \ldots, t_n)$, where p is a predicate of arity n and the t_i's are terms.
- A *(Datalog)* ***rule*** is of the form

$$\underbrace{A_0}_{\textbf{\textit{head}}} \leftarrow \underbrace{A_1, \ldots, A_n}_{\textbf{\textit{body}}}$$

where $n \geq 0$ and the A_i's are atoms.

Syntax: Datalog with Function Symbols

Definition

We are given (pairwise disjoint) sets of ***constants***, ***variables***, ***function symbols*** (with arity > 0), and ***predicates*** (with arity ≥ 0).

- A ***term*** is either a constant, a variable, or of the form $f(t_1, \ldots, t_m)$, where f is a function symbol of arity m and the t_i's are terms.
- An ***atom*** is of the form $p(t_1, \ldots, t_n)$, where p is a predicate of arity n and the t_i's are terms.
- A *(Datalog)* ***rule*** is of the form

$$\underbrace{A_0}_{\textbf{\textit{head}}} \leftarrow \underbrace{A_1, \ldots, A_n}_{\textbf{\textit{body}}}$$

  where $n \geq 0$ and the A_i's are atoms.
- A *(Datalog)* ***program*** is a finite set of Datalog rules.

Syntax: Datalog with Function Symbols

We consider ***safe*** programs: every variable in the head must appear in the body.

Syntax: Datalog with Function Symbols

We consider ***safe*** programs: every variable in the head must appear in the body.

Example (Safe program)

$$\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{Y}).$$

Example (Unsafe program)

$$\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathbf{Y}) \leftarrow \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{Z}).$$

Syntax: Datalog with Function Symbols

We consider ***safe*** programs: every variable in the head must appear in the body.

Example (Safe program)

$$\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{Y}).$$

Example (Unsafe program)

$$\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathbf{Y}) \leftarrow \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{Z}).$$

No disjunction and negation (for now).

Syntax: Datalog with Function Symbols

We consider ***safe*** programs: every variable in the head must appear in the body.

Example (Safe program)

$$\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{Y}).$$

Example (Unsafe program)

$$\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathbf{Y}) \leftarrow \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{Z}).$$

No disjunction and negation (for now).

Function symbols are **uninterpreted** (they are not evaluated).

Syntax: Datalog with Function Symbols

Definition

The ***arguments*** of a program P are expressions of the form $p[i]$ where p is a predicate appearing in P and $1 \leq i \leq arity(p)$.

Syntax: Datalog with Function Symbols

Definition

The ***arguments*** of a program P are expressions of the form $p[i]$ where p is a predicate appearing in P and $1 \leq i \leq arity(p)$.

Example

$$\begin{array}{lcl} \mathtt{p(X,Y)} & \leftarrow & \mathtt{b(X,Y)}. \\ \mathtt{q(f(X))} & \leftarrow & \mathtt{p(X,Y)}. \end{array}$$

The **arguments** of this program are **b[1]**, **b[2]**, **p[1]**, **p[2]**, and **q[1]**.

Termination Criteria

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate *p* so that all head variables in rules defining *p* are bound by predicates *p*′ with strictly lower level.

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X).} \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X).} \end{array}$$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X).} \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X).} \end{array}$$

$\boldsymbol{\lambda(\mathtt{r}) = 1}$;

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathbf{q(X)} & \leftarrow & \mathtt{p(X), r(X)}. \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)}. \end{array}$$

$\lambda(\mathtt{r}) = 1;$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$
\begin{array}{rcl}
\mathbf{q(X)} & \leftarrow & \mathtt{p(X)}, \mathbf{r(X)}. \\
\mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)}.
\end{array}
$$

$\lambda(\mathtt{r}) = 1;$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathbf{q(X)} & \leftarrow & \mathtt{p(X)}, \mathbf{r(X)}. \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)}. \end{array}$$

$\lambda(\mathtt{r}) = 1; \quad \boldsymbol{\lambda(\mathbf{q}) = 2};$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathrm{q}(\mathrm{X}) & \leftarrow & \mathrm{p}(\mathrm{X}), \mathrm{r}(\mathrm{X}). \\ \mathbf{p(f(X))} & \leftarrow & \mathrm{q}(\mathrm{X}). \end{array}$$

$\lambda(\mathrm{r}) = 1; \quad \lambda(\mathrm{q}) = 2;$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X).} \\ \mathbf{p(f(X))} & \leftarrow & \mathbf{q(X)}. \end{array}$$

$\lambda(\mathtt{r}) = 1; \quad \lambda(\mathtt{q}) = 2;$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$
\begin{array}{rcl}
\mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X).} \\
\mathbf{p(f(X))} & \leftarrow & \mathbf{q(X)}.
\end{array}
$$

$$\lambda(\mathtt{r}) = 1; \quad \lambda(\mathtt{q}) = 2; \quad \boldsymbol{\lambda(\mathbf{p}) = 3.}$$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X).} \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X).} \end{array}$$

$$\lambda(\mathtt{r}) = \mathtt{1}; \quad \lambda(\mathtt{q}) = \mathtt{2}; \quad \lambda(\mathtt{p}) = \mathtt{3}.$$

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X)}. \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)}. \end{array}$$

$$\lambda(\mathtt{r}) = \mathtt{1}; \quad \lambda(\mathtt{q}) = \mathtt{2}; \quad \lambda(\mathtt{p}) = \mathtt{3}.$$

The program is λ-restricted.

λ-Restricted Programs [GST07]

Basic Idea: Assign a level (i.e., an integer) $\lambda(p)$ to each predicate p so that all head variables in rules defining p are bound by predicates p' with strictly lower level.

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X).} \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X).} \end{array}$$

$$\lambda(\mathtt{r}) = \mathtt{1}; \quad \lambda(\mathtt{q}) = \mathtt{2}; \quad \lambda(\mathtt{p}) = \mathtt{3}.$$

The program is λ-restricted.

Example

$$\mathtt{p(X)} \quad \leftarrow \quad \mathtt{p(X).}$$

No function symbols $\Rightarrow$ **The evaluation always terminates.**

$\lambda(\mathtt{p}) > \lambda(\mathtt{p}) \Rightarrow$ **The program is not λ-restricted.**

Finite Domain Programs [CCIL08]

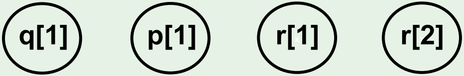
Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example (Argument Graph)

$$
\begin{array}{rcl}
\mathtt{q(X)} & \leftarrow & \mathtt{q(f(X)).} \\
\mathtt{p(f(X))} & \leftarrow & \mathtt{q(X), r(X, Y).}
\end{array}
$$



Finite Domain Programs [CCIL08]

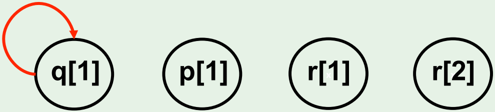
Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from *p*[*i*] to *q*[*j*] if there is a rule where a term is propagated from *p*[*i*] to *q*[*j*].

Example (Argument Graph)

$$\begin{array}{rcl} \mathtt{q}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X})). \\ \mathtt{p}(\mathtt{f}(\mathtt{X})) & \leftarrow & \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{X}, \mathtt{Y}). \end{array}$$



Finite Domain Programs [CCIL08]

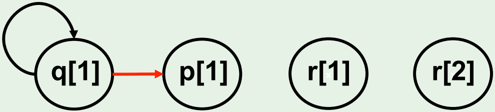
Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example (Argument Graph)

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{q(f(X))}. \\ \mathbf{p}(\mathtt{f}(\mathbf{X})) & \leftarrow & \mathbf{q}(\mathbf{X}), \mathtt{r(X, Y)}. \end{array}$$



Finite Domain Programs [CCIL08]

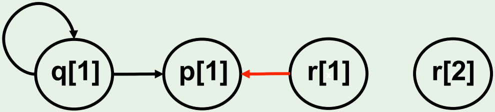
Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example (Argument Graph)

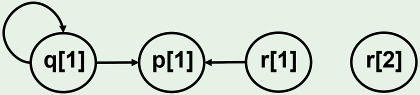
$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{q(f(X)).} \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X), r(X, Y).} \end{array}$$



Finite Domain Programs [CCIL08]

Example

$$\begin{array}{rcl} \mathtt{q}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X})). \\ \mathtt{p}(\mathtt{f}(\mathtt{X})) & \leftarrow & \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{X}, \mathtt{Y}). \end{array}$$



Finite domain arguments:

Finite Domain Programs [CCIL08]

Example

$$\begin{array}{rcl} \mathtt{q}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X})). \\ \mathtt{p}(\mathtt{f}(\mathtt{X})) & \leftarrow & \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{X}, \mathtt{Y}). \end{array}$$

$$\begin{tikzcd} \mathbf{q[1]} \arrow[loop left] \arrow[r] & \mathbf{p[1]} & \mathbf{r[1]} \arrow[l] & \mathbf{r[2]} \end{tikzcd}$$

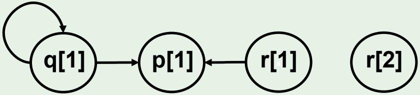
Finite domain arguments:

- $\mathtt{r}[1]$ and $\mathtt{r}[2]$, as they appear in no head.

Finite Domain Programs [CCIL08]

Example

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{q(f(X))}. \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X), r(X, Y)}. \end{array}$$



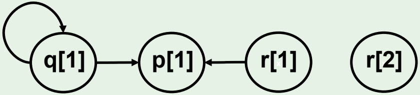
Finite domain arguments:

- $\mathtt{r[1]}$ and $\mathtt{r[2]}$, as they appear in no head.
- $\mathtt{q[1]}$, as the term in the head of the 1st rule is a subterm of that in the body.

Finite Domain Programs [CCIL08]

Example

$$
\begin{array}{rcl}
\mathtt{q}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X})). \\
\mathtt{p}(\mathtt{f}(\mathtt{X})) & \leftarrow & \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{X}, \mathtt{Y}).
\end{array}
$$



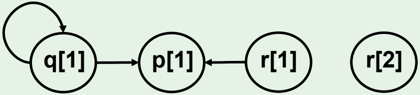
Finite domain arguments:

- $\mathtt{r}[1]$ and $\mathtt{r}[2]$, as they appear in no head.
- $\mathtt{q}[1]$, as the term in the head of the 1st rule is a subterm of that in the body.
- $\mathtt{p}[1]$, as $\mathtt{r}[1]$ is finite domain and is not “recursive” with $\mathtt{p}[1]$

Finite Domain Programs [CCIL08]

Example

$$\begin{array}{rcl} \mathtt{q}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X})). \\ \mathtt{p}(\mathtt{f}(\mathtt{X})) & \leftarrow & \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{X}, \mathtt{Y}). \end{array}$$

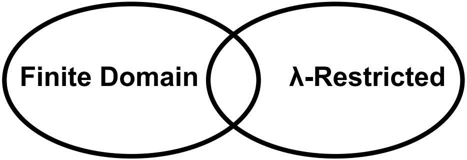


Finite domain arguments:

- $\mathtt{r}[1]$ and $\mathtt{r}[2]$, as they appear in no head.
- $\mathtt{q}[1]$, as the term in the head of the 1st rule is a subterm of that in the body.
- $\mathtt{p}[1]$, as $\mathtt{r}[1]$ is finite domain and is not "recursive" with $\mathtt{p}[1]$

All arguments are finite domain $\Rightarrow$ The program is finite domain.

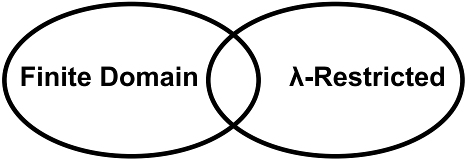
Relative Expressivity



Theorem

λ-Restricted $\nparallel$ Finite Domain.

Relative Expressivity



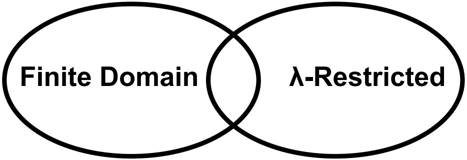
Theorem

λ-Restricted $\nparallel$ Finite Domain.

Example (Finite Domain but not λ-Restricted)

$$
\begin{array}{rcl}
\mathtt{q}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X})). \\
\mathtt{p}(\mathtt{f}(\mathtt{X})) & \leftarrow & \mathtt{q}(\mathtt{X}), \mathtt{r}(\mathtt{X}, \mathtt{Y}).
\end{array}
$$

Relative Expressivity



Theorem

λ-Restricted $\nparallel$ Finite Domain.

Example (Finite Domain but not λ-Restricted)

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{q(f(X))}. \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X), r(X, Y)}. \end{array}$$

Example (λ-Restricted but not Finite Domain)

$$\begin{array}{rcl} \mathtt{q(X)} & \leftarrow & \mathtt{p(X), r(X)}. \\ \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)}. \end{array}$$

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Term Depth

Depth $d(X, t)$ of a variable X in a term t containing X:

$$d(X, X) = 0$$
$$d(X, f(t_1, \ldots, t_m)) = 1 + \max_{1 \le i \le m \,:\, t_i \text{ contains } X} d(X, t_i).$$

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Term Depth

Depth $d(X, t)$ of a variable X in a term t containing X:

$$
\begin{aligned}
d(X,\, X) &= 0 \\
d(X,\, f(t_1, \ldots, t_m)) &= 1 + \max_{1 \le i \le m \,:\, t_i \textit{ contains } X} d(X, t_i).
\end{aligned}
$$

Example

$$d(\mathtt{X},\ \mathtt{f}(\mathtt{X}, \mathtt{g}(\mathtt{X}), \mathtt{Y}))$$

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Term Depth

Depth $d(X, t)$ of a variable X in a term t containing X:

$$
\begin{aligned}
d(X,\, X) &= 0 \\
d(X,\, f(t_1, \ldots, t_m)) &= 1 + \max_{1 \le i \le m \,:\, t_i \text{ contains } X} d(X, t_i).
\end{aligned}
$$

Example

$$d(\mathtt{X},\ \mathtt{f}(\mathtt{X}, \mathtt{g}(\mathtt{X}), \mathtt{Y})) = \mathbf{2}$$

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Term Depth

Depth $d(X,t)$ of a variable X in a term t containing X:

$$d(X,\, X) = 0$$
$$d(X,\, f(t_1,\ldots,t_m)) = 1 + \max_{1 \le i \le m \,:\, t_i \text{ contains } X} d(X, t_i).$$

Example

$$d(\mathtt{X},\ \mathbf{f}(\mathtt{X}, \mathbf{g}(\mathbf{X}), \mathtt{Y})) = \mathbf{2}$$

$$d(\mathtt{Y},\ \mathbf{f}(\mathtt{X}, \mathtt{g}(\mathtt{X}), \mathbf{Y})) = \mathbf{1}$$

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Example

$$\begin{array}{rcl} \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)} \\ \mathtt{q(X)} & \leftarrow & \mathtt{p(f(X))} \end{array}$$

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Example

$$\mathtt{p(f(X))} \leftarrow \mathtt{q(X)}$$

$$\mathtt{q(X)} \leftarrow \mathtt{p(f(X))}$$

We need to find a function ϕ
(assigning an integer to each argument) such that:

$\phi(\mathtt{p[1]}) \geq \phi(\mathtt{q[1]}) + 1 - 0$, and

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Example

$$\mathtt{p(f(X))} \leftarrow \mathtt{q(X)}$$

$$\mathtt{q(X)} \leftarrow \mathtt{p(f(X))}$$

We need to find a function ϕ
(assigning an integer to each argument) such that:

$\phi(\mathtt{p[1]}) \geq \phi(\mathtt{q[1]}) + \mathbf{1} - \mathbf{0}$, and

$\phi(\mathtt{q[1]}) \geq \phi(\mathtt{p[1]}) + \mathbf{0} - \mathbf{1}$.

Argument Restriction [LL09]

Basic idea: assign to each argument an upper bound of the depth of terms that may occur in that argument.

Example

$$\mathtt{p}(\overset{1}{\mathtt{f}(\mathtt{X})}) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}})$$

$$\mathtt{q}(\overset{0}{\mathtt{X}}) \leftarrow \mathtt{p}(\overset{1}{\mathtt{f}(\mathtt{X})})$$

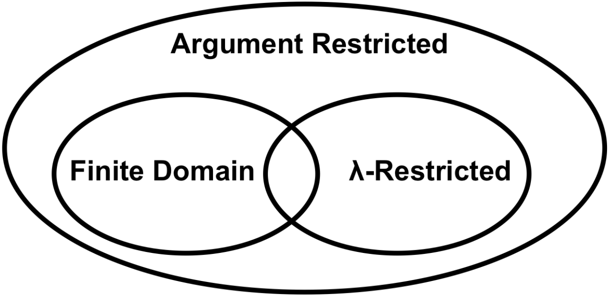
We need to find a function ϕ
(assigning an integer to each argument) such that:

$$\phi(\mathtt{p}[1]) \geq \phi(\mathtt{q}[1]) + 1 - 0, \text{ and}$$

$$\phi(\mathtt{q}[1]) \geq \phi(\mathtt{p}[1]) + 0 - 1.$$

The program is argument-restricted

Relative Expressivity



Theorem

- *Finite Domain* $\subsetneq$ *Argument Restricted.*
- λ*-Restricted* $\subsetneq$ *Argument Restricted.*

Argument Restriction [LL09]

Simple and easy (polynomial time) to compute.

Argument Restriction [LL09]

Simple and easy (polynomial time) to compute.

Limitation: No distinction between different function symbols.

Example

$$\mathtt{p}(\mathbf{f}(\mathtt{f}(\mathtt{X}))) \leftarrow \mathtt{p}(\mathbf{g}(\mathtt{X}))$$

We need to find a function ϕ such that

$$\phi(\mathtt{p}[1]) \geq \phi(\mathtt{p}[1]) + 1$$

No such ϕ exists. The program is not argument-restricted...
... but the program evaluation always terminates.

Argument restriction can be used as a starting point for more complex analysis.

Bounded Programs [GMT13b]

Bounded Programs [GMT13b]

Basic Idea:

- Start with a set A of "limited" arguments.
- Iteratively apply a (monotone) operator $\Psi(A)$ which derives more arguments as "limited".
- If, eventually, all arguments are derived as limited, then the program is *bounded*.

The operator relies on two tools:

- the **activation graph**, and
- the **labeled argument graph**.

Termination Analysis Tools — Activation Graph

Activation Graph

It describes "activation" of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

Termination Analysis Tools — Activation Graph

Activation Graph

It describes "activation" of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

Example

$$r_1: \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \quad \mathtt{p(g(X))} \leftarrow \mathtt{q(X)}$$

Termination Analysis Tools — Activation Graph

Activation Graph

It describes “activation” of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

Example

$$r_1: \ \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \ \mathtt{p(g(X))} \leftarrow \mathtt{q(X)}$$

Termination Analysis Tools — Activation Graph

Activation Graph

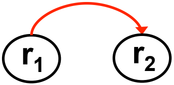
It describes "activation" of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

Example

r_1 : $\mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$

r_2 : $\mathtt{p(g(X))} \leftarrow \mathtt{q(X)}$



Termination Analysis Tools — Activation Graph

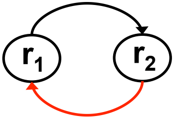
Activation Graph

It describes "activation" of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

Example

r_1 : $\mathtt{q(f(X))} \leftarrow \mathbf{p(X)}$
r_2 : $\mathbf{p(g(X))} \leftarrow \mathtt{q(X)}$



Termination Analysis Tools — Activation Graph

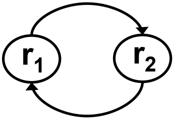
Activation Graph

It describes "activation" of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

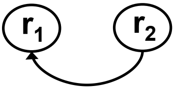
Example

$$r_1: \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \quad \mathtt{p(g(X))} \leftarrow \mathtt{q(X)}$$



Example

$$r_1: \quad \mathtt{q(\mathbf{f}(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \quad \mathtt{p(g(X))} \leftarrow \mathtt{q(\mathbf{g}(X))}$$



Termination Analysis Tools — Activation Graph

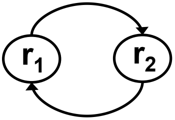
Activation Graph

It describes "activation" of rules.

- the nodes are the rules of the program, and
- there is an edge from r_i to r_j iff the head of r_i unifies with some body atom of r_j.

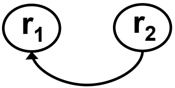
Example

$$r_1 : \quad \mathrm{q(f(X))} \leftarrow \mathrm{p(X)}$$
$$r_2 : \quad \mathrm{p(g(X))} \leftarrow \mathrm{q(X)}$$



Example

$$r_1 : \quad \mathrm{q(\mathbf{f}(X))} \leftarrow \mathrm{p(X)}$$
$$r_2 : \quad \mathrm{p(g(X))} \leftarrow \mathrm{q(\mathbf{g}(X))}$$



A rule might be applied an infinite number of times only if it depends on a cycle.

