Checking Termination of Logic Programs with Function Symbols Through Linear Constraints

Marco Calautti, Sergio Greco, Cristian Molinaro, Irina Trubitsyna

RuleML 2014 - Prague

August 18-20, 2014

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Example of execution:

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Example of execution:

$\texttt{len}([\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}], 0) \rightarrow$

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Example of execution:

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

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Example of execution:

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

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Example of execution:

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

$$\texttt{len}([\texttt{d}], \texttt{s}(\texttt{s}(\texttt{s}(0)))) \rightarrow$$

Context

- **Context: Bottom-up Evaluation of Logic Programs with Function Symbols**
  - The use of function symbols makes the language more expressive and the resulting encoding more readable and concise;

Example

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

Example of execution:

$$\texttt{len}([\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}], 0) \rightarrow \texttt{len}([\texttt{b}, \texttt{c}, \texttt{d}], \texttt{s}(0)) \rightarrow \texttt{len}([\texttt{c}, \texttt{d}], \texttt{s}(\texttt{s}(0))) \rightarrow$$
$$\texttt{len}([\texttt{d}], \texttt{s}(\texttt{s}(\texttt{s}(0)))) \rightarrow \mathbf{len([\,],s(s(s(s(0)))))}$$

Context

The price to pay is that the **bottom-up** evaluation of programs may be non terminating.

Context

The price to pay is that the **bottom-up** evaluation of programs may be non terminating.

Example

$$r_0: \quad \mathtt{nat}(0).$$
$$r_1: \quad \mathtt{nat}(\mathbf{s}(\mathtt{X})) \leftarrow \mathtt{nat}(\mathtt{X}).$$

The bottom-up evaluation of this program never terminates, producing $nat(0), nat(s(0)), nat(s(s(0))), \ldots$.

Context

The price to pay is that the **bottom-up** evaluation of programs may be non terminating.

Example

$$r_0: \quad \mathtt{nat(0)}.$$
$$r_1: \quad \mathtt{nat(s(X))} \leftarrow \mathtt{nat(X)}.$$

The bottom-up evaluation of this program never terminates, producing $nat(0), nat(s(0)), nat(s(s(0))), ...$

- Establishing if a program has a terminating bottom-up evaluation is an **undecidable problem**;

Context

The price to pay is that the **bottom-up** evaluation of programs may be non terminating.

Example

$$r_0: \quad \mathtt{nat(0)}.$$
$$r_1: \quad \mathtt{nat(s(X))} \leftarrow \mathtt{nat(X)}.$$

The bottom-up evaluation of this program never terminates, producing $nat(0), nat(s(0)), nat(s(s(0))), ...$

- Establishing if a program has a terminating bottom-up evaluation is an **undecidable problem**;
- Recent work has focused on finding sufficient conditions for the termination of logic programs.

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat(0)}.$

r_1 : $\mathtt{nat(s(X))} \leftarrow \mathtt{nat(X)}.$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\mathtt{X})) \leftarrow \mathtt{nat}(\overset{0}{\mathtt{X}}).$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{0}{\mathtt{X}}).$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

$$r_0 : \ \mathtt{nat}(0).$$

$$r_1 : \ \mathtt{nat}(\mathbf{s}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{1}{\mathtt{X}}).$$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{1}{\mathtt{X}}).$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$

No upper bound for $\mathtt{nat}[1]$.

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$

No upper bound for $\mathtt{nat}[1]$.

Example

r_0 : $\mathtt{p}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{q}(\mathtt{X}).$

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

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

r_0 : $\mathtt{nat}(0).$

r_1 : $\mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$

No upper bound for $\mathtt{nat}[1]$.

Example

r_0 : $\mathtt{p}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$

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

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

$$r_0 : \ \mathtt{nat}(0).$$
$$r_1 : \ \mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$$

No upper bound for $\mathtt{nat}[1]$.

Example

$$r_0 : \ \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$
$$r_1 : \ \mathtt{q}(\mathtt{X}) \leftarrow \mathtt{p}(\mathtt{f}(\mathtt{X})).$$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

$$r_0 : \ \mathtt{nat}(0).$$
$$r_1 : \ \mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$$

No upper bound for $\mathtt{nat}[1]$.

Example

$$r_0 : \ \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$
$$r_1 : \ \mathtt{q}(\mathtt{X}) \leftarrow \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})).$$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

$$r_0 : \ \mathtt{nat}(0).$$
$$r_1 : \ \mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$$

No upper bound for $\mathtt{nat}[1]$.

Example

$$r_0 : \ \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$
$$r_1 : \ \mathtt{q}(\overset{0}{\mathtt{X}}) \leftarrow \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})).$$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

Computes, for each predicate argument, an upper bound of the depth of terms that may occur in that argument.

Example

$$r_0: \ \mathtt{nat}(0).$$
$$r_1: \ \mathtt{nat}(\mathbf{s}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{nat}(\overset{2}{\mathtt{X}}).$$

No upper bound for $\mathtt{nat}[1]$.

