Termination Analysis of Logic Programs

Sergio Greco and Cristian Molinaro

DIMES, University of Calabria, Italy

July 25th, 2015
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?

- Rules with function symbols.
- 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.
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

- Codish, Taboch. A semantic basis for the termination analysis of logic programs. JLP (1999).
- Pedreschi, Ruggieri, Smaus. Classes of terminating logic programs. TPLP (2002).
Top-down Evaluation

- 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).
Bottom-up Evaluation

- Chomicki. A decidable class of logic programs with function symbols. TR 1990.
Bottom-up Evaluation

- Lierler, Lifschitz. One more decidable class of finitely ground programs. ICLP (2009).
- 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).
Top-down vs. Bottom-up Evaluation

Example

\[ p(X) \leftarrow p(X). \]

- Non-terminating top-down evaluation.
- Completely harmless under bottom-up evaluation.
Top-down vs. Bottom-up Evaluation

Example

\[ p(x) \leftarrow p(x). \]

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

We consider **bottom-up** evaluation.
Bottom-up Evaluation

Example

\[
\begin{align*}
\text{len}([a, b, c], 0). \\
\text{len}(\text{Tail}, \text{s}(N)) & \leftarrow \text{len}(\text{list}(\text{Head}, \text{Tail}), N).
\end{align*}
\]
Bottom-up Evaluation

Example

\[
\begin{align*}
\text{len}([a,b,c], 0). \\
\text{len}(\text{Tail}, s(N)) & \leftarrow \text{len}(\text{list}(\text{Head}, \text{Tail}), N).
\end{align*}
\]

Bottom-up evaluation:

\[
\begin{align*}
\text{len}([b, c], s(0)) & \leftarrow \text{len}([a, b, c], 0) \quad \text{yields} \quad \text{len}([b, c], s(0))
\end{align*}
\]
Bottom-up Evaluation

Example

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

Bottom-up evaluation:

\[
\begin{align*}
\text{len}([b, c], s(0)) & \leftarrow \text{len}([a, b, c], 0) & \text{yields } & \text{len}([b, c], s(0)) \\
\text{len}([c], s(s(0))) & \leftarrow \text{len}([b, c], s(0)) & \text{yields } & \text{len}([c], s(s(0))))
\end{align*}
\]
**Bottom-up Evaluation**

**Example**

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

**Bottom-up evaluation:**

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

Fixpoint, the evaluation TERMINATES.
**Bottom-up Evaluation**

**Example**

\[
\text{len}([a, b, c], 0).
\]

\[
\text{len}((\text{Tail}, s(N)) \leftarrow \text{len}(\text{list}(\text{Head}, \text{Tail}), N).
\]

**Bottom-up evaluation:**

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

\[
\text{len}([c], s(s(0))) \leftarrow \text{len}([b, c], s(0)) \quad \text{yields} \quad \text{len}([c], s(s(0)))
\]

\[
\text{len}([], s(s(s(0)))) \leftarrow \text{len}([c], s(s(0))) \quad \text{yields} \quad \text{len}([], s(s(s(0))))
\]

**Fixpoint, the evaluation TERMINATES.**
Bottom-up Evaluation

Example

\begin{align*}
\text{nat}(0). \\
\text{nat}(s(X)) & \leftarrow \text{nat}(X).
\end{align*}

Bottom-up evaluation:

\begin{align*}
\text{nat}(s(0)) & \leftarrow \text{nat}(0) & \text{yields} & \text{nat}(s(0))
\end{align*}
Bottom-up Evaluation

Example

\[
\begin{align*}
\text{nat}(0). \\
\text{nat}(s(X)) & \leftarrow \text{nat}(X).
\end{align*}
\]

Bottom-up evaluation:

\[
\begin{align*}
\text{nat}(s(0)) & \leftarrow \text{nat}(0) & \text{yields} & \text{nat}(s(0)) \\
\text{nat}(s(s(0))) & \leftarrow \text{nat}(s(0)) & \text{yields} & \text{nat}(s(s(0)))
\end{align*}
\]

The evaluation does NOT terminate.
Bottom-up Evaluation

Example

\[
\text{nat}(0).
\text{nat}(s(X)) \leftarrow \text{nat}(X).
\]

Bottom-up evaluation:

\[
\begin{align*}
\text{nat}(s(0)) & \leftarrow \text{nat}(0) \quad \text{yields} \quad \text{nat}(s(0)) \\
\text{nat}(s(s(0))) & \leftarrow \text{nat}(s(0)) \quad \text{yields} \quad \text{nat}(s(s(0))) \\
\text{nat}(s(s(s(0)))) & \leftarrow \text{nat}(s(s(0))) \quad \text{yields} \quad \text{nat}(s(s(s(0)))) \\
& \vdots
\end{align*}
\]

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 $\textit{terminating}$ 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 \textit{terminating} iff the evaluation of $P \cup D$ terminates for every finite set of facts $D$.

Termination Criteria

Define a decidable condition $C$ such that for every program $P$

$$P \text{ satisfies } C \implies P \text{ is terminating.}$$
Termination Criteria

- $\omega$-restricted programs [Syr01]
- $\lambda$-restricted programs [GST07]
- Finite domain programs [CCIL08]
- Argument-restricted programs [LL09]
- Safe and $\Gamma$-acyclic programs [CGST14]
- Mapping-restricted programs [CGT13]
- Bounded programs [GMT13b]
- Rule- and cycle-bounded programs [CGMT14]
- Program Adornment technique [GMT13a]
Termination Criteria

- $\omega$-restricted programs [Syr01]
- $\lambda$-restricted programs [GST07]
- Finite domain programs [CCIL08]
- Argument-restricted programs [LL09]
- Safe and $\Gamma$-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 $\geq 0$).
Definition

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

- 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.
Definition

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

- 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.
Definition

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

- 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

$$A_0 \leftarrow A_1, \ldots, A_n$$

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 \(\geq 0\)).

- 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

\[
A_0 \leftarrow A_1, \ldots, A_n
\]

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

\[ p(f(X), Y) \leftarrow q(X), r(Y). \]

**Example (Unsafe program)**

\[ p(f(X), Y) \leftarrow q(X), r(Z). \]
We consider **safe** programs: every variable in the head must appear in the body.

Example (**Safe** program)

\[ p(f(X), Y) \leftarrow q(X), r(Y). \]

Example (**Unsafe** program)

\[ p(f(X), Y) \leftarrow q(X), r(Z). \]

**No disjunction and negation** (for now).
We consider \textit{safe} programs: every variable in the head must appear in the body.

\textbf{Example (Safe program)}

\[
p(f(X), Y) \leftarrow q(X), r(Y).
\]

\textbf{Example (Unsafe program)}

\[
p(f(X), Y) \leftarrow q(X), r(Z).
\]

No disjunction and negation (for now).

Function symbols are \textit{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 $1 \leq i \leq \text{arity}(p)$.
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 $1 \leq i \leq \text{arity}(p)$.

Example

\[
\begin{align*}
p(X, Y) & \leftarrow b(X, Y). \\
q(f(X)) & \leftarrow p(X, Y).
\end{align*}
\]

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**

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]
**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**

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]

$\lambda(r) = 1$;
\(\lambda\)-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**

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]

\(\lambda(r) = 1;\)
**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**

$$
\begin{align*}
q(x) & \leftarrow p(x), r(x). \\
p(f(x)) & \leftarrow q(x).
\end{align*}
$$

$\lambda(r) = 1$;
\textbf{\textit{\textlambda}-Restricted Programs [GST07]}

\textbf{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.

\textbf{Example}

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]

\[\lambda(r) = 1; \quad \lambda(q) = 2;\]
**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**

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]

$\lambda(r) = 1; \quad \lambda(q) = 2;$
**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**

\[
q(X) \leftarrow p(X), r(X).
\]

\[
p(f(X)) \leftarrow q(X).
\]

$\lambda(r) = 1; \quad \lambda(q) = 2;$
**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

<table>
<thead>
<tr>
<th>Rule</th>
<th>$\lambda(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q(X) \leftarrow p(X), r(X).$</td>
<td>2</td>
</tr>
<tr>
<td>$p(f(X)) \leftarrow q(X).$</td>
<td>3</td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]

\( \lambda(r) = 1; \quad \lambda(q) = 2; \quad \lambda(p) = 3. \)
**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**

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X). \\
p(f(X)) & \leftarrow q(X).
\end{align*}
\]

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

The program is $\lambda$-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**

\[
\begin{align*}
q(X) & \leftarrow p(X), r(X) . \\
p(f(X)) & \leftarrow q(X) . \\
\lambda(r) & = 1; \\
\lambda(q) & = 2; \\
\lambda(p) & = 3.
\end{align*}
\]

The program is $\lambda$-restricted.

**Example**

\[
\begin{align*}
p(X) & \leftarrow p(X) . \\
\end{align*}
\]

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

\[
\lambda(p) > \lambda(p) \Rightarrow \text{The program is not } \lambda\text{-restricted.}
\]
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)

\[
\begin{align*}
q(X) & \leftarrow q(f(X)). \\
p(f(X)) & \leftarrow q(X), r(X, Y).
\end{align*}
\]
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)

\[
\begin{align*}
q(X) & \leftarrow q(f(X)). \\
p(f(X)) & \leftarrow q(X), r(X, Y).
\end{align*}
\]
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)

\[
\begin{align*}
q(X) & \leftarrow q(f(X)). \\
p(f(X)) & \leftarrow q(X), r(X, Y).
\end{align*}
\]
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)**

\[
\begin{align*}
q(X) & \leftarrow q(f(X)). \\
p(f(X)) & \leftarrow q(X), r(X, Y).
\end{align*}
\]
Finite Domain Programs [CCIL08]

Example

\[ q(x) \leftarrow q(f(x)). \]
\[ p(f(x)) \leftarrow q(x), r(x, y). \]

Finite domain arguments:

- \( q[1] \)
- \( p[1] \)
- \( r[1] \)
- \( r[2] \)
Finite Domain Programs [CCIL08]

Example

\[ q(X) \leftarrow q(f(X)). \]
\[ p(f(X)) \leftarrow q(X), r(X, Y). \]

Finite domain arguments:
- \( r[1] \) and \( r[2] \), as they appear in no head.
Finite Domain Programs [CCIL08]

Example

\[
\begin{align*}
q(X) & \leftarrow q(f(X)). \\
p(f(X)) & \leftarrow q(X), r(X, Y).
\end{align*}
\]

Finite domain arguments:

- \(r[1]\) and \(r[2]\), as they appear in no head.
- \(q[1]\), as the term in the head of the 1st rule is a subterm of that in the body.
Finite Domain Programs [CCIL08]

Example

\[
q(X) \leftarrow q(f(X)).  \\
p(f(X)) \leftarrow q(X), r(X, Y).
\]

Finite domain arguments:
- \(r[1]\) and \(r[2]\), as they appear in no head.
- \(q[1]\), as the term in the head of the 1st rule is a subterm of that in the body.
- \(p[1]\), as \(r[1]\) is finite domain and is not “recursive” with \(p[1]\).
Finite Domain Programs [CCIL08]

Example

\[
q(X) \leftarrow q(f(X)). \\
p(f(X)) \leftarrow q(X), r(X, Y).
\]

Finite domain arguments:
- \(r[1]\) and \(r[2]\), as they appear in no head.
- \(q[1]\), as the term in the head of the 1st rule is a subterm of that in the body.
- \(p[1]\), as \(r[1]\) is finite domain and is not “recursive” with \(p[1]\)

All arguments are finite domain \(\Rightarrow\) The program is finite domain.
Relative Expressivity

Theorem

\( \lambda \)-Restricted \( \not\models \) Finite Domain.
Relative Expressivity

Finite Domain \(\lambda\)-Restricted

Theorem

\(\lambda\)-Restricted \(\not\equiv\) Finite Domain.

Example (Finite Domain but not \(\lambda\)-Restricted)

\(q(X) \leftarrow q(f(X)).\)
\(p(f(X)) \leftarrow q(X), r(X, Y).\)
Relative Expressivity

Finite Domain \( \sqcap \) \( \lambda \)-Restricted

Theorem

\( \lambda \)-Restricted \( \parallel \) Finite Domain.

Example (Finite Domain but not \( \lambda \)-Restricted)

\[
q(X) \leftarrow q(f(X)). \\
p(f(X)) \leftarrow q(X), r(X, Y).
\]

Example (\( \lambda \)-Restricted but not Finite Domain)

\[
q(X) \leftarrow p(X), r(X). \\
p(f(X)) \leftarrow q(X).
\]
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 $X$:

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

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

**Example**

\[
d(X, f(X, g(X), Y))
\]
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 $X$:

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

**Example**

$$
d(X, f(X, g(X), Y)) = 2
$$
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 $X$:

\[
\begin{align*}
    d(X, X) &= 0 \\
    d(X, f(t_1, \ldots, t_m)) &= 1 + \max_{1 \leq i \leq m : t_i \text{ contains } X} d(X, t_i).
\end{align*}
\]

**Example**

\[
\begin{align*}
    d(X, f(X, g(X), Y)) &= 2 \\
    d(Y, f(X, g(X), Y)) &= 1
\end{align*}
\]
Argument Restriction [LL09]

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

**Example**

\[
\begin{align*}
p(f(X)) & \leftarrow q(X) \\
q(X) & \leftarrow p(f(X))
\end{align*}
\]
Argument Restriction [LL09]

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

**Example**

\[
\begin{align*}
p(f(X)) & \leftarrow q(X) \\
q(X) & \leftarrow p(f(X))
\end{align*}
\]

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

\[
\phi(p[1]) \geq \phi(q[1]) + 1 - 0, \text{ and}
\]
Argument Restriction [LL09]

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

**Example**

\[
\begin{align*}
p(f(X)) & \leftarrow q(X) \\
q(X) & \leftarrow p(f(X))
\end{align*}
\]

We need to find a function \( \phi \)

(assigning an integer to each argument) such that:

\[
\phi(p[1]) \geq \phi(q[1]) + 1 - 0, \quad \text{and} \\
\phi(q[1]) \geq \phi(p[1]) + 0 - 1.
\]
Argument Restriction [LL09]

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

Example

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

$$\phi(p[1]) \geq \phi(q[1]) + 1 - 0$$

and

$$\phi(q[1]) \geq \phi(p[1]) + 0 - 1.$$
Theorem

- \textit{Finite Domain} \subset \not\subset \textit{Argument Restricted}.
- \textit{\lambda\text{-Restricted}} \subset \not\subset \textit{Argument Restricted}.

Relative Expressivity
Argument Restriction [LL09]

Simple and easy (polynomial time) to compute.

Limitation:

No distinction between different function symbols.

Example:

\[ p(f(f(X))) \leftarrow p(g(X)) \]

We need to find a function \( \phi \) such that

\[ \phi(p[1]) \geq \phi(p[1]) + 1 \]

No such \( \phi \) 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.
Argument Restriction [LL09]

Simple and easy (polynomial time) to compute.