Termination Analysis Tools — Labeled Argument Graph

Labeled Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Termination Analysis Tools — Labeled Argument Graph

Labeled Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example

$$
\begin{array}{rrcl}
r_1: & \mathtt{q(f(X),X)} & \leftarrow & \mathtt{b(Y),\ p(X).} \\
r_2: & \mathtt{p(X)} & \leftarrow & \mathtt{q(f(X),Y).}
\end{array}
$$

Termination Analysis Tools — Labeled Argument Graph

Labeled Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example

$$
\begin{array}{lrcl}
r_1 : & \mathtt{q}(\mathtt{f}(\mathtt{X}),\mathtt{X}) & \leftarrow & \mathtt{b}(\mathtt{Y}),\ \mathtt{p}(\mathtt{X}). \\
r_2 : & \mathtt{p}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X}),\mathtt{Y}).
\end{array}
$$

Termination Analysis Tools — Labeled Argument Graph

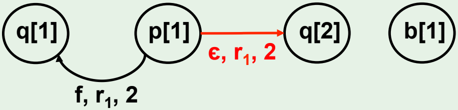
Labeled Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example

$$\begin{array}{rrcl} r_1: & \mathtt{q}(\mathtt{f}(\mathtt{X}), \mathtt{X}) & \leftarrow & \mathtt{b}(\mathtt{Y}), \mathtt{p}(\mathtt{X}). \\ r_2: & \mathtt{p}(\mathtt{X}) & \leftarrow & \mathtt{q}(\mathtt{f}(\mathtt{X}), \mathtt{Y}). \end{array}$$



Termination Analysis Tools — Labeled Argument Graph

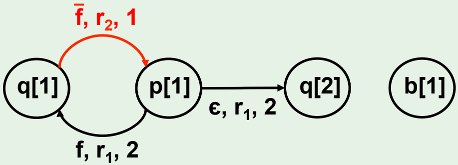
Labeled Argument Graph

It describes the propagation of values among arguments.

- the nodes are the arguments of the program, and
- there is an edge from $p[i]$ to $q[j]$ if there is a rule where a term is propagated from $p[i]$ to $q[j]$.

Example

$$
\begin{array}{llcl}
r_1: & \mathtt{q(f(X),X)} & \leftarrow & \mathtt{b(Y),\ p(X).} \\
r_2: & \mathbf{p(X)} & \leftarrow & \mathbf{q(f(X)},\mathtt{Y).}
\end{array}
$$



Cycle Classification

Two classifications of **cycles** in the **labeled argument graph**:

The aim is to identify "harmless" cycles.

Cycle Classification

Two classifications of **cycles** in the **labeled argument graph**:

The aim is to identify “harmless” cycles.

1. ***Balanced*** cycle
2. ***Growing*** cycle
3. ***Failing*** cycle

Cycle Classification

Two classifications of **cycles** in the **labeled argument graph**:

The aim is to identify "harmless" cycles.

1. ***Balanced*** cycle
2. ***Growing*** cycle
3. ***Failing*** cycle

1. ***Active*** cycle
2. ***Inactive*** cycle

Cycle Classification

Two classifications of **cycles** in the **labeled argument graph**:

The aim is to identify "harmless" cycles.

1. ***Balanced*** cycle
2. ***Growing*** cycle
3. ***Failing*** cycle

1. ***Active*** cycle
2. ***Inactive*** cycle

$\Uparrow$

The activation graph is also used

Balanced, Growing, and Failing Cycles

Classification on the basis of the **first component** of the edge labels.

Balanced / Growing / Failing Cycles

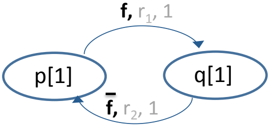
- ***Balanced cycle:*** a term propagated through the whole cycle remains the same.
- ***Growing cycle:*** a term propagated through the whole cycle grows.
- ***Failing cycle:*** a term propagated through the whole cycle decreases or cannot really go through the entire cycle.

Terms in an argument might grow infinitely only if this argument depends on a growing cycle.

Balanced, Growing, and Failing Cycles

Balanced cycle

$r_1 : \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$
$r_2 : \quad \mathtt{p(X)} \leftarrow \mathtt{q(f(X))}$



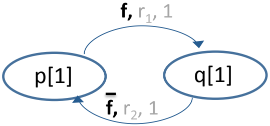
$f\bar{f} \approx \epsilon$

$\mathtt{p(a)}.$
$\mathtt{q(f(a))} \leftarrow \mathtt{p(a)}.$

Balanced, Growing, and Failing Cycles

Balanced cycle

$$r_1 : \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2 : \quad \mathtt{p(X)} \leftarrow \mathtt{q(f(X))}$$



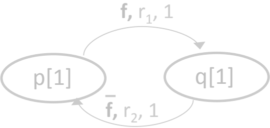
$$f\bar{f} \approx \epsilon$$

$$\mathtt{p(a).}$$
$$\mathtt{q(f(a))} \leftarrow \mathtt{p(a).}$$
$$\mathtt{p(a)} \leftarrow \mathtt{q(f(a)).}$$

Balanced, Growing, and Failing Cycles

Balanced cycle

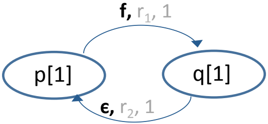
$r_1: \ \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$
$r_2: \ \mathtt{p(X)} \leftarrow \mathtt{q(f(X))}$



$f\bar{f} \approx \epsilon$

Growing cycle

$r_1: \ \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$
$r_2: \ \mathtt{p(X)} \leftarrow \mathtt{q(X)}$



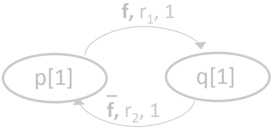
$f\epsilon \approx f$

$\mathtt{p(a)}.$
$\mathtt{q(f(a))} \leftarrow \mathtt{p(a)}.$

Balanced, Growing, and Failing Cycles

Balanced cycle

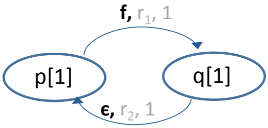
$r_1: \ \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$
$r_2: \ \mathtt{p(X)} \leftarrow \mathtt{q(f(X))}$



$f\bar{f} \approx \epsilon$

Growing cycle

$r_1: \ \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$
$r_2: \ \mathtt{p(X)} \leftarrow \mathtt{q(X)}$



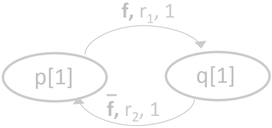
$f\epsilon \approx f$

$$\begin{array}{c} \mathtt{p(a).} \\ \mathtt{q(f(a))} \leftarrow \mathtt{p(a).} \\ \mathtt{p(f(a))} \leftarrow \mathtt{q(f(a)).} \end{array}$$

Balanced, Growing, and Failing Cycles

Balanced cycle

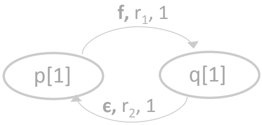
$$r_1: \; \texttt{q(f(X))} \leftarrow \texttt{p(X)}$$
$$r_2: \; \texttt{p(X)} \leftarrow \texttt{q(f(X))}$$



$$f\bar{f} \approx \epsilon$$

Growing cycle

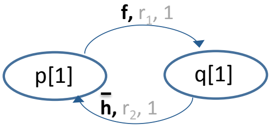
$$r_1: \; \texttt{q(f(X))} \leftarrow \texttt{p(X)}$$
$$r_2: \; \texttt{p(X)} \leftarrow \texttt{q(X)}$$



$$f\epsilon \approx f$$

Failing cycle

$$r_1: \; \texttt{q(f(X))} \leftarrow \texttt{p(X)}$$
$$r_2: \; \texttt{p(X)} \leftarrow \texttt{q(h(X))}$$



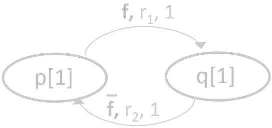
$$f\bar{h}$$

$$\texttt{p(a).}$$
$$\texttt{q(f(a))} \leftarrow \texttt{p(a).}$$

Balanced, Growing, and Failing Cycles

Balanced cycle

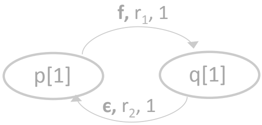
$$r_1: \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \quad \mathtt{p(X)} \leftarrow \mathtt{q(f(X))}$$



$$f\bar{f} \approx \epsilon$$

Growing cycle

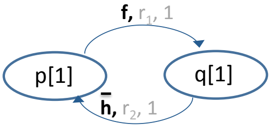
$$r_1: \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \quad \mathtt{p(X)} \leftarrow \mathtt{q(X)}$$



$$f\epsilon \approx f$$

Failing cycle

$$r_1: \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}$$
$$r_2: \quad \mathtt{p(X)} \leftarrow \mathtt{q(h(X))}$$



$$f\bar{h}$$

$$\mathtt{p(a)}.$$
$$\mathtt{q(f(a))} \leftarrow \mathtt{p(a)}.$$
$$\mathtt{p(a)} \leftarrow \mathtt{q(h(a))}.$$

Active and Inactive Cycles

- Classification on the basis of the **second component** of the edge labels.
- The activation graph is also used.

Active / Inactive Cycles

- ***Active*** cycle: the corresponding rules form a cycle in the activation graph.
- ***Inactive*** cycle: otherwise.

Active and Inactive Cycles

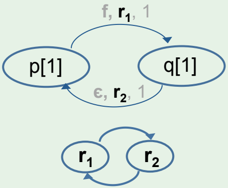
- Classification on the basis of the **second component** of the edge labels.
- The activation graph is also used.

Active / Inactive Cycles

- ***Active*** cycle: the corresponding rules form a cycle in the activation graph.
- ***Inactive*** cycle: otherwise.

Example (**Active** Cycle)

$$r_1 : \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}.$$
$$r_2 : \quad \mathtt{p(X)} \leftarrow \mathtt{q(X)}.$$



Active and Inactive Cycles

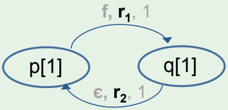
- Classification on the basis of the **second component** of the edge labels.
- The activation graph is also used.

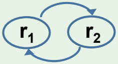
Active / Inactive Cycles

- ***Active*** cycle: the corresponding rules form a cycle in the activation graph.
- ***Inactive*** cycle: otherwise.

Example (**Active** Cycle)

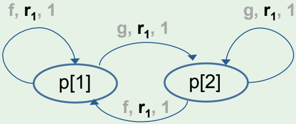
$$\begin{aligned} r_1 &: \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}. \\ r_2 &: \quad \mathtt{p(X)} \leftarrow \mathtt{q(X)}. \end{aligned}$$

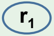




Example (**Inactive** Cycles)

$$r_1 : \quad \mathtt{p(f(X), g(X))} \leftarrow \mathtt{p(X, X)}.$$





Active and Inactive Cycles

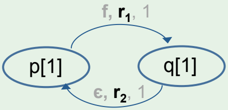
- Classification on the basis of the **second component** of the edge labels.
- The activation graph is also used.

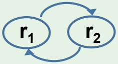
Active / Inactive Cycles

- ***Active*** cycle: the corresponding rules form a cycle in the activation graph.
- ***Inactive*** cycle: otherwise.

Example (**Active** Cycle)

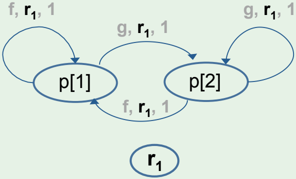
$$r_1 : \quad \mathtt{q(f(X))} \leftarrow \mathtt{p(X)}.$$
$$r_2 : \quad \mathtt{p(X)} \leftarrow \mathtt{q(X)}.$$





Example (**Inactive** Cycles)

$$r_1 : \quad \mathtt{p(f(X), g(X))} \leftarrow \mathtt{p(X, X)}.$$



Only active cycles may be "dangerous".

Argument-bounded Cycles

The depth of terms in an argument might grow only if this argument depends on an **active growing cycle**.

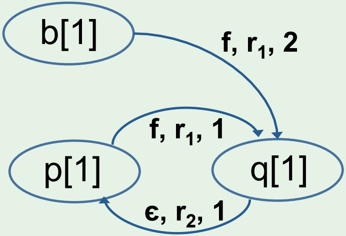
Argument-bounded Cycles

The depth of terms in an argument might grow only if this argument depends on an **active growing cycle**.

However ...

Example

$$\begin{array}{llll} r_1: & \mathtt{q(f(X))} & \leftarrow & \mathtt{p(X)}, \boldsymbol{b(X)} \\ r_2: & \mathtt{p(X)} & \leftarrow & \mathtt{q(X)}. \end{array}$$



Since b[1] is limited, the number of values propagated in q[1] is finite.

Twin Cycles

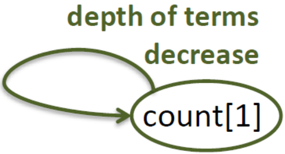
Example (List length)

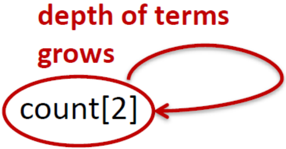
r_0 : $\texttt{count}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0}).$

r_1 : $\texttt{count}(\texttt{L},\texttt{s}(\texttt{I})) \leftarrow \texttt{count}([\texttt{X}|\texttt{L}],\texttt{I}).$

Query goal : $\texttt{count}([\,],\texttt{N}).$

$\texttt{count}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0})$
$\texttt{count}([\texttt{b},\texttt{c}],\texttt{s}(\texttt{0}))$
$\texttt{count}([\texttt{c}],\texttt{s}(\texttt{s}(\texttt{0})))$
$\texttt{count}([\,],\texttt{s}(\texttt{s}(\texttt{s}(\texttt{0}))))$





Twin Cycles

Example (List length)

r_0 : $\texttt{count}([\texttt{a}, \texttt{b}, \texttt{c}], \texttt{0}).$

r_1 : $\texttt{count}(\texttt{L}, \texttt{s}(\texttt{I})) \leftarrow \texttt{count}([\texttt{X}|\texttt{L}], \texttt{I}).$

Query goal : $\texttt{count}([\,], \texttt{N}).$

$\texttt{count}([\texttt{a}, \texttt{b}, \texttt{c}], \texttt{0})$
$\texttt{count}([\texttt{b}, \texttt{c}], \texttt{s}(\texttt{0}))$
$\texttt{count}([\texttt{c}], \texttt{s}(\texttt{s}(\texttt{0})))$
$\texttt{count}([\,], \texttt{s}(\texttt{s}(\texttt{s}(\texttt{0}))))$

The arguments may influence each other even if they do not exchange values

The growth of $\texttt{count}[2]$ is bounded by the reduction of $\texttt{count}[1]$.

Twin Cycles

Example (Append)

$$
\begin{array}{r}
\texttt{magic\_append}([\texttt{a},\texttt{b}],[\texttt{c},\texttt{d}]). \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \\
\texttt{magic\_append}(\texttt{L1},\texttt{L2}) \leftarrow \texttt{magic\_append}([\texttt{X}|\texttt{L1}],\texttt{L2}). \\
\texttt{append}([\,],\texttt{L},\texttt{L}) \leftarrow \texttt{magic\_append}([\,],\texttt{L}). \qquad\quad \\
\texttt{append}([\texttt{X}|\texttt{L1}],\texttt{L2},[\texttt{X}|\texttt{L3}]) \leftarrow \texttt{magic\_append}([\texttt{X}|\texttt{L1}],\texttt{L2}), \\
\texttt{append}(\texttt{L1},\texttt{L2},\texttt{L3}). \qquad\quad
\end{array}
$$

Twin Cycles

Cycles are classified on the basis of **the last two components** of the edge labels.

Twin Cycles

Cycles are classified on the basis of **the last two components** of the edge labels.

***Twin* cycles** describe the propagation of values through the same atoms of the same rules (the last two components of the edge labels coincide).

Twin Cycles

Cycles are classified on the basis of **the last two components** of the edge labels.

***Twin* cycles** describe the propagation of values through the same atoms of the same rules (the last two components of the edge labels coincide).

Example

$$r_1 : \; \mathtt{q}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{Y})).$$
$$r_2 : \; \mathtt{p}(\mathtt{f}(\mathtt{X}), \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}, \mathtt{Y}).$$

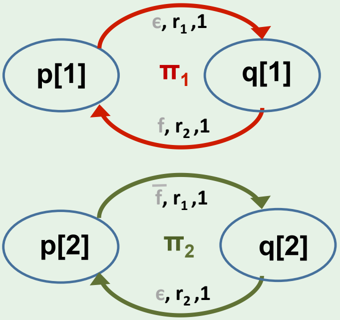
Twin Cycles

Cycles are classified on the basis of **the last two components** of the edge labels.

***Twin* cycles** describe the propagation of values through the same atoms of the same rules (the last two components of the edge labels coincide).

Example

r_1 : $\mathtt{q}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{Y}))$.
r_2 : $\mathtt{p}(\mathtt{f}(\mathtt{X}), \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}, \mathtt{Y})$.



Twin Cycles

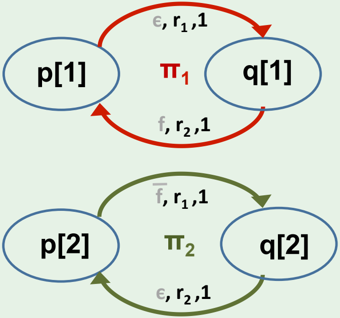
Cycles are classified on the basis of **the last two components** of the edge labels.

***Twin* cycles** describe the propagation of values through the same atoms of the same rules (the last two components of the edge labels coincide).

Example

$$r_1 : \ \mathtt{q}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{Y})).$$
$$r_2 : \ \mathtt{p}(\mathtt{f}(\mathtt{X}), \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}, \mathtt{Y}).$$

- π_1 and π_2 are ***twin* cycles**.



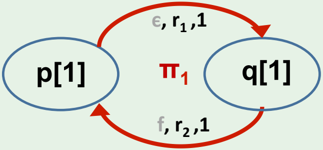
Twin Cycles

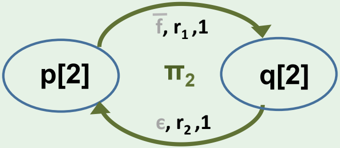
Cycles are classified on the basis of **the last two components** of the edge labels.

***Twin* cycles** describe the propagation of values through the same atoms of the same rules (the last two components of the edge labels coincide).

Example

$$r_1 : \ \mathtt{q(X, Y)} \leftarrow \mathtt{p(X, f(Y))}.$$
$$r_2 : \ \mathtt{p(f(X), Y)} \leftarrow \mathtt{q(X, Y)}.$$





- π_1 and π_2 are ***twin* cycles**.
- The **growth** of values in π_1 is **bounded by** the **reduction** in π_2.

Bounded Programs

- Start with a set *A* of limited arguments.
- Then, add an argument $p[i]$ if, for every cycle π on which $p[i]$ depends:
  1. π is not active or not growing;
  2. π has a twin cycle π' which is not balanced and goes only through arguments in *A*; or
  3. π is argument-bounded.

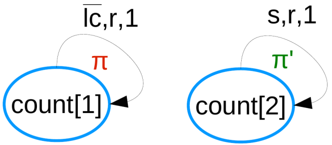
Bounded Programs

- Start with a set A of limited arguments.
- Then, add an argument $p[i]$ if, for every cycle π on which $p[i]$ depends:
  1. π is not active or not growing;
  2. π has a twin cycle π' which is not balanced and goes only through arguments in A; or
  3. π is argument-bounded.

Example

$\quad \mathtt{count}([\mathtt{a}, \mathtt{b}, \mathtt{c}], \mathtt{0}).$

$r: \ \mathtt{count}(\mathtt{L}, \mathtt{s}(\mathtt{I})) \leftarrow \mathtt{count}([\mathtt{X}|\mathtt{L}], \mathtt{I}).$



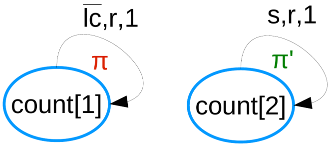
- Both π and π' are active cycles;

Bounded Programs

- Start with a set *A* of limited arguments.
- Then, add an argument $p[i]$ if, for every cycle π on which $p[i]$ depends:
  1. π is not active or not growing;
  2. π has a twin cycle π' which is not balanced and goes only through arguments in *A*; or
  3. π is argument-bounded.

Example

$$\texttt{count}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0}).$$
$$r: \texttt{count}(\texttt{L},\texttt{s}(\texttt{I})) \leftarrow \texttt{count}([\texttt{X}|\texttt{L}],\texttt{I}).$$



- Both π and π' are active cycles;
- π is failing, then `count[1]` is limited (Condition 1);

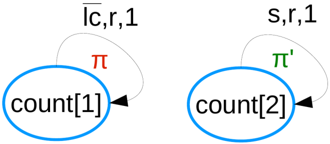
Bounded Programs

- Start with a set A of limited arguments.
- Then, add an argument $p[i]$ if, for every cycle π on which $p[i]$ depends:
  1. π is not active or not growing;
  2. π has a twin cycle π' which is not balanced and goes only through arguments in A; or
  3. π is argument-bounded.

Example

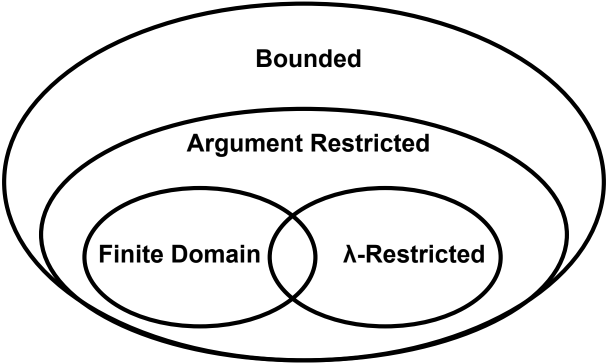
$\texttt{count}([\texttt{a},\texttt{b},\texttt{c}],\texttt{0}).$

$r:\ \texttt{count}(\texttt{L},\texttt{s}(\texttt{I})) \leftarrow \texttt{count}([\texttt{X}|\texttt{L}],\texttt{I}).$



- Both π and π' are active cycles;
- π is failing, then $\texttt{count}[1]$ is limited (Condition 1);
- π' is a twin of π. Since π is not balanced and its arguments are limited, $\texttt{count}[2]$ is also limited (Condition 2).

Relative Expressivity



Theorem

Argument Restricted $\subsetneq$ *Bounded*.

Rule-bounded programs [CGMT14]

Many practical programs contain rules where the "size" of the head atom **does not increase** w.r.t. the "size" of a body atom.