Example

$$r_0: \ \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$
$$r_1: \ \mathtt{q}(\overset{0}{\mathtt{X}}) \leftarrow \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})).$$

$\mathtt{q}[1] \longrightarrow 0$, $\mathtt{p}[1] \longrightarrow 1$

Argument-restricted criterion [Lierler&Lifschitz, ICLP'09]

- **Pros**:
  - Simple: it just needs to compute a particular level mapping for each argument;
  - Efficient: Computing an argument ranking (if exists) requires polynomial time.
- **Cons**:
  - Only characterizes **the depth of terms** inside arguments **alone**;
  - No distinction between different function symbols;
  - **Few practical (terminating) programs** are recognized.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$

$$r_2 : \texttt{len}(\texttt{Tail}, \mathbf{s}(\texttt{N})) \leftarrow \texttt{len}(\mathbf{list}(\texttt{Head}, \texttt{Tail}), \texttt{N}).$$

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$

$$r_2 : \texttt{len}(\texttt{Tail}, \mathbf{s}(\texttt{N})) \leftarrow \texttt{len}(\mathbf{list}(\texttt{Head}, \texttt{Tail}), \texttt{N}).$$

The program is terminating, but not recognized by argument-restriction.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$

$$r_2 : \texttt{len}(\texttt{Tail}, \mathbf{s}(\texttt{N})) \leftarrow \texttt{len}(\mathbf{list}(\texttt{Head}, \texttt{Tail}), \overset{0}{\texttt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$\mathtt{r}_1 : \mathtt{len}(\mathtt{List}, 0) \leftarrow \mathtt{input}(\mathtt{List}).$$

$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \overset{1}{\mathbf{s}}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{0}{\mathtt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$\mathtt{r}_1 : \mathtt{len}(\mathtt{List}, 0) \leftarrow \mathtt{input}(\mathtt{List}).$$

$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \overset{1}{\mathbf{s}}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{1}{\mathtt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$
$$r_2 : \texttt{len}(\texttt{Tail}, \overset{2}{\texttt{s}}(\texttt{N})) \leftarrow \texttt{len}(\textbf{list}(\texttt{Head}, \texttt{Tail}), \overset{1}{\texttt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$

$$r_2 : \texttt{len}(\texttt{Tail}, \overset{2}{\texttt{s}}(\texttt{N})) \leftarrow \texttt{len}(\texttt{list}(\texttt{Head}, \texttt{Tail}), \overset{2}{\texttt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$
$$r_2 : \texttt{len}(\texttt{Tail}, \overset{2}{\texttt{s}}(\texttt{N})) \leftarrow \texttt{len}(\texttt{list}(\texttt{Head}, \texttt{Tail}), \overset{2}{\texttt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

- Very simple notion of atom size (we can do better): sum all the terms depths.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$r_1 : \texttt{len}(\texttt{List}, 0) \leftarrow \texttt{input}(\texttt{List}).$$
$$r_2 : \texttt{len}(\texttt{Tail}, \overset{2}{\texttt{s}}(\texttt{N})) \leftarrow \texttt{len}(\texttt{list}(\texttt{Head}, \texttt{Tail}), \overset{2}{\texttt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

- Very simple notion of atom size (we can do better): sum all the terms depths.
  - $\texttt{size}(\texttt{head}(r_2)) = 0 + 1$;
  - $\texttt{size}(\texttt{body}(r_2)) = 1 + 0$.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$\mathtt{r}_1 : \mathtt{len}(\mathtt{List}, 0) \leftarrow \mathtt{input}(\mathtt{List}).$$

$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \overset{2}{\mathbf{s}}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{2}{\mathtt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

- Very simple notion of atom size (we can do better): sum all the terms depths.
  - $\mathtt{size}(\mathtt{head}(\mathtt{r}_2)) = 0 + 1$;
  - $\mathtt{size}(\mathtt{body}(\mathtt{r}_2)) = 1 + 0$.

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

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$\mathtt{r}_1 : \mathtt{len}(\mathtt{List}, 0) \leftarrow \mathtt{input}(\mathtt{List}).$$

$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \overset{2}{\mathbf{s}}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{2}{\mathtt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

- Very simple notion of atom size (we can do better): sum all the terms depths.
  - $\mathtt{size}(\mathtt{head}(\mathtt{r}_2)) = 0 + 1$;
  - $\mathtt{size}(\mathtt{body}(\mathtt{r}_2)) = 1 + 0$.

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

- The overall size of the head will never increase **w.r.t. the body**.

The idea

Idea: Many practical programs contain rules that propagate terms **without increasing** the **overall** size of the head atom.

Example

$$\mathtt{r}_1 : \mathtt{len}(\mathtt{List}, 0) \leftarrow \mathtt{input}(\mathtt{List}).$$
$$\mathtt{r}_2 : \mathtt{len}(\mathtt{Tail}, \overset{2}{\mathbf{s}}(\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{2}{\mathtt{N}}).$$

The program is terminating, but not recognized by argument-restriction.