**Limitation:** No distinction between different function symbols.

**Example**

\[ p(f(f(x))) \leftarrow p(g(x)) \]

We need to find a function \( \phi \) such that

\[ \phi(p[1]) \geq \phi(p[1]) + 1 \]

**No such \( \phi \) 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]

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.
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.
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 $r_j$. 
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

$r_1 : q(f(X)) ← p(X)$
$r_2 : p(g(X)) ← q(X)$
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

\[
\begin{align*}
  r_1 &: \quad q(f(X)) &\leftarrow &p(X) \\
  r_2 &: \quad p(g(X)) &\leftarrow &q(X)
\end{align*}
\]
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

\[
\begin{align*}
  r_1 &: \quad q(f(X)) & \leftarrow & \quad p(X) \\
  r_2 &: \quad p(g(X)) & \leftarrow & \quad q(X)
\end{align*}
\]
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

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

\[ r_1 : \quad q(f(X)) \leftarrow p(X) \]
\[ r_2 : \quad p(g(X)) \leftarrow q(X) \]

Example

\[ r_1 : \quad q(f(X)) \leftarrow p(X) \]
\[ r_2 : \quad p(g(X)) \leftarrow q(g(X)) \]
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

$r_1 : q(f(X)) ← p(X)$
$r_2 : p(g(X)) ← q(X)$

Example

$r_1 : q(f(X)) ← p(X)$
$r_2 : p(g(X)) ← q(g(X))$

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

\[ r_1: q(f(X), X) \leftarrow b(Y), p(X). \]

\[ r_2: p(X) \leftarrow q(f(X), Y). \]
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{align*}
r_1 & : \quad q(f(X), X) \leftarrow b(Y), p(X). \\
\end{align*}
\begin{align*}
r_2 & : \quad p(X) \leftarrow q(f(X), Y).
\end{align*}
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

\[
\begin{align*}
    r_1 : & \quad q(f(X), X) \leftarrow b(Y), p(X). \\
    r_2 : & \quad p(X) \leftarrow q(f(X), Y).
\end{align*}
\]
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

\[
\begin{align*}
  r_1 : & \quad q(f(X), X) \leftarrow b(Y), p(X). \\
  r_2 : & \quad p(X) \leftarrow q(f(X), Y).
\end{align*}
\]
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

\[
\begin{align*}
r_1 : & \quad q(f(X), X) \leftarrow b(Y), p(X). \\
r_2 : & \quad p(X) \leftarrow q(f(X), Y).
\end{align*}
\]

 
\( f, r_2, 1 \)

\( q[1] \)  
\( p[1] \)  
\( \epsilon, r_1, 2 \)  
\( q[2] \)  
\( b[1] \)
Two classifications of cycles in the labeled argument graph:

The aim is to identify “harmless” cycles.
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
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

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 cycle

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(f(X)) \]

\[ f, r_1, 1 \]
\[ \bar{f}, r_2, 1 \]

\[ f\bar{f} \approx \epsilon \]

p(a).
q(f(a)) \leftarrow p(a).
Balanced, Growing, and Failing Cycles

Balanced cycle

\[ r_1 : \text{q}(f(X)) \leftarrow \text{p}(X) \]
\[ r_2 : \text{p}(X) \leftarrow \text{q}(f(X)) \]

\[ f, r_1, 1 \]
\[ \text{p}[1] \rightarrow \text{q}[1] \]
\[ \text{f}, r_2, 1 \]
\[ f\bar{f} \approx \epsilon \]

\[ \text{p}(a). \]
\[ \text{q}(f(a)) \leftarrow \text{p}(a). \]
\[ \text{p}(a) \leftarrow \text{q}(f(a)). \]
Balanced, Growing, and Failing Cycles

**Balanced cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(f(X)) \]

- \( f, r_1, 1 \)
- \( f, r_2, 1 \)

\[ f \bar{f} \approx \epsilon \]

**Growing cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(X) \]

- \( f, r_1, 1 \)
- \( \epsilon, r_2, 1 \)

\[ f \epsilon \approx f \]

\[ p(a). \]
\[ q(f(a)) \leftarrow p(a). \]
Balanced, Growing, and Failing Cycles

**Balanced cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(f(X)) \]

**Growing cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(f(X)) \]

p(a).
q(f(a)) \leftarrow p(a).
p(f(a)) \leftarrow q(f(a)).

\[ f \epsilon \approx f \]

\[ f \bar{f} \approx \epsilon \]
Balanced, Growing, and Failing Cycles

**Balanced cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(f(X)) \]

**Growing cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(X) \]

**Failing cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(h(X)) \]

\[ f, r_1, 1 \]
\[ f \epsilon \approx f \]
\[ f \bar{h} \]

\[ \bar{f} \approx \epsilon \]
Balanced, Growing, and Failing Cycles

**Balanced cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(f(X)) \]

**Growing cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(X) \]

**Failing cycle**

\[ r_1 : q(f(X)) \leftarrow p(X) \]
\[ r_2 : p(X) \leftarrow q(h(X)) \]

\[ f \bar{f} \approx \epsilon \]
\[ f \epsilon \approx f \]
\[ f \bar{h} \]
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)**

\[
\begin{align*}
    r_1 & : q(f(X)) \leftarrow p(X). \\
    r_2 & : p(X) \leftarrow q(X).
\end{align*}
\]

![Diagram of Active Cycle](image-url)
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)**

\[
\begin{align*}
    r_1 : & \quad q(f(X)) \leftarrow p(X). \\
    r_2 : & \quad p(X) \leftarrow q(X).
\end{align*}
\]

\[
\begin{tikzpicture}
    \node (p1) at (0,0) {$p[1]$};
    \node (q1) at (2,0) {$q[1]$};
    \node (r1) at (1,-1) {$r_1$};
    \node (r2) at (1,1) {$r_2$};
    \draw[->] (p1) -- (r1) node[midway,above] {$f$, $r_1$, 1};
    \draw[->] (r1) -- (q1) node[midway,above] {$e$, $r_2$, 1};
    \draw[->] (q1) -- (r2) node[midway,above] {$e$, $r_2$, 1};
    \draw[->] (r2) -- (p1) node[midway,above] {$f$, $r_1$, 1};
\end{tikzpicture}
\]

**Example (Inactive Cycles)**

\[
\begin{align*}
    r_1 : & \quad p(f(X), g(X)) \leftarrow p(X, X).
\end{align*}
\]

\[
\begin{tikzpicture}
    \node (p1) at (0,0) {$p[1]$};
    \node (p2) at (2,0) {$p[2]$};
    \node (r1) at (1,-1) {$r_1$};
    \draw[->] (p1) -- (r1) node[midway,above] {$f$, $r_1$, 1};
    \draw[->] (r1) -- (p2) node[midway,above] {$g$, $r_1$, 1};
    \draw[->] (p2) -- (p1) node[midway,above] {$f$, $r_1$, 1};
\end{tikzpicture}
\]

Only active cycles may be "dangerous."
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)

\[ r_1 : q(f(X)) \leftarrow p(X). \]
\[ r_2 : p(X) \leftarrow q(X). \]

#### Example (Inactive Cycles)

\[ r_1 : p(f(X), g(X)) \leftarrow p(X, X). \]

Only active cycles may be “dangerous”.

S. Greco and C. Molinaro  
Termination Analysis of Logic Programs  
July 25th, 2015  
45 / 184
Argument-bounded Cycles

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

Example

\[ r_1 : q(f(X)) \leftarrow p(X), b(X) \]
\[ r_2 : p(X) \leftarrow q(X) \]

Since \( b[1] \) is limited, the number of values propagated in \( q[1] \) is finite.
Argument-bounded Cycles

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

However . . .

**Example**

\[ r_1 : \ q(f(X)) \leftarrow p(X), b(X) \]
\[ r_2 : \ p(X) \leftarrow q(X). \]

Since \( b[1] \) is limited, the number of values propagated in \( q[1] \) is finite.
**Twin Cycles**

### Example (List length)

- $r_0: \text{count}([a, b, c], 0)$.
- $r_1: \text{count}(L, s(I)) \leftarrow \text{count}([X|L], I)$.

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

The depth of terms for $\text{count}[1]$ decreases, while the depth of terms for $\text{count}[2]$ grows.
Example (List length)

\[
\begin{align*}
r_0 : & \quad \text{count\}(\text{[a, b, c]}, 0). & \quad \text{count\}(\text{[a, b, c]}, 0) \\
r_1 : & \quad \text{count\}(\text{L, s(I)}) \leftarrow \text{count\}([X|L], I). & \quad \text{count\}([b, c], s(0)) \\
\text{Query goal} : & \quad \text{count\}([], N). & \quad \text{count\}([], s(s(s(0))))
\end{align*}
\]

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

The growth of count[2] is bounded by the reduction of count[1].
Example (Append)

\[
\begin{align*}
&\text{magic_append}([a, b], [c, d]). \\
&\text{magic_append}(L1, L2) \leftarrow \text{magic_append}([X|L1], L2). \\
&\text{append}([], L, L) \leftarrow \text{magic_append}([], L). \\
&\text{append}([X|L1], L2, [X|L3]) \leftarrow \text{magic_append}([X|L1], L2), \\
&\text{append}(L1, L2, L3).
\end{align*}
\]
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).

Example

\[ r_1: q(X, Y) \leftarrow p(X, f(Y)) \]
\[ r_2: p(f(X), Y) \leftarrow q(X, Y) \]

\( \pi_1 \) and \( \pi_2 \) are twin cycles. The growth of values in \( \pi_1 \) is bounded by the reduction in \( \pi_2 \).
**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**

\[
\begin{align*}
  r_1 : & \quad q(X, Y) \leftarrow p(X, f(Y)). \\
  r_2 : & \quad p(f(X), Y) \leftarrow q(X, Y).
\end{align*}
\]
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**

\[
\begin{align*}
    r_1 : & \quad q(X, Y) \leftarrow p(X, f(Y)). \\
    r_2 : & \quad p(f(X), Y) \leftarrow q(X, Y).
\end{align*}
\]
**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**

\[
\begin{align*}
    r_1 & : q(X, Y) \leftarrow p(X, f(Y)). \\
    r_2 & : p(f(X), Y) \leftarrow q(X, Y).
\end{align*}
\]

\(\pi_1\) and \(\pi_2\) 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**

\[
\begin{align*}
  r_1 &: \quad q(X, Y) \leftarrow p(X, f(Y)). \\
  r_2 &: \quad p(f(X), Y) \leftarrow q(X, Y).
\end{align*}
\]

- \( \pi_1 \) and \( \pi_2 \) are *twin cycles*.
- The growth of values in \( \pi_1 \) is bounded by the reduction in \( \pi_2 \).
Bounded Programs

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

Example count($[a, b, c], 0$).

```plaintext
count(L, s(I)) ← count([X|L], I).
```
Both $\pi$ and $\pi'$ are active cycles; $\pi$ is failing, then $\text{count}[1]$ is limited (Condition 1); $\pi'$ is a twin of $\pi$. Since $\pi$ is not balanced and its arguments are limited, $\text{count}[2]$ is also limited (Condition 2).
Bounded Programs

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

Example

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

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

- Both \( \pi \) and \( \pi' \) are active cycles;
Bounded Programs

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

Example

\[
\text{count}([a,b,c], 0).
\]
\[
\overset{r}{\text{count}(L, s(I)) ← count([X|L], I)}.
\]

- Both $\pi$ and $\pi'$ are active cycles;
- $\pi$ is failing, then $\text{count}[1]$ is limited (Condition 1);
Bounded Programs

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

Example

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

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

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

Argument Restricted $\subseteq$ 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)**

\[
\begin{align*}
\text{bub}(L, [], []) & \leftarrow \text{input}(L). \\
\text{bub}([Y|T], [X|Cur], Sol) & \leftarrow \text{bub}([X|[Y|T]], Cur, Sol), X \leq Y. \\
\text{bub}([X|T], [Y|Cur], Sol) & \leftarrow \text{bub}([X|[Y|T]], Cur, Sol), Y < X. \\
\text{bub}(Cur, [], [X|Sol]) & \leftarrow \text{bub}([X|[]], Cur, Sol).
\end{align*}
\]
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)

\[
\begin{align*}
bub(L, [], []) & \leftarrow \text{input}(L). \\
bub([Y|T], [X|Cur], Sol) & \leftarrow \text{bub}([X|[Y|T]], Cur, Sol), X \leq Y. \\
bub([X|T], [Y|Cur], Sol) & \leftarrow \text{bub}([X|[Y|T]], Cur, Sol), Y < X. \\
bub(Cur, [], [X|Sol]) & \leftarrow \text{bub}([X|[]], Cur, Sol).
\end{align*}
\]
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)

\[
\text{visit}(\text{Tree},[],[]) \leftarrow \text{input}(\text{Tree}).
\]
\[
\text{visit}(<\text{Left},[\text{Root}|\text{Visited}], [\text{Right}|\text{ToVisit}]>) \leftarrow
\quad \text{visit}(\text{tree}(\text{Root}, \text{Left}, \text{Right}), \text{Visited}, \text{ToVisit}).
\]
\[
\text{visit}(\text{Next}, \text{Visited}, \text{ToVisit}) \leftarrow \text{visit}(<\text{null}, \text{Visited}, [\text{Next}|\text{ToVisit}]>).
\]
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)**

\[
\begin{align*}
\text{visit}(\text{Tree}, [], []) & \leftarrow \text{input}(\text{Tree}). \\
\text{visit}(\text{Left}, [\text{Root}|\text{Visited}], [\text{Right}|\text{ToVisit}]) & \leftarrow \\
& \quad \text{visit}(\text{tree}(\text{Root}, \text{Left}, \text{Right}), \text{Visited}, \text{ToVisit}). \\
\text{visit}(\text{Next}, \text{Visited}, \text{ToVisit}) & \leftarrow \text{visit}(\text{null}, \text{Visited}, [\text{Next}|\text{ToVisit}]).
\end{align*}
\]

**Example (List Concatenation)**

\[
\begin{align*}
\text{reverse}(L_1, []) & \leftarrow \text{input1}(L_1). \\
\text{reverse}(L_1, [X|L_2]) & \leftarrow \text{reverse}([X|L_1], L_2). \\
\text{append}(L_1, L_2) & \leftarrow \text{reverse}([], L_1), \text{input2}(L_2). \\
\text{append}(L_1, [X|L_2]) & \leftarrow \text{append}([X|L_1], L_2).
\end{align*}
\]
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:

\[ t = f(X, c, g(Y, Z)) \]
Rule-bounded programs - Notions of Size

Term size:

\[ t = f( X, c, g(Y, Z) ) \]

\[ \Downarrow \]

\[ size(t) = 3 + (x + 0 + size(g(Y, Z))) \]

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.

\[ A = p(t_1, \ldots, t_n) \]

\[ \Downarrow \]

\[ size(A) = \alpha p_1 \cdot size(t_1) + \ldots + \alpha p_n \cdot size(t_n) \]

Integer coefficients \( \alpha p_1, \ldots, \alpha p_n \) will be chosen depending on the program structure.
Rule-bounded programs - Notions of Size

Term size:

\[ t = f(X, c, g(Y, Z)) \]

\[ \Downarrow \]

\[ \text{size}(t) = 3 + (x + 0 + \text{size}(g(Y, Z))) = 3 + (x + 0 + (2 + y + z)) \]
Rule-bounded programs - Notions of Size

Term size:

\[ t = f( X, c, g(Y, Z) ) \]

\[ \Downarrow \]

\[ \text{size}(t) = 3 + (x + 0 + \text{size}(g(Y, Z))) \]

\[ = 3 + (x + 0 + (2 + y + z)) \]

**Intuition:** A template for all possible sizes the term may have during the program evaluation.
Rule-bounded programs - Notions of Size

Term size:

\[ t = f( X, c, g( Y, Z ) ) \]
\[ \downarrow \]
\[ size(t) = 3 + ( x + 0 + size(g( Y, Z ))) \]
\[ = 3 + ( x + 0 + ( 2 + y + z ) ) \]

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:

\[ t = f(X, c, g(Y, Z)) \]

\[ \Downarrow \]

\[ size(t) = 3 + (x + 0 + size(g(Y, Z))) \]

\[ = 3 + (x + 0 + (2 + y + z)) \]

**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:

\[ t = f(X, c, g(Y, Z)) \]

\[ \Rightarrow \]

\[ \text{size}(t) = 3 + (x + 0 + \text{size}(g(Y, Z))) \]

\[ = 3 + (x + 0 + (2 + y + z)) \]

**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.

\[ A = p(t_1, \ldots, t_n) \]

\[ \Rightarrow \]

\[ \text{size}(A) = \alpha_{p_1} \cdot \text{size}(t_1) + \ldots + \alpha_{p_n} \cdot \text{size}(t_n) \]
Rule-bounded programs - Notions of Size

Term size:

\[ t = f(X, c, g(Y, Z)) \]
\[ \Downarrow \]
\[ size(t) = 3 + (x + 0 + size(g(Y, Z))) \]
\[ = 3 + (x + 0 + (2 + y + z)) \]

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.