Example (Bubble Sort)

$$\begin{array}{l}
\mathtt{bub}(\mathtt{L},[\,],[\,]) \leftarrow \mathtt{input}(\mathtt{L}).\\
\mathtt{bub}([\mathtt{Y}|\mathtt{T}],[\mathtt{X}|\mathtt{Cur}],\mathtt{Sol}) \leftarrow \mathtt{bub}([\mathtt{X}|[\mathtt{Y}|\mathtt{T}]],\mathtt{Cur},\mathtt{Sol}),\ \mathtt{X} \leq \mathtt{Y}.\\
\mathtt{bub}([\mathtt{X}|\mathtt{T}],[\mathtt{Y}|\mathtt{Cur}],\mathtt{Sol}) \leftarrow \mathtt{bub}([\mathtt{X}|[\mathtt{Y}|\mathtt{T}]],\mathtt{Cur},\mathtt{Sol}),\ \mathtt{Y} < \mathtt{X}.\\
\mathtt{bub}(\mathtt{Cur},[\,],[\mathtt{X}|\mathtt{Sol}]) \leftarrow \mathtt{bub}([\mathtt{X}|[\,]],\mathtt{Cur},\mathtt{Sol}).
\end{array}$$

Rule-bounded programs [CGMT14]

Many practical programs contain rules where the “size” of the head atom **does not increase** w.r.t. the “size” of a body atom.

Example (Bubble Sort)

$$\mathtt{bub}(\mathtt{L}, [\,], [\,]) \leftarrow \mathtt{input}(\mathtt{L}).$$
$$\mathbf{bub([Y|T],[X|Cur],Sol)} \leftarrow \mathbf{bub([X|[Y|T]],Cur,Sol)},\ \mathtt{X} \leq \mathtt{Y}.$$
$$\mathtt{bub}([\mathtt{X}|\mathtt{T}], [\mathtt{Y}|\mathtt{Cur}], \mathtt{Sol}) \leftarrow \mathtt{bub}([\mathtt{X}|[\mathtt{Y}|\mathtt{T}]], \mathtt{Cur}, \mathtt{Sol}),\ \mathtt{Y} < \mathtt{X}.$$
$$\mathtt{bub}(\mathtt{Cur}, [\,], [\mathtt{X}|\mathtt{Sol}]) \leftarrow \mathtt{bub}([\mathtt{X}|[\,]], \mathtt{Cur}, \mathtt{Sol}).$$

Rule-bounded programs [CGMT14]

Many practical programs contain rules where the "size" of the head atom **does not increase** w.r.t. the "size" of a body atom.

Example (Tree Visit)

$$\texttt{visit}(\texttt{Tree}, [\,], [\,]) \leftarrow \texttt{input}(\texttt{Tree}).$$
$$\texttt{visit}(\texttt{Left}, [\texttt{Root}|\texttt{Visited}], [\texttt{Right}|\texttt{ToVisit}]) \leftarrow$$
$$\texttt{visit}(\texttt{tree}(\texttt{Root}, \texttt{Left}, \texttt{Right}), \texttt{Visited}, \texttt{ToVisit}).$$
$$\texttt{visit}(\texttt{Next}, \texttt{Visited}, \texttt{ToVisit}) \leftarrow \texttt{visit}(\texttt{null}, \texttt{Visited}, [\texttt{Next}|\texttt{ToVisit}]).$$

Rule-bounded programs [CGMT14]

Many practical programs contain rules where the "size" of the head atom **does not increase** w.r.t. the "size" of a body atom.

Example (Tree Visit)

$$\mathtt{visit}(\mathtt{Tree}, [\,], [\,]) \leftarrow \mathtt{input}(\mathtt{Tree}).$$
$$\mathtt{visit}(\mathtt{Left}, [\mathtt{Root}|\mathtt{Visited}], [\mathtt{Right}|\mathtt{ToVisit}]) \leftarrow \mathtt{visit}(\mathtt{tree}(\mathtt{Root}, \mathtt{Left}, \mathtt{Right}), \mathtt{Visited}, \mathtt{ToVisit}).$$
$$\mathtt{visit}(\mathtt{Next}, \mathtt{Visited}, \mathtt{ToVisit}) \leftarrow \mathtt{visit}(\mathtt{null}, \mathtt{Visited}, [\mathtt{Next}|\mathtt{ToVisit}]).$$

Example (List Concatenation)

$$\begin{array}{lcl}
\mathtt{reverse}(\mathtt{L}_1, [\,]) & \leftarrow & \mathtt{input1}(\mathtt{L}_1). \\
\mathtt{reverse}(\mathtt{L}_1, [\mathtt{X}|\mathtt{L}_2]) & \leftarrow & \mathtt{reverse}([\mathtt{X}|\mathtt{L}_1], \mathtt{L}_2). \\
\mathtt{append}(\mathtt{L}_1, \mathtt{L}_2) & \leftarrow & \mathtt{reverse}([\,], \mathtt{L}_1),\ \mathtt{input2}(\mathtt{L}_2). \\
\mathtt{append}(\mathtt{L}_1, [\mathtt{X}|\mathtt{L}_2]) & \leftarrow & \mathtt{append}([\mathtt{X}|\mathtt{L}_1], \mathtt{L}_2).
\end{array}$$

Rule-bounded programs [CGMT14]

- **Basic idea:** check if the size of the head is bounded by the size of a body atom.
- **Linear constraints** are used to check this condition.
- **Question:** How do we measure the size of an atom?

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\,\mathtt{X},\ \mathtt{c},\ \mathtt{g}(\mathtt{Y},\mathtt{Z})\,)$$

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\,\mathtt{X},\ \mathtt{c},\ \mathtt{g}(\mathtt{Y},\mathtt{Z})\,)$$

$$\Downarrow$$

$$size(t) \;=\; \mathbf{3} + (\mathtt{x}\ +\ \mathtt{0}\ +\ size(\mathtt{g}(\mathtt{Y},\mathtt{Z})))$$

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\,\mathtt{X},\ \mathtt{c},\ \mathtt{g}(\mathtt{Y},\mathtt{Z})\,)$$

$$\Downarrow$$

$$\begin{aligned} \mathit{size}(t) &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + \mathit{size}(\mathtt{g}(\mathtt{Y},\mathtt{Z}))) \\ &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + (\mathtt{2} + \mathtt{y} + \mathtt{z})) \end{aligned}$$

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\mathtt{X}, \mathtt{c}, \mathtt{g}(\mathtt{Y}, \mathtt{Z}))$$

$$\Downarrow$$

$$\begin{aligned} \mathit{size}(t) &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + \mathit{size}(\mathtt{g}(\mathtt{Y}, \mathtt{Z}))) \\ &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + (\mathtt{2} + \mathtt{y} + \mathtt{z})) \end{aligned}$$

Intuition: A template for all possible sizes the term may have during the program evaluation.

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\mathtt{X}, \mathtt{c}, \mathtt{g}(\mathtt{Y}, \mathtt{Z}))$$

$$\Downarrow$$

$$\begin{aligned} \mathit{size}(t) &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + \mathit{size}(\mathtt{g}(\mathtt{Y}, \mathtt{Z}))) \\ &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + (\mathtt{2} + \mathtt{y} + \mathtt{z})) \end{aligned}$$

Intuition: A template for all possible sizes the term may have during the program evaluation.

Atom size: Linear combination of the size of its terms.

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\,\mathtt{X},\ \mathtt{c},\ \mathtt{g}(\mathtt{Y},\mathtt{Z})\,)$$

$$\Downarrow$$

$$\begin{aligned} \mathit{size}(t) &= \mathbf{3} + (\mathtt{x} \ + \ \mathtt{0} \ + \ \mathit{size}(\mathtt{g}(\mathtt{Y},\mathtt{Z}))) \\ &= \mathbf{3} + (\mathtt{x} \ + \ \mathtt{0} \ + \ (\mathtt{2} + \mathtt{y} + \mathtt{z})) \end{aligned}$$

Intuition: A template for all possible sizes the term may have during the program evaluation.

Atom size: Linear combination of the size of its terms.

$$A = p(t_1, \ldots, t_n)$$

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\,\mathtt{X},\ \mathtt{c},\ \mathtt{g}(\mathtt{Y},\mathtt{Z})\,)$$

$$\Downarrow$$

$$\begin{aligned} \mathit{size}(t) &= \mathbf{3} + (\mathtt{x} \ + \ \mathtt{0} \ + \ \mathit{size}(\mathtt{g}(\mathtt{Y},\mathtt{Z}))) \\ &= \mathbf{3} + (\mathtt{x} \ + \ \mathtt{0} \ + \ (\mathtt{2} + \mathtt{y} + \mathtt{z})) \end{aligned}$$

Intuition: A template for all possible sizes the term may have during the program evaluation.

Atom size: Linear combination of the size of its terms.

$$A = p(t_1, \ldots, t_n)$$

$$\Downarrow$$

$$\mathit{size}(A) = \alpha_{p_1} \cdot \mathit{size}(t_1) + \ldots + \alpha_{p_n} \cdot \mathit{size}(t_n)$$

Rule-bounded programs - Notions of Size

Term size:

$$t = \mathbf{f}(\mathtt{X}, \mathtt{c}, \mathtt{g}(\mathtt{Y}, \mathtt{Z}))$$
$$\Downarrow$$
$$\begin{aligned} \mathit{size}(t) &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + \mathit{size}(\mathtt{g}(\mathtt{Y}, \mathtt{Z}))) \\ &= \mathbf{3} + (\mathtt{x} + \mathtt{0} + (\mathtt{2} + \mathtt{y} + \mathtt{z})) \end{aligned}$$

Intuition: A template for all possible sizes the term may have during the program evaluation.

Atom size: Linear combination of the size of its terms.

$$A = p(t_1, \ldots, t_n)$$
$$\Downarrow$$
$$\mathit{size}(A) = \alpha_{p_1} \cdot \mathit{size}(t_1) + \ldots + \alpha_{p_n} \cdot \mathit{size}(t_n)$$

Integer coefficients $\alpha_{\mathrm{p}_1}, \ldots, \alpha_{\mathrm{p}_n}$ will be chosen depending on the program structure.

Rule-bounded program - Example

Example (List Length)

$$r_1 : \mathtt{len}([\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}], \mathtt{0}).$$
$$r_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

Rule-bounded program - Example

Example (List Length)

$$r_1 : \texttt{len}([\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}], \texttt{0}).$$
$$r_2 : \texttt{len}(\texttt{Tail}, \mathbf{s}(\texttt{N})) \leftarrow \texttt{len}(\mathbf{list}(\texttt{Head}, \texttt{Tail}), \texttt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

Rule-bounded program - Example

Example (List Length)

$$\texttt{r}_1 : \texttt{len}([\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}], \texttt{0}).$$
$$\texttt{r}_2 : \texttt{len}(\texttt{Tail}, \mathbf{s}(\texttt{N})) \leftarrow \texttt{len}(\textbf{list}(\texttt{Head}, \texttt{Tail}), \texttt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail)$$

Rule-bounded program - Example

Example (List Length)

$$\mathtt{r}_1 : \mathtt{len}([\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}], \mathtt{0}).$$
$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n$$

Rule-bounded program - Example

Example (List Length)

$$\mathtt{r}_1 : \mathtt{len}([\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}], \mathtt{0}).$$
$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n \geq \alpha_1 \cdot tail$$

Rule-bounded program - Example

Example (List Length)

$$r_1 : \mathtt{len}([\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}], 0).$$
$$r_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

Rule-bounded program - Example

Example (List Length)

$$r_1 : \mathtt{len}([a, b, c, d], 0).$$
$$r_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

Find α_1 and α_2 s.t. the inequality holds for all *head*, *tail*, *n* $\in \mathbb{N}$

Rule-bounded program - Example

Example (List Length)

$$r_1 : \mathtt{len}([\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}], 0).$$
$$r_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

Find α_1 and α_2 s.t. the inequality holds for all $head, tail, n \in \mathbb{N}$

$$2 \cdot \alpha_1 + \alpha_1 \cdot head + \cancel{\alpha_1 \cdot tail} + \cancel{\alpha_2 \cdot n} \geq \cancel{\alpha_1 \cdot tail} + \alpha_2 + \cancel{\alpha_2 \cdot n}$$

Rule-bounded program - Example

Example (List Length)

r_1 : len([a, b, c, d], 0).
r_2 : len(Tail, **s**(N)) $\leftarrow$ len(**list**(Head, Tail), N).

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

Find α_1 and α_2 s.t. the inequality holds for all *head*, *tail*, *n* $\in \mathbb{N}$

$$2 \cdot \alpha_1 + \alpha_1 \cdot head + \cancel{\alpha_1 \cdot tail} + \cancel{\alpha_2 \cdot n} \geq \cancel{\alpha_1 \cdot tail} + \alpha_2 + \cancel{\alpha_2 \cdot n}$$

$$2 \cdot \alpha_1 \geq \alpha_2$$

Rule-bounded program - Example

Example (List Length)

$$\mathtt{r}_1 : \mathtt{len}([\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}], \mathtt{0}).$$
$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \mathtt{N}).$$

We need to check:

$$size(body(r_2)) \geq size(head(r_2))$$

$$\alpha_1 \cdot (2 + head + tail) + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

Find α_1 and α_2 s.t. the inequality holds for all $head, tail, n \in \mathbb{N}$

$$2 \cdot \alpha_1 + \alpha_1 \cdot head + \cancel{\alpha_1 \cdot tail} + \cancel{\alpha_2 \cdot n} \geq \cancel{\alpha_1 \cdot tail} + \alpha_2 + \cancel{\alpha_2 \cdot n}$$
$$2 \cdot \alpha_1 \geq \alpha_2$$

We can choose $\alpha_1 = \alpha_2 = 1 \Rightarrow$ **the program is rule-bounded**.

Rule-bounded programs - Another Example

Example (Bubble sort)

$\texttt{sort}([\texttt{b},\texttt{a},\texttt{d},\texttt{h},\texttt{e}],[\,],[\,]).$

$\texttt{sort}([\texttt{Y}|\texttt{T}],[\texttt{X}|\texttt{Temp}],\texttt{Sorted}) \leftarrow \texttt{sort}([\texttt{X}|[\texttt{Y}|\texttt{T}]]),\texttt{Temp},\texttt{Sorted}),\texttt{X} \leq \texttt{Y}.$

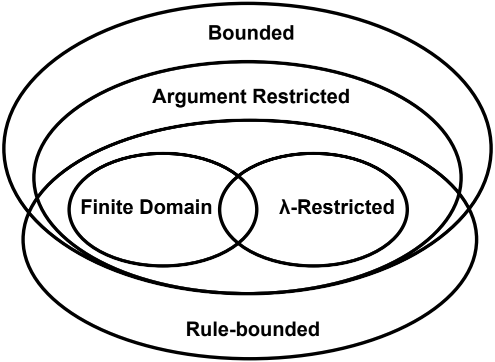
$\texttt{sort}([\texttt{X}|\texttt{T}],[\texttt{Y}|\texttt{Temp}],\texttt{Sorted}) \leftarrow \texttt{sort}([\texttt{X}|[\texttt{Y}|\texttt{T}]]),\texttt{Temp},\texttt{Sorted}),\texttt{Y} < \texttt{X}.$

$\texttt{sort}(\texttt{Temp},[\,],[\texttt{X}|\texttt{Sorted}]) \leftarrow \texttt{sort}([\texttt{X}],\texttt{Temp},\texttt{Sorted}).$

$$\begin{cases} \alpha_1 \cdot (4+x+y+t) + \alpha_2 \cdot \mathit{temp} + \alpha_3 \cdot \mathit{sorted} \geq \\ \qquad\qquad \alpha_1 \cdot (2+y+t) + \alpha_2 \cdot (2+x+\mathit{temp}) + \alpha_3 \cdot \mathit{sorted} \\ \alpha_1 \cdot (4+x+y+t) + \alpha_2 \cdot \mathit{temp} + \alpha_3 \cdot \mathit{sorted} \geq \\ \qquad\qquad \alpha_1 \cdot (2+x+t) + \alpha_2 \cdot (2+y+\mathit{temp}) + \alpha_3 \cdot \mathit{sorted} \\ \alpha_1 \cdot (2+x) + \alpha_2 \cdot \mathit{temp} + \alpha_3 \cdot \mathit{sorted} \geq \\ \qquad\qquad \alpha_1 \cdot \mathit{temp} + \alpha_3 \cdot (2+x+\mathit{sorted}) \end{cases}$$

A possible solution is $\alpha_1 = 2$, $\alpha_2 = 2$, $\alpha_3 = 1$

Relative Expressivity



Theorem

- *Finite Domain* $\subsetneq$ *Rule-bounded.*
- λ*-Restricted* $\subsetneq$ *Rule-bounded.*
- *Argument Restricted* $\nparallel$ *Rule-bounded.*
- *Bounded* $\nparallel$ *Rule-bounded.*

Program Adornment [GMT13a]

Program Adornment [GMT13a]

- The technique can be used in conjunction with current termination criteria allowing them to detect more programs having a terminating evaluation.
- The technique transforms a program P into an (adorned) "equivalent" program P^μ.
- The aim is to apply termination criteria to the adorned program P^μ rather than the original program P.

Program Adornment

- Suppose we want to check if the evaluation of a program P terminates by applying a criterion C.
- We first transform P into an adorned program P^{μ}.
- Then, we apply criterion C to P^{μ} (rather than the original program P).
- (Soundness) If P^{μ} satisfies criterion C, then the evaluation of the original program P terminates.
- This approach strictly enlarges the class of programs identified by criterion C.

Example

Original program

$$p(X,X) \leftarrow base(X)$$

$$q(X,Y) \leftarrow p(X,Y)$$

$$p(f(X),g(X)) \leftarrow q(X,X)$$

Example

Original program

$p(X,X) \leftarrow base(X)$

$q(X,Y) \leftarrow p(X,Y)$

$p(f(X),g(X)) \leftarrow q(X,X)$

Adorned program

$p^{\varepsilon\varepsilon}(X,X) \leftarrow base^{\varepsilon}(X)$

$q^{\varepsilon\varepsilon}(X,Y) \leftarrow p^{\varepsilon\varepsilon}(X,Y)$

$p^{f_1 g_1}(f(X),g(X)) \leftarrow q^{\varepsilon\varepsilon}(X,X)$

$q^{f_1 g_1}(X,Y) \leftarrow p^{f_1 g_1}(X,Y)$

Each adorned rule is obtained from a rule in the original program by adding adornments which keep track of the structure of the terms that can be propagated during the bottom-up evaluation.

Example

Original program

$p(X,X) \leftarrow base(X)$

$q(X,Y) \leftarrow p(X,Y)$

$p(f(X),g(X)) \leftarrow q(X,X)$

Adorned program

$p^{\varepsilon\varepsilon}(X,X) \leftarrow base^{\varepsilon}(X)$

$q^{\varepsilon\varepsilon}(X,Y) \leftarrow p^{\varepsilon\varepsilon}(X,Y)$

$p^{f_1 g_1}(f(X),g(X)) \leftarrow q^{\varepsilon\varepsilon}(X,X)$

$q^{f_1 g_1}(X,Y) \leftarrow p^{f_1 g_1}(X,Y)$

The adorned program is "equivalent" to the original one in the following sense: the minimal model of the original program can be obtained from the minimal model of the adorned program by dropping adornments.

Adornment Algorithm

Original program

$$p(X, f(X)) \leftarrow base(X)$$

$$p(X, f(X)) \leftarrow p(Y, X), base(Y)$$

$$p(X, Y) \leftarrow p(f(X), f(Y))$$

Adornment Algorithm

Original program

$$p(X, f(X)) \leftarrow base(X)$$
$$p(X, f(X)) \leftarrow p(Y, X), base(Y)$$
$$p(X, Y) \leftarrow p(f(X), f(Y))$$

Adorned program

Adorned predicate symbols

$base^{\varepsilon}$

Adornment definitions

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program

Adorned predicate symbols

$base^{\varepsilon}$

Adornment definitions

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program

$\leftarrow base^{\varepsilon}(X)$

Adorned predicate symbols

$base^{\varepsilon}$

Adornment definitions

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program

$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$

Adorned predicate symbols	***Adornment definitions***
$base^{\varepsilon}$	
$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$

Adornment Algorithm

Original program

$$p(X, f(X)) \leftarrow base(X)$$
$$p(X, f(X)) \leftarrow p(Y, X), base(Y)$$
$$p(X, Y) \leftarrow p(f(X), f(Y))$$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$

Adornment Algorithm

Original program

$$p(X, f(X)) \leftarrow base(X)$$

$$p(X, f(X)) \leftarrow p(Y, X), base(Y)$$

$$p(X, Y) \leftarrow p(f(X), f(Y))$$

Adorned program

$$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$$

$$\leftarrow p^{\varepsilon f_1}(Y, X), base^{\varepsilon}(Y)$$

Adorned predicate symbols	***Adornment definitions***
$base^{\varepsilon}$	
$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}(X, f(X)) \leftarrow p^{\varepsilon f_1}(Y, X), base^{\varepsilon}(Y)$	$p^{f_1 f_2}$	$f_2 = f(f_1)$

Adornment Algorithm

Original program

$$p(X, f(X)) \leftarrow base(X)$$
$$p(X, f(X)) \leftarrow p(Y, X), base(Y)$$
$$p(X, Y) \leftarrow p(f(X), f(Y))$$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}(X, f(X)) \leftarrow p^{\varepsilon f_1}(Y, X), base^{\varepsilon}(Y)$	$p^{f_1 f_2}$	$f_2 = f(f_1)$

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}(X, f(X)) \leftarrow p^{\varepsilon f_1}(Y, X), base^{\varepsilon}(Y)$	$p^{f_1 f_2}$	$f_2 = f(f_1)$

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program

$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$

$p^{f_1 f_2}(X, f(X)) \leftarrow p^{\varepsilon f_1}(Y, X), base^{\varepsilon}(Y)$

$\leftarrow p^{f_1 f_2}(f(X), f(Y))$

Adorned predicate symbols	Adornment definitions
$base^{\varepsilon}$	
$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}$	$f_2 = f(f_1)$

Adornment Algorithm

Original program

$p(X, f(X)) \leftarrow base(X)$

$p(X, f(X)) \leftarrow p(Y, X), base(Y)$

$p(X, Y) \leftarrow p(f(X), f(Y))$

Adorned program

	Adorned predicate symbols	Adornment definitions
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X, f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}(X, f(X)) \leftarrow p^{\varepsilon f_1}(Y, X), base^{\varepsilon}(Y)$	$p^{f_1 f_2}$	$f_2 = f(f_1)$
$p^{\varepsilon f_1}(X, Y) \leftarrow p^{f_1 f_2}(f(X), f(Y))$		

Adornment Algorithm

Original program

$$p(X,f(X)) \leftarrow base(X)$$
$$p(X,f(X)) \leftarrow p(Y,X), base(Y)$$
$$p(X,Y) \leftarrow p(f(X),f(Y))$$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X,f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}(X,f(X)) \leftarrow p^{\varepsilon f_1}(Y,X), base^{\varepsilon}(Y)$	$p^{f_1 f_2}$	$f_2 = f(f_1)$
$p^{\varepsilon f_1}(X,Y) \leftarrow p^{f_1 f_2}(f(X),f(Y))$		

The adornment algorithm terminates because no new *coherently adorned* body conjunction can be generated.