- Very simple notion of atom size (we can do better): sum all the terms depths.
  - $\mathtt{size}(\mathtt{head}(\mathtt{r}_2)) = 0 + 1$;
  - $\mathtt{size}(\mathtt{body}(\mathtt{r}_2)) = 1 + 0$.

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

- The overall size of the head will never increase **w.r.t. the body**.

If *all* rules satisfy this condition, the program is terminating.

Firing Graph

Not all rules that increase the head size are **dangerous**.

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

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

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

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p(f(X))} \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$

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

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\overset{1}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$

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

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\overset{1}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}(\overset{1}{\mathtt{g}}(\mathtt{X})).$$

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\overset{1}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{q}(\overset{0}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\overset{1}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{p}(\overset{1}{\mathtt{g}}(\mathtt{X})).$$

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{p}(\mathtt{g}(\overset{1}{\mathtt{X}})).$$

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\overset{2}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\overset{1}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{p}(\overset{1}{\mathtt{g}}(\mathtt{X})).$$

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\mathtt{f}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{p}(\mathtt{g}(\overset{1}{\mathtt{X}})).$$

The program is not argument-restricted.

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\mathtt{f}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{p}(\mathtt{g}(\overset{1}{\mathtt{X}})).$$

The program is not argument-restricted.

However, rule r_1 will never "trigger" r_2.

Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\overset{2}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\overset{1}{\mathtt{f}}(\mathtt{X})) \leftarrow \mathtt{p}(\overset{1}{\mathtt{g}}(\mathtt{X})).$$

The program is not argument-restricted.

However, rule r_1 will never "trigger" r_2.

Definition (Firing Graph)

Directed graph whose nodes are the rules of the program and there is an edge from r_1 to r_2 if r_1 fires r_2.

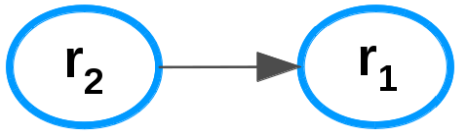
Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\mathtt{f}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{p}(\mathtt{g}(\overset{1}{\mathtt{X}})).$$



The program is not argument-restricted.

However, rule r_1 will never "trigger" r_2.

Definition (Firing Graph)

Directed graph whose nodes are the rules of the program and there is an edge from r_1 to r_2 if r_1 fires r_2.

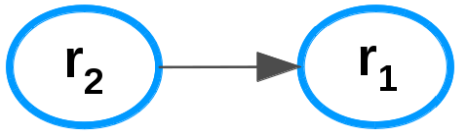
Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\mathtt{f}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{p}(\mathtt{g}(\overset{1}{\mathtt{X}})).$$



The program is not argument-restricted.

However, rule r_1 will never "trigger" r_2.

Definition (Firing Graph)

Directed graph whose nodes are the rules of the program and there is an edge from r_1 to r_2 if r_1 fires r_2.

There is no cyclic execution of r_1 and r_2.

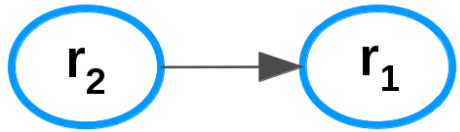
Firing Graph

Not all rules that increase the head size are **dangerous**.

Example

$$r_1 : \mathtt{p}(\mathtt{f}(\overset{2}{\mathtt{X}})) \leftarrow \mathtt{q}(\overset{1}{\mathtt{X}}).$$

$$r_2 : \mathtt{q}(\mathtt{f}(\overset{1}{\mathtt{X}})) \leftarrow \mathtt{p}(\mathtt{g}(\overset{1}{\mathtt{X}})).$$



The program is not argument-restricted.

However, rule r_1 will never "trigger" r_2.

Definition (Firing Graph)

Directed graph whose nodes are the rules of the program and there is an edge from r_1 to r_2 if r_1 fires r_2.

There is no cyclic execution of r_1 and r_2.

Goal: Check termination of the SCCs of the Firing Graph.

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation.

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

The rule $\mathtt{p}(\mathtt{X}, \mathtt{c}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{c}))$ is rewritten to $\mathtt{p}(\mathtt{X}, \mathtt{Z}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})), \mathtt{b}(\mathtt{Z})$. Where the only extension of $\mathtt{b}$ is $\mathtt{b}(\mathtt{c})$.

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

The rule $\mathtt{p(X, c)} \leftarrow \mathtt{p(X, f(X, Y, c))}$ is rewritten to $\mathtt{p(X, Z)} \leftarrow \mathtt{p(X, f(X, Y, Z))}, \mathtt{b(Z)}$. Where the only extension of $\mathtt{b}$ is $\mathtt{b(c)}$.

- Variable Y does not partecipate to the propagation of values, thus its size is 0;

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

The rule $\mathtt{p}(\mathtt{X}, \mathtt{c}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{c}))$ is rewritten to $\mathtt{p}(\mathtt{X}, \mathtt{Z}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})), \mathtt{b}(\mathtt{Z})$. Where the only extension of $\mathtt{b}$ is $\mathtt{b}(\mathtt{c})$.