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

Integer coefficients \( \alpha_{p_1}, \ldots, \alpha_{p_n} \) will be chosen depending on the program structure.
Rule-bounded program - Example

Example (List Length)

\[
\begin{align*}
  r_1 &: \text{len}([a,b,c,d], 0). \\
  r_2 &: \text{len}(\text{Tail}, \text{s}(N)) \leftarrow \text{len}(\text{list}(\text{Head}, \text{Tail}), N).
\end{align*}
\]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[ \text{size}(%\text{body}(r_2)) \geq \text{size}(%\text{head}(r_2)) \]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[ \text{size}(\text{body}(r_2)) \geq \text{size}(\text{head}(r_2)) \]

\[ \alpha_1 \cdot (2 + \text{head} + \text{tail}) \]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[ \text{size}(\text{body}(r_2)) \geq \text{size}(\text{head}(r_2)) \]

\[ \alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[
\text{size}(\text{body}(r_2)) \geq \text{size}(\text{head}(r_2))
\]

\[
\alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail}
\]
Example (List Length)

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

We need to check:

\[ \text{size(body}(r_2)) \geq \text{size(head}(r_2)) \]
\[ \alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 \cdot (1 + n) \]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[ \text{size}(\text{body}(\text{r}_2)) \geq \text{size}(\text{head}(\text{r}_2)) \]
\[ \alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 \cdot (1 + n) \]

Find \( \alpha_1 \) and \( \alpha_2 \) s.t. the inequality holds for all \( \text{head}, \text{tail}, n \in \mathbb{N} \).
Rule-bounded program - Example

**Example (List Length)**

\[
\begin{align*}
  r_1 & : \text{len}([a, b, c, d], 0). \\
  r_2 & : \text{len}(\text{Tail}, \text{s}(N)) \leftarrow \text{len}(\text{list}(\text{Head}, \text{Tail}), N).
\end{align*}
\]

We need to check:

\[
\text{size(body}(r_2)) \geq \text{size(head}(r_2))
\]

\[
\alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 \cdot (1 + n)
\]

Find \(\alpha_1\) and \(\alpha_2\) s.t. the inequality holds for all \(\text{head}, \text{tail}, n \in \mathbb{N}\)

\[
2 \cdot \alpha_1 + \alpha_1 \cdot \text{head} + \alpha_1 \cdot \text{tail} + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 + \alpha_2 \cdot n
\]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[ \text{size(body}(r_2)) \geq \text{size(head}(r_2)) \]
\[ \alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 \cdot (1 + n) \]

Find \( \alpha_1 \) and \( \alpha_2 \) s.t. the inequality holds for all \( \text{head}, \text{tail}, n \in \mathbb{N} \)

\[ 2 \cdot \alpha_1 + \alpha_1 \cdot \text{head} + \alpha_1 \cdot \text{tail} + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 + \alpha_2 \cdot n \]
\[ 2 \cdot \alpha_1 \geq \alpha_2 \]
Rule-bounded program - Example

Example (List Length)

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

We need to check:

\[ \text{size(body}(r_2)) \geq \text{size(head}(r_2)) \]
\[ \alpha_1 \cdot (2 + \text{head} + \text{tail}) + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 \cdot (1 + n) \]

Find \( \alpha_1 \) and \( \alpha_2 \) s.t. the inequality holds for all \( \text{head}, \text{tail}, n \in \mathbb{N} \)

\[ 2 \cdot \alpha_1 + \alpha_1 \cdot \text{head} + \alpha_1 \cdot \text{tail} + \alpha_2 \cdot n \geq \alpha_1 \cdot \text{tail} + \alpha_2 + \alpha_2 \cdot n \]
\[ 2 \cdot \alpha_1 \geq \alpha_2 \]

We can choose \( \alpha_1 = \alpha_2 = 1 \Rightarrow \text{the program is rule-bounded.} \)
Example (Bubble sort)

\[
\text{sort}([b, a, d, h, e], [], []). \\
\text{sort}([Y|T], [X|Temp], \text{Sorted}) \leftarrow \text{sort}([X|[Y\mid T]], \text{Temp, Sorted}), X \leq Y. \\
\text{sort}([X|T], [Y|Temp], \text{Sorted}) \leftarrow \text{sort}([X|[Y\mid T]], \text{Temp, Sorted}), Y < X. \\
\text{sort}(\text{Temp, []}, [X|\text{Sorted}]) \leftarrow \text{sort}([X], \text{Temp, Sorted})).
\]

\[
\begin{align*}
\alpha_1 \cdot (4 + x + y + t) + \alpha_2 \cdot \text{temp} + \alpha_3 \cdot \text{sorted} & \geq \\
\alpha_1 \cdot (2 + y + t) + \alpha_2 \cdot (2 + x + \text{temp}) + \alpha_3 \cdot \text{sorted} \\
\alpha_1 \cdot (4 + x + y + t) + \alpha_2 \cdot \text{temp} + \alpha_3 \cdot \text{sorted} & \geq \\
\alpha_1 \cdot (2 + x + t) + \alpha_2 \cdot (2 + y + \text{temp}) + \alpha_3 \cdot \text{sorted} \\
\alpha_1 \cdot (2 + x) + \alpha_2 \cdot \text{temp} + \alpha_3 \cdot \text{sorted} & \geq \\
\alpha_1 \cdot \text{temp} + \alpha_3 \cdot (2 + x + \text{sorted})
\end{align*}
\]

A possible solution is \(\alpha_1 = 2, \alpha_2 = 2, \alpha_3 = 1\)
Relative Expressivity

Theorem

- $\text{Finite Domain } \subsetneq \text{Rule-bounded}$.
- $\lambda$-Restricted $\subsetneq \text{Rule-bounded}$.
- Argument Restricted $\nsubseteq$ Rule-bounded.
- Bounded $\nsubseteq$ Rule-bounded.
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 $P_{\mu}$. The aim is to apply termination criteria to the adorned program $P_{\mu}$ rather than the original program $P$. 

S. Greco and C. Molinaro
Termination Analysis of Logic Programs
July 25th, 2015 61 / 184
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 $P^\mu$.

The aim is to apply termination criteria to the adorned program $P^\mu$ 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 $P^\mu$.
- Then, we apply criterion $C$ to $P^\mu$ (rather than the original program $P$).
- (Soundness) If $P^\mu$ 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**

\[
p(X,X) \leftarrow \text{base}(X)
\]
\[
q(X,Y) \leftarrow p(X,Y)
\]
\[
p(f(X),g(X)) \leftarrow q(X,X)
\]
Example

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.

**Original program**

\[
\begin{align*}
  p(X,X) & \leftarrow \text{base}(X) \\
  q(X,Y) & \leftarrow p(X,Y) \\
  p(f(X),g(X)) & \leftarrow q(X,X)
\end{align*}
\]

**Adorned program**

\[
\begin{align*}
  p^{εε}(X,X) & \leftarrow \text{base}^ε(\text{X}) \\
  q^{εε}(X,Y) & \leftarrow p^{εε}(X,Y) \\
  p^{f_{1g_1}}(f(X),g(X)) & \leftarrow q^{εε}(X,X) \\
  q^{f_{1g_1}}(X,Y) & \leftarrow p^{f_{1g_1}}(X,Y)
\end{align*}
\]
Example

**Original program**

\[
p(X, X) \leftarrow \text{base}(X)
\]

\[
q(X, Y) \leftarrow p(X, Y)
\]

\[
p(f(X), g(X)) \leftarrow q(X, X)
\]

**Adorned program**

\[
p^{\varepsilon\varepsilon}(X, X) \leftarrow \text{base}^\varepsilon(X)
\]

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

\[
p^{f_{ig1}}(f(X), g(X)) \leftarrow q^{\varepsilon\varepsilon}(X, X)
\]

\[
q^{f_{ig1}}(X, Y) \leftarrow p^{f_{ig1}}(X, Y)
\]

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**

\[ p(X,f(X)) \leftarrow \text{base}(X) \]
\[ p(X,f(X)) \leftarrow p(Y,X),\text{base}(Y) \]
\[ p(X,Y) \leftarrow p(f(X),f(Y)) \]
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**

\[base^e\]

**Adornment definitions**
Adornment Algorithm

Original program

\[ p(X, f(X)) \leftarrow \text{base}(X) \]
\[ p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y) \]
\[ p(X, Y) \leftarrow p(f(X), f(Y)) \]

Adorned program

Adorned predicate symbols

\[ \text{base}^e \]

Adornment definitions
Adornment Algorithm

**Original program**

\[
p(X, f(X)) \leftarrow \text{base}(X)
\]

\[
p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y)
\]

\[
p(X, Y) \leftarrow p(f(X), f(Y))
\]

**Adorned program**

\[
\leftarrow \text{base}^\varepsilon(X)
\]

Adorned predicate symbols

\[
\text{base}^\varepsilon
\]

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

\[
p^{e_{f_1}}(X, f(X)) \leftarrow base^e(X)
\]

Adorned predicate symbols

\[
base^e
\]

Adornment definitions

\[
p^{e_{f_1}}
\]

\[
f_1 = f(\varepsilon)
\]
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

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

Adorned predicate symbols

\[ base^\varepsilon \]
\[ p^{\varepsilon f_1} \]

Adornment definitions

\[ f_1 = f(\varepsilon) \]
Adornment Algorithm

**Original program**

\[
p(X, f(X)) \leftarrow \text{base}(X)
\]
\[
p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y)
\]
\[
p(X, Y) \leftarrow p(f(X), f(Y))
\]

**Adorned program**

\[
p^\varepsilon_{f_1}(X, f(X)) \leftarrow \text{base}^\varepsilon(X)
\]

**Adorned predicate symbols**

- \( \text{base}^\varepsilon \)
- \( p^\varepsilon_{f_1} \)

**Adornment definitions**

\[
f_1 = f(\varepsilon)
\]
Adorned Algorithm

Original program
\[ p(X, f(X)) \leftarrow \text{base}(X) \]
\[ p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y) \]
\[ p(X, Y) \leftarrow p(f(X), f(Y)) \]

Adorned program
\[ p^{eff_1}(X, f(X)) \leftarrow \text{base}^e(X) \]
\[ \leftarrow p^{eff_1}(Y, X), \text{base}^e(Y) \]

Adorned predicate symbols
- \( \text{base}^e \)
- \( p^{eff_1} \)

Adornment definitions
- \( f_1 = f(\varepsilon) \)
Adornment Algorithm

Original program

\[ p(X, f(X)) \leftarrow \text{base}(X) \]
\[ p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y) \]
\[ p(X, Y) \leftarrow p(f(X), f(Y)) \]

Adorned program

\[ p^{e_{f_1}}(X, f(X)) \leftarrow \text{base}^e(X) \]
\[ p^{f_1f_2}(X, f(X)) \leftarrow p^{e_{f_1}}(Y, X), \text{base}^e(Y) \]

Adorned predicate symbols

- \text{base}^e
- \text{p}^{e_{f_1}}
- \text{p}^{f_1f_2}

Adornment definitions

- \[ f_1 = f(\varepsilon) \]
- \[ f_2 = f(f_1) \]
**Adornment Algorithm**

**Original program**

\[ p(X, f(X)) \leftarrow \text{base}(X) \]
\[ p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y) \]
\[ p(X, Y) \leftarrow p(f(X), f(Y)) \]

**Adorned program**

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

<table>
<thead>
<tr>
<th>Adorned predicate symbols</th>
<th>Adornment definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{base}^\varepsilon)</td>
<td>(f_1 = f(\varepsilon))</td>
</tr>
<tr>
<td>(p^{\varepsilon f_1})</td>
<td>(f_2 = f(f_1))</td>
</tr>
<tr>
<td>(p^{f_1 f_2})</td>
<td></td>
</tr>
</tbody>
</table>
Adorned Algorithm

**Original program**

\[
p(X, f(X)) \leftarrow \text{base}(X)
\]

\[
p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y)
\]

\[
p(X, Y) \leftarrow p(f(X), f(Y))
\]

**Adorned program**

\[
p^{e_1}(X, f(X)) \leftarrow \text{base}^e(X)
\]

\[
p^{f_1f_2}(X, f(X)) \leftarrow p^{e_1}(Y, X), \text{base}^e(Y)
\]

**Adorned predicate symbols**

\[
\begin{align*}
\text{base}^e \\
p^{e_1} \\
p^{f_1f_2}
\end{align*}
\]

**Adornment definitions**

\[
\begin{align*}
f_1 & = f(e) \\
f_2 & = f(f_1)
\end{align*}
\]
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**

\[
p^{\epsilon_1}(X, f(X)) \leftarrow base^\epsilon(X)
\]

\[
p^{\epsilon_2}(X, f(X)) \leftarrow p^{\epsilon_1}(Y, X), base^\epsilon(Y)
\]

\[\downarrow\]

\[
p^{\epsilon_2}(f(X), f(Y))
\]

**Adorned predicate symbols**

- \(base^\epsilon\)
- \(p^{\epsilon_1}\)
- \(p^{\epsilon_2}\)

**Adornment definitions**

\[
f_1 = f(\epsilon)
\]

\[
f_2 = f(f_1)
\]
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

\[ p^{e_f_1}(X, f(X)) \leftarrow base^e(X) \]
\[ p^{f_1,f_2}(X, f(X)) \leftarrow p^{e_f_1}(Y, X), base^e(Y) \]
\[ p^{e_f_1}(X, Y) \leftarrow p^{f_1,f_2}(f(X), f(Y)) \]

Adorned predicate symbols

\[ base^e \]
\[ p^{e_f_1} \]
\[ p^{f_1,f_2} \]

Adornment definitions

\[ f_1 = f(\epsilon) \]
\[ f_2 = f(f_1) \]
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**

<table>
<thead>
<tr>
<th>Adorned predicate symbols</th>
<th>Adornment definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p^{\varepsilon f_1}(X, f(X)) \leftarrow base^\varepsilon(X) )</td>
<td>( p^{\varepsilon f_1} )</td>
</tr>
<tr>
<td>( p^{f_1 f_2}(X, f(X)) \leftarrow p^{\varepsilon f_1}(Y, X), base^\varepsilon(Y) )</td>
<td>( f_1 = f(\varepsilon) )</td>
</tr>
<tr>
<td>( p^{\varepsilon f_1}(X, Y) \leftarrow p^{f_1 f_2}(f(X), f(Y)) )</td>
<td>( f_2 = f(f_1) )</td>
</tr>
</tbody>
</table>

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

**Original program**
\[ p(X, f(X)) \leftarrow \text{base}(X) \]
\[ p(X, f(X)) \leftarrow p(Y, X), \text{base}(Y) \]
\[ p(X, Y) \leftarrow p(f(X), f(Y)) \]

**Adorned program**
\[ p^{\varepsilon_{f_1}}(X, f(X)) \leftarrow \text{base}^\varepsilon(X) \]
\[ p^{f_1f_2}(X, f(X)) \leftarrow p^{\varepsilon_{f_1}}(Y, X), \text{base}^\varepsilon(Y) \]
\[ p^{\varepsilon_{f_1}}(X, Y) \leftarrow p^{f_1f_2}(f(X), f(Y)) \]

**Adorned predicate symbols**
- base\(^\varepsilon\)
- \(p^{\varepsilon_{f_1}}\)
- \(p^{f_1f_2}\)

**Adornment definitions**
- \(f_1 = f(\varepsilon)\)
- \(f_2 = f(f_1)\)

\[ p^{f_1f_2}(Y, X), \text{base}^\varepsilon(Y) \] is not coherently adorned because \(Y\) is associated with the two different adornment symbols \(f_1\) and \(\varepsilon\)
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theorem</strong></td>
</tr>
<tr>
<td><em>The adornment algorithm always terminates.</em></td>
</tr>
</tbody>
</table>
Theorem

The adornment algorithm always terminates.

Theorem

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

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>The adornment algorithm always terminates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $P$ be a program and $P^\mu$ the adorned version of $P$. Then, the least model of $P$ is equal to the least model of $P^\mu$ with adornments dropped from predicates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $P$ be a program and $P^\mu$ the adorned version of $P$. If $P^\mu$ satisfies a termination criterion $C$, then the evaluation of $P \cup D$ terminates for any finite set of (flat) database facts $D$.</td>
</tr>
</tbody>
</table>
Properties

Theorem

The adornment algorithm always terminates.

Theorem

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

Theorem

Let $P$ be a program and $P^\mu$ the adorned version of $P$. If $P^\mu$ 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

Definition

A Datalog ∨, ¬ rule is of the form

\[ A_1 \lor \cdots \lor A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n \]

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

A Datalog ∨, ¬ program is a finite set of Datalog ∨, ¬ rules.

Semantics: Stable Model Semantics.

We want to check if a Datalog ∨, ¬ program has a finite number of stable models, each of finite size and that can be computed.
Dealing with Negation and Disjunction

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A Datalog(^\lor,\neg) rule</strong> is of the form</td>
</tr>
<tr>
<td>(A_1 \lor \cdots \lor A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n)</td>
</tr>
</tbody>
</table>

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

**A Datalog\(^\lor,\neg\) program** is a finite set of Datalog\(^\lor,\neg\) rules.
Dealing with Negation and Disjunction

Definition

A Datalog\(^{\lor, \neg}\) rule is of the form

\[
A_1 \lor \cdots \lor A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n
\]

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

A Datalog\(^{\lor, \neg}\) program is a finite set of Datalog\(^{\lor, \neg}\) rules.

Semantics: **Stable Model Semantics.**

We want to check if a Datalog\(^{\lor, \neg}\) 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 $\lor, \neg$ 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\(^{\lor,\neg}\) 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
\[
A_1 \lor \cdots \lor A_m \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_n
\]
in \(P\) is replaced with \(m\) Datalog rules
\[
A_i \leftarrow B_1, \ldots, B_k \quad 1 \leq i \leq m
\]
Dealing with Negation and Disjunction

We want to check if a Datalog $\lor, \neg$ 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

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

in $P$ is replaced with $m$ Datalog rules

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

**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., $\text{FDNC}$ 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.