Adornment Algorithm

Original program

$$p(X,f(X)) \leftarrow base(X)$$

$$p(X,f(X)) \leftarrow p(Y,X), base(Y)$$

$$p(X,Y) \leftarrow p(f(X),f(Y))$$

Adorned program	***Adorned predicate symbols***	***Adornment definitions***
	$base^{\varepsilon}$	
$p^{\varepsilon f_1}(X,f(X)) \leftarrow base^{\varepsilon}(X)$	$p^{\varepsilon f_1}$	$f_1 = f(\varepsilon)$
$p^{f_1 f_2}(X,f(X)) \leftarrow p^{\varepsilon f_1}(Y,X), base^{\varepsilon}(Y)$	$p^{f_1 f_2}$	$f_2 = f(f_1)$
$p^{\varepsilon f_1}(X,Y) \leftarrow p^{f_1 f_2}(f(X),f(Y))$		

$p^{f_1 f_2}(Y,X), base^{\varepsilon}(Y)$ is not coherently adorned because *Y* is associated with the two different adornment symbols f_1 and ε

Properties

Theorem

The adornment algorithm always terminates.

Properties

Theorem

The adornment algorithm always terminates.

Theorem

Let P be a program and P^{μ} the adorned version of P. Then, the least model of P is equal to the least model of P^{μ} with adornments dropped from predicates.

Properties

Theorem

The adornment algorithm always terminates.

Theorem

Let P be a program and P^{μ} the adorned version of P. Then, the least model of P is equal to the least model of P^{μ} with adornments dropped from predicates.

Theorem

Let P be a program and P^{μ} the adorned version of P. If P^{μ} satisfies a termination criterion C, then the evaluation of $P \cup D$ terminates for any finite set of (flat) database facts D.

Properties

Theorem

The adornment algorithm always terminates.

Theorem

Let P be a program and P^μ the adorned version of P. Then, the least model of P is equal to the least model of P^μ with adornments dropped from predicates.

Theorem

Let P be a program and P^μ the adorned version of P. If P^μ satisfies a termination criterion C, then the evaluation of $P \cup D$ terminates for any finite set of (flat) database facts D.

Theorem

By applying a termination criterion to adorned programs we are able to identify more programs whose evaluation terminates.

Dealing with Negation and Disjunction

Dealing with Negation and Disjunction

Definition

A *Datalog*$^{\vee,\neg}$ *rule* is of the form

$$A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n$$

where $m > 0$, $k \geq 0$, $n \geq 0$, and the A_i's, B_i's, C_i's are atoms.

A *Datalog*$^{\vee,\neg}$ *program* is a finite set of Datalog$^{\vee,\neg}$ rules.

Dealing with Negation and Disjunction

Definition

A *Datalog*$^{\vee,\neg}$ *rule* is of the form

$$A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n$$

where $m > 0$, $k \geq 0$, $n \geq 0$, and the A_i's, B_i's, C_i's are atoms.

A *Datalog*$^{\vee,\neg}$ *program* is a finite set of Datalog$^{\vee,\neg}$ rules.

Semantics: **Stable Model Sematics.**

We want to check if a Datalog$^{\vee,\neg}$ program has a finite number of stable models, each of finite size and that can be computed.

Dealing with Negation and Disjunction

We want to check if a Datalog$^{\vee,\neg}$ program P has a finite number of stable models, each of finite size and that can be computed.

We derive a Datalog program $st(P)$ from P as follows.

Dealing with Negation and Disjunction

We want to check if a Datalog$^{\vee,\neg}$ program P has a finite number of stable models, each of finite size and that can be computed.

We derive a Datalog program $st(P)$ from P as follows.
Each rule

$$A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n$$

in P is replaced with m Datalog rules

$$A_i \leftarrow B_1, \ldots, B_k \qquad\qquad 1 \le i \le m$$

Dealing with Negation and Disjunction

We want to check if a Datalog$^{\vee,\neg}$ program P has a finite number of stable models, each of finite size and that can be computed.

We derive a Datalog program $st(P)$ from P as follows.
Each rule

$$A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n$$

in P is replaced with m Datalog rules

$$A_i \leftarrow B_1, \ldots, B_k \qquad\qquad 1 \leq i \leq m$$

Proposition

If $st(P)$ satisfies a termination criterion, then P has a finite number of stable models, each of them is of finite size and can be computed.

Conclusions

- The evaluation of logic programs with function symbols might not terminate, and establishing termination is not decidable.
- One solution: (Sufficient) Termination Conditions.
- Related lines of research:
  - Ensure decidability of some reasoning tasks even if there might be infinite and infinitely many stable models (e.g., $\mathbb{FDNC}$ programs [ES10, Bon11], Finitary Programs [Bon04], Finitely Recursive Programs [BBC09]).
  - Finite well-founded model [RS14].

Directions for Future Work

1. Combining termination criteria.
   One approach: identify arguments that are “limited” even when the program is not entirely recognized as terminating.
   - support the user in the problem formulation;
   - provide limited arguments to other techniques which can leverage this kind of information.

Directions for Future Work

1. Combining termination criteria.
One approach: identify arguments that are “limited” even when the program is not entirely recognized as terminating.
    - support the user in the problem formulation;
    - provide limited arguments to other techniques which can leverage this kind of information.
2. Exploiting negation and disjunction.

Example

$$p(f(X)) \leftarrow p(X), \neg p(X)$$

will be analyzed like

$$p(f(X)) \leftarrow p(X), \cancel{\neg p(X)}$$

Directions for Future Work

1. Combining termination criteria.
   One approach: identify arguments that are "limited" even when the program is not entirely recognized as terminating.
   - support the user in the problem formulation;
   - provide limited arguments to other techniques which can leverage this kind of information.
2. Exploiting negation and disjunction.

Example

$$p(f(X)) \leftarrow p(X), \neg p(X)$$

will be analyzed like

$$p(f(X)) \leftarrow p(X), \cancel{\neg p(X)}$$

3. Interpreted function symbols.

Directions for Future Work

1. Combining termination criteria.
   One approach: identify arguments that are "limited" even when the program is not entirely recognized as terminating.
   - support the user in the problem formulation;
   - provide limited arguments to other techniques which can leverage this kind of information.
2. Exploiting negation and disjunction.

Example

$$p(f(X)) \leftarrow p(X), \neg p(X)$$

will be analyzed like

$$p(f(X)) \leftarrow p(X), \cancel{\neg p(X)}$$

3. Interpreted function symbols.
4. Testing Local Stratification.

Thanks!

Questions?

Part II

Existential Rules

Existential rules

Special rules whose head atoms:

- may have existentially quantified variables,
- may be equality conditions (between two variables).

Existential rules

Special rules whose head atoms:

- may have existentially quantified variables,
- may be equality conditions (between two variables).

Used in a variety of contexts:

- in databases to define integrity constraints;
- in data integration and data exchange to define schema mappings;
- for knowledge representation and ontological reasoning (Datalog$^{\pm}$).

Integrity constraints in databases

Example

$$emp(Emp\#, Name, Address) \qquad worksFor(Emp\#, Prj\#)$$

Integrity constraints in databases

Example

$$emp(Emp\#, Name, Address) \qquad worksFor(Emp\#, Prj\#)$$

- Inclusion dependencies and foreign keys:

$$worksFor(E, P) \rightarrow \exists N \, \exists A \; emp(E, N, A)$$

- Functional Dependencies and internal keys

$$emp(E, N_1, Pr_1) \wedge emp(E, N_2, Pr_2) \rightarrow N_1 = N_2$$

Schema Mappings in Data Exchange

Data Exchange: Transform data structured under a source schema into data structured under a different target schema.

Example

Company A	Company B
$empA(Emp\#, Name, Address, Salary)$	$empB(Emp\#, Name, Phone, Salary)$

Schema Mappings in Data Exchange

Data Exchange: Transform data structured under a source schema into data structured under a different target schema.

Example

Company A	Company B
$empA(Emp\#, Name, Address, Salary)$	$empB(Emp\#, Name, Phone, Salary)$

Company A is acquired by Company B

$$empA(E, N, A, S) \rightarrow \exists P \; empB(E, N, P, S)$$

Encoding Ontologies

- Plain Datalog allows for encoding some ontological axioms:
- TGDs can also express other important ontological axioms:

Encoding Ontologies

- Plain Datalog allows for encoding some ontological axioms:
- TGDs can also express other important ontological axioms:

- Concept Inclusions:
  $$\forall X\ emp(X) \rightarrow person(X)$$
- (Inverse) Relation Inclusion:
  $$\forall X\ \forall Y\ manages(X,Y) \rightarrow isManaged(Y,X)$$
- Relation Transitivity:
  $$\forall X\ \forall Y\ \forall Z\ mgs(X,Y), mgs(Y,Z) \rightarrow mgs(X,Z)$$
- Participation:
  $$\forall X\ emp(X) \rightarrow \exists Y\ report(X,Y)$$
- Disjointness:
  $$\forall X\ emp(X), customer(X) \rightarrow false$$
- Functionality:
  $$\forall X\ \forall Y\ \forall Z\ reports(X,Y), reports(X,Z) \rightarrow Y = Z$$

The Problem

Answering queries under constraints

The Problem

Input:

- *A database D (set of ground facts),*
- *A set of data dependencies (integrity constraints) Σ,*
- *A (boolean) conjunctive query Q*

Question:

- $D \cup \Sigma \models Q$

Answering queries under constraints

The Problem

Input:

- *A database D (set of ground facts),*
- *A set of data dependencies (integrity constraints) Σ,*
- *A (boolean) conjunctive query Q*

Question:

- $D \cup \Sigma \models Q$

Very old problem: CQ answering over incomplete databases

Answering queries under constraints

The Problem

Input:

- *A database D (set of ground facts),*
- *A set of data dependencies (integrity constraints) Σ,*
- *A (boolean) conjunctive query Q*

Question:

- $D \cup \Sigma \models Q$

Very old problem: CQ answering over incomplete databases

Undecidable in the general case

Answering queries under constraints

The Problem

Data dependencies:

- *Tuple generating dependencies (TGDs):*

$$\forall \overline{X}\ \forall \overline{Y}\ \varphi(\overline{X}, \overline{Y}) \rightarrow \exists \overline{Z}\ \psi(\overline{X}, \overline{Z})$$

- *Equality generating dependencies (EGDs):*

$$\forall \overline{X}\ \varphi(\overline{X}) \rightarrow X_1 = X_2$$

$\varphi(\overline{X}, \overline{Y}), \varphi(\overline{X})$ and $\psi(\overline{Z}, \overline{X})$ are conjunctions of atoms, $X_1, X_2 \in \overline{X}$.

Answering queries under constraints

The Problem

Data dependencies:

- *Tuple generating dependencies (TGDs):*

$$\forall \overline{X}\ \forall \overline{Y}\ \varphi(\overline{X}, \overline{Y}) \rightarrow \exists \overline{Z}\ \psi(\overline{X}, \overline{Z})$$

- *Equality generating dependencies (EGDs):*

$$\forall \overline{X}\ \varphi(\overline{X}) \rightarrow X_1 = X_2$$

$\varphi(\overline{X}, \overline{Y}), \varphi(\overline{X})$ and $\psi(\overline{Z}, \overline{X})$ are conjunctions of atoms, $X_1, X_2 \in \overline{X}$.

$K = D \cup \Sigma$ is called *knowledge base*.

Answering queries under constraints

Example (Models and answers)

- Database: $D = \{person(john)\}$
- Data dependencies Σ:

$$\forall X \; person(X) \rightarrow \exists Z \; fatherOf(Z, X)$$
$$\forall X \, \forall Y \; fatherOf(X, Y), person(Y) \rightarrow person(X)$$

Answering queries under constraints

Example (Models and answers)

- Database: $D = \{person(john)\}$
- Data dependencies Σ:

$$\forall X\ person(X) \rightarrow \exists Z\ fatherOf(Z, X)$$
$$\forall X\ \forall Y\ fatherOf(X, Y), person(Y) \rightarrow person(X)$$

- Queries:

$$Q_1 = \exists X\ fatherOf(X, john)$$
$$Q_2 = \exists X\ fatherOf(john, X)$$

Answering queries under constraints

Example (Models and answers)

- Database: $D = \{person(john)\}$
- Data dependencies Σ:

$$\forall X\ person(X) \rightarrow \exists Z\ fatherOf(Z, X)$$
$$\forall X\ \forall Y\ fatherOf(X, Y), person(Y) \rightarrow person(X)$$

- Queries:

$$Q_1 = \exists X\ fatherOf(X, john)$$
$$Q_2 = \exists X\ fatherOf(john, X)$$

- Answers:

$$D \cup \Sigma \models Q_1 \qquad certain(Q_1, (D, \Sigma)) = "yes"$$
$$D \cup \Sigma \not\models Q_2 \qquad certain(Q_2, (D, \Sigma)) = "no"$$

All models of $D \cup \Sigma$ contain an atom $fatherOf(x, john)$,

Datalog$^{\pm}$ (Syntax)

Datalog variant for ontological reasoning allowing in the head:

- existential variables (TGDs)
- Equality atoms (EGDs)
- Constant *false* (Denial constraints)

Also denoted as *Datalog*$[\exists, =, F]$

More expressive than several ontological reasoning languages (e.g. UML Class Diagrams, DL-Lite, $\mathcal{ELHI}^{\neg}$, F-Logic Lite).

Query answering under Datalog$^{\pm}$ is undecidable

Query answering is undecidable

$\Rightarrow$

Determine decidable classes of queries

Answering queries over incomplete databases

Definition (Incomplete databases/Naive tables)

Databases may be *incomplete*, that is may contain (labeled) nulls (of the form $\perp_i$), representing the presence of unknown values.

Answering queries over incomplete databases

Definition (Incomplete databases/Naive tables)

Databases may be *incomplete*, that is may contain (labeled) nulls (of the form $\perp_i$), representing the presence of unknown values.

Definition (Possible worlds (under CWA))

Given a possibly incomplete database D, **POSS**(**D**) denotes the set of ground databases obtained from D by replacing nulls with constants.

Answering queries over incomplete databases

Definition (Incomplete databases/Naive tables)

Databases may be *incomplete*, that is may contain (labeled) nulls (of the form $\perp_i$), representing the presence of unknown values.

Definition (Possible worlds (under CWA))

Given a possibly incomplete database D, **POSS(D)** denotes the set of ground databases obtained from D by replacing nulls with constants.

Example (POSS(D))

- $D = \{person(john), person(frank), fatherOf(\perp_1, john)\}$

Answering queries over incomplete databases

Definition (Incomplete databases/Naive tables)

Databases may be *incomplete*, that is may contain (labeled) nulls (of the form $\perp_i$), representing the presence of unknown values.

Definition (Possible worlds (under CWA))

Given a possibly incomplete database D, **POSS(D)** denotes the set of ground databases obtained from D by replacing nulls with constants.

Example (POSS(D))

- $D = \{person(john), person(frank), fatherOf(\perp_1, john)\}$
- $POSS(D)$ (under CWA) contains:
  - $\{person(john), person(frank), fatherOf(john, john)\}$
  - $\{person(john), person(frank), fatherOf(frank, john)\}$

Answering queries over incomplete databases

Definition (certain answer)

- **certain(D)** = database derived from D by deleting tuples with nulls.
- $\mathbf{certain(Q, D)} = \bigcap \{ \mathbf{Q(R)} \mid \mathbf{R} \in \mathbf{POSS(D)} \}$

Answering queries over incomplete databases

Definition (certain answer)

- **certain(D)** = database derived from D by deleting tuples with nulls.
- $\mathbf{certain(Q, D)} = \bigcap \{ \mathbf{Q(R)} \mid \mathbf{R} \in \mathbf{POSS(D)} \}$

Theorem (weak representation systems)

For union of conjunctive queries

$$\mathbf{certain(Q(D))} = \mathbf{certain(Q, D)}$$

Certain answers can be computed by

1. Evaluating (naively) $Q(D)$
2. Removing tuples with nulls

Answering queries under constraints

Definition (Model)

Given a knowledge base $K = D \cup \Sigma$, M is a **model** of K if $M \models K$.

Definition (Homomorphism)

Mapping $h : \mathit{Nulls} \rightarrow \mathit{Nulls} \cup \mathit{Constants}$.

Answering queries under constraints

Definition (Model)

Given a knowledge base $K = D \cup \Sigma$, M is a **model** of K if $M \models K$.

Definition (Homomorphism)

Mapping $h : \mathit{Nulls} \rightarrow \mathit{Nulls} \cup \mathit{Constants}$.

Definition (Possible worlds under OWA)

$\mathbf{POSS(M)} = \{\ \mathbf{R} \mid \mathbf{h(M)} \subseteq \mathbf{R} \wedge \mathbf{R}\ \textbf{is ground}\ \}$.

Answering queries under constraints

Definition (Model)

Given a knowledge base $K = D \cup \Sigma$, M is a **model** of K if $M \models K$.

Definition (Homomorphism)

Mapping $h : \mathit{Nulls} \rightarrow \mathit{Nulls} \cup \mathit{Constants}$.

Definition (Possible worlds under OWA)

$\mathbf{POSS(M)} = \{\ \mathbf{R} \mid \mathbf{h(M)} \subseteq \mathbf{R} \wedge \mathbf{R}\ \textbf{is ground}\ \}$.

Definition (certain answer)

$\mathbf{certain(Q, (D, \Sigma))} = \bigcap \{\ \mathbf{Q(R)} \mid \mathbf{R} \in \mathbf{POSS(M)}\ \wedge \mathbf{M}\ \text{is a model of}\ \mathbf{D} \cup \Sigma\ \}$

Universal models

Definition (Models comparison)

Given two models M_1 and M_2 we say that M_1 is ***at least as general*** as M_2 ($M_1 \sqsupseteq M_2$) if $\exists h$ such that $h(M_1) \subseteq M_2$.
M_1 is ***more general*** than M_2 ($M_1 \sqsupset M_2$) if $M_1 \sqsupseteq M_2$ and $M_2 \not\sqsupseteq M_1$.

Universal models

Definition (Models comparison)

Given two models M_1 and M_2 we say that M_1 is ***at least as general*** as M_2 ($M_1 \sqsupseteq M_2$) if $\exists h$ such that $h(M_1) \subseteq M_2$.
M_1 is ***more general*** than M_2 ($M_1 \sqsupset M_2$) if $M_1 \sqsupseteq M_2$ and $M_2 \not\sqsupseteq M_1$.

Theorem

- $M_1 \sqsupseteq M_2 \quad \textit{iff} \quad POSS(M_1) \supseteq POSS(M_2)$,
- $M_1 \subseteq M_2 \quad \Rightarrow \quad M_1 \sqsupseteq M_2$.

Universal models

Definition (Universal model)

M is an **universal model** (or **universal solution**) if for every model N, $M \sqsupseteq N$ (i.e. $\exists h\ s.t.\ h(M) \subseteq N$).

Universal models

Definition (Universal model)

M is an **universal model** (or **universal solution**) if for every model N, $M \sqsupseteq N$ (i.e. $\exists h$ $s.t.$ $h(M) \subseteq N$).

Theorem (Main Th.)