- Variable Y does not partecipate to the propagation of values, thus its size is 0;
- Variables X and Z are propagated, their size is represented by the mathematical variables x and z;

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

The rule $\mathtt{p}(\mathtt{X}, \mathtt{c}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{c}))$ is rewritten to $\mathtt{p}(\mathtt{X}, \mathtt{Z}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})), \mathtt{b}(\mathtt{Z})$. Where the only extension of $\mathtt{b}$ is $\mathtt{b}(\mathtt{c})$.

- Variable Y does not partecipate to the propagation of values, thus its size is 0;
- Variables X and Z are propagated, their size is represented by the mathematical variables x and z;
- Finally, since the size of X is x and the size of Z is z, the size of the complex term $\mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})$ is $(1 + x) + 0 + (1 + z)$.

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

The rule $\mathtt{p}(\mathtt{X}, \mathtt{c}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{c}))$ is rewritten to $\mathtt{p}(\mathtt{X}, \mathtt{Z}) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})), \mathtt{b}(\mathtt{Z})$. Where the only extension of $\mathtt{b}$ is $\mathtt{b}(\mathtt{c})$.

- Variable Y does not partecipate to the propagation of values, thus its size is 0;
- Variables X and Z are propagated, their size is represented by the mathematical variables x and z;
- Finally, since the size of X is x and the size of Z is z, the size of the complex term $\mathtt{f}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})$ is $(1 + x) + 0 + (1 + z)$.

It is clear that this notion of size is dependent on the considered rule.

Term size

Intuition: The size of a term is a template for all possible sizes the term may have during the program evaluation. We assume *constant-free* programs.

The rule $\mathtt{p(X, c)} \leftarrow \mathtt{p(X, f(X, Y, c))}$ is rewritten to $\mathtt{p(X, Z)} \leftarrow \mathtt{p(X, f(X, Y, Z))}, \mathtt{b(Z)}$. Where the only extension of $\mathtt{b}$ is $\mathtt{b(c)}$.

- Variable Y does not partecipate to the propagation of values, thus its size is 0;
- Variables X and Z are propagated, their size is represented by the mathematical variables x and z;
- Finally, since the size of X is x and the size of Z is z, the size of the complex term $\mathtt{f(X, Y, Z)}$ is $(1 + x) + 0 + (1 + z)$.

It is clear that this notion of size is dependent on the considered rule.

size(t,r) is the size of term t w.r.t. rule r

Atom size

The size of an atom is a **linear combination of its terms sizes**.

Atom size

The size of an atom is a **linear combination of its terms sizes**.

Given a rule r and an atom $A = p(t_1, \ldots, t_n)$ in r, the **size of** A is:

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

where $\alpha_{p_1}, \ldots, \alpha_{p_n}$ are **positive integers**.

Atom size

The size of an atom is a **linear combination of its terms sizes**.

Given a rule r and an atom $A = p(t_1, \ldots, t_n)$ in r, the **size of** A is:

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

where $\alpha_{p_1}, \ldots, \alpha_{p_n}$ are **positive integers**.

We allow for the coefficients α_i to be not fixed a priori.

Atom size

The size of an atom is a **linear combination of its terms sizes**.

Given a rule r and an atom $A = p(t_1, \ldots, t_n)$ in r, the **size of** A is:

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

where $\alpha_{p_1}, \ldots, \alpha_{p_n}$ are **positive integers**.

We allow for the coefficients α_i to be not fixed a priori.

They will be chosen depending on the program structure.

Rule-bounded programs

Definition

A program $\mathcal{P}$ is *rule-bounded* if for every SCC $\mathcal{C}$, every rule r in $\mathcal{C}$ is such that:

$$size(body(r), r) \geq size(head(r), r)$$

for some fixed set of parameters α_i.

Rule-bounded programs

Example

Consider the length program:

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

Rule-bounded programs

Example

Consider the length program:

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

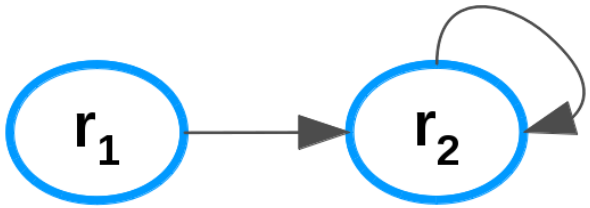


Figure: Firing Graph

Rule-bounded programs

Example

Consider the length program:

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

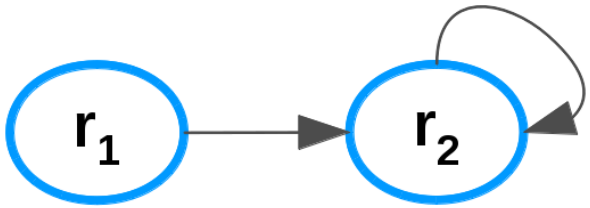


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

Rule-bounded programs

Example

Consider the length program:

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

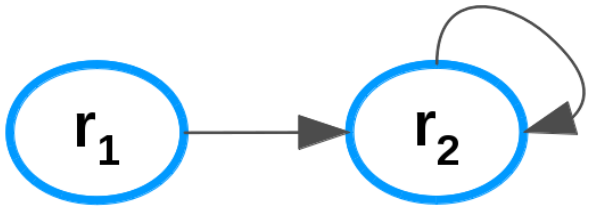


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)]$$

Rule-bounded programs

Example

Consider the length program:

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

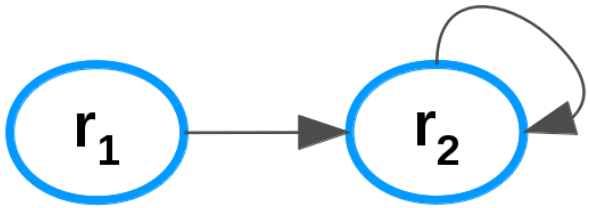


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n$$

Rule-bounded programs

Example

Consider the length program:

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

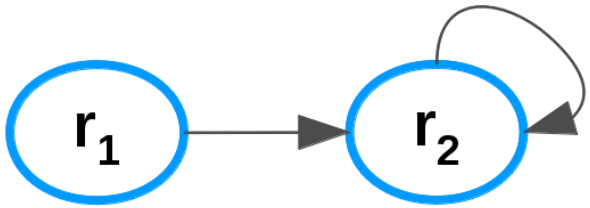


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n \geq \alpha_1 \cdot tail$$

Rule-bounded programs

Example

Consider the length program:

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

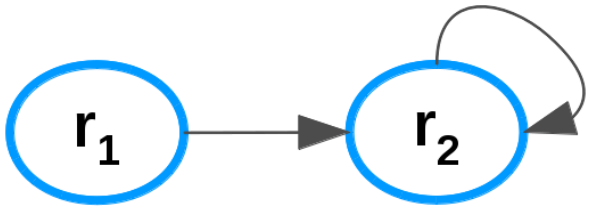


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

Rule-bounded programs

Example

Consider the length program:

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

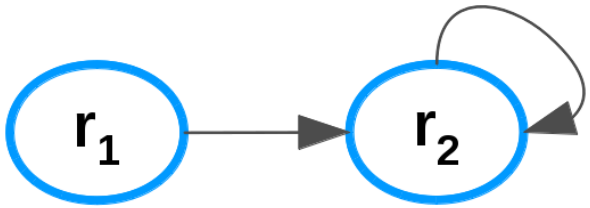


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

$$\alpha_1 + \alpha_1 \cdot tail + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 + \alpha_2 \cdot n$$

Rule-bounded programs

Example

Consider the length program:

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

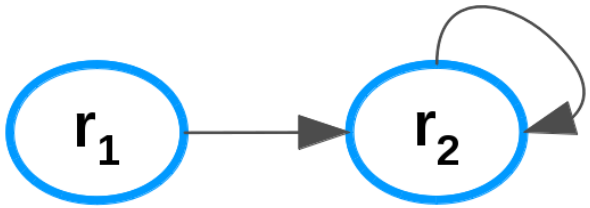


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

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

Rule-bounded programs

Example

Consider the length program:

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

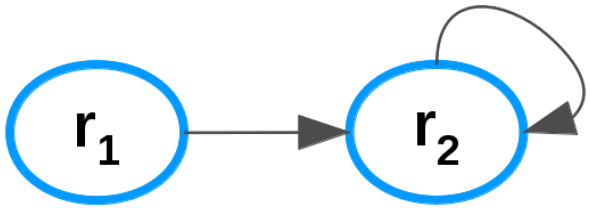


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

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

We can choose $\alpha_1 = \alpha_2 = 1$

Rule-bounded programs

Example

Consider the length program:

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

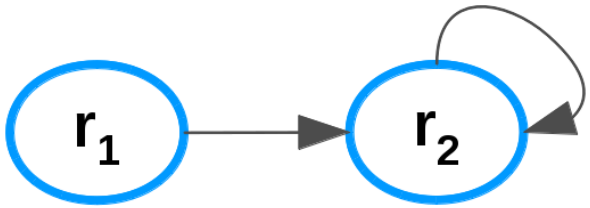


Figure: Firing Graph

The only SCC is $\mathcal{C} = \{r_2\}$, thus we need to check:

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

$$\alpha_1 \cdot [0 + (1 + tail)] + \alpha_2 \cdot n \geq \alpha_1 \cdot tail + \alpha_2 \cdot (1 + n)$$

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

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

Rule-bounded programs

Theorem

The bottom-up evaluation of a rule-bounded program always terminates.

Rule-bounded programs

Theorem

The bottom-up evaluation of a rule-bounded program always terminates.

Theorem

The complexity of checking whether a program is rule-bounded is NP.