2. Exploiting negation and disjunction.
   Example:
   \[ p(f(X)) \leftarrow p(X), \neg p(X) \]
   will be analyzed like
   \[ p(f(X)) \leftarrow p(X), \neg p(X) \]

3. Interpreted function symbols.

4. Testing Local Stratification.

S. Greco and C. Molinaro
Termination Analysis of Logic Programs
July 25th, 2015 85 / 184
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

\[ p(f(X)) \leftarrow p(X), \neg p(X) \]
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

\[ p(f(X)) \leftarrow p(X), \neg p(X) \]

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

\[ p(f(X)) \leftarrow p(X), \neg p(X) \]

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

$$emp(Emp\#, \ Name, \ Address) \quad worksFor(Emp\#, \ Prj\#)$$
Integrity constraints in databases

Example

\[ \text{emp}(\text{Emp}\#, \text{Name}, \text{Address}) \] \[ \text{worksFor}(\text{Emp}\#, \text{Prj}\#) \]

- Inclusion dependencies and foreign keys:
  \[ \text{worksFor}(E, P) \to \exists N \exists A \text{emp}(E, N, A) \]

- Functional Dependencies and internal keys
  \[ \text{emp}(E, N_1, Pr_1) \land \text{emp}(E, N_2, Pr_2) \to N_1 = N_2 \]
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

empA(Emp#, Name, Address, Salary)

Company B

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

<table>
<thead>
<tr>
<th>Company A</th>
<th>Company B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$empA(Emp#, \ Name, \ Address, \ Salary)$</td>
<td>$empB(Emp#, \ Name, \ Phone, \ Salary)$</td>
</tr>
</tbody>
</table>

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:

\[
\forall X \text{emp}(X) \rightarrow \text{person}(X)
\]

(Inverse) Relation Inclusion:

\[
\forall X \forall Y \text{manages}(X, Y) \rightarrow \text{isManaged}(Y, X)
\]

Relation Transitivity:

\[
\forall X \forall Y \forall Z \text{mgs}(X, Y), \text{mgs}(Y, Z) \rightarrow \text{mgs}(X, Z)
\]

Participation:

\[
\forall X \text{emp}(X) \rightarrow \exists Y \text{report}(X, Y)
\]

Disjointness:

\[
\forall X \text{emp}(X), \text{customer}(X) \rightarrow \text{false}
\]

Functionality:

\[
\forall X \forall Y \forall Z \text{reports}(X, Y), \text{reports}(X, Z) \rightarrow Y = Z
\]
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) $\Sigma$,
- 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) $\Sigma$,
- 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) \( \Sigma \),
- 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):**
  \[
  \forall X \forall Y \varphi(X, Y) \rightarrow \exists Z \psi(X, Z)
  \]

- **Equality generating dependencies (EGDs):**
  \[
  \forall X \varphi(X) \rightarrow X_1 = X_2
  \]

\(\varphi(X, Y), \varphi(X)\) and \(\psi(Z, X)\) are conjunctions of atoms, \(X_1, X_2 \in X\).
Answering queries under constraints

The Problem

Data dependencies:

- **Tuple generating dependencies (TGDs):**

\[ \forall X \forall Y \varphi(X, Y) \rightarrow \exists Z \psi(X, Z) \]

- **Equality generating dependencies (EGDs):**

\[ \forall X \varphi(X) \rightarrow X_1 = X_2 \]

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

\( K = D \cup \Sigma \) is called *knowledge base*.
Answering queries under constraints

Example (Models and answers)

- **Database:** \( D = \{ \text{person}(john) \} \)
- **Data dependencies** \( \Sigma \):

\[
\forall X \text{ person}(X) \rightarrow \exists Z \text{ fatherOf}(Z, X) \\
\forall X \forall Y \text{ fatherOf}(X, Y), \text{ person}(Y) \rightarrow \text{ person}(X)
\]
Answering queries under constraints

Example (Models and answers)

- **Database:** \( D = \{ \text{person}(\text{john}) \} \)
- **Data dependencies** \( \Sigma \):
  \[
  \forall X \ \text{person}(X) \rightarrow \exists Z \ \text{fatherOf}(Z, X) \\
  \forall X \ \forall Y \ \text{fatherOf}(X, Y), \text{person}(Y) \rightarrow \text{person}(X)
  \]
- **Queries:**
  \[
  Q_1 = \exists X \ \text{fatherOf}(X, \text{john}) \\
  Q_2 = \exists X \ \text{fatherOf}(\text{john}, X)
  \]
Answering queries under constraints

Example (Models and answers)

- Database: \( D = \{ \text{person}(john) \} \)
- Data dependencies \( \Sigma \):
  \[
  \forall X \text{ person}(X) \rightarrow \exists Z \text{ fatherOf}(Z, X) \\
  \forall X \forall Y \text{ fatherOf}(X, Y), \text{ person}(Y) \rightarrow \text{ person}(X)
  \]
- Queries:
  \[
  Q_1 = \exists X \text{ fatherOf}(X, john) \\
  Q_2 = \exists X \text{ fatherOf}(john, X)
  \]
- Answers:
  \[
  D \cup \Sigma \models Q_1 \quad \text{certain}(Q_1, (D, \Sigma)) = "yes" \\
  D \cup \Sigma \not\models Q_2 \quad \text{certain}(Q_2, (D, \Sigma)) = "no"
  \]

All models of \( D \cup \Sigma \) contain an atom \( \text{fatherOf}(x, john) \).
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, $\mathcal{ELHI}$, F-Logic Lite).

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

⇒

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 \( \bot_i \))*), representing the presence of unknown values.

Definition (Possible worlds (under CWA))
Given a possibly incomplete database \( D \), \( \text{POSS}(D) \) 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 $D$, $\text{POSS}(D)$ denotes the set of ground databases obtained from $D$ by replacing nulls with constants.

Example (POSS(D))

- $D = \{ \text{person}(john), \text{person}(frank), \text{fatherOf}(\perp_1, john) \}$
Databases may be *incomplete*, that is may contain *(labeled)* nulls (of the form $\bot_i$), representing the presence of unknown values.

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

Example (POSS(D))

- $D = \{\text{person}(\text{john}), \text{person}(\text{frank}), \text{fatherOf}(\bot_1, \text{john})\}$
- $\text{POSS}(D)$ (under CWA) contains:
  - $\{\text{person}(\text{john}), \text{person}(\text{frank}), \text{fatherOf}(\text{john}, \text{john})\}$
  - $\{\text{person}(\text{john}), \text{person}(\text{frank}), \text{fatherOf}(\text{frank}, \text{john})\}$
Answering queries over incomplete databases

**Definition (certain answer)**

- $\text{certain}(D) = \text{database derived from } D \text{ by deleting tuples with nulls.}$
- $\text{certain}(Q, D) = \bigcap \{ Q(R) \mid R \in \text{POSS}(D) \}$
Answering queries over incomplete databases

**Definition (certain answer)**

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

**Theorem (weak representation systems)**

*For union of conjunctive queries*

\[ \text{certain}(Q(D)) = \text{certain}(Q, D) \]

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 $h : \text{Nulls} \rightarrow \text{Nulls} \cup \text{Constants}$.
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 $h : \text{Nulls} \rightarrow \text{Nulls} \cup \text{Constants}$.

Definition (Possible worlds under OWA)
$\text{POSS}(M) = \{ R \mid h(M) \subseteq R \land R \text{ is ground} \}$. 
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 \( h : \text{Nulls} \rightarrow \text{Nulls} \cup \text{Constants} \).

Definition (Possible worlds under OWA)
\[ \text{POSS}(M) = \{ R \mid h(M) \subseteq R \land R \text{ is ground} \} \]

Definition (certain answer)
\[ \text{certain}(Q, (D, \Sigma)) = \bigcap \{ Q(R) \mid R \in \text{POSS}(M) \land M \text{ is a model of } D \cup \Sigma \} \]
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 $M_2$ ($M_1 \sqsubset M_2$) 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 \textit{at least as general} as $M_2$ ($M_1 \supseteq M_2$) if $\exists h$ such that $h(M_1) \subseteq M_2$.

$M_1$ is \textit{more general} than $M_2$ ($M_1 \supset M_2$) if $M_1 \supseteq M_2$ and $M_2 \not\supseteq M_1$.

Theorem

- $M_1 \supseteq M_2$ iff $\text{POSS}(M_1) \supseteq \text{POSS}(M_2)$,
- $M_1 \subseteq M_2$ $\Rightarrow$ $M_1 \supseteq M_2$. 
Universal models

Definition (Universal model)

*M* is an **universal model** (or **universal solution**) if for every model *N*,

\[ M \sqsupseteq N \text{ (i.e. } \exists h \text{ s.t. } h(M) \subseteq N) \].
Universal models

Definition (Universal model)

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

Theorem (Main Th.)

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

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

Recall that:

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

Example (Models and answers)

- **Database**: $D = \{person(john)\}$
- **Data dependencies $\Sigma$:**
  \[
  \forall X \ person(X) \rightarrow \exists Z \ fatherOf(Z, X) \\
  \forall X \ \forall Y \ fatherOf(X, Y), \ person(Y) \rightarrow person(X)
  \]
- **Models (under OWA):**
  \[
  M_1 = \{person(john), fatherOf(john, john)\} \\
  M_2 = \{person(john), fatherOf(⊥_1, john), person(⊥_1)\} \\
  M_3 = \{person(john), fatherOf(⊥_2, john), person(⊥_2)\} \\
  M_4 = \{person(john), fatherOf(⊥_1, john), person(frank)\} \\
  ... 
  \]
Universal models

Example (Models and answers)

- Database: \( D = \{ \text{person}(john) \} \)
- Data dependencies \( \Sigma \):

\[
\forall X \ \text{person}(X) \rightarrow \exists Z \ \text{fatherOf}(Z, X) \\
\forall X \ \forall Y \ \text{fatherOf}(X, Y), \text{person}(Y) \rightarrow \text{person}(X)
\]

- Models (under OWA):

\[
M_1 = \{ \text{person}(john), \text{fatherOf}(john, john) \} \\
M_2 = \{ \text{person}(john), \text{fatherOf}(\bot_1, john), \text{person}(\bot_1) \} \\
M_3 = \{ \text{person}(john), \text{fatherOf}(\bot_2, john), \text{person}(\bot_2) \} \\
M_4 = \{ \text{person}(john), \text{fatherOf}(\bot_1, john), \text{person}(\text{frank}) \}
\]

\[ 
M_2 \sqsupset M_1, M_2 \sqsupset M_4, M_2 \sqsupseteq M_3, \quad M_3 \sqsubset M_1, M_3 \sqsupset M_4, M_3 \sqsupseteq M_2
\]

\( M_2 \) 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.
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
The Chase

Chase algorithm \textit{chase}(D, \Sigma)

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 extension \( h' \) 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 \textit{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

\[ D : \quad \sum : \]

\[
\begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) & \quad S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*}
\]
Chase - Enforcing data dependencies: a first example

Example

\[ D : \quad \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y E(X, Y) \]
\[ S(a) \quad S(X) \land E(X, Y) \rightarrow N(Y) \]

\[ \text{chase}(D, \Sigma) = \{ N(a), S(a) \} \]
Example

\[ \begin{align*}
D & : \\
\Sigma & : \\
N(a) & : N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) & : S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*} \]

\[ \text{chase}(D, \Sigma) = \{ N(a), S(a) \} \]
Example

\[
\begin{align*}
D & : & \Sigma & : \\
N(a) & : & N(X) & \rightarrow \exists Y \ E(X, Y) \\
S(a) & : & S(X) \land E(X, Y) & \rightarrow N(Y)
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ \ N(a), \ S(a), \ E(a, \bot_1) \}
\]
Example

\[ D : \quad \Sigma : \]

\[
\begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) & \quad S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ \ N(a), \ S(a), \ E(a, \perp_1) \}
\]
Chase - Enforcing data dependencies: a first example

Example

<table>
<thead>
<tr>
<th>$D:$</th>
<th>$\Sigma:$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N(a)$</td>
<td>$N(X) \rightarrow \exists Y \ E(X, Y)$</td>
</tr>
<tr>
<td>$S(a)$</td>
<td>$S(X) \land E(X, Y) \rightarrow N(Y)$</td>
</tr>
</tbody>
</table>

$\text{chase}(D, \Sigma) = \{ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1) \}$
Example

\[ D : \quad \Sigma : \]

\[
\begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) & \quad S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*}
\]

\[
chase(D, \Sigma) = \{ \ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1) \}
\]
Chase - Enforcing data dependencies: a first example

Example

\[
\begin{align*}
D : & \quad \Sigma : \\
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) & \quad S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*}
\]

\[\text{chase}(D, \Sigma) = \{ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2) \}\]

All TGDs are satisfied: STOP.

This and every other chase sequence terminates.

S. Greco and C. Molinaro
Termination Analysis of Logic Programs
July 25\textsuperscript{th}, 2015 112 / 184
Chase - Enforcing data dependencies: a first example

Example

\[
D : \quad \Sigma : \\
N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) \quad S(X) \land E(X, Y) \rightarrow N(Y)
\]

\[
\text{chase}(D, \Sigma) = \{ \ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2) \}
\]

All dependencies are satisfied: STOP.

This and every other chase sequence terminates.
Chase - Enforcing data dependencies: a first example

Example

\[
D : \quad \Sigma :
\]

\[
N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) \quad S(X) \land E(X, Y) \rightarrow N(Y)
\]

\[
\text{chase}(D, \Sigma) = \{ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2) \}
\]
Example

\[ D : \]
\[ N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \]
\[ S(a) \quad S(X) \land E(X, Y) \rightarrow N(Y) \]

\[ \text{chase}(D, \Sigma) = \{ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2) \} \]

All dependencies are satisfied: STOP.

This and every other chase sequence terminates.
Chase - Enforcing data dependencies: a first example

Example

\[ D : \begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y E(X, Y) \\
S(a) & \quad S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*} \]

\[ \text{chase}(D, \Sigma) = \{ N(a), S(a), E(a, \bot_1), N(\bot_1), E(\bot_1, \bot_2) \} \]
Chase - Enforcing data dependencies: a first example

Example

\[ D : \quad \Sigma : \]

\[
\begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
S(a) & \quad S(X) \land E(X, Y) \rightarrow N(Y)
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ \ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2) \}
\]
Chase - Enforcing data dependencies: a first example

Example

\[ D : \quad \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \]

\[ S(a) \quad S(X) \wedge E(X, Y) \rightarrow N(Y) \]

\[ \text{chase}(D, \Sigma) = \{ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2), \ N(\bot_2), \ldots \} \]

There is no terminating chase sequence.
Example

\[
D : \quad \Sigma :
\]
\[
N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y)
\]
\[
S(a) \quad \exists X \ E(X, Y) \rightarrow N(Y)
\]

\[
\text{chase}(D, \Sigma) = \{ \ N(a), \ S(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2), \ N(\bot_2), \ \ldots \}\n\]

There is **no** terminating chase sequence.
Example (\(\exists\) a finite sequence)

\[
\begin{align*}
D : & \\
\text{airport}(a) & \\
r_1 : & \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
r_2 : & \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
r_3 : & \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \(D\):

\[
\begin{align*}
\text{flight}(a, \bot) & \\
\text{airport}(\bot) & \\
\text{flight}(\bot, a) & \\
\end{align*}
\]

No further rule is applicable: STOP.
Chase – Terminating Sequence

Example (∃ a finite sequence)

\[
\begin{align*}
D : & \\
\text{airport}(a) & \\
r_1 : & \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
r_2 : & \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
r_3 : & \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \( D \):

\[
\begin{align*}
\text{flight}(a, \perp_1) \\
\text{airport}(\perp_1) \\
\text{flight}(\perp_1, a)
\end{align*}
\]

No further rule is applicable: STOP.
Chase – Terminating Sequence

Example ($\exists$ a finite sequence)

\[
\begin{align*}
D : & \quad \Sigma : \\
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) & \\
r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to $D$:

- \text{flight}(a, \bot_1)
- \text{airport}(\bot_1)
- \text{flight}(\bot_1, a)

No further rule is applicable: STOP.
Chase – Terminating Sequence

Example (∃ a finite sequence)

\[
\begin{align*}
D : & \quad \Sigma : \\
& \quad \quad \text{airport}(a) \\
& \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
& \quad r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
& \quad r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \\
\end{align*}
\]

The following facts are added to \( D \):

\[
\text{flight}(a, \bot_1)
\]
Example (∃ a finite sequence)

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$airport(a)$</td>
<td>$r_1 : airport(X) \rightarrow \exists Y \text{ flight}(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2 : \text{ flight}(X, Y) \rightarrow airport(X) \land airport(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3 : \text{ flight}(X, Y) \rightarrow \text{ flight}(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D$:

$flight(a, \bot_1)$
### Example (∃ a finite sequence)

<table>
<thead>
<tr>
<th></th>
<th>( D : )</th>
<th>( \Sigma : )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{airport}(a) )</td>
<td>( r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) )</td>
<td>( r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) )</td>
</tr>
</tbody>
</table>

The following facts are added to \( D \):

\( \text{flight}(a, \bot_1) \)
**Example (exists a finite sequence)**

\[
D : \\
\exists \ a \ finite \ sequence
\]

\[
Σ :
\]