For every UCQ Q and for every arbitrary universal model **M** *of* $D \cup \Sigma$

$$\mathbf{certain(Q, (D, \Sigma)) = certain(Q, M) = certain(Q(M))}$$

Recall that:

$$certain(Q,(D,\Sigma) = \bigcap\{\, Q(R) \mid R \in POSS(M) \wedge M \text{ is a model of } D \cup \Sigma \,\}$$

Universal models

Example (Models and answers)

- Database: $D = \{person(john)\}$
- Data dependencies Σ:

$$\forall X\ person(X) \rightarrow \exists Z\ fatherOf(Z, X)$$
$$\forall X\ \forall Y\ fatherOf(X, Y), person(Y) \rightarrow person(X)$$

- Models (under OWA):

$$M_1 = \{person(john), fatherOf(john, john)\}$$
$$M_2 = \{person(john), fatherOf(\perp_1, john), person(\perp_1)\}$$
$$M_3 = \{person(john), fatherOf(\perp_2, john), person(\perp_2)\}$$
$$M_4 = \{person(john), fatherOf(\perp_1, john), person(frank)\}$$
$$\ldots$$

Universal models

Example (Models and answers)

- Database: $D = \{person(john)\}$
- Data dependencies Σ:

$$\forall X\ person(X) \rightarrow \exists Z\ fatherOf(Z, X)$$
$$\forall X\ \forall Y\ fatherOf(X, Y), person(Y) \rightarrow person(X)$$

- Models (under OWA):

$$M_1 = \{person(john), fatherOf(john, john)\}$$
$$M_2 = \{person(john), fatherOf(\perp_1, john), person(\perp_1)\}$$
$$M_3 = \{person(john), fatherOf(\perp_2, john), person(\perp_2)\}$$
$$M_4 = \{person(john), fatherOf(\perp_1, john), person(frank)\}$$
$$\ldots$$

$M_2 \sqsupset M_1$, $M_2 \sqsupset M_4$, $M_2 \sqsupseteq M_3$, $M_3 \sqsupset M_1$, $M_3 \sqsupset M_4$, $M_3 \sqsupseteq M_2$

M_2 and M_3 are universal models.

The Chase

Fixpoint algorithm designed to enforce satisfaction of dependencies.

The execution of the chase involves

- adding new facts (possibly with null values) to satisfy TGDs,
- replacing nulls (with constants or other null values) to satisfy EGDs.

The Chase

Several problems can be solved using the chase algorithm:

- Checking query containment under dependencies
- Checking implication of dependencies
- Checking lossless decomposition of database schema
- Computing universal solutions in data exchange
- Computing certain answers in data integration
- Ontology Querying
- Database repair
- ...

The Chase

Chase algorithm *chase*(D, Σ)

Iteratively, let K be the current instance $(K = D$ at step 0),

- select nondeterministically a constraint $r \in \Sigma$ and an homomorphism h such that $K \not\models h(r)$ (i.e. $K \models body(h(r))$ and there is no estension h' of h such that $K \models head(h'(r))$).
- enforce the satisfaction of $h(r)$ by either i) adding a tuple (if r is a TGD), or ii) replacing a null value (if r is an EGD), or "fail" (if r is an EGD which cannot be enforced).

A chase step from K_1 and r_1 with homomorphism h to K_2 is denoted as $K_1 \xrightarrow{r_1, h_1} K_2$.

The result of $chase(D, \Sigma)$ is nondeterministic and is either

- a (possibly infinite) universal model;
- fail, if $D \cup \Sigma$ does not have universal models.

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X, Y)$
$S(a)$	$S(X) \wedge E(X, Y) \rightarrow N(Y)$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$S(X) \wedge E(X,Y) \rightarrow N(Y)$

$chase(D, \Sigma) = \{N(a), S(a)$

Chase - Enforcing data dependencies: a first example

Example

D : Σ :

$N(a)$ $\qquad N(X) \rightarrow \exists Y\ E(X, Y)$

$S(a)$ $\qquad S(X) \wedge E(X, Y) \rightarrow N(Y)$

$chase(D, \Sigma) = \{\ N(a),\ S(a)$

Chase - Enforcing data dependencies: a first example

Example

D : Σ :

$$N(a) \qquad N(X) \rightarrow \exists Y\ E(X, Y)$$
$$S(a) \qquad S(X) \wedge E(X, Y) \rightarrow N(Y)$$

$chase(D, \Sigma) = \{\ N(a),\ S(a),\ E(a, \perp_1)$

Chase - Enforcing data dependencies: a first example

Example

D : $\qquad\qquad$ Σ :

$$\begin{array}{ll} N(a) & N(X) \rightarrow \exists Y\ E(X,Y) \\ S(a) & S(X) \wedge E(X,Y) \rightarrow N(Y) \end{array}$$

$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1)$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X, Y)$
$S(a)$	$S(X) \wedge E(X, Y) \rightarrow N(Y)$

$$chase(D, \Sigma) = \{\ N(a),\ S(a),\ E(a, \perp_1),\ N(\perp_1)$$

Chase - Enforcing data dependencies: a first example

Example

$D:$	$\Sigma:$
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$S(X) \wedge E(X,Y) \rightarrow N(Y)$

$chase(D, \Sigma) = \{\ N(a),\ S(a),\ E(a, \perp_1),\ N(\perp_1)$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$S(X) \wedge E(X,Y) \rightarrow N(Y)$

$$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2)$$

Chase - Enforcing data dependencies: a first example

Example

$D:$	$\Sigma:$
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$S(X) \wedge E(X,Y) \rightarrow N(Y)$

$$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2)$$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X, Y)$
$S(a)$	$S(X) \wedge E(X, Y) \rightarrow N(Y)$

$$chase(D, \Sigma) = \{\ N(a),\ S(a),\ E(a, \perp_1),\ N(\perp_1),\ E(\perp_1, \perp_2)\ \}$$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y \; E(X, Y)$
$S(a)$	$S(X) \wedge E(X, Y) \rightarrow N(Y)$

$$chase(D, \Sigma) = \{ \; N(a), \; S(a), \; E(a, \perp_1), \; N(\perp_1), \; E(\perp_1, \perp_2) \; \}$$

All dependencies are satisfied: STOP.

This and every other chase sequence terminates.

Chase - Enforcing data dependencies: a first example

Example

$$
\begin{array}{lcl}
D: & & \Sigma: \\
N(a) & & N(X) \rightarrow \exists Y\, E(X,Y) \\
S(a) & & \cancel{S(X)} \wedge E(X,Y) \rightarrow N(Y)
\end{array}
$$

$$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2)$$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$\cancel{S(X)} \wedge E(X,Y) \rightarrow N(Y)$

$$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2)$$

Chase - Enforcing data dependencies: a first example

Example

$D:$	$\Sigma:$
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$\cancel{S(X)} \wedge E(X,Y) \rightarrow N(Y)$

$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2),\ N(\perp_2),$

Chase - Enforcing data dependencies: a first example

Example

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$
$S(a)$	$\cancel{S(X)} \wedge E(X,Y) \rightarrow N(Y)$

$$chase(D,\Sigma) = \{\ N(a),\ S(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2),\ N(\perp_2), \ldots\}$$

There is **no** terminating chase sequence.

Chase – Terminating Sequence

Example (∃ a finite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D :

$airport(a)$

Σ :

$$r_1 : airport(X) \rightarrow \exists Y\ flight(X, Y)$$
$$r_2 : flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3 : flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

Chase – Terminating Sequence

Example (∃ a finite sequence)

D :

$airport(a)$

Σ :

$$r_1 : \; airport(X) \rightarrow \exists Y \; flight(X, Y)$$
$$r_2 : \; flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3 : \; flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y \ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Terminating Sequence

Example (∃ a finite sequence)

D :

$airport(a)$

Σ :

r_1 : $airport(X) \rightarrow \exists Y\ flight(X, Y)$

r_2 : $flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$

r_3 : $flight(X, Y) \rightarrow flight(Y, X)$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D :

$airport(a)$

Σ :

$$\begin{aligned} r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\ r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\ r_3 &: flight(X, Y) \rightarrow flight(Y, X) \end{aligned}$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Terminating Sequence

Example (∃ a finite sequence)

D :

$airport(a)$

Σ :

r_1 : $airport(X) \rightarrow \exists Y\ flight(X, Y)$

r_2 : $flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$

r_3 : $flight(X, Y) \rightarrow flight(Y, X)$

The following facts are added to D :

$flight(a, \perp_1)$

$airport(\perp_1)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D : Σ :

$airport(a)$

$$
\begin{aligned}
r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D :

$airport(a)$

Σ :

$$r_1: \ airport(X) \rightarrow \exists Y \ flight(X, Y)$$
$$r_2: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3: \ flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D : $\qquad\qquad$ Σ :

$airport(a)$

$$
\begin{aligned}
r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$

$airport(\perp_1)$

Chase – Terminating Sequence

Example (∃ a finite sequence)

D :

$airport(a)$

Σ :

$$\begin{aligned} r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\ r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\ r_3 &: flight(X, Y) \rightarrow flight(Y, X) \end{aligned}$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, a)$

Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

D : $\qquad\qquad$ Σ :

$airport(a)$

$$
\begin{aligned}
r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, a)$

No further rule is applicable: STOP.

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y \ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

r_1 : $airport(X) \rightarrow \exists Y\ flight(X, Y)$

r_2 : $flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$

r_3 : $flight(X, Y) \rightarrow flight(Y, X)$

The following facts are added to D :

Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

D :

$airport(a)$

Σ :

$$r_1 : \ airport(X) \rightarrow \exists Y \ flight(X, Y)$$
$$r_2 : \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3 : \ flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D : | Σ :

$airport(a)$

$$r_1 : \ airport(X) \rightarrow \exists Y \ flight(X, Y)$$
$$r_2 : \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3 : \ flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y \ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

D :

$airport(a)$

Σ :

r_1 : $airport(X) \rightarrow \exists Y\ flight(X, Y)$
r_2 : $flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$
r_3 : $flight(X, Y) \rightarrow flight(Y, X)$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

r_1 : $airport(X) \rightarrow \exists Y\ flight(X, Y)$

r_2 : $flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$

r_3 : $flight(X, Y) \rightarrow flight(Y, X)$

The following facts are added to D :

$flight(a, \perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D : Σ :

$airport(a)$

$$r_1: \; airport(X) \rightarrow \exists Y \; flight(X,Y)$$
$$r_2: \; flight(X,Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3: \; flight(X,Y) \rightarrow flight(Y,X)$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y \ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \; airport(X) \rightarrow \exists Y \; flight(X, Y) \\
r_2 &: \; flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \; flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$

$airport(\perp_1)$

Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

D :

$airport(a)$

Σ :

$$r_1 : \ airport(X) \rightarrow \exists Y \ flight(X, Y)$$
$$r_2 : \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3 : \ flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$r_1: \ airport(X) \rightarrow \exists Y \ flight(X, Y)$$
$$r_2: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3: \ flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, \perp_2)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$\begin{aligned} r_1 &: \ airport(X) \rightarrow \exists Y\ flight(X, Y) \\ r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\ r_3 &: \ flight(X, Y) \rightarrow flight(Y, X) \end{aligned}$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, \perp_2)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: \ airport(X) \rightarrow \exists Y \ flight(X, Y) \\
r_2 &: \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: \ flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, \perp_2)$

Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

D :

$airport(a)$

Σ :

$$
\begin{aligned}
r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\
r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\
r_3 &: flight(X, Y) \rightarrow flight(Y, X)
\end{aligned}
$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, \perp_2)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D : Σ :

$airport(a)$

$$\begin{aligned} r_1 &: airport(X) \rightarrow \exists Y\ flight(X, Y) \\ r_2 &: flight(X, Y) \rightarrow airport(X) \wedge airport(Y) \\ r_3 &: flight(X, Y) \rightarrow flight(Y, X) \end{aligned}$$

The following facts are added to D :

$flight(a, \perp_1)$

$airport(\perp_1)$

$flight(\perp_1, \perp_2)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$r_1 : \ airport(X) \rightarrow \exists Y \ flight(X, Y)$

$r_2 : \ flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$

$r_3 : \ flight(X, Y) \rightarrow flight(Y, X)$

The following facts are added to D :

$flight(a, \perp_1)$

$airport(\perp_1)$

$flight(\perp_1, \perp_2)$

$airport(\perp_2)$

Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

D :

$airport(a)$

Σ :

$$r_1 : \; airport(X) \rightarrow \exists Y \; flight(X, Y)$$
$$r_2 : \; flight(X, Y) \rightarrow airport(X) \wedge airport(Y)$$
$$r_3 : \; flight(X, Y) \rightarrow flight(Y, X)$$

The following facts are added to D :

$flight(a, \perp_1)$
$airport(\perp_1)$
$flight(\perp_1, \perp_2)$
$airport(\perp_2)$
$\vdots$

By iteratively applying r_1 and r_2 **the chase never terminates.**

Chase Termination

- Checking whether there is **at least one** terminating chase sequence vs. **all** chase sequences are terminating;
- for **a given instance** D vs. **for every instance.**

Chase Termination

Theorem

Consider a set Σ of TGDs:

- *It is undecidable whether,* ***for every instance*** D, ***some*** *chase sequence of* D *with* Σ *terminates [GO13].*
- *It is undecidable whether,* ***for every instance*** D, ***all*** *chase sequences of* D *with* Σ *terminate [GM14].*

Chase Termination

Theorem ([DNR08])

Given a set Σ of TGDs and a (fixed) instance D:

- *It is undecidable whether some chase sequence of D with Σ terminates.*
- *It is undecidable whether all chase sequences of D with Σ terminate.*

Sufficient Conditions

One Solution: Identify sufficient conditions guaranteeing chase termination.

Sufficient Conditions

One Solution: Identify sufficient conditions guaranteeing chase termination.

Many have been proposed:

- Weak Acyclicity [FKMP05]
- Stratification [DNR08] and C-Stratification [MSL09]
- Safety and Inductive Restriction [MSL09]
- Super-weak Acyclicity [Mar09]
- Local Stratification [GST11, GST15]
- Adornment Techniques [GS10, GST15]
- Model-Faithful Acyclicity [GHK^{+}13]
- Acyclic Graph Rule Dependencies [BLMS11]

Sufficient Conditions

One Solution: Identify sufficient conditions guaranteeing chase termination.

Many have been proposed:

- Weak Acyclicity [FKMP05]
- Stratification [DNR08] and C-Stratification [MSL09]
- Safety and Inductive Restriction [MSL09]
- Super-weak Acyclicity [Mar09]
- Local Stratification [GST11, GST15]
- Adornment Techniques [GS10, GST15]
- Model-Faithful Acyclicity [GHK$^+$13]
- Acyclic Graph Rule Dependencies [BLMS11]

From now on we consider only TGDs

Chase Variants

Oblivious and **Semi-oblivious**

- The set of dependencies is *skolemized*.
- The resulting logic program is evaluated.
- The oblivious and semi-oblivious chases adopt two different skolemizations.

Example

$$r: \; N(X,Y) \rightarrow \exists K, Z \; E(X,K,Z)$$

- **Oblivious** Chase. Skolemization:

$$N(X,Y) \rightarrow E(X, f_r^K(X,Y), f_r^Z(X,Y))$$

- **Semi-oblivious** Chase. Skolemization:

$$N(X,Y) \rightarrow E(X, f_r^K(X), f_r^Z(X))$$

Chase Variants

Example (Complex terms represent nulls)

D :	Σ :
$E(a,b)$	$E(X,Y) \rightarrow \exists Z\ E(X,Z)$

Standard	**Semi-oblivious**	**Oblivious**

Chase Variants

Example (Complex terms represent nulls)

$$D: \quad E(a,b) \qquad \Sigma: \quad E(X,Y) \rightarrow \exists Z\ E(X,Z)$$

Standard	**Semi-oblivious**	**Oblivious**
No chase step ($D \models \Sigma$)		

Chase Variants

Example (Complex terms represent nulls)

$$D: \quad E(a,b) \qquad\qquad \Sigma: \quad E(X,Y) \to \exists Z\ E(X,Z)$$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \to E(X,f(X))$	
No chase step ($D \models \Sigma$)		

Chase Variants

Example (Complex terms represent nulls)

D :	Σ :
$E(a,b)$	$E(X,Y) \rightarrow \exists Z\ E(X,Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \rightarrow E(X,f(X))$	
No chase step ($D \models \Sigma$)	$E(a,b)$	

Chase Variants

Example (Complex terms represent nulls)

D :	Σ :
$E(a, b)$	$E(X, Y) \rightarrow \exists Z\ E(X, Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X, Y) \rightarrow E(X, f(X))$	
No chase step $(D \models \Sigma)$	$E(a, b)$ $E(a, f(a))$	

Chase Variants

Example (Complex terms represent nulls)

D : $E(a,b)$

Σ : $E(X,Y) \rightarrow \exists Z\ E(X,Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \rightarrow E(X,f(X))$	
No chase step ($D \models \Sigma$)	$E(a,b)$ $E(a,f(a))$	
	STOP (fixpoint)	

Chase Variants

Example (Complex terms represent nulls)

D : $E(a,b)$

Σ : $E(X,Y) \rightarrow \exists Z\ E(X,Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \rightarrow E(X,f(X))$	$E(X,Y) \rightarrow E(X,f(X,Y))$
No chase step ($D \models \Sigma$)	$E(a,b)$ $E(a,f(a))$	
	STOP (fixpoint)	

Chase Variants

Example (Complex terms represent nulls)

$$D: \quad E(a,b) \qquad\qquad \Sigma: \quad E(X,Y) \rightarrow \exists Z\ E(X,Z)$$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \rightarrow E(X,f(X))$	$E(X,Y) \rightarrow E(X,f(X,Y))$
No chase step ($D \models \Sigma$)	$E(a,b)$ $E(a,f(a))$	$E(a,b)$
	STOP (fixpoint)	

Chase Variants

Example (Complex terms represent nulls)

D :	Σ :
$E(a, b)$	$E(X, Y) \rightarrow \exists Z\ E(X, Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X, Y) \rightarrow E(X, f(X))$	$E(X, Y) \rightarrow E(X, f(X, Y))$
No chase step ($D \models \Sigma$)	$E(a, b)$ $E(a, f(a))$	$E(a, b)$ $E(a, f(a, b))$
	STOP (fixpoint)	

Chase Variants

Example (Complex terms represent nulls)

D :	Σ :
$E(a,b)$	$E(X,Y) \rightarrow \exists Z\ E(X,Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \rightarrow E(X,f(X))$	$E(X,Y) \rightarrow E(X,f(X,Y))$
No chase step ($D \models \Sigma$)	$E(a,b)$ $E(a,f(a))$	$E(a,b)$ $E(a,f(a,b))$ $E(a,f(a,f(a,b)))$
	STOP (fixpoint)	

Chase Variants

Example (Complex terms represent nulls)

$$D: \quad E(a,b) \qquad \Sigma: \quad E(X,Y) \rightarrow \exists Z\; E(X,Z)$$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X,Y) \rightarrow E(X,f(X))$	$E(X,Y) \rightarrow E(X,f(X,Y))$
No chase step $(D \models \Sigma)$	$E(a,b)$ $E(a,f(a))$ STOP (fixpoint)	$E(a,b)$ $E(a,f(a,b))$ $E(a,f(a,f(a,b)))$ $E(a,f(a,f(a,f(a,b))))$

Chase Variants

Example (Complex terms represent nulls)

D : $E(a, b)$

Σ : $E(X, Y) \rightarrow \exists Z\, E(X, Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X, Y) \rightarrow E(X, f(X))$	$E(X, Y) \rightarrow E(X, f(X, Y))$
No chase step ($D \models \Sigma$)	$E(a, b)$ $E(a, f(a))$ STOP (fixpoint)	$E(a, b)$ $E(a, f(a, b))$ $E(a, f(a, f(a, b)))$ $E(a, f(a, f(a, f(a, b))))$ $\vdots$ NO Termination (no fixpoint)

Chase Variants

Example (Complex terms represent nulls)

D : $E(a, b)$

Σ : $E(X, Y) \rightarrow \exists Z\ E(X, Z)$

Standard	**Semi-oblivious**	**Oblivious**
	$E(X, Y) \rightarrow E(X, f(X))$	$E(X, Y) \rightarrow E(X, f(X, Y))$
No chase step ($D \models \Sigma$)	$E(a, b)$ $E(a, \perp_1)$	$E(a, b)$ $E(a, \perp_2)$ $E(a, \perp_3)$ $E(a, \perp_4)$ $\vdots$
	STOP (fixpoint)	NO Termination (no fixpoint)

Chase Variants

Core Chase [DNR08]
Minimal universal models.

Core chase step:

1. Enforce all dependencies "in parallel".
2. "Retract" the result (homomorphism $h : M \rightarrow M$).

Theorem (Completeness of the Core Chase [DNR08])

If D is an instance and Σ is a set of TGDs. then there exists a universal model for Σ and I iff the core chase of I with Σ terminates and yields such a model.

Chase Variants

- $CT^{c}_{\forall}$: class of sets of TGDs Σ s.t., for every instance, **all** c-chase sequences terminate.
- $CT^{c}_{\exists}$: class of of sets of TGDs Σ s.t., for every instance, **at least one** c-chase sequence terminates.

Theorem ([Mei10, One13] For TGDs only)

$$CT^{obl}_{\forall} = CT^{obl}_{\exists} \subsetneq CT^{sobl}_{\forall} = CT^{sobl}_{\exists} \subsetneq CT^{std}_{\forall} \subsetneq CT^{std}_{\exists} \subsetneq CT^{core}_{\forall} = CT^{core}_{\exists}$$

Function Symbols vs. TGDs

Termination Criteria for programs with function symbols can be applied to TGDs:

Step 1. Skolemize TGDs.

Example

$$\begin{array}{rl} r: & p(X,Y) \rightarrow \exists K, Z\; q(X,K,Z) \\ sk(r): & p(X,Y) \rightarrow q(X, f_r^K(X), f_r^Z(X)) \end{array}$$

Step 2. Apply termination criteria to skolemized TGDs.

Given a set Σ of TGDs, let $sk(\Sigma) = \{sk(r) \mid r \in \Sigma\}$.

Termination of the bottom-up evaluation of $sk(\Sigma)$ (i.e., the semi-oblivious chase) $\Rightarrow$ Termination of the chase of Σ [One13].

Function Symbols vs. TGDs

Example

$$r: \quad p(X,Y) \rightarrow \exists Z\, p(X,Z)$$

Function Symbols vs. TGDs

Example

$$r: \quad p(X, Y) \rightarrow \exists Z p(X, Z)$$

Step 1. Skolemize r:

$$sk(r): \quad p(X, Y) \rightarrow p(X, f_r^Z(X))$$

We get a logic program with function symbols.