Examples of rule-bounded programs

Example (Bubble sort)

$$
\begin{array}{lll}
r_0: & \texttt{sort(List,[],[])} & \leftarrow \quad \texttt{input(List).} \\
r_1: & \texttt{sort([Y|T],[X|Temp],Sorted)} & \leftarrow \quad \texttt{sort([X|[Y|T]]),Temp,Sorted),X} \leq \texttt{Y.} \\
r_2: & \texttt{sort([X|T],[Y|Temp],Sorted)} & \leftarrow \quad \texttt{sort([X|[Y|T]]),Temp,Sorted),Y} < \texttt{X.} \\
r_3: & \texttt{sort(Temp,[],[X|Sorted])} & \leftarrow \quad \texttt{sort([X],Temp,Sorted).}
\end{array}
$$

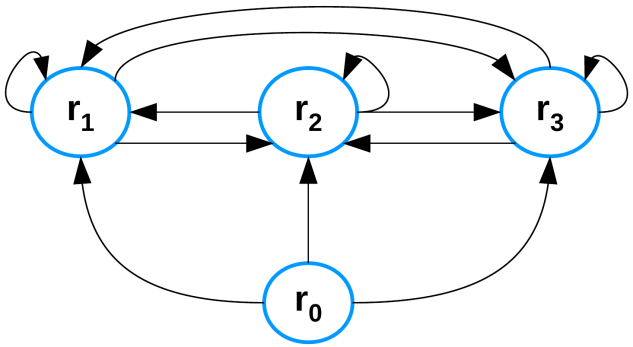


Figure: Bubble sort Firing Graph

$$\text{SCC } \mathcal{C} = \{r_1, r_2, r_3\}$$

Examples of rule-bounded programs

Example (Bubble sort)

$$
\begin{array}{lll}
r_0: & \texttt{sort(List, Nil, Nil)} & \leftarrow \quad \texttt{input(List), b(Nil).} \\
r_1: & \texttt{sort([Y|T], [X|Temp], Sorted)} & \leftarrow \quad \texttt{sort([X|[Y|T]]), Temp, Sorted), X} \leq \texttt{Y.} \\
r_2: & \texttt{sort([X|T], [Y|Temp], Sorted)} & \leftarrow \quad \texttt{sort([X|[Y|T]]), Temp, Sorted), Y} < \texttt{X.} \\
r_3: & \texttt{sort(Temp, Nil, [X|Sorted])} & \leftarrow \quad \texttt{sort([X|Nil], Temp, Sorted), b(Nil).}
\end{array}
$$

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

Examples of rule-bounded programs

Example (Bubble sort)

$$
\begin{array}{lll}
r_0: & \texttt{sort(List, Nil, Nil)} & \leftarrow \quad \texttt{input(List), b(Nil).} \\
r_1: & \texttt{sort([Y|T], [X|Temp], Sorted)} & \leftarrow \quad \texttt{sort([X|[Y|T]]), Temp, Sorted), X} \leq \texttt{Y.} \\
r_2: & \texttt{sort([X|T], [Y|Temp], Sorted)} & \leftarrow \quad \texttt{sort([X|[Y|T]]), Temp, Sorted), Y} < \texttt{X.} \\
r_3: & \texttt{sort(Temp, Nil, [X|Sorted])} & \leftarrow \quad \texttt{sort([X|Nil], Temp, Sorted), b(Nil).}
\end{array}
$$

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

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

Examples of rule-bounded programs

Example (Tree visit)

$r_0 : \texttt{visit}(\texttt{Tree}, [\,], [\,]) \leftarrow \texttt{input}(\texttt{Tree}).$

$r_1 : \texttt{visit}(\texttt{Left}, [\texttt{Root}|\texttt{Visited}], [\texttt{Right}|\texttt{ToVisit}]) \leftarrow \texttt{visit}(\texttt{tree}(\texttt{Root}, \texttt{Left}, \texttt{Right}), \texttt{Visited}, \texttt{ToVisit}).$

$r_2 : \texttt{visit}(\texttt{Next}, \texttt{Visited}, \texttt{ToVisit}) \leftarrow \texttt{visit}(\texttt{nil}, \texttt{Visited}, [\texttt{Next}|\texttt{ToVisit}]).$

$$\begin{cases} \alpha_1 \cdot (3 + \mathit{root} + \mathit{left} + \mathit{right}) + \alpha_2 \cdot \mathit{visited} + \alpha_3 \cdot \mathit{tovisit} \geq \\ \qquad \alpha_1 \cdot \mathit{left} + \alpha_2 \cdot (2 + \mathit{root} + \mathit{visited}) + \alpha_3 \cdot (2 + \mathit{right} + \mathit{tovisit}) \\ \alpha_1 \cdot 0 + \alpha_2 \cdot \mathit{visited} + \alpha_3 \cdot (2 + \mathit{next} + \mathit{tovisit}) \geq \\ \qquad \alpha_1 \cdot \mathit{next} + \alpha_2 \cdot \mathit{visited} + \alpha_3 \cdot \mathit{tovisit} \end{cases}$$

Examples of rule-bounded programs

Example (Tree visit)

$r_0 : \texttt{visit}(\texttt{Tree}, [\,], [\,]) \leftarrow \texttt{input}(\texttt{Tree}).$