- `airport(a)`
- `r₁ : airport(X) → ∃Y flight(X, Y)`
- `r₂ : flight(X, Y) → airport(X) ∧ airport(Y)`
- `r₃ : flight(X, Y) → flight(Y, X)`

The following facts are added to \( D : \)

- `flight(a, ⊥₁)`
Chase – Terminating Sequence

Example (exists a finite sequence)

\[ D \quad \Sigma \]

\[
\begin{align*}
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
\text{flight}(X, Y) & \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
\text{flight}(X, Y) & \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \( D \):

\[
\text{flight}(a, \perp_1)
\]
Chase – Terminating Sequence

Example (∃ a finite sequence)

\[ D : \]

\[
\begin{align*}
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
\end{align*}
\]

\[
\begin{align*}
\text{flight}(X, Y) & \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
\text{flight}(X, Y) & \rightarrow \text{flight}(Y, X)
\end{align*}
\]

\[
\begin{align*}
\text{flight}(a, \bot_1) \\
\text{airport}(\bot_1)
\end{align*}
\]

The following facts are added to \( D \):

\[
\begin{align*}
\text{flight}(a, \bot_1) \\
\text{airport}(\bot_1)
\end{align*}
\]
### Example (∃ a finite sequence)

<table>
<thead>
<tr>
<th>$D:$</th>
<th>$\Sigma:$</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport(a)</td>
<td>$r_1: \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2: \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3: \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D:$

- $\text{flight}(a, \bot_1)$
- $\text{airport}(\bot_1)$
Example (exists a finite sequence)

\[ D : \quad \Sigma : \]

\[
\begin{align*}
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
& \quad r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
& \quad r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \( D \):

\[
\begin{align*}
\text{flight}(a, \bot_1) \\
\text{airport}(\bot_1)
\end{align*}
\]
Chase – Terminating Sequence

Example (exists a finite sequence)

\[ D : \]
\[ r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \]
\[ r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \]
\[ r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \]

The following facts are added to \( D \):

\[ \text{flight}(a, \perp_1) \]
\[ \text{airport}(\perp_1) \]
Chase – Terminating Sequence

Example (∃ a finite sequence)

\[ D : \]

- \( \text{airport}(a) \)

\[ \Sigma : \]

- \( r_1 : \text{airport}(X) \rightarrow \exists Y \text{flight}(X, Y) \)
- \( r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \)
- \( r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \)

The following facts are added to \( D : \)

- \( \text{flight}(a, \bot_1) \)
- \( \text{airport}(\bot_1) \)
- \( \text{flight}(\bot_1, a) \)
Chase – Terminating Sequence

Example (exists a finite sequence)

\[ D : \]

\[ \Sigma : \]

\[ \text{airport}(a) \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \]
\[ r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \]
\[ r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \]

The following facts are added to \( D \):

\[ \text{flight}(a, \bot_1) \]
\[ \text{airport}(\bot_1) \]
\[ \text{flight}(\bot_1, a) \]

No further rule is applicable: STOP.
**Chase – Non-terminating Sequence**

Example ($\exists$ an infinite sequence)

<table>
<thead>
<tr>
<th>$D:$</th>
<th>$\Sigma:$</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport($a$)</td>
<td>$r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D:$
Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

\[
D : \quad \Sigma :
\]

\[
\begin{align*}
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
& \quad r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
& \quad r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \(D\):

\[
\text{flight}(a, \bot_1) \\
\text{airport}(\bot_1)
\]
Chase – Non-terminating Sequence

Example (\(\exists\) an infinite sequence)

\[
\begin{align*}
\text{D} &: \\
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
& \quad r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
& \quad r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \(D\):

...
Chase – Non-terminating Sequence

Example (\(\exists\) an infinite sequence)

\[D:\]

\(\text{airport}(a)\)

\(r_1: \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y)\)

\(r_2: \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)\)

\(r_3: \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)\)

The following facts are added to \(D:\)

\(\text{flight}(a, \bot_1)\)
### Chase – Non-terminating Sequence

#### Example (\(\exists\) an infinite sequence)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>D</strong></td>
<td><strong>(\Sigma)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>airport(a)</strong></td>
<td><strong>r_1</strong> : (airport(X) \rightarrow \exists Y) flight((X, Y))**</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>r_2</strong> : (flight(X, Y) \rightarrow airport(X) \land airport(Y))**</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>r_3</strong> : (flight(X, Y) \rightarrow flight(Y, X))**</td>
</tr>
<tr>
<td></td>
<td><strong>flight(a, \bot_1)</strong></td>
<td>The following facts are added to <strong>D</strong>:</td>
</tr>
</tbody>
</table>

The following facts are added to **D**:

- \(flight(a, \bot_1)\)

S. Greco and C. Molinaro
Termination Analysis of Logic Programs
July 25th, 2015 116 / 184
Chase – Non-terminating Sequence

Example (\(\exists\) an infinite sequence)

<table>
<thead>
<tr>
<th>(D)</th>
<th>(\Sigma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(airport(a))</td>
<td>(r_1: \text{airport}(X) \rightarrow \exists Y \text{flight}(X, Y))</td>
</tr>
<tr>
<td></td>
<td>(r_2: \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y))</td>
</tr>
<tr>
<td></td>
<td>(r_3: \text{flight}(X, Y) \rightarrow \text{flight}(Y, X))</td>
</tr>
</tbody>
</table>

The following facts are added to \(D\):

\(\text{flight}(a, \bot_1)\)
Example (\(\exists\) an infinite sequence)

\[
\begin{align*}
D : & & \Sigma : \\
\text{airport}(a) & & r_1 : \text{airport}(X) \rightarrow \exists Y \, \text{flight}(X, Y) \\
r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \wedge \text{airport}(Y) \\
r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \(D\):

\[
\text{flight}(a, \bot_1)
\]
Chase – Non-terminating Sequence

Example (\(\exists\) an infinite sequence)

\[ D : \]

\[ \text{airport}(a) \]

\[ r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \]

\[ r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \]

\[ r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \]

\[ \Sigma : \]

The following facts are added to \(D:\)

\[ \text{flight}(a, \bot_1) \]
Example (∃ an infinite sequence)

\[ D : \]

\[ \Sigma : \]

\[ \text{airport}(a) \]

\[ r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \]

\[ r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \]

\[ r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \]

The following facts are added to \( D \):

\[ \text{flight}(a, \bot_1) \]

\[ \text{airport}(\bot_1) \]
Chase – Non-terminating Sequence

Example (\(\exists\) an infinite sequence)

\[
D : \quad \Sigma :
\]

\[
\begin{align*}
\text{airport}(a) & \quad r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \\
r_2 : \text{flight}(X, Y) & \rightarrow \text{airport}(X) \land \text{airport}(Y) \\
r_3 : \text{flight}(X, Y) & \rightarrow \text{flight}(Y, X)
\end{align*}
\]

The following facts are added to \(D\):

\[
\begin{align*}
\text{flight}(a, \bot_1) \\
\text{airport}(\bot_1)
\end{align*}
\]
Example (exists an infinite sequence)

\[ D : \]
\[ \Sigma : \]

\( \text{airport}(a) \)
\[ r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y) \]
\[ r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y) \]
\[ r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X) \]

The following facts are added to \( D : \)

\( \text{flight}(a, \bot_1) \)
\( \text{airport}(\bot_1) \)
### Chase – Non-terminating Sequence

**Example (∃ an infinite sequence)**

<table>
<thead>
<tr>
<th>$D:$</th>
<th>$\Sigma:$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{airport}(a)$</td>
<td>$r_1: \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2: \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3: \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D$:

- $\text{flight}(a, \bot_1)$
- $\text{airport}(\bot_1)$
Example (∃ an infinite sequence)

\[ D : \quad \Sigma : \]

- \(\text{airport}(a)\)
- \(r_1 : \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y)\)
- \(r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)\)
- \(r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)\)

The following facts are added to \(D:\)

- \(\text{flight}(a, \bot_1)\)
- \(\text{airport}(\bot_1)\)
- \(\text{flight}(\bot_1, \bot_2)\)
Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

<table>
<thead>
<tr>
<th>D</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport(a)</td>
<td>r₁ : airport(X) → ∃Y flight(X, Y)</td>
</tr>
<tr>
<td></td>
<td>r₂ : flight(X, Y) → airport(X) ∧ airport(Y)</td>
</tr>
<tr>
<td></td>
<td>r₃ : flight(X, Y) → flight(Y, X)</td>
</tr>
</tbody>
</table>

The following facts are added to D:
- flight(a, ⊥₁)
- airport(⊥₁)
- flight(⊥₁, ⊥₂)
Chase – Non-terminating Sequence

Example ($\exists$ an infinite sequence)

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport($a$)</td>
<td>$r_1 : \text{airport}(X) \rightarrow \exists Y , \text{flight}(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2 : \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D$:

- flight($a, \bot_1$)
- airport($\bot_1$)
- flight($\bot_1, \bot_2$)
Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

<table>
<thead>
<tr>
<th>$D:$</th>
<th>$\Sigma:$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$airport(a)$</td>
<td>$r_1: \quad airport(X) \rightarrow \exists Y ; flight(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2: \quad flight(X, Y) \rightarrow airport(X) \land airport(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3: \quad flight(X, Y) \rightarrow flight(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D:$

- $flight(a, \bot_1)$
- $airport(\bot_1)$
- $flight(\bot_1, \bot_2)$
### Chase – Non-terminating Sequence

**Example (∃ an infinite sequence)**

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$airport(a)$</strong></td>
<td>$r_1 : airport(X) \rightarrow \exists Y \text{ flight}(X, Y)$</td>
</tr>
<tr>
<td>$r_2 : \text{flight}(X, Y) \rightarrow airport(X) \land airport(Y)$</td>
<td></td>
</tr>
<tr>
<td>$r_3 : \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)$</td>
<td></td>
</tr>
</tbody>
</table>

The following facts are added to $D$:

- $\text{flight}(a, \bot_1)$
- $\text{airport}(\bot_1)$
- $\text{flight}(\bot_1, \bot_2)$

By iteratively applying $r_1$ and $r_2$ the chase never terminates.
### Example ($\exists$ an infinite sequence)

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$airport(a)$</td>
<td></td>
</tr>
<tr>
<td>$r_1 : airport(X) \rightarrow \exists Y \ flight(X, Y)$</td>
<td></td>
</tr>
<tr>
<td>$r_2 : flight(X, Y) \rightarrow airport(X) \land airport(Y)$</td>
<td></td>
</tr>
<tr>
<td>$r_3 : flight(X, Y) \rightarrow flight(Y, X)$</td>
<td></td>
</tr>
</tbody>
</table>

The following facts are added to $D$:

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

By iteratively applying $r_1$ and $r_2$, the chase never terminates.
Chase – Non-terminating Sequence

Example (∃ an infinite sequence)

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport($a$)</td>
<td>$r_1: \text{airport}(X) \rightarrow \exists Y \text{ flight}(X, Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_2: \text{flight}(X, Y) \rightarrow \text{airport}(X) \land \text{airport}(Y)$</td>
</tr>
<tr>
<td></td>
<td>$r_3: \text{flight}(X, Y) \rightarrow \text{flight}(Y, X)$</td>
</tr>
</tbody>
</table>

The following facts are added to $D$:

- $\text{flight}(a, \bot_1)$
- $\text{airport}(\bot_1)$
- $\text{flight}(\bot_1, \bot_2)$
- $\text{airport}(\bot_2)$
  
  ... 

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.
Theorem

Consider a set $\Sigma$ of TGDs:

- It is undecidable whether, for every instance $D$, some chase sequence of $D$ with $\Sigma$ terminates [GO13].
- It is undecidable whether, for every instance $D$, all chase sequences of $D$ with $\Sigma$ terminate [GM14].
Chase Termination

Theorem ([DNR08])

Given a set $\Sigma$ of TGDs and a (fixed) instance $D$:

- It is undecidable whether some chase sequence of $D$ with $\Sigma$ terminates.
- It is undecidable whether all chase sequences of $D$ with $\Sigma$ terminate.
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]
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:

  \[ N(X, Y) \rightarrow E(X, f^K_r(X, Y), f^Z_r(X, Y)) \]

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

  \[ N(X, Y) \rightarrow E(X, f^K_r(X), f^Z_r(X)) \]
### Chase Variants

**Example (Complex terms represent nulls)**

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
</table>

STOP (fixpoint)

... NO Termination (no fixpoint)
### Chase Variants

**Example (Complex terms represent nulls)**

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>(\quad)</td>
<td>(\quad)</td>
</tr>
</tbody>
</table>
# Chase Variants

## Example (Complex terms represent nulls)

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>(E(X, Y) \rightarrow E(X, f(X)))</td>
<td></td>
</tr>
</tbody>
</table>
## Chase Variants

### Example (Complex terms represent nulls)

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
</table>
| **No chase step**  
\((D \models \Sigma)\) | \(E(X, Y) \rightarrow E(X, f(X))\) | \(E(a, b)\) |
### Chase Variants

#### Example (Complex terms represent nulls)

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step $(D \models \Sigma)$</td>
<td>$E(X, Y) \rightarrow E(X, f(X))$</td>
<td>$E(a, b)$, $E(a, f(a))$</td>
</tr>
</tbody>
</table>

$$D : \Sigma :$$
- $D(a, b)$
- $E(X, Y) \rightarrow \exists Z E(X, Z)$
### Chase Variants

**Example (Complex terms represent nulls)**

\[
D : \quad \Sigma :
\]

\[
E(a, b) \quad E(X, Y) \rightarrow \exists Z \ E(X, Z)
\]

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>(E(X, Y) \rightarrow E(X, f(X)))</td>
<td>(E(a, b))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(E(a, f(a)))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STOP (fixpoint)</td>
</tr>
</tbody>
</table>
### Chase Variants

**Example (Complex terms represent nulls)**

<table>
<thead>
<tr>
<th></th>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$ :</td>
<td>$E(a, b)$</td>
<td>$E(X, Y) \rightarrow \exists Z \ E(X, Z)$</td>
<td></td>
</tr>
<tr>
<td>$\Sigma$ :</td>
<td>$E(X, Y) \rightarrow E(X, f(X))$</td>
<td>$E(X, Y) \rightarrow E(X, f(X, Y))$</td>
<td></td>
</tr>
<tr>
<td>No chase step</td>
<td>$E(a, b)$</td>
<td>$E(a, f(a))$</td>
<td></td>
</tr>
<tr>
<td>$(D \models \Sigma)$</td>
<td>STOP (fixpoint)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Chase Variants

### Example (Complex terms represent nulls)

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>(E(X, Y) \rightarrow E(X, f(X)))</td>
<td>(E(X, Y) \rightarrow E(X, f(X, Y)))</td>
</tr>
<tr>
<td>(D : E(a, b))</td>
<td>(E(a, f(a)))</td>
<td>(E(a, b))</td>
</tr>
<tr>
<td>(E(X, Y) \rightarrow \exists Z E(X, Z))</td>
<td>STOP (fixpoint)</td>
<td></td>
</tr>
</tbody>
</table>

---

S. Greco and C. Molinaro

Termination Analysis of Logic Programs

July 25th, 2015 123 / 184
Chase Variants

Example (Complex terms represent nulls)

\[ D : E(a, b) \]
\[ E(X, Y) \rightarrow \exists Z \ E(X, Z) \]

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>(E(X, Y) \rightarrow E(X, f(X)))</td>
<td>(E(X, Y) \rightarrow E(X, f(X, Y)))</td>
</tr>
<tr>
<td>(E(a, b)) (E(a, f(a)))</td>
<td>(E(a, b)) (E(a, f(a)))</td>
<td>(E(a, b)) (E(a, f(a, b)))</td>
</tr>
<tr>
<td>STOP (fixpoint)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chase Variants

Example (Complex terms represent nulls)

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>[ E(X, Y) \rightarrow E(X, f(X)) ] [ E(a, b) ] [ E(a, f(a)) ]</td>
<td>[ E(X, Y) \rightarrow E(X, f(X, Y)) ] [ E(a, b) ] [ E(a, f(a, b)) ] [ E(a, f(a, f(a, b))) ]</td>
</tr>
<tr>
<td>STOP (fixpoint)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chase Variants

Example (Complex terms represent nulls)

\[ D : \begin{align*}
E(a, b) & \\
E(X, Y) \rightarrow \exists Z \ E(X, Z)
\end{align*} \]

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chase step ((D \models \Sigma))</td>
<td>(E(X, Y) \rightarrow E(X, f(X)))</td>
<td>(E(X, Y) \rightarrow E(X, f(X, Y)))</td>
</tr>
<tr>
<td></td>
<td>(E(a, b))</td>
<td>(E(a, b))</td>
</tr>
<tr>
<td></td>
<td>(E(a, f(a)))</td>
<td>(E(a, f(a, b)))</td>
</tr>
<tr>
<td></td>
<td>(\text{STOP (fixpoint)})</td>
<td>(E(a, f(a, f(a, b))))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(E(a, f(a, f(a, f(a, b)))))</td>
</tr>
</tbody>
</table>
### Chase Variants

#### Example (Complex terms represent nulls)

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Semi-oblivious</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No chase step</strong></td>
<td>(E(X, Y) \rightarrow E(X, f(X)))</td>
<td>(E(X, Y) \rightarrow E(X, f(X, Y)))</td>
</tr>
<tr>
<td>((D \models \Sigma))</td>
<td>(E(a, b))</td>
<td>(E(a, b))</td>
</tr>
<tr>
<td></td>
<td>(E(a, f(a)))</td>
<td>(E(a, f(a, b)))</td>
</tr>
<tr>
<td></td>
<td>STOP (fixpoint)</td>
<td>(E(a, f(a, f(a, b))))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(E(a, f(a, f(a, f(a, b)))))</td>
</tr>
</tbody>
</table>

STOP (fixpoint)  

NO Termination  (no fixpoint)
Chase Variants

Example (Complex terms represent nulls)

\[
D : \quad \sum : \\
E(a, b) \quad E(X, Y) \rightarrow \exists Z \ E(X, Z)
\]

Standard

No chase step \((D \models \sum)\)

\[
E(X, Y) \rightarrow E(X, f(X))\\
E(a, b)\\
E(a, \bot_1)
\]

STOP (fixpoint)

Semi-oblivious

\[
E(a, b)\\
E(a, \bot_1)
\]

Oblivious

\[
E(a, b)\\
E(a, \bot_2)\\
E(a, \bot_3)\\
E(a, \bot_4)
\]

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 $\Sigma$ is a set of TGDs, then there exists a universal model for $\Sigma$ and $I$ iff the core chase of $I$ with $\Sigma$ terminates and yields such a model.
Chase Variants

- \( \text{CT}^c_{\forall} \): class of sets of TGDs \( \Sigma \) s.t., for every instance, \textbf{all} \( c \)-chase sequences terminate.
- \( \text{CT}^c_{\exists} \): class of sets of TGDs \( \Sigma \) s.t., for every instance, \textbf{at least one} \( c \)-chase sequence terminates.