Function Symbols vs. TGDs

Example

$$r: \quad p(X, Y) \rightarrow \exists Z p(X, Z)$$

Step 1. Skolemize r:

$$sk(r): \quad p(X, Y) \rightarrow p(X, f_r^Z(X))$$

We get a logic program with function symbols.

Step 2. Analyze $sk(r)$ by applying a termination criterion.

Function Symbols vs. TGDs

Example

$$r: \quad p(X, Y) \rightarrow \exists Z p(X, Z)$$

Step 1. Skolemize r:

$$sk(r): \quad p(X, Y) \rightarrow p(X, f_r^Z(X))$$

We get a logic program with function symbols.

Step 2. Analyze $sk(r)$ by applying a termination criterion.

- E.g., $sk(r)$ is argument-restricted.

Function Symbols vs. TGDs

Example

$$r: \quad p(X, Y) \rightarrow \exists Z p(X, Z)$$

Step 1. Skolemize r:

$$sk(r): \quad p(X, Y) \rightarrow p(X, f_r^Z(X))$$

We get a logic program with function symbols.

Step 2. Analyze $sk(r)$ by applying a termination criterion.

- E.g., $sk(r)$ is argument-restricted.
- Thus, the bottom-up evaluation of $sk(r)$ always terminates.

Function Symbols vs. TGDs

Example

$$r : \quad p(X, Y) \rightarrow \exists Z p(X, Z)$$

Step 1. Skolemize r:

$$sk(r) : \quad p(X, Y) \rightarrow p(X, f_r^Z(X))$$

We get a logic program with function symbols.

Step 2. Analyze $sk(r)$ by applying a termination criterion.

- E.g., $sk(r)$ is argument-restricted.
- Thus, the bottom-up evaluation of $sk(r)$ always terminates.
- That is, the semi-oblivious chase of r always terminates.

Function Symbols vs. TGDs

Example

$$r: \quad p(X,Y) \rightarrow \exists Z p(X,Z)$$

Step 1. Skolemize r:

$$sk(r): \quad p(X,Y) \rightarrow p(X, f_r^Z(X))$$

We get a logic program with function symbols.

Step 2. Analyze $sk(r)$ by applying a termination criterion.

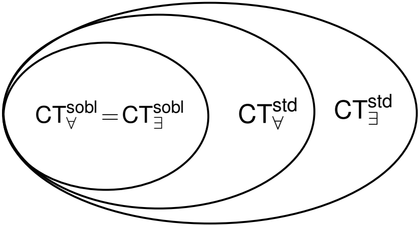
- E.g., $sk(r)$ is argument-restricted.
- Thus, the bottom-up evaluation of $sk(r)$ always terminates.
- That is, the semi-oblivious chase of r always terminates.
- Thus, the standard chase of r always terminates.

Function Symbols vs. TGDs

Limitations: Recall that:

Theorem ([Mei10, One13])

$$\mathrm{CT}_{\forall}^{\mathrm{sobl}} = \mathrm{CT}_{\exists}^{\mathrm{sobl}} \subsetneq \mathrm{CT}_{\forall}^{\mathrm{std}} \subsetneq \mathrm{CT}_{\exists}^{\mathrm{std}}$$

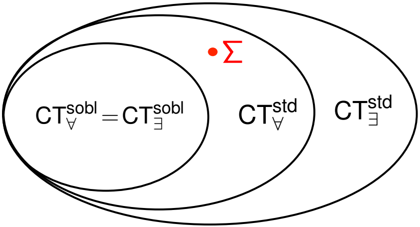


Function Symbols vs. TGDs

Limitations: Recall that:

Theorem ([Mei10, One13])

$$\mathrm{CT}^{\mathrm{sobl}}_{\forall} = \mathrm{CT}^{\mathrm{sobl}}_{\exists} \subsetneq \mathrm{CT}^{\mathrm{std}}_{\forall} \subsetneq \mathrm{CT}^{\mathrm{std}}_{\exists}$$



Function Symbols vs. TGDs

What about applying criteria for TGDs to programs with function symbols?

Function Symbols vs. TGDs

What about applying criteria for TGDs to programs with function symbols?

The latter are more general than skolemized TGDs.

Each function symbol occurs:

Skolemized TGDs	**Programs with function symbols**
once	arbitrary number of times
only in the head	in the body and/or head
no nesting	arbitrary nesting

Termination Criteria

Weak Acyclicity [FKMP05] [standard chase]

Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. normal edges represent the propagation of values between arguments;
  2. special edges $\xrightarrow{*}$ represent the generation of nulls.

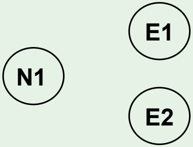
Weak Acyclicity [FKMP05] [standard chase]

Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. normal edges represent the propagation of values between arguments;
  2. special edges $\xrightarrow{*}$ represent the generation of nulls.

Example

$$\Sigma = \begin{array}{l} N(X) \rightarrow \exists Y\ E(X, Y) \\ E(X, Y) \rightarrow N(Y) \end{array}$$



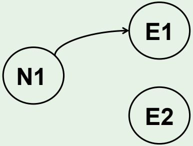
Weak Acyclicity [FKMP05] [standard chase]

Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. normal edges represent the propagation of values between arguments;
  2. special edges $\xrightarrow{*}$ represent the generation of nulls.

Example

$$\Sigma = \begin{array}{l} \boldsymbol{N}(\boldsymbol{X}) \rightarrow \exists Y\ \boldsymbol{E}(\boldsymbol{X}, Y) \\ E(X, Y) \rightarrow N(Y) \end{array}$$



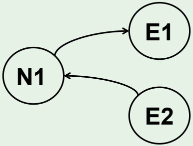
Weak Acyclicity [FKMP05] [standard chase]

Dependency Graph $dep(\Sigma)$

- Nodes are predicate arguments.
- Two kinds of edges:
  1. normal edges represent the propagation of values between arguments;
  2. special edges $\xrightarrow{*}$ represent the generation of nulls.

Example

$$\Sigma = \begin{array}{l} N(X) \rightarrow \exists Y\ E(X,Y) \\ \boldsymbol{E}(X,\boldsymbol{Y}) \rightarrow \boldsymbol{N}(\boldsymbol{Y}) \end{array}$$



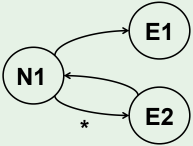
Weak Acyclicity [FKMP05] [standard chase]

Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. normal edges represent the propagation of values between arguments;
  2. special edges $\xrightarrow{*}$ represent the generation of nulls.

Example

$$\Sigma = \begin{array}{l} \boldsymbol{N}(\boldsymbol{X}) \rightarrow \exists Y\ \boldsymbol{E}(X, \boldsymbol{Y}) \\ E(X, Y) \rightarrow N(Y) \end{array}$$



Weak Acyclicity [FKMP05] [standard chase]

Definition

A set of dependencies is **weakly acyclic** if **there is no cycle going through a special edge** in the dependency graph.

Weak Acyclicity [FKMP05] [standard chase]

Definition

A set of dependencies is **weakly acyclic** if **there is no cycle going through a special edge** in the dependency graph.

Theorem

*If Σ is weakly acyclic, then **for every instance** I, **every chase sequence terminates** (and has a polynomial length in the size of I).*

Safety [MSL09] [standard chase]

Affected Positions *aff*(Σ) [CGK13]

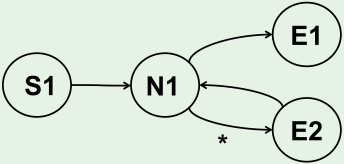
Overestimation of positions that may contain null values.

Propagation Graph *prop*(Σ)

Restriction of the dependency graph containing only affected positions.

Example

$$\Sigma = \begin{array}{l} r_1 : N(X) \to \exists \mathbf{Y}\ E(X, \mathbf{Y}) \\ r_2 : S(\mathbf{Y}) \wedge E(X, \mathbf{Y}) \to N(\mathbf{Y}) \end{array}$$



Safety [MSL09] [standard chase]

Affected Positions $aff(\Sigma)$ [CGK13]

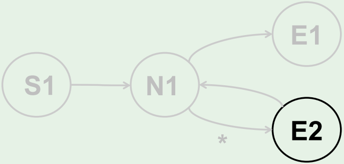
Overestimation of positions that may contain null values.

Propagation Graph $prop(\Sigma)$

Restriction of the dependency graph containing only affected positions.

Example

$$\Sigma = \begin{array}{l} r_1 : N(X) \rightarrow \exists \mathbf{Y}\ E(X, \mathbf{Y}) \\ r_2 : S(\mathbf{Y}) \wedge E(X, \mathbf{Y}) \rightarrow N(\mathbf{Y}) \end{array}$$



Affected Positions $aff(\Sigma)$

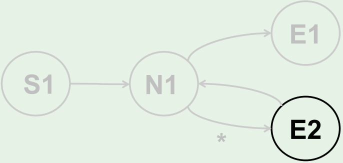
Overestimation of positions that may contain null values.

Propagation Graph $prop(\Sigma)$

Restriction of the dependency graph containing only affected positions.

Example

$$\Sigma = \begin{array}{l} r_1 : N(X) \rightarrow \exists \mathbf{Y}\ E(X, \mathbf{Y}) \\ r_2 : S(\mathbf{Y}) \wedge E(X, \mathbf{Y}) \rightarrow N(\mathbf{Y}) \end{array}$$



- $aff(\Sigma) = \{E_2\}$
- $prop(\Sigma) = (\{E_2\}, \emptyset)$

Safety [MSL09] [standard chase]

Affected Positions $aff(\Sigma)$

Overestimation of positions that may contain null values.

Propagation Graph $prop(\Sigma)$

Restriction of dependency graph containing only affected positions.

Safety

A set of dependencies is **safe** if **the propagation graph does not contain cycles with special edges.**

Safety [MSL09] [standard chase]

Affected Positions $aff(\Sigma)$

Overestimation of positions that may contain null values.

Propagation Graph $prop(\Sigma)$

Restriction of dependency graph containing only affected positions.

Safety

A set of dependencies is **safe** if **the propagation graph does not contain cycles with special edges.**

Theorem

If Σ is safe, then **for every instance I, every chase sequence terminates** (and has a polynomial length in the size of I).

Stratification [DNR08] [standard chase]

Chase Graph $G(\Sigma)$

- It represents how dependencies fire each other.
- Nodes: the dependencies in Σ.
- Edges: there is an edge from r_1 to r_2 ($r_1 \prec r_2$) if r_1 may "fire" r_2.

Stratification [DNR08] [standard chase]

Chase Graph $G(\Sigma)$

- It represents how dependencies fire each other.
- Nodes: the dependencies in Σ.
- Edges: there is an edge from r_1 to r_2 ($r_1 \prec r_2$) if r_1 may "fire" r_2.

Definition (Chase Graph $G(\Sigma)$)

$r_1 \prec r_2$ if $\exists$ *instance* K_1 and homomorphisms h_1 and h_2 such that

1) $K_1 \xrightarrow{r_1, h_1} K_2$ (chase step - $K_1 \not\models h_1(r_1)$),

2) $K_2 \not\models h_2(r_2)$,

3) $K_1 \models h_2(r_2)$.

Stratification [DNR08] [standard chase]

Chase Graph $G(\Sigma)$

- It represents how dependencies fire each other.
- Nodes: the dependencies in Σ.
- Edges: there is an edge from r_1 to r_2 ($r_1 \prec r_2$) if r_1 may "fire" r_2.

Example

$$\Sigma = \begin{array}{l} r_1 : N(X) \rightarrow \exists Y \; E(X,Y) \\ r_2 : S(Y) \wedge E(X,Y) \rightarrow N(Y) \end{array}$$

- *there exists a scenario where firing r_2 causes r_1 to fire ($r_2 \prec r_1$).*
- $r_1 \not\prec r_2$, $r_1 \not\prec r_1$ *and* $r_2 \not\prec r_2$.
- *The chase graph is acyclic and Σ is stratified.*

Stratification [DNR08] [standard chase]

Stratification

A set of dependencies is *stratified* if **every cycle in the chase graph** $G(\Sigma)$ **is weakly acyclic.**

Stratification [DNR08] [standard chase]

Stratification

A set of dependencies is *stratified* if **every cycle in the chase graph** $G(\Sigma)$ **is weakly acyclic.**

Theorem

If Σ *is stratified then, for every instance* I*,* ***there exists at least one chase sequence that terminates*** *(and whose length is polynomial in the size of* I*).*

C-Stratification [MSL09] vs Stratification

- A variation called *c-stratification* guarantees the termination of **every** chase sequence.
- Same approach of stratification, but the oblivious chase is used.
- C-Stratification

  $r_1 \prec_c r_2$ if:

  1) $K_1 \xrightarrow{*, r_1, h_1} K_2$ (oblivious step),

  2) $K_2 \not\models h_2(r_2)$,

  3) $K_1 \models h_2(r_2)$.

- Stratification

  $r_1 \prec r_2$ if:

  1) $K_1 \xrightarrow{r_1, h_1} K_2$ (standard step),

  2) $K_2 \not\models h_2(r_2)$,

  3) $K_1 \models h_2(r_2)$.

C-Stratification [MSL09] vs Stratification

- A variation called *c-stratification* guarantees the termination of **every** chase sequence.
- Same approach of stratification, but the oblivious chase is used.
- C-Stratification

  $r_1 \prec_c r_2$ if:

  1) $K_1 \xrightarrow{*,r_1,h_1} K_2$ (oblivious step),

  2) $K_2 \not\models h_2(r_2)$,

  3) $K_1 \models h_2(r_2)$.

- Stratification

  $r_1 \prec r_2$ if:

  1) $K_1 \xrightarrow{r_1,h_1} K_2$ (standard step),

  2) $K_2 \not\models h_2(r_2)$,

  3) $K_1 \models h_2(r_2)$.

Theorem

*If Σ is c-stratified then, for every instance I, **all chase sequences terminate** and their length is polynomial in the size of I).*

For any Σ, $G(\Sigma) \subseteq G_c(\Sigma) \quad \Rightarrow \quad \mathcal{S}tr \supseteq \mathcal{CS}tr$

Inductive Restriction [MSL09] [oblivious chase]

- It improves the firing relation by considering possible propagation of null values.
- It tests safety on the (nontrivial) strongly connected components of the graph.
- It generalizes both safety and c-stratification.

Theorem

If Σ is inductively restricted, then **for every instance** I, **every chase sequence terminates** (and has a polynomial length in the size of I).

Super-weak Acyclicity [Mar09] [semi-obliv. chase]

- Builds a *trigger graph* whose edges define relations among dependencies. An edge $r_i \rightsquigarrow r_j$ means that a null value introduced by a dependency r_i is propagated (directly or indirectly) into the body of r_j.
- Different nulls in positions for the same variable $\Rightarrow$ dependencies are not fired

Super-weak Acyclicity [Mar09] [semi-obliv. chase]

- Builds a *trigger graph* whose edges define relations among dependencies. An edge $r_i \rightsquigarrow r_j$ means that a null value introduced by a dependency r_i is propagated (directly or indirectly) into the body of r_j.
- Different nulls in positions for the same variable $\Rightarrow$ dependencies are not fired

Example

$$
\begin{aligned}
r_1 &: N(X) \rightarrow \exists Y, Z\ E(X, Y, Z) \\
r_2 &: E(X, Y, Z) \rightarrow G(X, Y, Z) \\
r_3 &: G(X, Y, Y) \rightarrow N(Y)
\end{aligned}
$$

Σ neither safe not stratified.

Super-weak Acyclicity [Mar09] [semi-obliv. chase]

- Builds a *trigger graph* whose edges define relations among dependencies. An edge $r_i \rightsquigarrow r_j$ means that a null value introduced by a dependency r_i is propagated (directly or indirectly) into the body of r_j.
- Different nulls in positions for the same variable $\Rightarrow$ dependencies are not fired

Example

$$
\begin{aligned}
r_1 &: N(X) \rightarrow \exists Y, Z\; E(X, Y, Z) \\
r_2 &: E(X, Y, Z) \rightarrow G(X, Y, Z) \\
r_3 &: G(X, Y, Y) \rightarrow N(Y)
\end{aligned}
$$

Σ neither safe not stratified.

$$
P(\Sigma) = \begin{cases}
r'_1 : N(X) \rightarrow E(X, f_Y^{r_1}(X), f_Z^{r_1}(X)) \\
r'_2 : E(X, Y, Z) \rightarrow \exists Y, Z\; G(X, Y, Z) \\
r'_3 : G(X, Y, Y) \rightarrow N(Y)
\end{cases}
$$

Super-weak Acyclicity [Mar09]

Super-weak Acyclicity

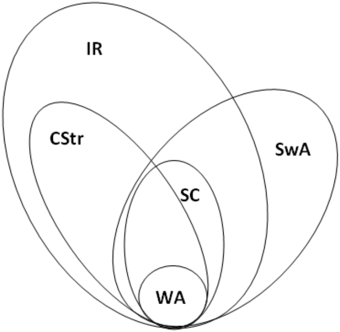
A set of dependencies is *super-weak acyclic* if **the trigger relation is acyclic.**

Theorem

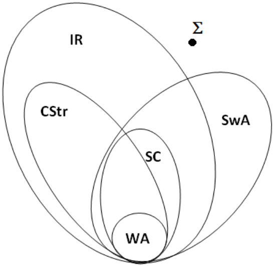
If Σ is super-weak acyclic, then **for every instance** I, **every chase sequence terminates** (and has a polynomial length in the size of I).

Relative Expressivity

- $\mathcal{WA}$: Weak Acylicity
- $\mathcal{SC}$: Safety
- $\mathcal{CS}tr$: C-stratification
- $\mathcal{IR}$: Inductive Restriction
- $\mathcal{S}w\mathcal{A}$: Super-weak Acylicity



Limitations



Example

$$
\begin{aligned}
r_1 &: \; N(X) \rightarrow \exists\, Y \, \exists Z \; E(X, Y) \wedge S(Z, Y) \\
r_2 &: \; E(X, Y) \wedge S(X, Y) \rightarrow N(Y) \\
r_3 &: \; E(X, Y) \rightarrow E(Y, X)
\end{aligned}
$$

Improvements of (C-)Stratification

- Builds a *firing graph* $\Gamma(\Sigma) = (\Sigma, E)$ representing how constraints fire each other.
- $(r_1, r_2) \in E$ if $r_1 < r_2$ (firing r_1 can cause r_2 to fire)
- $r_1 < r_2$ if:

  1) $K_1 \stackrel{r_1, h_1}{\rightarrow} K_2$,

  2) $K_2 \cup S \not\models h_2(r_2)$,

  3) $K_1 \cup S \models h_2(r_2)$ and

  4) $Null(S) \cap (Null(K_2) - Null(K_1)) = \emptyset$.

- $r_1 \prec r_2$ if:

  1) $K_1 \stackrel{r_1, h_1}{\rightarrow} K_2$,

  2) $K_2 \not\models h_2(r_2)$,

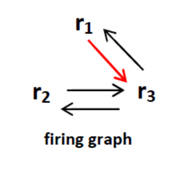
  3) $K_1 \models h_2(r_2)$.

As r_1 could cause the firing of r_2 not immediately, S is a set of atoms which could have been inferred after the firing of r_1.

Improvements of (C-)Stratification

Example

$$\Sigma = \begin{array}{ll} r_1 : & R(x) \rightarrow \exists y\, T(x,y) \\ r_2 : & R(x) \rightarrow T(x,x) \\ r_3 : & T(x,y) \wedge T(x,x) \rightarrow R(y) \end{array}$$

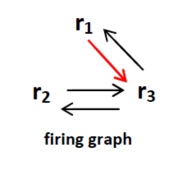


- $K_1 = \{R(a)\}$ and $K_2 = \{R(a), T(a, \perp_1)\}$
- $S = \{T(a,a)\}$
- $r_3 : \; T(a, \perp_1) \wedge T(a,a) \rightarrow R(\perp_1)$
- r_3 is fired by r_1, then we have $r_1 < r_3$

Improvements of (C-)Stratification

Example

$$\Sigma = \begin{array}{ll} r_1: & R(x) \rightarrow \exists y\, T(x,y) \\ r_2: & R(x) \rightarrow T(x,x) \\ r_3: & T(x,y) \wedge T(x,x) \rightarrow R(y) \end{array}$$

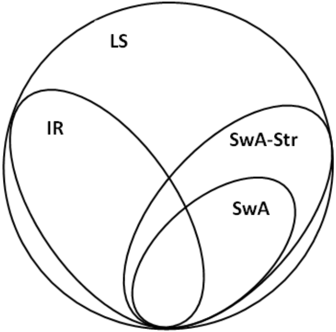


- $K_1 = \{R(a)\}$ and $K_2 = \{R(a), T(a, \perp_1)\}$
- $S = \{T(a,a)\}$
- $r_3: \; T(a, \perp_1) \wedge T(a,a) \rightarrow R(\perp_1)$
- r_3 is fired by r_1, then we have $r_1 < r_3$

Local Stratification

- *WA-Str* (resp. *SC-Str*, *SwA-Str*) tests *WA* (resp. *SC*, *SwA*) over components of $\Gamma(\Sigma)$
- *Local Stratification* (*LC*) combines *SwA* with $\Gamma(\Sigma)$: in analyzing how nulls may be propagated from a rule r_i to a rule r_j, also checks whether $r_i < r_j$ transitively.

Criteria Relationships



Rewriting Techniques

Constraints Rewriting Technique [GST15]

Idea

- Rewrite Σ into an 'equivalent' adorned set Σ^α and verify the structural properties for chase termination on Σ^α (similarly to LPs)
- Rewrite Σ into a set of dependencies useful to analyze the structure of terms during the execution.