$r_1 : \texttt{visit}(\texttt{Left}, [\texttt{Root}|\texttt{Visited}], [\texttt{Right}|\texttt{ToVisit}]) \leftarrow \texttt{visit}(\texttt{tree}(\texttt{Root}, \texttt{Left}, \texttt{Right}), \texttt{Visited}, \texttt{ToVisit}).$

$r_2 : \texttt{visit}(\texttt{Next}, \texttt{Visited}, \texttt{ToVisit}) \leftarrow \texttt{visit}(\texttt{nil}, \texttt{Visited}, [\texttt{Next}|\texttt{ToVisit}]).$

$$\begin{cases} \alpha_1 \cdot (3 + \mathit{root} + \mathit{left} + \mathit{right}) + \alpha_2 \cdot \mathit{visited} + \alpha_3 \cdot \mathit{tovisit} \geq \\ \qquad \alpha_1 \cdot \mathit{left} + \alpha_2 \cdot (2 + \mathit{root} + \mathit{visited}) + \alpha_3 \cdot (2 + \mathit{right} + \mathit{tovisit}) \\ \alpha_1 \cdot 0 + \alpha_2 \cdot \mathit{visited} + \alpha_3 \cdot (2 + \mathit{next} + \mathit{tovisit}) \geq \\ \qquad \alpha_1 \cdot \mathit{next} + \alpha_2 \cdot \mathit{visited} + \alpha_3 \cdot \mathit{tovisit} \end{cases}$$

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

Dealing with non-increasing cycles

Example

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

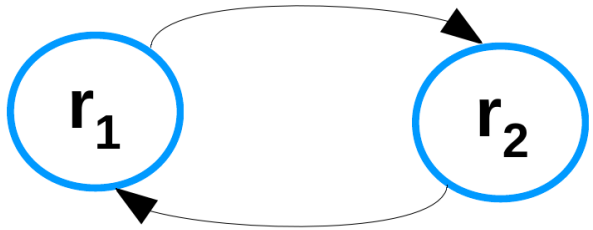


Figure: Firing Graph

The program terminates, but...

Dealing with non-increasing cycles

Example

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

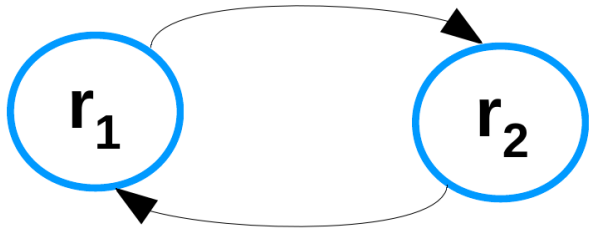


Figure: Firing Graph

The program terminates, but...

Rule r_2 increases its head size $\Rightarrow$ program is not rule-bounded.

Dealing with non-increasing cycles

Example

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

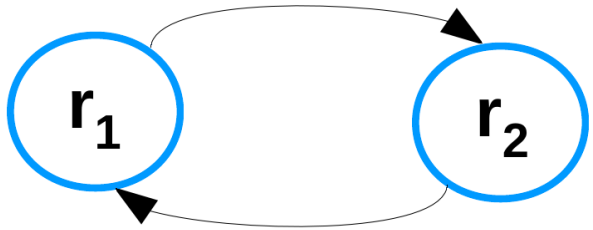


Figure: Firing Graph

The program terminates, but...

Rule r_2 increases its head size $\Rightarrow$ program is not rule-bounded.

But the **cycle** does not increase the size of propagated values.

Dealing with non-increasing cycles

Example

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

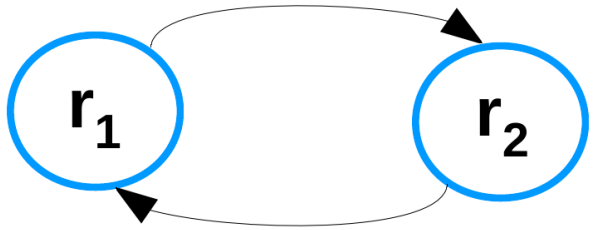


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

Dealing with non-increasing cycles

Example

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

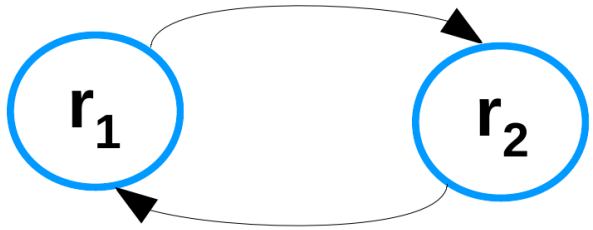


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\mathtt{q(f(X),Y)} \longrightarrow \mathtt{p(X,Y)}}_{r_1}$$