Theorem ([Mei10, One13] For TGDs only)

\[
\text{CT}^\text{obl}_{\forall} = \text{CT}^\text{obl}_{\exists} \subsetneq \text{CT}^\text{sobl}_{\forall} = \text{CT}^\text{sobl}_{\exists} \subsetneq \text{CT}^\text{std}_{\forall} \subsetneq \text{CT}^\text{std}_{\exists} \subsetneq \text{CT}^\text{core}_{\forall} = \text{CT}^\text{core}_{\exists}
\]
Function Symbols vs. TGDs

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

**Step 1. Skolemize TGDs.**

Example

\[
  r : \quad p(X, Y) \rightarrow \exists K, Z \ q(X, K, Z) \\
  sk(r) : \quad p(X, Y) \rightarrow q(X, f^K_r(X), f^Z_r(X))
\]

**Step 2. Apply termination criteria to skolemized TGDs.**

Given a set \( \Sigma \) 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 \( \Sigma \) [One13].
**Example**

\[ r : \quad p(X, Y) \rightarrow \exists Z p(X, Z) \]
**Function Symbols vs. TGDs**

**Example**

\[ r : \ p(X, Y) \rightarrow \exists Z p(X, Z) \]

**Step 1. Skolemize** \( r \):

\[ sk(r) : \ p(X, Y) \rightarrow p(X, f^Z_r(X)) \]

We get a logic program with function symbols.
Example

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

Step 1. Skolemize $$r$$:

$$sk(r) : \ p(X, Y) \rightarrow p(X, f^Z_X(Y))$$

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** \( r \):

\[ \text{sk}(r) : \quad p(X, Y) \rightarrow p(X, f^Z_r(X)) \]

We get a logic program with function symbols.

**Step 2. Analyze** \( \text{sk}(r) \) **by applying a termination criterion.**

- E.g., \( \text{sk}(r) \) is argument-restricted.
Example

\[ r : \ p(X, Y) \rightarrow \exists Z p(X, Z) \]

Step 1. Skolemize \( r \):

\[ sk(r) : \ p(X, Y) \rightarrow p(X, f^Z_r(X)) \]

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.
Example

\[ r : \ p(X, Y) \rightarrow \exists Z p(X, Z) \]

Step 1. Skolemize \( r \):

\[ sk(r) : \ p(X, Y) \rightarrow p(X, f^Z_r(X)) \]

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

\[ r : \ p(X, Y) \rightarrow \exists Z p(X, Z) \]

Step 1. Skolemize \( r \):

\[ sk(r) : \ p(X, Y) \rightarrow p(X, f^Z_i(X)) \]

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])

\[
\begin{align*}
\text{CT}_{\forall}^{\text{sobl}} & = \text{CT}_{\exists}^{\text{sobl}} \\
\subset & \text{CT}_{\forall}^{\text{std}} \\
\subset & \text{CT}_{\exists}^{\text{std}}
\end{align*}
\]
Limitations: Recall that:

Theorem ([Mei10, One13])

\[
\begin{align*}
\text{CT}_{\forall} \text{sobl} & = \text{CT}_{\exists} \text{sobl} \not\subset \text{CT}_{\forall} \text{std} \not\subset \text{CT}_{\exists} \text{std} \\
\end{align*}
\]
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:

<table>
<thead>
<tr>
<th>Skolemized TGDs</th>
<th>Programs with function symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>once</td>
<td>arbitrary number of times</td>
</tr>
<tr>
<td>only in the head</td>
<td>in the body and/or head</td>
</tr>
<tr>
<td>no nesting</td>
<td>arbitrary nesting</td>
</tr>
</tbody>
</table>
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 $\rightarrow^*$ represent the generation of nulls.

Example: $\Sigma = N(X) \rightarrow \exists Y E(X, Y) E(X, Y) \rightarrow N(Y)$
Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. Normal edges represent the propagation of values between arguments;
  2. Special edges $\rightarrow^*$ represent the generation of nulls.

Example

$$\Sigma = \begin{align*}
N(X) & \rightarrow \exists Y \ E(X, Y) \\
E(X, Y) & \rightarrow N(Y)
\end{align*}$$
Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. normal edges represent the propagation of values between arguments;
  2. special edges \( \rightarrow^* \) represent the generation of nulls.

Example

\[ \Sigma = \begin{align*}
N(X) & \rightarrow \exists Y \ E(X, Y) \\
E(X, Y) & \rightarrow N(Y)
\end{align*} \]
Weak Acyclicity [FKMP05] [standard chase]

Dependency Graph $\text{dep}(\Sigma)$

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

Example

$\Sigma = \begin{align*}
  N(X) &\rightarrow \exists Y \ E(X, Y) \\
  E(X, Y) &\rightarrow N(Y)
\end{align*}$

Example graph:

- Node $N1$ connected to $E1$.
- Node $E1$ connected to $N1$.
- Node $E2$ connected to $E1$.
Dependency Graph

- Nodes are predicate arguments.
- Two kinds of edges:
  1. Normal edges represent the propagation of values between arguments;
  2. Special edges $\rightarrow^*$ represent the generation of nulls.

Example

$$\Sigma = N(X) \rightarrow \exists Y E(X, Y)$$
$$E(X, Y) \rightarrow N(Y)$$
Definition

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

Theorem
If \( \Sigma \) is weakly acyclic, then for every instance \( I \), every chase sequence terminates (and has a polynomial length in the size of \( I \)).
Affected Positions $\text{aff}(\Sigma)$ [CGK13]
Overestimation of positions that may contain null values.

Propagation Graph $\text{prop}(\Sigma)$
Restriction of the dependency graph containing only affected positions.

Example

$$\Sigma = \begin{align*} r_1 : N(X) & \rightarrow \exists Y E(X, Y) \\ r_2 : S(Y) & \land E(X, Y) \rightarrow N(Y) \end{align*}$$
Affected Positions $\text{aff}(\Sigma)$ [CGK13]

Overestimation of positions that may contain null values.

Propagation Graph $\text{prop}(\Sigma)$

Restriction of the dependency graph containing only affected positions.

Example

$$\Sigma = \begin{align*}
  r_1 & : N(X) \rightarrow \exists Y \ E(X, Y) \\
  r_2 & : S(Y) \land E(X, Y) \rightarrow N(Y)
\end{align*}$$
Safety [MSL09] [standard chase]

Affected Positions $\text{aff}(\Sigma)$

Overestimation of positions that may contain null values.

Propagation Graph $\text{prop}(\Sigma)$

Restriction of the dependency graph containing only affected positions.

Example

$$\Sigma = \begin{align*}
  r_1 : & N(X) \rightarrow \exists Y \ E(X, Y) \\
  r_2 : & S(Y) \land E(X, Y) \rightarrow N(Y)
\end{align*}$$

- $\text{aff}(\Sigma) = \{E_2\}$
- $\text{prop}(\Sigma) = (\{E_2\}, \emptyset)$
### Affected Positions $\text{aff}(\Sigma)$
Overestimation of positions that may contain null values.

### Propagation Graph $\text{prop}(\Sigma)$
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.
Affected Positions $\text{aff}(\Sigma)$
Overestimation of positions that may contain null values.

Propagation Graph $\text{prop}(\Sigma)$
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 $\Sigma$ is safe, then for every instance $I$, every chase sequence terminates (and has a polynomial length in the size of $I$).
Chase Graph $G(\Sigma)$

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

Example $\Sigma = r_1 : N(X) \rightarrow \exists Y E(X,Y) \land E(X,Y) \rightarrow N(Y)$

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 $\Sigma$ is stratified.
It represents how dependencies fire each other.

Nodes: the dependencies in $\Sigma$.

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) $K_1 \xrightarrow{r_1 \ y_1 h_1} K_2$ (chase step - $K_1 \not\models h_1(r_1)$),

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

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

Example

$\Sigma = r_1 : N(X) \rightarrow \exists Y E(X, Y)$

$r_2 : S(Y) \land E(X, Y) \rightarrow N(Y)$

there exists a scenario where firing $r_2$ causes $r_1$ to fire ($r_2 \prec r_1$).
Stratification [DNR08] [standard chase]

Chase Graph $G(\Sigma)$

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

Example

\[
\Sigma = \\
r_1 : N(X) \Rightarrow \exists Y \ E(X, Y) \\
r_2 : S(Y) \land E(X, Y) \Rightarrow N(Y)
\]

- 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 $\Sigma$ is stratified.
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 $\Sigma$ 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

\[ r_1 \prec_c r_2 \text{ if:} \]
1) \( K_1 \overset{*}{\overset{r_1}{\rightarrow}} h_1 K_2 \) (oblivious step),
2) \( K_2 \not\models h_2(r_2) \),
3) \( K_1 \models h_2(r_2) \).

Stratification

\[ r_1 \prec r_2 \text{ if:} \]
1) \( K_1 \overset{h_1}{\vdash} K_2 \) (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 \textit{c-stratification} guarantees the termination of every chase sequence.
- Same approach of stratification, but the oblivious chase is used.

C-Stratification

\[ r_1 \prec_c r_2 \text{ if:} \]
1) \( K_1 \overset{*}{\rightarrow} r_1 h_1 K_2 \) (oblivious step),
2) \( K_2 \not\models h_2(r_2) \),
3) \( K_1 \models h_2(r_2) \).

Stratification

\[ r_1 \prec r_2 \text{ if:} \]
1) \( K_1 \overset{h_1}{\rightarrow} r_2 K_2 \) (standard step),
2) \( K_2 \not\models h_2(r_2) \),
3) \( K_1 \models h_2(r_2) \).

Theorem

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

For any \( \Sigma \), \( G(\Sigma) \subseteq G_c(\Sigma) \) \( \Rightarrow \) \( \text{Str} \supseteq \text{CStr} \)
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 $\Sigma$ 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{align*}
  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{align*}

$\Sigma$ neither safe not stratified.
Super-weak Acyclicity [Mar09]  [semi-obliv. chase]

- Builds a *trigger graph* whose edges define relations among dependencies. An edge \( r_i \leadsto r_j \) 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**

\[
\begin{align*}
    r_1 : N(X) &\rightarrow \exists Y, Z \ E(X, Y, Z) \\
r_2 : E(X, Y, Z) &\rightarrow G(X, Y, Z) \\
r_3 : G(X, Y, Y) &\rightarrow N(Y)
\end{align*}
\]

\( \Sigma \) neither safe not stratified.

\[
P(\Sigma) = \begin{cases} 
    r'_1 : N(X) &\rightarrow E(X, f^r_Y(X), f^r_Z(X)) \\
    r'_2 : E(X, Y, Z) &\rightarrow \exists Y, Z \ G(X, Y, Z) \\
    r'_3 : G(X, Y, Y) &\rightarrow N(Y)
\end{cases}
\]
Super-weak Acyclicity

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

---

Theorem

If $\Sigma$ 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 Acyclicity
- $SC$: Safety
- $CStr$: C-stratification
- $IR$: Inductive Restriction
- $SwA$: Super-weak Acyclicity
Limitations

Example

\begin{align*}
  r_1 : \quad N(X) & \rightarrow \exists Y \exists Z \ E(X, Y) \land S(Z, Y) \\
  r_2 : \quad E(X, Y) \land S(X, Y) & \rightarrow N(Y) \\
  r_3 : \quad E(X, Y) & \rightarrow E(Y, X)
\end{align*}
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. $K_1 \overset{r_1, h_1}{\rightarrow} K_2$,
  2. $K_2 \cup S \not\models h_2(r_2)$,
  3. $K_1 \cup S \models h_2(r_2)$ and
  4. $\text{Null}(S) \cap (\text{Null}(K_2) - \text{Null}(K_1)) = \emptyset$.

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 $r_1$.

- $r_1 < r_2$ if:
  1. $K_1 \overset{r_1, h_1}{\rightarrow} K_2$,
  2. $K_2 \not\models h_2(r_2)$,
  3. $K_1 \models h_2(r_2)$.
Improvements of (C-)Stratification

Example

\[ \Sigma = \begin{align*}
  r_1 : & \ R(x) \rightarrow \exists y \ T(x, y) \\
  r_2 : & \ R(x) \rightarrow T(x, x) \\
  r_3 : & \ T(x, y) \land T(x, x) \rightarrow R(y)
\end{align*} \]

- \( K_1 = \{ R(a) \} \) and \( K_2 = \{ R(a), T(a, \bot_1) \} \)
- \( S = \{ T(a, a) \} \)
- \( r_3 : T(a, \bot_1) \land T(a, a) \rightarrow R(\bot_1) \)
- \( r_3 \) is fired by \( r_1 \), then we have \( r_1 < r_3 \)
Improvements of (C-)Stratification

Example

\[ \Sigma = \]

- \( r_1 : R(x) \rightarrow \exists y T(x, y) \)
- \( r_2 : R(x) \rightarrow T(x, x) \)
- \( r_3 : T(x, y) \land T(x, x) \rightarrow R(y) \)

- \( K_1 = \{ R(a) \} \) and \( K_2 = \{ R(a), T(a, \bot_1) \} \)
- \( S = \{ T(a, a) \} \)
- \( r_3 : T(a, \bot_1) \land T(a, a) \rightarrow R(\bot_1) \)
- \( r_3 \) 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 $\Sigma$ into an ‘equivalent’ adorned set $\Sigma^\alpha$ and verify the structural properties for chase termination on $\Sigma^\alpha$ (similarly to LPs)
- Rewrite $\Sigma$ into a set of dependencies useful to analyze the structure of terms during the execution.
### Rewriting Algorithm [GST15]

#### Example

\[ \Sigma : \]

\[ r_1 : N(X) \rightarrow \exists Y \ E(X, Y) \]
\[ r_2 : S(X) \land E(X, Y) \rightarrow N(Y) \]

\[ \text{Adn}(\Sigma) : \]

\[
\begin{align*}
    s_1 & : N(X) \rightarrow N^b(X) \\
    s_2 & : S(X) \rightarrow S^b(X) \\
    s_3 & : E(X, Y) \rightarrow E^{bb}(X, Y) \\
    r'_1 & : N^b(X) \rightarrow \exists Y \ E^{bf_1}(X, Y) \quad f_1 = f_{r_1}^Y(b) \\
    r'_2 & : S^b(X) \land E^{bb}(X, Y) \rightarrow N^b(Y) \\
    r''_2 & : S^b(X) \land E^{bf_1}(X, Y) \rightarrow N^{f_1}(Y) \\
    r''_1 & : N^{f_1}(X) \rightarrow \exists Y \ E^{f_1 f_2}(X, Y) \quad f_2 = f_{r_1}^Y(f_1)
\end{align*}
\]
Model-Faithful Acyclicity (MFA) [GHK$^+$13]