Rewriting Algorithm [GST15]

Example

Σ :

$$
\begin{array}{l}
r_1 : \; N(X) \rightarrow \exists Y \; E(X,Y) \\
r_2 : \; S(X) \wedge E(X,Y) \rightarrow N(Y)
\end{array}
$$

$Adn(\Sigma)$:

$$
\begin{array}{lcll}
s_1 : \; N(X) & \rightarrow & N^b(X) & \\
s_2 : \; S(X) & \rightarrow & S^b(X) & \\
s_3 : \; E(X,Y) & \rightarrow & E^{bb}(X,Y) & \\
r_1' : \; N^b(X) & \rightarrow & \exists Y \; E^{b\mathbf{f_1}}(X,Y) & \mathbf{f_1} = f_{r_1}^Y(b) \\
r_2' : \; S^b(X) \wedge E^{bb}(X,Y) & \rightarrow & N^b(Y) & \\
r_2'' : \; S^b(X) \wedge E^{bf_1}(X,Y) & \rightarrow & N^{f_1}(Y) & \\
r_1'' : \; N^{f_1}(X) & \rightarrow & \exists Y \; E^{f_1\mathbf{f_2}}(X,Y) & \mathbf{f_2} = f_{r_1}^Y(f_1)
\end{array}
$$

Model-Faithful Acyclicity (MFA) [GHK+13]

Example

$$\Sigma:$$

$$r: \quad A(X) \rightarrow \exists Y, Z\ R(X,Y) \wedge B(Z)$$

Model-Faithful Acyclicity (MFA) [GHK+13]

Example

$$\Sigma :$$

$$r : \quad A(X) \rightarrow \exists Y, Z\ R(X, Y) \wedge B(Z)$$

$$MFA(\Sigma) :$$

$$A(X) \rightarrow \exists Y, Z\ R(X, Y) \wedge B(Z)$$

Model-Faithful Acyclicity (MFA) [GHK+13]

Example

$$\Sigma :$$

$$r : \quad A(X) \rightarrow \exists Y, Z \; R(X, Y) \wedge B(Z)$$

$$MFA(\Sigma) :$$

$$\begin{aligned} A(X) \rightarrow \exists Y, Z \; & R(X, Y) \wedge B(Z) \wedge \\ & F_r^Y(Y) \wedge F_r^Z(Z) \wedge S(X, Y) \wedge S(X, Z) \end{aligned}$$

Model-Faithful Acyclicity (MFA) [GHK+13]

Example

$$\Sigma :$$

$$r : \quad A(X) \to \exists Y, Z\ R(X, Y) \wedge B(Z)$$

$$MFA(\Sigma) :$$

$$\begin{array}{rcl} & & A(X) \to \exists Y, Z\ R(X, Y) \wedge B(Z) \wedge \\ & & F_r^Y(Y) \wedge F_r^Z(Z) \wedge S(X, Y) \wedge S(X, Z) \\ S(X_1, X_2) & \to & D(X_1, X_2) \\ D(X_1, X_2) \wedge S(X_2, X_3) & \to & D(X_1, X_3) \end{array}$$

Model-Faithful Acyclicity (MFA) [GHK+13]

Example

$$\Sigma :$$

$$r : \quad A(X) \rightarrow \exists Y, Z \; R(X, Y) \wedge B(Z)$$

$$MFA(\Sigma) :$$

$$\begin{aligned}
A(X) &\rightarrow \exists Y, Z \; R(X, Y) \wedge B(Z) \wedge \\
&\quad F_r^Y(Y) \wedge F_r^Z(Z) \wedge S(X, Y) \wedge S(X, Z) \\
S(X_1, X_2) &\rightarrow D(X_1, X_2) \\
D(X_1, X_2) \wedge S(X_2, X_3) &\rightarrow D(X_1, X_3) \\
F_r^Y(X_1) \wedge D(X_1, X_2) \wedge F_r^Y(X_2) &\rightarrow C \\
F_r^Z(X_1) \wedge D(X_1, X_2) \wedge F_r^Z(X_2) &\rightarrow C
\end{aligned}$$

Model-Faithful Acyclicity (MFA) [GHK+13]

Example

$$\Sigma :$$

$$r : \quad A(X) \rightarrow \exists Y, Z\ R(X, Y) \wedge B(Z)$$

$$MFA(\Sigma) :$$

$$\begin{aligned}
& A(X) \rightarrow \exists Y, Z\ R(X, Y) \wedge B(Z) \wedge \\
& \qquad F_r^Y(Y) \wedge F_r^Z(Z) \wedge S(X, Y) \wedge S(X, Z) \\
S(X_1, X_2) & \rightarrow D(X_1, X_2) \\
D(X_1, X_2) \wedge S(X_2, X_3) & \rightarrow D(X_1, X_3) \\
F_r^Y(X_1) \wedge D(X_1, X_2) \wedge F_r^Y(X_2) & \rightarrow C \\
F_r^Z(X_1) \wedge D(X_1, X_2) \wedge F_r^Z(X_2) & \rightarrow C
\end{aligned}$$

If $I \cup MFA(\Sigma) \models C$ then a cyclic term is derived during the semi-oblivious chase execution of I and Σ.

Model-Faithful Acyclicity (MFA) [GHK+13]

Definition

Σ is MFA w.r.t. an instance I if $I \cup MFA(\Sigma) \not\models C$.

Definition

The *critical instance* I_Σ for Σ is the instance containing all facts that can be built using:

- all predicates in Σ,
- all constants in the body of a dependency in Σ, and
- one special fresh constant $*$.

Theorem ([Mar09])

The semi-oblivious chase of Σ and I terminates for every I iff the semi-oblivious chase of Σ and I_Σ terminates.

Theorem

If Σ is MFA w.r.t. I_Σ, then **for every instance** I, **every (semi-oblivious) chase sequence terminates**.

Related Approaches

So far we have discussed **sufficient conditions** ensuring chase termination.
Other lines of research:

- Identify restricted classes of dependencies for which the termination problem is decidable [CGP15].
- Identify restricted classes of dependencies guaranteeing decidability of query answering (even if the chase does not terminate).
  - Guarded and Weakly Guarded Datalog$^{\pm}$ [CGK13]
  - Sticky Datalog$^{\pm}$ [CGP10]
  - Forward and Backward chaining [BLMS11]

Adding EGDs

EGDs – Syntax

An Equality-Generating Dependency is of the form:

$$\forall \overline{X}\ \varphi(\overline{X}) \rightarrow X_1 = X_2$$

where $\varphi(\overline{X})$ is a conjunction of atoms and $X_1, X_2 \in \overline{X}$.

Example

$\forall M_1, M_2, P\ directs(M_1, P) \wedge directs(M_2, P) \rightarrow M_1 = M_2$

EGDs and Chase Termination

1. In some cases the presence of EGDs allows us to have a terminating c-chase sequence when the set consisting only of the TGDs does not have one;
2. In some cases in the presence of EGDs there is no terminating c-chase sequence, but the set consisting only of the TGDs does have one.

Chase and EGDs

Adding EGDs leads to termination

Example (No EGDs)

D : $N(a)$

Σ :

$$N(X) \to \exists Y\ E(X, Y)$$
$$E(X, Y) \to N(Y)$$

Chase and EGDs

Adding EGDs leads to termination

Example (No EGDs)

$$D: \qquad\qquad \Sigma:$$

$$N(a) \qquad \begin{array}{l} N(X) \to \exists Y\ E(X,Y) \\ E(X,Y) \to N(Y) \end{array}$$

$$chase(D,\Sigma) = \{N(a),\ E(a,\perp_1),\ N(\perp_1),\ E(\perp_1,\perp_2),\ \ldots$$

There is no terminating chase sequence.

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :

$N(a)$

Σ :

$$N(X) \rightarrow \exists Y \; E(X,Y)$$
$$E(X,Y) \rightarrow N(Y)$$
$$\boldsymbol{E(X,Y) \rightarrow X = Y}$$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs **+ EGDs**)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X, Y)$
	$E(X, Y) \rightarrow N(Y)$
	$E(X, Y) \rightarrow X = Y$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X, Y)$ $E(X, Y) \rightarrow N(Y)$ $E(X, Y) \rightarrow X = Y$

$chase(D, \Sigma) = \{N(a),$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :

$N(a)$

Σ :

$N(X) \rightarrow \exists Y\ E(X, Y)$
$E(X, Y) \rightarrow N(Y)$
$E(X, Y) \rightarrow X = Y$

$chase(D, \Sigma) = \{N(a),$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

$D:$

$N(a)$

$\Sigma:$

$N(X) \rightarrow \exists Y\ E(X,Y)$
$E(X,Y) \rightarrow N(Y)$
$E(X,Y) \rightarrow X = Y$

$chase(D,\Sigma) = \{N(a), E(a,\perp_1),$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X, Y)$
	$E(X, Y) \rightarrow N(Y)$
	$E(X, Y) \rightarrow X = Y$

$chase(D, \Sigma) = \{N(a),\ E(a, \perp_1),$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :

$N(a)$

Σ :

$N(X) \rightarrow \exists Y\ E(X, Y)$
$E(X, Y) \rightarrow N(Y)$
$E(X, Y) \rightarrow X = Y$

$chase(D, \Sigma) = \{N(a), E(a, \perp_1),$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + **EGDs**)

D :

$N(a)$

Σ :

$$N(X) \rightarrow \exists Y\ E(X, Y)$$
$$E(X, Y) \rightarrow N(Y)$$
$$E(X, Y) \rightarrow X = Y$$

$chase(D, \Sigma) = \{N(a), E(a, a)$

Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\ E(X,Y)$ $E(X,Y) \rightarrow N(Y)$ $E(X,Y) \rightarrow X = Y$

$$chase(D,\Sigma) = \{N(a),\ E(a,a)\}$$

No further dependency is applicable: STOP.

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (**No EGDs**)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\, \exists Z\ E(X,Y,Z)$ $E(X,Y,Y) \rightarrow N(Y)$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (No EGDs)

$$D : \qquad \qquad \Sigma :$$

$$N(a) \qquad \begin{array}{l} N(X) \rightarrow \exists Y\, \exists Z\; E(X,Y,Z) \\ E(X,Y,Y) \rightarrow N(Y) \end{array}$$

$chase(D,\Sigma) = \{N(a),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (No EGDs)

D : Σ :

$N(a)$ $\quad$ $N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$

$E(X, Y, Y) \rightarrow N(Y)$

$chase(D, \Sigma) = \{N(a),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (**No EGDs**)

D : Σ :

$N(a)$ $\quad$ $N(X) \rightarrow \exists Y \, \exists Z \; E(X, Y, Z)$

$E(X, Y, Y) \rightarrow N(Y)$

$$chase(D, \Sigma) = \{N(a), \;\; E(a, \perp_1, \perp_2)$$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (No EGDs)

D : Σ :

$N(a)$ $N(X) \rightarrow \exists Y\, \exists Z\, E(X, Y, Z)$

$E(X, Y, Y) \rightarrow N(Y)$

$$chase(D, \Sigma) = \{N(a),\ \ E(a, \perp_1, \perp_2)\ \}$$

No further dependency is applicable: STOP.

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\, \exists Z\; E(X,Y,Z)$ $E(X,Y,Y) \rightarrow N(Y)$ $\boldsymbol{E(X,Y,Z) \rightarrow Y = Z}$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$ $E(X, Y, Y) \rightarrow N(Y)$ $E(X, Y, Z) \rightarrow Y = Z$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : $\quad\quad \Sigma$:

$N(a) \qquad N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$

$$E(X, Y, Y) \rightarrow N(Y)$$

$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :

$N(a)$

Σ :

$N(X) \rightarrow \exists Y \, \exists Z \; E(X, Y, Z)$
$E(X, Y, Y) \rightarrow N(Y)$
$E(X, Y, Z) \rightarrow Y = Z$

$chase(D, \Sigma) = \{N(a),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : Σ :

$N(a)$

$$N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a),\ \ E(a, \perp_1, \perp_2),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y \, \exists Z \, E(X,Y,Z)$
	$E(X,Y,Y) \rightarrow N(Y)$
	$E(X,Y,Z) \rightarrow Y = Z$

$chase(D, \Sigma) = \{N(a), \;\; E(a, \perp_1, \perp_2),$

Chase and EGDs

Adding EGDs → No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :

$N(a)$

Σ :

$$N(X) \rightarrow \exists Y\, \exists Z\, E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a),\ E(a, \perp_1, \perp_2),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + **EGDs**)

D : $\quad\quad\quad\quad$ Σ :

$N(a)$

$$N(X) \rightarrow \exists Y\, \exists Z\, E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a), E(a, \perp_1, \perp_1),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$
	$E(X, Y, Y) \rightarrow N(Y)$
	$E(X, Y, Z) \rightarrow Y = Z$

$chase(D, \Sigma) = \{N(a),\ E(a, \perp_1, \perp_1),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : Σ :

$N(a)$ $\quad N(X) \rightarrow \exists Y\, \exists Z\, E(X, Y, Z)$

$E(X, Y, Y) \rightarrow N(Y)$

$E(X, Y, Z) \rightarrow Y = Z$

$chase(D, \Sigma) = \{N(a), E(a, \perp_1, \perp_1),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : Σ :

$N(a)$

$$N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a), E(a, \perp_1, \perp_1), N(\perp_1),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : Σ :

$N(a)$

$$N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a),\ E(a, \perp_1, \perp_1),\ N(\perp_1),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y\, \exists Z\; E(X, Y, Z)$
	$E(X, Y, Y) \rightarrow N(Y)$
	$E(X, Y, Z) \rightarrow Y = Z$

$chase(D, \Sigma) = \{N(a),\, E(a, \perp_1, \perp_1),\, N(\perp_1),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + **EGDs**)

D : $\quad\quad\quad$ Σ :

$$N(a) \qquad \begin{array}{l} N(X) \rightarrow \exists Y\, \exists Z\, E(X, Y, Z) \\ E(X, Y, Y) \rightarrow N(Y) \\ E(X, Y, Z) \rightarrow Y = Z \end{array}$$

$chase(D, \Sigma) = \{N(a), E(a, \perp_1, \perp_1), N(\perp_1), E(\perp_1, \perp_2, \perp_3),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : Σ :

$N(a)$

$$N(X) \rightarrow \exists Y\, \exists Z\; E(X,Y,Z)$$
$$E(X,Y,Y) \rightarrow N(Y)$$
$$E(X,Y,Z) \rightarrow Y = Z$$

$chase(D,\Sigma) = \{N(a), E(a,\perp_1,\perp_1), N(\perp_1), E(\perp_1,\perp_2,\perp_3),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D :	Σ :
$N(a)$	$N(X) \rightarrow \exists Y \, \exists Z \; E(X, Y, Z)$
	$E(X, Y, Y) \rightarrow N(Y)$
	$E(X, Y, Z) \rightarrow Y = Z$

$chase(D, \Sigma) = \{N(a), E(a, \perp_1, \perp_1), N(\perp_1), E(\perp_1, \perp_2, \perp_3),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs **+ EGDs**)

D : Σ :

$N(a)$

$$N(X) \rightarrow \exists Y\, \exists Z\, E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$chase(D, \Sigma) = \{N(a),\ E(a, \perp_1, \perp_1),\ N(\perp_1),\ E(\perp_1, \perp_2, \perp_2),$

Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

D : Σ :

$N(a)$

$$N(X) \rightarrow \exists Y \, \exists Z \; E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$$chase(D, \Sigma) = \{N(a), \, E(a, \perp_1, \perp_1), \, N(\perp_1), \, E(\perp_1, \perp_2, \perp_2), \ldots$$

No termination

Relationship between $\mathsf{CT}^{c}_{\forall}$ and $\mathsf{CT}^{c}_{\exists}$

For TGDs only:

$$\mathsf{CT}^{obl}_{\forall} = \mathsf{CT}^{obl}_{\exists} \subset \mathsf{CT}^{sobl}_{\forall} = \mathsf{CT}^{sobl}_{\exists} \subset \mathsf{CT}^{std}_{\forall} \subset \mathsf{CT}^{std}_{\exists} \subset \mathsf{CT}^{core}_{\forall} = \mathsf{CT}^{core}_{\exists}$$

Relationship between $CT^{c}_{\forall}$ and $CT^{c}_{\exists}$

For TGDs only:

$$CT^{obl}_{\forall} = CT^{obl}_{\exists} \subset CT^{sobl}_{\forall} = CT^{sobl}_{\exists} \subset CT^{std}_{\forall} \subset CT^{std}_{\exists} \subset CT^{core}_{\forall} = CT^{core}_{\exists}$$

Known techniques can (and some actually do!) consider the class $CT^{c}_{\forall}$ for a simpler chase (e.g., oblivious).

Relationship between $\mathsf{CT}^{\mathsf{c}}_{\forall}$ and $\mathsf{CT}^{\mathsf{c}}_{\exists}$

For TGDs only:

$$\mathsf{CT}^{\mathsf{obl}}_{\forall} = \mathsf{CT}^{\mathsf{obl}}_{\exists} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall} = \mathsf{CT}^{\mathsf{sobl}}_{\exists} \subset \mathsf{CT}^{\mathsf{std}}_{\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\exists} \subset \mathsf{CT}^{\mathsf{core}}_{\forall} = \mathsf{CT}^{\mathsf{core}}_{\exists}$$

Known techniques can (and some actually do!) consider the class $\mathsf{CT}^{c}_{\forall}$ for a simpler chase (e.g., oblivious).

Then, membership in $\mathsf{CT}^{\mathsf{std}}_{\forall}$, $\mathsf{CT}^{\mathsf{std}}_{\exists}$, $\mathsf{CT}^{\mathsf{core}}_{\forall}$, and $\mathsf{CT}^{\mathsf{core}}_{\exists}$ is implied.

Relationship between $CT^{c}_{\forall}$ and $CT^{c}_{\exists}$

For TGDs only:

$$CT^{obl}_{\forall} = CT^{obl}_{\exists} \subset CT^{sobl}_{\forall} = CT^{sobl}_{\exists} \subset CT^{std}_{\forall} \subset CT^{std}_{\exists} \subset CT^{core}_{\forall} = CT^{core}_{\exists}$$

Known techniques can (and some actually do!) consider the class $CT^{c}_{\forall}$ for a simpler chase (e.g., oblivious).

Then, membership in $CT^{std}_{\forall}$, $CT^{std}_{\exists}$, $CT^{core}_{\forall}$, and $CT^{core}_{\exists}$ is implied.

Question: Does this still hold for TGDs+EGDs?

Relationship between $CT^{c}_{\forall}$ and $CT^{c}_{\exists}$

For TGDs only:

$$CT^{obl}_{\forall} = CT^{obl}_{\exists} \subset CT^{sobl}_{\forall} = CT^{sobl}_{\exists} \subset CT^{std}_{\forall} \subset CT^{std}_{\exists} \subset CT^{core}_{\forall} = CT^{core}_{\exists}$$

Known techniques can (and some actually do!) consider the class $CT^{c}_{\forall}$ for a simpler chase (e.g., oblivious).

Then, membership in $CT^{std}_{\forall}$, $CT^{std}_{\exists}$, $CT^{core}_{\forall}$, and $CT^{core}_{\exists}$ is implied.

Question: Does this still hold for TGDs+EGDs?

$$CT^{obl}_{\forall} \subset CT^{obl}_{\exists} \nparallel CT^{sobl}_{\forall} \subset CT^{sobl}_{\exists} \nparallel CT^{std}_{\forall} \subset CT^{std}_{\exists} \subset CT^{core}_{\forall} = CT^{core}_{\exists}$$

Relationship between $CT^c_\forall$ and $CT^c_\exists$

For TGDs only:

$$CT^{obl}_\forall = CT^{obl}_\exists \subset CT^{sobl}_\forall = CT^{sobl}_\exists \subset CT^{std}_\forall \subset CT^{std}_\exists \subset CT^{core}_\forall = CT^{core}_\exists$$

Known techniques can (and some actually do!) consider the class $CT^c_\forall$ for a simpler chase (e.g., oblivious).

Then, membership in $CT^{std}_\forall$, $CT^{std}_\exists$, $CT^{core}_\forall$, and $CT^{core}_\exists$ is implied.

Question: Does this still hold for TGDs+EGDs?

$$CT^{obl}_\forall \subset CT^{obl}_\exists \nparallel CT^{sobl}_\forall \subset CT^{sobl}_\exists \nparallel CT^{std}_\forall \subset CT^{std}_\exists \subset CT^{core}_\forall = CT^{core}_\exists$$

$$CT^{obl}_\forall \subset CT^{sobl}_\forall \subset CT^{std}_\forall \subset CT^{core}_\forall$$

$$CT^{obl}_\exists \subset CT^{sobl}_\exists \subset CT^{std}_\exists \subset CT^{core}_\exists$$

Termination Criteria and EGDs

- Many techniques are valid for TGDs only;

Termination Criteria and EGDs

- Many techniques are valid for TGDs only;
- But they can be applied by simulating EGDs with TGDs:
  - Natural Simulation [Gottlob et al., PODS06];
  - Substitution-free simulation [Marnette, PODS09].

Termination Criteria and EGDs

- Many techniques are valid for TGDs only;
- But they can be applied by simulating EGDs with TGDs:
  - Natural Simulation [Gottlob et al., PODS06];
  - Substitution-free simulation [Marnette, PODS09].
- However, the behaviour of EGDs cannot be fully simulated via TGDs...

EGDs Simulation

Example

Σ :

$$
\begin{array}{lllll}
r_1 : & A(x) \wedge B(x) & \rightarrow & C(x) \\
r_2 : & C(x) & \rightarrow & \exists y\, A(x) \wedge B(y) \\
r_3 : & C(x) & \rightarrow & \exists y\, A(y) \wedge B(x) \\
r_4 : & A(x) \wedge A(y) & \rightarrow & x = y \\
r_5 : & B(x) \wedge B(y) & \rightarrow & x = y
\end{array}
$$

Every chase sequence is terminating, for any variation of the chase.