Example

$\Sigma :$

\[ r : \quad A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]
Model-Faithful Acyclicity (MFA) [GHK+13]

Example

\[ \Sigma : \]

\[ r : \quad A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]

\[ MFA(\Sigma) : \]

\[ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]
Model-Faithful Acyclicity (MFA) [GHK+13]

Example

\[ R : \ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]

\[ MFA(\Sigma) : \]

\[ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \land F_Y(Y) \land F_Z(Z) \land S(X, Y) \land S(X, Z) \]
Model-Faithful Acyclicity (MFA) [GHK+13]

Example

Σ :

\[ r : \quad A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]

MFA(Σ) :

\[ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \land \]
\[ F_r^Y(Y) \land F_r^Z(Z) \land S(X, Y) \land S(X, Z) \]
\[ S(X_1, X_2) \rightarrow D(X_1, X_2) \]
\[ D(X_1, X_2) \land S(X_2, X_3) \rightarrow D(X_1, X_3) \]
Model-Faithful Acyclicity (MFA) [GHK⁺13]

Example

\[ \Sigma : \]

\[ r : \ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]

\[ MFA(\Sigma) : \]

\[ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \land \]

\[ F_Y^r(Y) \land F_Z^r(Z) \land S(X, Y) \land S(X, Z) \]

\[ S(X_1, X_2) \rightarrow D(X_1, X_2) \]

\[ D(X_1, X_2) \land S(X_2, X_3) \rightarrow D(X_1, X_3) \]

\[ F_Y^r(X_1) \land D(X_1, X_2) \land F_Y^r(X_2) \rightarrow C \]

\[ F_Z^r(X_1) \land D(X_1, X_2) \land F_Z^r(X_2) \rightarrow C \]
Model-Faithful Acyclicity (MFA) [GHK+13]

Example

\[ \sum : \]

\[ r : \quad A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \]

\[ \textit{MFA}(\sum) : \]

\[ A(X) \rightarrow \exists Y, Z \ R(X, Y) \land B(Z) \land \]

\[ F^Y_r(Y) \land F^Z_r(Z) \land S(X, Y) \land S(X, Z) \]

\[ S(X_1, X_2) \rightarrow D(X_1, X_2) \]

\[ D(X_1, X_2) \land S(X_2, X_3) \rightarrow D(X_1, X_3) \]

\[ F^Y_r(X_1) \land D(X_1, X_2) \land F^Y_r(X_2) \rightarrow C \]

\[ F^Z_r(X_1) \land D(X_1, X_2) \land F^Z_r(X_2) \rightarrow C \]

If \( I \cup \textit{MFA}(\sum) \models C \) then a cyclic term is derived during the semi-oblivious chase execution of \( I \) and \( \sum \).
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 \( I_{\Sigma} \) for \( \Sigma \) is the instance containing all facts that can be built using:
- all predicates in \( \Sigma \),
- all constants in the body of a dependency in \( \Sigma \), and
- one special fresh constant \( * \).

Theorem ([Mar09])

The semi-oblivious chase of \( \Sigma \) and \( I \) terminates for every \( I \) iff the semi-oblivious chase of \( \Sigma \) and \( I_{\Sigma} \) terminates.

Theorem

If \( \Sigma \) is MFA w.r.t. \( I_{\Sigma} \), 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:

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

where $\varphi(\bar{X})$ is a conjunction of atoms and $X_1, X_2 \in \bar{X}$.

Example

$$\forall M_1, M_2, P \, \text{directs}(M_1, P) \wedge \text{directs}(M_2, P) \rightarrow M_1 = M_2$$
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)

\[ D : \quad \Sigma : \]
\[ N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \]
\[ E(X, Y) \rightarrow N(Y) \]
Chase and EGDs

Adding EGDs leads to termination

Example (No EGDs)

\[ D : \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \]
\[ E(X, Y) \rightarrow N(Y) \]

\[ \text{chase}(D, \Sigma) = \{ N(a), \ E(a, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2), \ \ldots \} \]

There is no terminating chase sequence.
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$$D : \quad \Sigma :$$

$N(a)$

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

$E(X, Y) \rightarrow N(Y)$

$E(X, Y) \rightarrow X = Y$
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$D : \Sigma :$

$N(a)$  

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

$E(X, Y) \rightarrow N(Y)$

$E(X, Y) \rightarrow X = Y$
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to \( \Sigma \) ...

Example (TGDs + EGDs)

\[
D : \quad \Sigma :
\]

\[
\begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
E(X, Y) & \rightarrow N(Y) \\
E(X, Y) & \rightarrow X = Y
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ N(a),
\]
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$D : \Sigma :$

$N(a)$

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

$E(X, Y) \rightarrow N(Y)$

$E(X, Y) \rightarrow X = Y$

$\text{chase}(D, \Sigma) = \{ N(a) , \}$
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$$D : \quad \Sigma :$$

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

$$\text{chase}(D, \Sigma) = \{ N(a), E(a, \bot_1) \},$$
Chase and EGDs
Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[ D : \quad \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \]
\[ E(X, Y) \rightarrow N(Y) \]
\[ E(X, Y) \rightarrow X = Y \]

\[ \text{chase}(D, \Sigma) = \{ N(a), E(a, \bot_1) \}, \]
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$D : \Sigma :$

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

$\text{chase}(D, \Sigma) = \{ N(a), E(a, \bot_1) \},$
Chase and EGDs
Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

**Example (TGDs + EGDs)**

\[
D : \quad \Sigma :
\]

\[
N(a) \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
E(X, Y) \rightarrow N(Y) \\
E(X, Y) \rightarrow X = Y
\]

\[
\text{chase}(D, \Sigma) = \{N(a), E(a, a)\}
\]
Chase and EGDs

Adding EGDs leads to termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[
\begin{align*}
D : & \quad \Sigma : \\
N(a) & \quad N(X) \rightarrow \exists Y \ E(X, Y) \\
& \quad E(X, Y) \rightarrow N(Y) \\
& \quad E(X, Y) \rightarrow X = Y \\
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ N(a), E(a, a) \}
\]

No further dependency is applicable: STOP.
Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (No EGDs)

<table>
<thead>
<tr>
<th>D :</th>
<th>$\Sigma$ :</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N(a)$</td>
<td>$N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z)$</td>
</tr>
<tr>
<td></td>
<td>$E(X, Y, Y) \rightarrow N(Y)$</td>
</tr>
</tbody>
</table>
Chase and EGDs

Adding EGDs → No termination

Example (No EGDs)

\[ D : \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y \exists Z E(X, Y, Z) \]
\[ E(X, Y, Y) \rightarrow N(Y) \]

\[ \text{chase}(D, \Sigma) = \{ N(a) \}, \]
Chase and EGDs

Adding EGDs → No termination

Example (No EGDs)

\[
D : \quad \Sigma : \\
N(a) \quad N(X) \rightarrow \exists Y \exists Z E(X, Y, Z) \\
E(X, Y, Y) \rightarrow N(Y)
\]

\[
\text{chase}(D, \Sigma) = \{N(a),
\]

No further dependency is applicable: STOP.
Chase and EGDs

Adding EGDs → No termination

Example (No EGDs)

\[ D : \quad \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y \exists Z E(X, Y, Z) \]
\[ E(X, Y, Y) \rightarrow N(Y) \]

\[ \text{chase}(D, \Sigma) = \{ N(a), E(a, \bot_1, \bot_2) \} \]

No further dependency is applicable: STOP.
Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Example (No EGDs)

\[
D : \quad \Sigma : \\
N(a) \quad N(X) \rightarrow \exists Y \exists Z E(X, Y, Z) \\
E(X, Y, Y) \rightarrow N(Y)
\]

\[
\text{chase}(D, \Sigma) = \{ N(a), \ E(a, \bot_1, \bot_2) \} 
\]

No further dependency is applicable: STOP.
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...
Chase and EGDs
Adding EGDs $\rightarrow$ No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs $+$ EGDs)

$$D : \quad \Sigma :$$

$$N(a) \quad N(X) \rightarrow \exists Y \exists Z \; E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

**Example (TGDs + EGDs)**

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N(a)$</td>
<td>$N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z)$</td>
</tr>
<tr>
<td></td>
<td>$E(X, Y, Y) \rightarrow N(Y)$</td>
</tr>
<tr>
<td></td>
<td>$E(X, Y, Z) \rightarrow Y = Z$</td>
</tr>
</tbody>
</table>
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

### Example (TGDs + EGDs)

\[
\begin{align*}
D : & \quad \Sigma : \\
N(a) & \quad N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z) \\
 & \quad E(X, Y, Y) \rightarrow N(Y) \\
 & \quad E(X, Y, Z) \rightarrow Y = Z
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ N(a) \},
\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[
D : \quad \Sigma :
\]

\[
N(a) \quad N(X) \rightarrow \exists Y \exists Z E(X, Y, Z)
\]

\[
E(X, Y, Y) \rightarrow N(Y)
\]

\[
E(X, Y, Z) \rightarrow Y = Z
\]

\[
\text{chase}(D, \Sigma) = \{N(a), \}
\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[
\begin{align*}
D : & \quad \Sigma : \\
N(a) & \quad N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z) \\
& \quad E(X, Y, Y) \rightarrow N(Y) \\
& \quad E(X, Y, Z) \rightarrow Y = Z
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{ N(a), \ E(a, \bot_1, \bot_2) , \}
\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[
\begin{align*}
D : & \quad \Sigma : \\
N(a) & \quad N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z) \\
& \quad E(X, Y, Y) \rightarrow N(Y) \\
& \quad E(X, Y, Z) \rightarrow Y = Z \\
\end{align*}
\]

\[\text{chase}(D, \Sigma) = \{N(a), E(a, \perp_1, \perp_2)\},\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$$D : \quad \Sigma :$$

$N(a)$  

$N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z)$  

$E(X, Y, Y) \rightarrow N(Y)$  

$E(X, Y, Z) \rightarrow Y = Z$

$$\text{chase}(D, \Sigma) = \{ N(a), \ E(a, \bot_1, \bot_2) \},$$
Chase and EGDs
Adding EGDs → No termination

Adding an EGD to Σ ...

Example (TGDs + EGDs)

\[
D : \quad \Sigma :
\]
\[
N(a) \quad N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z)
\]
\[
E(X, Y, Y) \rightarrow N(Y)
\]
\[
E(X, Y, Z) \rightarrow Y = Z
\]

\[
\text{chase}(D, \Sigma) = \{N(a), E(a, \bot_1, \bot_1)\},
\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[
D : \\
\Sigma : \\
N(a) \\
N(X) \rightarrow \exists Y \exists Z E(X, Y, Z) \\
E(X, Y, Y) \rightarrow N(Y) \\
E(X, Y, Z) \rightarrow Y = Z
\]

\[
\text{chase}(D, \Sigma) = \{N(a), E(a, \bot_1, \bot_1)\},
\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to \( \Sigma \) ...

Example (TGDs + EGDs)

\[
\begin{align*}
D : & \quad \Sigma : \\
N(a) & \quad N(X) \rightarrow \exists Y \exists Z E(X, Y, Z) \\
& \quad E(X, Y, Y) \rightarrow N(Y) \\
& \quad E(X, Y, Z) \rightarrow Y = Z
\end{align*}
\]

\[
\text{chase}(D, \Sigma) = \{N(a), E(a, \bot_1, \bot_1)\},
\]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$D : \Sigma :
\begin{align*}
N(a) & \quad N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z) \\
E(X, Y, Y) & \rightarrow N(Y) \\
E(X, Y, Z) & \rightarrow Y = Z
\end{align*}$

$\text{chase}(D, \Sigma) = \{N(a), E(a, \bot, \bot), N(\bot)\}$
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

\[ D : \quad \Sigma : \]

\[ N(a) \quad N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z) \]
\[ E(X, Y, Y) \rightarrow N(Y) \]
\[ E(X, Y, Z) \rightarrow Y = Z \]

\[ \text{chase}(D, \Sigma) = \{ N(a), \ E(a, \bot_1, \bot_1), \ N(\bot_1), \} \]
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$$D : \quad \Sigma :$$

$N(a)$

$$N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z)$$
$$E(X, Y, Y) \rightarrow N(Y)$$
$$E(X, Y, Z) \rightarrow Y = Z$$

$\text{chase}(D, \Sigma) = \{ N(a), E(a, \bot_1, \bot_1), N(\bot_1), \}$
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$$D : \quad \Sigma :$$

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

$$E(X, Y, Y) \rightarrow N(Y)$$

$$E(X, Y, Z) \rightarrow Y = Z$$

chase($D, \Sigma$) = \{ $N(a)$, $E(a, \bot_1, \bot_1)$, $N(\bot_1)$, $E(\bot_1, \bot_2, \bot_3)$, ... \}
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

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

$$
\text{chase}(D, \Sigma) = \{N(a), E(a, \bot_1, \bot_1), N(\bot_1), E(\bot_1, \bot_2, \bot_3)\},
$$
Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

$$D : \quad \Sigma :$$

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

$$E(X, Y, Y) \rightarrow N(Y)$$

$$E(X, Y, Z) \rightarrow Y = Z$$

$$\text{chase}(D, \Sigma) = \{N(a), E(a, \bot_1, \bot_1), N(\bot_1), E(\bot_1, \bot_2, \bot_3)\},$$
Chase and EGDs

Adding EGDs → No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N(a)$</td>
<td>$N(X) \rightarrow \exists Y \exists Z \ E(X, Y, Z)$</td>
</tr>
<tr>
<td></td>
<td>$E(X, Y, Y) \rightarrow N(Y)$</td>
</tr>
<tr>
<td></td>
<td>$E(X, Y, Z) \rightarrow Y = Z$</td>
</tr>
</tbody>
</table>

$\text{chase}(D, \Sigma) = \{ N(a), E(a, \bot_1, \bot_1), N(\bot_1), E(\bot_1, \bot_2, \bot_2) \}$
Chase and EGDs

Adding EGDs $\rightarrow$ No termination

Adding an EGD to $\Sigma$ ...

Example (TGDs + EGDs)

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

$$\text{chase}(D, \Sigma) = \{ N(a), \ E(a, \bot_1, \bot_1), \ N(\bot_1), \ E(\bot_1, \bot_2, \bot_2), \ldots \}$$

No termination
Relationship between $\text{CT}^c_\forall$ and $\text{CT}^c_\exists$

For TGDs only:

$$\text{CT}^\text{obl}_\forall = \text{CT}^\text{obl}_\exists \subset \text{CT}^\text{sobl}_\forall = \text{CT}^\text{sobl}_\exists \subset \text{CT}^\text{std}_\forall \subset \text{CT}^\text{std}_\exists \subset \text{CT}^\text{core}_\forall = \text{CT}^\text{core}_\exists$$
Relationship between $\text{CT}_\forall^c$ and $\text{CT}_\exists^c$

For TGDs only:

\[ \text{CT}_\forall^{obl} = \text{CT}_\exists^{obl} \subset \text{CT}_\forall^{sobl} = \text{CT}_\exists^{sobl} \subset \text{CT}_\forall^{std} \subset \text{CT}_\exists^{std} \subset \text{CT}_\forall^{core} = \text{CT}_\exists^{core} \]

Known techniques can (and some actually do!) consider the class $\text{CT}_\forall^c$ for a simpler chase (e.g., oblivious).
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.
Relationship between $\text{CT}_\forall^c$ and $\text{CT}_\exists^c$

For TGDs only:

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

Known techniques can (and some actually do!) consider the class $\text{CT}_\forall^c$ for a simpler chase (e.g., oblivious).

Then, membership in $\text{CT}_\forall^\text{std}$, $\text{CT}_\exists^\text{std}$, $\text{CT}_\forall^\text{core}$, and $\text{CT}_\exists^\text{core}$ is implied.

**Question**: Does this still hold for TGDs+EGDs?
Relationship between $\text{CT}_\forall^c$ and $\text{CT}_\exists^c$

For TGDs only:

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

Known techniques can (and some actually do!) consider the class $\text{CT}_\forall^c$ for a simpler chase (e.g., oblivious).

Then, membership in $\text{CT}_\forall^c$, $\text{CT}_\exists^c$, $\text{CT}_\forall^c$, and $\text{CT}_\exists^c$ is implied.

**Question**: Does this still hold for TGDs+EGDss?
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?

$$CT^{obl}_\forall \subset CT^{obl}_\exists \subset CT^{sobl}_\forall \subset CT^{sobl}_\exists \subset CT^{std}_\forall \subset CT^{std}_\exists \subset CT^{core}_\forall = CT^{core}_\exists$$
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].
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...
Example

\[ \Sigma : \]

\[ r_1 : A(x) \land B(x) \rightarrow C(x) \]

\[ r_2 : C(x) \rightarrow \exists y A(x) \land B(y) \]

\[ r_3 : C(x) \rightarrow \exists y A(y) \land B(x) \]

\[ r_4 : A(x) \land A(y) \rightarrow x = y \]

\[ r_5 : B(x) \land B(y) \rightarrow x = y \]

Every chase sequence is terminating, for any variation of the chase.
Every chase sequence is terminating, for any variation of the chase.

However, both the natural and the substitution-free simulations of $\Sigma$ have no terminating chase sequence.
EGDs Simulation

Substitution-free simulation [Mar09]

Example

\[
A(X) \land B(X) \rightarrow C(X) \\
C(X) \rightarrow \exists Y A(X) \land B(Y) \\
C(X) \rightarrow \exists Y A(Y) \land B(X) \\
A(X) \land A(Y) \rightarrow X = Y \\
B(X) \land B(Y) \rightarrow X = Y
\]
EGDs Simulation

Substitution-free simulation [Mar09]

Example

\[ A(X) \land B(X) \rightarrow C(X) \]
\[ C(X) \rightarrow \exists Y \ A(X) \land B(Y) \]
\[ C(X) \rightarrow \exists Y \ A(Y) \land B(X) \]
\[ A(X) \land A(Y) \rightarrow X = Y \]
\[ B(X) \land B(Y) \rightarrow X = Y \]

\[ Eq(X, Y) \rightarrow Eq(Y, X) \]
\[ Eq(X, Y) \land Eq(Y, Z) \rightarrow Eq(X, Z) \]
\[ A(X) \rightarrow Eq(X, X) \]
\[ B(X) \rightarrow Eq(X, X) \]
\[ C(X) \rightarrow Eq(X, X) \]

Every chase sequence is terminating.

No terminating chase sequence for the substitution-free simulations.

S. Greco and C. Molinaro
Termination Analysis of Logic Programs
July 25th, 2015 175 / 184
EGDs Simulation

Substitution-free simulation [Mar09]

Example

\[ A(X) \land B(X) \rightarrow C(X) \quad A(X) \land B(X_2) \land Eq(X, X_2) \rightarrow C(X) \]
\[ C(X) \rightarrow \exists Y A(X) \land B(Y) \]
\[ C(X) \rightarrow \exists Y A(Y) \land B(X) \]
\[ A(X) \land A(Y) \rightarrow X = Y \]
\[ B(X) \land B(Y) \rightarrow X = Y \]

\[ Eq(X, Y) \rightarrow Eq(Y, X) \]
\[ Eq(X, Y) \land Eq(Y, Z) \rightarrow Eq(X, Z) \]
\[ A(X) \rightarrow Eq(X, X) \]
\[ B(X) \rightarrow Eq(X, X) \]
\[ C(X) \rightarrow Eq(X, X) \]
EGDs Simulation

Substitution-free simulation [Mar09]

Example

\[ A(X) \land B(X) \rightarrow C(X) \quad A(X) \land B(X_2) \land Eq(X, X_2) \rightarrow C(X) \]
\[ C(X) \rightarrow \exists Y \ A(X) \land B(Y) \quad C(X) \rightarrow \exists Y \ A(Y) \land B(X) \]
\[ A(X) \land A(Y) \rightarrow X = Y \quad Eq(X, Y) \quad B(X) \land B(Y) \rightarrow X = Y \quad Eq(X, Y) \]

\[ Eq(X, Y) \rightarrow Eq(Y, X) \quad Eq(X, Y) \land Eq(Y, Z) \rightarrow Eq(X, Z) \]
\[ A(X) \rightarrow Eq(X, X) \quad B(X) \rightarrow Eq(X, X) \quad C(X) \rightarrow Eq(X, X) \]
**EGDs Simulation**

**Substitution-free simulation** [Mar09]

### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(X) \land B(X) \rightarrow C(X))</td>
<td>(A(X) \land B(X_2) \land \text{Eq}(X, X_2) \rightarrow C(X))</td>
</tr>
<tr>
<td>(C(X) \rightarrow \exists Y \ A(Y) \land B(Y))</td>
<td>(C(X) \rightarrow \exists Y \ A(Y) \land B(X))</td>
</tr>
<tr>
<td>(A(X) \land A(Y) \rightarrow X = Y \text{ Eq}(X, Y))</td>
<td>(A(X) \land A(Y) \rightarrow X = Y \text{ Eq}(X, Y))</td>
</tr>
<tr>
<td>(B(X) \land B(Y) \rightarrow X = Y \text{ Eq}(X, Y))</td>
<td>(B(X) \land B(Y) \rightarrow X = Y \text{ Eq}(X, Y))</td>
</tr>
<tr>
<td>(\text{Eq}(X, Y))</td>
<td>(\rightarrow \text{Eq}(Y, X))</td>
</tr>
<tr>
<td>(\text{Eq}(X, Y) \land \text{Eq}(Y, Z))</td>
<td>(\rightarrow \text{Eq}(X, Z))</td>
</tr>
<tr>
<td>(A(X))</td>
<td>(\rightarrow \text{Eq}(X, X))</td>
</tr>
<tr>
<td>(B(X))</td>
<td>(\rightarrow \text{Eq}(X, X))</td>
</tr>
<tr>
<td>(C(X))</td>
<td>(\rightarrow \text{Eq}(X, X))</td>
</tr>
</tbody>
</table>

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**

\[
\begin{align*}
p(X) \land q(X) & \rightarrow r(X) \\
r(X) & \rightarrow \exists Y p(X) \land q(Y) \\
r(X) & \rightarrow \exists Y p(Y) \land q(X) \\
p(X) \land p(Y) & \rightarrow X = Y \\
q(X) \land q(Y) & \rightarrow X = Y
\end{align*}
\]

**Non - Terminating**

\[
\begin{align*}
p(X) \land q(X_2) \land eq(X, X_2) & \rightarrow r(X) \\
r(X) & \rightarrow \exists Y p(X) \land q(Y) \\
r(X) & \rightarrow \exists Y p(Y) \land q(X) \\
p(X) \land p(Y) & \rightarrow eq(X, Y) \\
q(X) \land q(Y) & \rightarrow eq(X, Y) \\
eq(X, Y) & \rightarrow eq(Y, X) \\
eq(X, Y) \land eq(Y, Z) & \rightarrow eq(X, Z) \\
p(X) & \rightarrow eq(X, X) \\
q(X) & \rightarrow eq(X, X) \\
r(X) & \rightarrow eq(X, X)
\end{align*}
\]
Dealing with EGDs

Example

\[ D = \{ N(a) \}, \Sigma : \]

\[
\begin{align*}
N(x) & \rightarrow \exists y \ E(x, y) \\
E(x, y) & \rightarrow N(y) \\
E(x, y) & \rightarrow x = y
\end{align*}
\]
Dealing with EGDs

Example

\[ D = \{ N(a) \}, \Sigma : \]

\[
\begin{align*}
N(x) & \rightarrow \exists y \ E(x, y) \\
E(x, y) & \rightarrow N(y) \\
E(x, y) & \rightarrow x = y
\end{align*}
\]

\[ N(a) \]
Dealing with EGDs

Example

\[ D = \{ N(a) \} \], \Sigma : \]

\[ N(x) \rightarrow \exists y \ E(x, y) \]
\[ E(x, y) \rightarrow N(y) \]
\[ E(x, y) \rightarrow x = y \]

\[ N(a) \rightarrow E(a, \bot_1) \]
Dealing with EGDs

Example

\[ D = \{ N(a) \}, \Sigma : \]

\[
\begin{align*}
N(x) & \rightarrow \exists y \ E(x, y) \\
E(x, y) & \rightarrow N(y) \\
E(x, y) & \rightarrow x = y
\end{align*}
\]

\[ N(a) \rightarrow E(a, \perp_1) \]
Dealing with EGDs

Example

\[ D = \{ N(a) \}, \Sigma : \]

\[
N(x) \rightarrow \exists y \ E(x, y) \\
E(x, y) \rightarrow N(y) \\
E(x, y) \rightarrow x = y
\]

\[ N(a) \rightarrow E(a, \bot_1) \rightarrow a = \bot_1 \]
Dealing with EGDs

Example

\[ D = \{ N(a) \}, \Sigma : \]

\[
\begin{align*}
N(x) & \rightarrow \exists y \ E(x, y) \\
E(x, y) & \rightarrow N(y) \\
E(x, y) & \rightarrow x = y
\end{align*}
\]

\[ N(a) \rightarrow E(a, a) \rightarrow a = \bot_1 \]
Dealing with EGDs

Example

\[ D = \{ N(a) \}, \Sigma : \]

\[
N(x) \rightarrow \exists y \ E(x,y) \\
E(x,y) \rightarrow N(y) \\
E(x,y) \rightarrow x = y
\]

\[ N(a) \rightarrow E(a,a) \rightarrow a = \bot_1 \rightarrow \text{all constraints satisfied!} \]
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \ E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]
Rewriting TGDs and EGDs

Example

\[
\begin{align*}
    r_1 &: \; N(x) & \rightarrow & \exists y \; E(x, y) \\
    r_2 &: \; E(x, y) & \rightarrow & N(y) \\
    r_3 &: \; E(x, y) & \rightarrow & x = y \\
    r'_3 &: \; E^{bb}(x, y) \\
\end{align*}
\]
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \; E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]

\[ r'_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]
Rewriting TGDs and EGDs

**Example**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$N(x) \rightarrow \exists y E(x, y)$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$E(x, y) \rightarrow N(y)$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$E(x, y) \rightarrow x = y$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r'_3$</td>
<td>$E^{bb}(x, y) \rightarrow x = y$</td>
</tr>
<tr>
<td>$r'_2$</td>
<td>$E^{bb}(x, y)$</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$N(x)$</td>
<td>$\exists y \ E(x, y)$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$E(x, y)$</td>
<td>$N(y)$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$E(x, y)$</td>
<td>$x = y$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r'_3$</td>
<td>$E^{bb}(x, y)$</td>
<td>$x = y$</td>
</tr>
<tr>
<td>$r'_2$</td>
<td>$E^{bb}(x, y)$</td>
<td>$N^b(y)$</td>
</tr>
</tbody>
</table>

No cyclic symbol $f_i$ occurs in the constraints above. Thus, there exists a terminating standard chase sequence.
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \quad E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]

\[ r_3' : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]
\[ r_2' : \quad E^{bb}(x, y) \quad \rightarrow \quad N^{b}(y) \]
\[ r_1' : \quad N^{b}(x) \]

No cyclic symbol \( f \) occurs in the constraints above.

Thus, there exists a terminating standard chase sequence.
Rewriting TGDs and EGDs

Example

\[ r_1 : \text{N}(x) \quad \rightarrow \quad \exists y \ E(x, y) \]
\[ r_2 : \text{E}(x, y) \quad \rightarrow \quad \text{N}(y) \]
\[ r_3 : \text{E}(x, y) \quad \rightarrow \quad x = y \]

\[ r'_3 : \text{E}^{bb}(x, y) \quad \rightarrow \quad x = y \]
\[ r'_2 : \text{E}^{bb}(x, y) \quad \rightarrow \quad \text{N}^{b}(y) \]
\[ r'_1 : \text{N}^{b}(x) \quad \rightarrow \quad \exists y \ E^{bf_1}(x, y) \quad f_1 = f_{r_1}^y(b) \]
Rewriting TGDs and EGDs

Example

\[ r_1 : \ N(x) \rightarrow \exists \ y \ E(x, y) \]
\[ r_2 : \ E(x, y) \rightarrow \ N(y) \]
\[ r_3 : \ E(x, y) \rightarrow x = y \]
\[ r'_3 : \ E_{bb}(x, y) \rightarrow x = y \]
\[ r'_2 : \ E_{bb}(x, y) \rightarrow N^b(y) \]
\[ r' : \ N^b(x) \rightarrow \exists \ y \ E^{bf_1}(x, y) \quad f_1 = f_{r_1}(b) \]
\[ r''_3 : \ E^{bf_1}(x, y) \]
### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>( N(x) )</td>
<td>( \exists y \ E(x, y) )</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>( E(x, y) )</td>
<td>( N(y) )</td>
</tr>
<tr>
<td>( r_3 )</td>
<td>( E(x, y) )</td>
<td>( x = y )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r'_3 )</td>
<td>( E^{bb}(x, y) )</td>
<td>( x = y )</td>
</tr>
<tr>
<td>( r'_2 )</td>
<td>( E^{bb}(x, y) )</td>
<td>( N^b(y) )</td>
</tr>
<tr>
<td>( r'_1 )</td>
<td>( N^b(x) )</td>
<td>( \exists y \ E^{bf_1}(x, y) ) ( f_1 = f_{r_1}^y(b) )</td>
</tr>
<tr>
<td>( r''_3 )</td>
<td>( E^{bf_1}(x, y) )</td>
<td>( x = y )</td>
</tr>
</tbody>
</table>
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \; E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]

\[ r'_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]
\[ r'_2 : \quad E^{bb}(x, y) \quad \rightarrow \quad N^b(y) \]
\[ r'_1 : \quad N^b(x) \quad \rightarrow \quad \exists y \; E^{bf_1}(x, y) \quad f_1 = f_{r_1}^b(b) \]
\[ r''_3 : \quad E^{bf_1}(x, y) \quad \rightarrow \quad x = y \quad b = f_1 \]
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \ E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]

\[ r'_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]
\[ r'_2 : \quad E^{bb}(x, y) \quad \rightarrow \quad N^b(y) \]
\[ r'_1 : \quad N^b(x) \quad \rightarrow \quad \exists y \ E^{bb}(x, y) \quad f_1 = f_{r_1}^y(b) \]
\[ r''_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \quad b = f_1 \]
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad \text{N}(x) \quad \rightarrow \quad \exists y \ E(x, y) \]
\[ r_2 : \quad \text{E}(x, y) \quad \rightarrow \quad \text{N}(y) \]
\[ r_3 : \quad \text{E}(x, y) \quad \rightarrow \quad x = y \]

\[ r'_3 : \quad \text{E}^{\text{bb}}(x, y) \quad \rightarrow \quad x = y \]
\[ r'_2 : \quad \text{E}^{\text{bb}}(x, y) \quad \rightarrow \quad \text{N}^{\text{b}}(y) \]
\[ r'_1 : \quad \text{N}^{\text{b}}(x) \quad \rightarrow \quad \exists y \ \text{E}^{\text{bb}}(x, y) \]
\[ r''_3 : \quad \text{E}^{\text{bb}}(x, y) \quad \rightarrow \quad x = y \]
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \quad E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]

\[ r'_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]
\[ r'_2 : \quad E^{bb}(x, y) \quad \rightarrow \quad N^b(y) \]
\[ r'_1 : \quad N^b(x) \quad \rightarrow \quad \exists y \quad E^{bb}(x, y) \]
\[ r''_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]

No cyclic symbol \( f_i \) occurs in the constraints above.
Rewriting TGDs and EGDs

Example

\[ r_1 : \quad N(x) \quad \rightarrow \quad \exists y \; E(x, y) \]
\[ r_2 : \quad E(x, y) \quad \rightarrow \quad N(y) \]
\[ r_3 : \quad E(x, y) \quad \rightarrow \quad x = y \]
\[ r'_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]
\[ r'_2 : \quad E^{bb}(x, y) \quad \rightarrow \quad N^b(y) \]
\[ r'_1 : \quad N^b(x) \quad \rightarrow \quad \exists y \; E^{bb}(x, y) \]
\[ r''_3 : \quad E^{bb}(x, y) \quad \rightarrow \quad x = y \]

No cyclic symbol \( f_i \) occurs in the constraints above.
Thus, there exists a terminating standard chase sequence.
Rewriting TGDs and EGDs

Example

\[ r_1 : \ N(x) \rightarrow \exists y \ E(x, y) \]
\[ r_2 : \ E(x, y) \rightarrow N(y) \]
\[ r_3 : \ E(x, y) \rightarrow x = y \]

\[ r'_3 : \ E^{bb}(x, y) \rightarrow x = y \]
\[ r'_2 : \ E^{bb}(x, y) \rightarrow N^{b}(y) \]
\[ r'_1 : \ N^{b}(x) \rightarrow \exists y \ E^{bb}(x, y) \]
\[ r''_3 : \ E^{bb}(x, y) \rightarrow x = y \]

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.
The rewriting algorithm always terminates;

\[ \Sigma^\alpha \in \mathcal{C}_\exists^{\text{std}} \text{ implies } \Sigma \in \mathcal{C}_\exists^{\text{std}}; \]

Furthermore, if \( \forall \text{ cyclic } f_i \text{ in } \Sigma^\alpha \), then \( \Sigma \in \mathcal{C}_\exists^{\text{std}} \);
Current and Future directions

- Determine decidable classes of data dependencies,
- Consider and extended framework (\textit{Datalog}[\exists, =, F, \neg])
- Define criteria guaranteeing 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?


[CGMT14] M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Checking termination of logic programs with function