EGDs Simulation

Example

Σ :

$$
\begin{array}{lllll}
r_1 : & A(x) \wedge B(x) & \rightarrow & C(x) \\
r_2 : & C(x) & \rightarrow & \exists y\, A(x) \wedge B(y) \\
r_3 : & C(x) & \rightarrow & \exists y\, A(y) \wedge B(x) \\
r_4 : & A(x) \wedge A(y) & \rightarrow & x = y \\
r_5 : & B(x) \wedge B(y) & \rightarrow & x = y
\end{array}
$$

Every chase sequence is terminating, for any variation of the chase.

However, both the natural and the substitution-free simulations of Σ have no terminating chase sequence.

EGDs Simulation

Substitution-free simulation [Mar09]

Example

$$A(X) \wedge B(X) \rightarrow C(X)$$
$$C(X) \rightarrow \exists Y\, A(X) \wedge B(Y)$$
$$C(X) \rightarrow \exists Y\, A(Y) \wedge B(X)$$
$$A(X) \wedge A(Y) \rightarrow X = Y$$
$$B(X) \wedge B(Y) \rightarrow X = Y$$

EGDs Simulation

Substitution-free simulation [Mar09]

Example

$$
\begin{array}{l}
A(X) \wedge B(X) \rightarrow C(X) \\
C(X) \rightarrow \exists Y\, A(X) \wedge B(Y) \\
C(X) \rightarrow \exists Y\, A(Y) \wedge B(X) \\
A(X) \wedge A(Y) \rightarrow X = Y \\
B(X) \wedge B(Y) \rightarrow X = Y
\end{array}
$$

$$
\begin{array}{lcl}
Eq(X, Y) & \rightarrow & Eq(Y, X) \\
Eq(X, Y) \wedge Eq(Y, Z) & \rightarrow & Eq(X, Z) \\
A(X) & \rightarrow & Eq(X, X) \\
B(X) & \rightarrow & Eq(X, X) \\
C(X) & \rightarrow & Eq(X, X)
\end{array}
$$

EGDs Simulation

Substitution-free simulation [Mar09]

Example

$$\cancel{A(X) \wedge B(X) \rightarrow C(X)} \qquad A(X) \wedge B(X_2) \wedge Eq(X, X_2) \rightarrow C(X)$$
$$C(X) \rightarrow \exists Y\, A(X) \wedge B(Y)$$
$$C(X) \rightarrow \exists Y\, A(Y) \wedge B(X)$$
$$A(X) \wedge A(Y) \rightarrow X = Y$$
$$B(X) \wedge B(Y) \rightarrow X = Y$$

$$\begin{array}{lcl} Eq(X, Y) & \rightarrow & Eq(Y, X) \\ Eq(X, Y) \wedge Eq(Y, Z) & \rightarrow & Eq(X, Z) \\ A(X) & \rightarrow & Eq(X, X) \\ B(X) & \rightarrow & Eq(X, X) \\ C(X) & \rightarrow & Eq(X, X) \end{array}$$

EGDs Simulation

Substitution-free simulation [Mar09]

Example

$$\cancel{A(X) \wedge B(X) \rightarrow C(X)} \quad A(X) \wedge B(X_2) \wedge Eq(X, X_2) \rightarrow C(X)$$
$$\cancel{C(X) \rightarrow \exists Y\, A(X) \wedge B(Y)}$$
$$C(X) \rightarrow \exists Y\, A(Y) \wedge B(X)$$
$$A(X) \wedge A(Y) \rightarrow \cancel{X = Y} \;\; Eq(X, Y)$$
$$B(X) \wedge B(Y) \rightarrow \cancel{X = Y} \;\; Eq(X, Y)$$

$$\begin{array}{lcl}
Eq(X, Y) & \rightarrow & Eq(Y, X) \\
Eq(X, Y) \wedge Eq(Y, Z) & \rightarrow & Eq(X, Z) \\
A(X) & \rightarrow & Eq(X, X) \\
B(X) & \rightarrow & Eq(X, X) \\
C(X) & \rightarrow & Eq(X, X)
\end{array}$$

EGDs Simulation

Substitution-free simulation [Mar09]

Example

$$\cancel{A(X) \wedge B(X) \rightarrow C(X)} \quad A(X) \wedge B(X_2) \wedge Eq(X, X_2) \rightarrow C(X)$$

$$C(X) \rightarrow \exists Y\, A(X) \wedge B(Y)$$

$$C(X) \rightarrow \exists Y\, A(Y) \wedge B(X)$$

$$A(X) \wedge A(Y) \rightarrow \cancel{X = Y} \;\; Eq(X, Y)$$

$$B(X) \wedge B(Y) \rightarrow \cancel{X = Y} \;\; Eq(X, Y)$$

$$\begin{array}{lcl} Eq(X, Y) & \rightarrow & Eq(Y, X) \\ Eq(X, Y) \wedge Eq(Y, Z) & \rightarrow & Eq(X, Z) \\ A(X) & \rightarrow & Eq(X, X) \\ B(X) & \rightarrow & Eq(X, X) \\ C(X) & \rightarrow & Eq(X, X) \end{array}$$

Every chase sequence is terminating.

No terminating chase sequence for the substitution-free simulations.

Function Symbols vs. EGDs

Step 1. Replace EGDs with TGDs via Substitution-free simulation [Mar09].
Step 2. Proceed as with TGDs.

Function Symbols vs. EGDs

Step 1. Replace EGDs with TGDs via Substitution-free simulation [Mar09].
Step 2. Proceed as with TGDs.

Recall that:

Example

Terminating

$$
\begin{array}{l}
p(X) \wedge q(X) \rightarrow r(X) \\
r(X) \rightarrow \exists Y\, p(X) \wedge q(Y) \\
r(X) \rightarrow \exists Y\, p(Y) \wedge q(X) \\
p(X) \wedge p(Y) \rightarrow X = Y \\
q(X) \wedge q(Y) \rightarrow X = Y
\end{array}
$$

Non − Terminating

$$
\begin{array}{l}
p(X) \wedge q(X_2) \wedge eq(X, X_2) \rightarrow r(X) \\
r(X) \rightarrow \exists Y\, p(X) \wedge q(Y) \\
r(X) \rightarrow \exists Y\, p(Y) \wedge q(X) \\
p(X) \wedge p(Y) \rightarrow eq(X, Y) \\
q(X) \wedge q(Y) \rightarrow eq(X, Y)
\end{array}
$$

$$
\begin{array}{l}
eq(X, Y) \rightarrow eq(Y, X) \\
eq(X, Y) \wedge eq(Y, Z) \rightarrow eq(X, Z) \\
p(X) \rightarrow eq(X, X) \\
q(X) \rightarrow eq(X, X) \\
r(X) \rightarrow eq(X, X)
\end{array}
$$

Dealing with EGDs

Example

$D = \{N(a)\}$, Σ :

$$\begin{array}{lcl} N(x) & \rightarrow & \exists y\, E(x,y) \\ E(x,y) & \rightarrow & N(y) \\ E(x,y) & \rightarrow & x = y \end{array}$$

Dealing with EGDs

Example

$D = \{N(a)\}$, Σ :

$$\begin{array}{lcl} N(x) & \rightarrow & \exists y\ E(x,y) \\ E(x,y) & \rightarrow & N(y) \\ E(x,y) & \rightarrow & x = y \end{array}$$

$N(a)$

Dealing with EGDs

Example

$D = \{N(a)\}, \Sigma:$

$$\begin{array}{lcl} N(x) & \rightarrow & \exists y\ E(x,y) \\ E(x,y) & \rightarrow & N(y) \\ E(x,y) & \rightarrow & x = y \end{array}$$

$N(a) \rightarrow E(a, \ \perp_1 \ )$

Dealing with EGDs

Example

$D = \{N(a)\}, \Sigma :$

$$\begin{array}{lcl} N(x) & \rightarrow & \exists y\ E(x,y) \\ E(x,y) & \rightarrow & N(y) \\ \color{red}{E(x,y)} & \color{red}{\rightarrow} & \color{red}{x = y} \end{array}$$

$N(a) \rightarrow E(a,\ \perp_1\ )$

Dealing with EGDs

Example

$D = \{N(a)\}, \Sigma:$

$$
\begin{array}{lcl}
N(x) & \rightarrow & \exists y\, E(x,y) \\
E(x,y) & \rightarrow & N(y) \\
E(x,y) & \rightarrow & x = y
\end{array}
$$

$N(a) \rightarrow E(a, \perp_1) \rightarrow a = \perp_1$

Dealing with EGDs

Example

$D = \{N(a)\}, \Sigma :$

$$\begin{array}{lcl} N(x) & \rightarrow & \exists y\ E(x,y) \\ E(x,y) & \rightarrow & N(y) \\ E(x,y) & \rightarrow & x = y \end{array}$$

$N(a) \rightarrow E(a, a) \rightarrow a = \perp_1$

Dealing with EGDs

Example

$D = \{N(a)\}, \Sigma :$

$$
\begin{array}{lcl}
N(x) & \rightarrow & \exists y\ E(x,y) \\
E(x,y) & \rightarrow & N(y) \\
E(x,y) & \rightarrow & x = y
\end{array}
$$

$N(a) \rightarrow E(a, a) \rightarrow a = \perp_1 \rightarrow$ all constraints satisfied!

Rewriting TGDs and EGDs

Example

$$
\begin{array}{rlcl}
r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\
r_2: & E(x,y) & \rightarrow & N(y) \\
r_3: & E(x,y) & \rightarrow & x = y
\end{array}
$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{lll} r_1: N(x) & \rightarrow & \exists y\, E(x,y) \\ r_2: E(x,y) & \rightarrow & N(y) \\ r_3: E(x,y) & \rightarrow & x = y \end{array}$$

$$r_3': E^{bb}(x,y)$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{lllll} r_1 : & N(x) & \rightarrow & \exists y\, E(x,y) \\ r_2 : & E(x,y) & \rightarrow & N(y) \\ r_3 : & E(x,y) & \rightarrow & x = y \end{array}$$

$$r'_3 : \; E^{bb}(x,y) \quad \rightarrow \quad x = y$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{lllll} r_1 : & N(x) & \rightarrow & \exists y\, E(x,y) \\ r_2 : & E(x,y) & \rightarrow & N(y) \\ r_3 : & E(x,y) & \rightarrow & x = y \end{array}$$

$$\begin{array}{llll} r'_3 : & E^{bb}(x,y) & \rightarrow & x = y \\ r'_2 : & E^{bb}(x,y) & & \end{array}$$

Rewriting TGDs and EGDs

Example

$$
\begin{array}{rlcl}
r_1: & N(x) & \rightarrow & \exists y\, E(x,y) \\
r_2: & \color{red}{E(x,y)} & \color{red}{\rightarrow} & \color{red}{N(y)} \\
r_3: & E(x,y) & \rightarrow & x = y
\end{array}
$$

$$
\begin{array}{rlcl}
r'_3: & E^{bb}(x,y) & \rightarrow & x = y \\
r'_2: & E^{bb}(x,y) & \rightarrow & N^{b}(y)
\end{array}
$$

Rewriting TGDs and EGDs

Example

$$
\begin{array}{rcl}
r_1 : \; N(x) & \rightarrow & \exists y \; E(x,y) \\
r_2 : \; E(x,y) & \rightarrow & N(y) \\
r_3 : \; E(x,y) & \rightarrow & x = y
\end{array}
$$

$$
\begin{array}{rcl}
r'_3 : \; E^{bb}(x,y) & \rightarrow & x = y \\
r'_2 : \; E^{bb}(x,y) & \rightarrow & N^b(y) \\
r'_1 : \; N^b(x) & &
\end{array}
$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{rlcl} r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\ r_2: & E(x,y) & \rightarrow & N(y) \\ r_3: & E(x,y) & \rightarrow & x = y \end{array}$$

$$\begin{array}{rlcll} r'_3: & E^{bb}(x,y) & \rightarrow & x = y & \\ r'_2: & E^{bb}(x,y) & \rightarrow & N^{b}(y) & \\ r'_1: & N^{b}(x) & \rightarrow & \exists y\ E^{bf_1}(x,y) & f_1 = f^{y}_{r_1}(b) \end{array}$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{rlcl} r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\ r_2: & E(x,y) & \rightarrow & N(y) \\ r_3: & E(x,y) & \rightarrow & x = y \end{array}$$

$$\begin{array}{rlcl} r_3': & E^{bb}(x,y) & \rightarrow & x = y \\ r_2': & E^{bb}(x,y) & \rightarrow & N^{b}(y) \\ r_1': & N^{b}(x) & \rightarrow & \exists y\ E^{bf_1}(x,y) \quad f_1 = f_{r_1}^{y}(b) \\ r_3'': & E^{bf_1}(x,y) & & \end{array}$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{rlcl} r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\ r_2: & E(x,y) & \rightarrow & N(y) \\ r_3: & E(x,y) & \rightarrow & x = y \end{array}$$

$$\begin{array}{rlcl} r_3': & E^{bb}(x,y) & \rightarrow & x = y \\ r_2': & E^{bb}(x,y) & \rightarrow & N^{b}(y) \\ r_1': & N^{b}(x) & \rightarrow & \exists y\ E^{bf_1}(x,y) \quad f_1 = f_{r_1}^{y}(b) \\ r_3'': & E^{bf_1}(x,y) & \rightarrow & x = y \end{array}$$

Rewriting TGDs and EGDs

Example

$$
\begin{array}{llcl}
r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\
r_2: & E(x,y) & \rightarrow & N(y) \\
r_3: & E(x,y) & \rightarrow & x = y
\end{array}
$$

$$
\begin{array}{llcll}
r'_3: & E^{bb}(x,y) & \rightarrow & x = y & \\
r'_2: & E^{bb}(x,y) & \rightarrow & N^{b}(y) & \\
r'_1: & N^{b}(x) & \rightarrow & \exists y\ E^{bf_1}(x,y) & f_1 = f^{y}_{r_1}(b) \\
r''_3: & E^{bf_1}(x,y) & \rightarrow & x = y & b = f_1
\end{array}
$$

Rewriting TGDs and EGDs

Example

$$
\begin{array}{llcl}
r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\
r_2: & E(x,y) & \rightarrow & N(y) \\
r_3: & E(x,y) & \rightarrow & x = y
\end{array}
$$

$$
\begin{array}{llcll}
r'_3: & E^{bb}(x,y) & \rightarrow & x = y & \\
r'_2: & E^{bb}(x,y) & \rightarrow & N^{b}(y) & \\
r'_1: & N^{b}(x) & \rightarrow & \exists y\ E^{bb}(x,y) & \cancel{f_1 = f^{y}_{r_1}(b)} \\
r''_3: & E^{bb}(x,y) & \rightarrow & x = y & b = f_1
\end{array}
$$

Rewriting TGDs and EGDs

Example

$$
\begin{array}{rlcl}
r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\
r_2: & E(x,y) & \rightarrow & N(y) \\
r_3: & E(x,y) & \rightarrow & x = y
\end{array}
$$

$$
\begin{array}{rlcl}
r_3': & E^{bb}(x,y) & \rightarrow & x = y \\
r_2': & E^{bb}(x,y) & \rightarrow & N^{b}(y) \\
r_1': & N^{b}(x) & \rightarrow & \exists y\ E^{bb}(x,y) \\
r_3'': & E^{bb}(x,y) & \rightarrow & x = y
\end{array}
$$

Rewriting TGDs and EGDs

Example

$$\begin{array}{rlcl} r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\ r_2: & E(x,y) & \rightarrow & N(y) \\ r_3: & E(x,y) & \rightarrow & x = y \end{array}$$

$$\begin{array}{rlcl} r_3': & E^{bb}(x,y) & \rightarrow & x = y \\ r_2': & E^{bb}(x,y) & \rightarrow & N^b(y) \\ r_1': & N^b(x) & \rightarrow & \exists y\ E^{bb}(x,y) \\ r_3'': & E^{bb}(x,y) & \rightarrow & x = y \end{array}$$

No cyclic symbol f_i occurs in the constraints above.

Rewriting TGDs and EGDs

Example

$$\begin{array}{rlcl} r_1: & N(x) & \rightarrow & \exists y\ E(x,y) \\ r_2: & E(x,y) & \rightarrow & N(y) \\ r_3: & E(x,y) & \rightarrow & x = y \end{array}$$

$$\begin{array}{rlcl} r'_3: & E^{bb}(x,y) & \rightarrow & x = y \\ r'_2: & E^{bb}(x,y) & \rightarrow & N^{b}(y) \\ r'_1: & N^{b}(x) & \rightarrow & \exists y\ E^{bb}(x,y) \\ r''_3: & E^{bb}(x,y) & \rightarrow & x = y \end{array}$$

No cyclic symbol f_i occurs in the constraints above.

Thus, there exists a terminating standard chase sequence.

Rewriting TGDs and EGDs

Example

$$
\begin{array}{rcll}
r_1 : & N(x) & \rightarrow & \exists y\ E(x,y) \\
r_2 : & E(x,y) & \rightarrow & N(y) \\
r_3 : & E(x,y) & \rightarrow & x = y
\end{array}
$$

$$
\begin{array}{rcll}
r'_3 : & E^{bb}(x,y) & \rightarrow & x = y \\
r'_2 : & E^{bb}(x,y) & \rightarrow & N^{b}(y) \\
r'_1 : & N^{b}(x) & \rightarrow & \exists y\ E^{bb}(x,y) \\
r''_3 : & E^{bb}(x,y) & \rightarrow & x = y
\end{array}
$$

No cyclic symbol f_i occurs in the constraints above.

Thus, there exists a terminating standard chase sequence.

In this sequence, EGDs are applied as soon as possible.

Results

- The rewriting algorithm always terminates;
- $\Sigma^\alpha \in \mathtt{CT}_\exists^{\mathit{std}}$ implies $\Sigma \in \mathtt{CT}_\exists^{\mathit{std}}$;
- Furthermore, if $\nexists$ cyclic f_i in Σ^α, then $\Sigma \in \mathtt{CT}_\exists^{\mathit{std}}$;

Current and Future directions

- Determine decidable classes of data dependencies,
- Consider and extended framework (*Datalog*$[\exists, =, F, \neg]$),
- Define criteria guaranteing termination of one chase sequence,
- Determine how to compute one of the terminating sequences,
- Further exploiting of EGDs
- Complexity (not discussed here)
- Support for design tools.

Thanks!

Questions?

[BBC09] S. Baselice, P. A. Bonatti, and G. Criscuolo. On finitely recursive programs. *TPLP*, 9(2):213–238, 2009.

[BLMS11] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[Bon04] P. A. Bonatti. Reasoning with infinite stable models. *Artificial Intelligence*, 156(1):75–111, 2004.

[Bon11] P. A. Bonatti. On the decidability of fdnc programs. *Intelligenza Artificiale*, 5(1):89–93, 2011.

[CCIL08] F. Calimeri, S. Cozza, G. Ianni, and N. Leone. Computable functions in ASP: Theory and implementation. In *ICLP*, pages 407–424, 2008.

[CGK13] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *JAIR*, 48:115–174, 2013.

[CGMT14] M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Checking termination of logic programs with function symbols through linear constraints. In *RuleML*, pages 97–111, 2014.

[CGP10] Andrea Calì, Georg Gottlob, and Andreas Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[CGP15] Marco Calautti, Georg Gottlob, and Andreas Pieris. Chase termination for guarded existential rules. In *PODS*, pages 91–103, 2015.

[CGST14] M. Calautti, S. Greco, F. Spezzano, and I. Trubitsyna. Checking termination of bottom-up evaluation of logic programs with function symbols. *TPLP*, 2014.

[CGT13] M. Calautti, S. Greco, and I. Trubitsyna. Detecting decidable classes of finitely ground logic programs with function symbols. In *PPDP*, 2013.

[DNR08] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[ES10] Thomas Eiter and Mantas Simkus. FDNC: decidable nonmonotonic disjunctive logic programs with function symbols. *TOCL*, 11(2), 2010.

[FKMP05] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

[GHK+13] Bernardo Cuenca Grau, Ian Horrocks, Markus Krotzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *JAIR*, 47:741–808, 2013.

[GM14] T. Gogacz and J. Marcinkowski. All-instances termination of chase is undecidable. In *ICALP*, pages 293–304, 2014.

[GMT13a] S. Greco, C. Molinaro, and I. Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. *TPLP*, 13(4-5):737–752, 2013.

[GMT13b] Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Bounded programs: A new decidable class of logic programs with function symbols. In *IJCAI*, 2013.

[GO13] Gösta Grahne and Adrian Onet. Anatomy of the chase. *CoRR*, abs/1303.6682, 2013.

[GS10] S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.

[GST07] M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In *LPNMR*, pages 266–271, 2007.

[GST11] Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratifi- cation criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.

[GST15] Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Checking chase termination: Cyclicity analysis and rewriting techniques. *TKDE*, 27(3):621–635, 2015.

[LL09] Y. Lierler and V. Lifschitz. One more decidable class of finitely ground programs. In *ICLP*, pages 489–493, 2009.

[Mar09] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[Mei10] Michael Meier. *On the Termination of the Chase Algorithm*. Albert-Ludwigs-Universitat Freiburg (Germany), 2010.

[MSL09] M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *CoRR*, abs/0906.4228, 2009.

[One13] Adrian Onet. The chase procedure and its applications in data exchange. In *Data Exchange, Integration, and Streams*, pages 1–37. 2013.

[RS14] F. Riguzzi and T. Swift. Terminating evaluation of logic programs with finite three-valued models. *ACM TOCL*, 2014.

[Syr01] T. Syrjanen. Omega-restricted logic programs. In *LPNMR*, pages 267–279, 2001.