Dealing with non-increasing cycles

Example

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

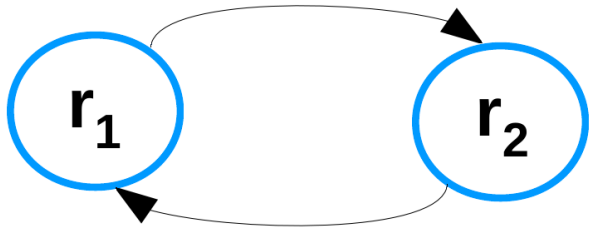


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\mathtt{q(f(X),Y)} \rightarrow \mathtt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\mathtt{p(W,Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_2}$$

Dealing with non-increasing cycles

Example

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

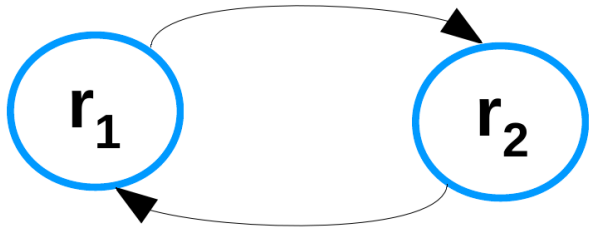


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\texttt{q(f(X),Y)} \longrightarrow \texttt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\texttt{p(W,Z)} \longrightarrow \texttt{q(W,f(Z))}}_{r_2}$$

Dealing with non-increasing cycles

Example

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

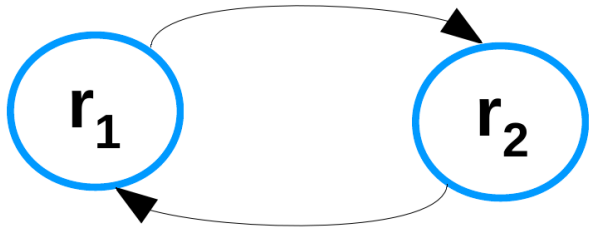


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\texttt{q(f(X),Y)} \longrightarrow \texttt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\texttt{p(W,Z)} \longrightarrow \texttt{q(W,f(Z))}}_{r_2}$$

$$\texttt{X/W, Y/Z}$$

Dealing with non-increasing cycles

Example

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

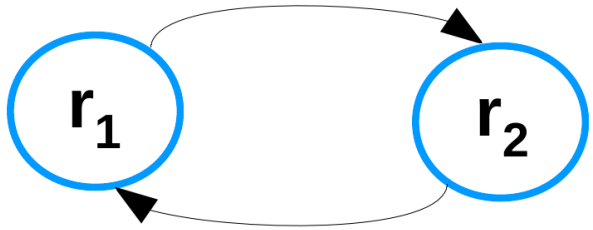


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\mathtt{q(f(X),Y)} \longrightarrow \mathtt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\mathtt{p(W,Z)} \longrightarrow \mathtt{q(W,f(Z))}}_{r_2}$$

$$\mathtt{X/W,\ Y/Z}$$

$$\underbrace{\mathtt{q(f(W),Z)} \longrightarrow \mathtt{q(W,f(Z))}}_{r_{12}}$$

Dealing with non-increasing cycles

Example

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

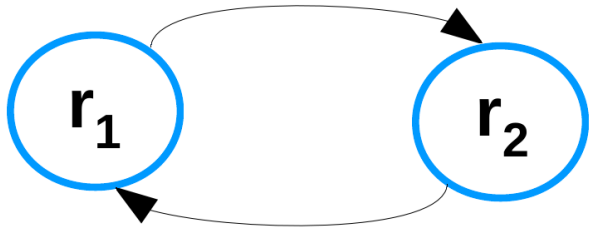


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\mathtt{q(f(X),Y)} \rightarrow \mathtt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\mathtt{p(W,Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_2}$$

$$\mathtt{X/W,\ Y/Z}$$

$$\underbrace{\mathtt{q(f(W),Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_{12}}$$

$$\alpha_{q_1} \cdot (1 + w) + \alpha_{q_2} \cdot z \geq \alpha_{q_1} \cdot w + \alpha_{q_2} \cdot (1 + z)$$

Dealing with non-increasing cycles

Example

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

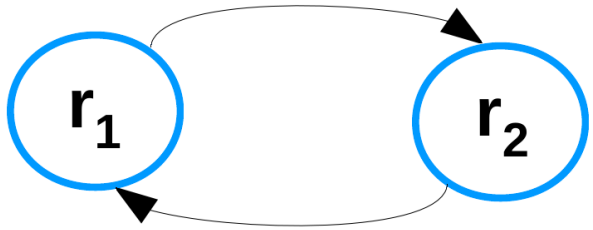


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\mathtt{q(f(X),Y)} \rightarrow \mathtt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\mathtt{p(W,Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_2}$$

$$\mathtt{X/W,\ Y/Z}$$

$$\underbrace{\mathtt{q(f(W),Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_{12}}$$

$$\alpha_{q_1} \cdot (1 + w) + \alpha_{q_2} \cdot z \geq \alpha_{q_1} \cdot w + \alpha_{q_2} \cdot (1 + z)$$

$$\alpha_{q_1} \geq \alpha_{q_2}$$

Dealing with non-increasing cycles

Example

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

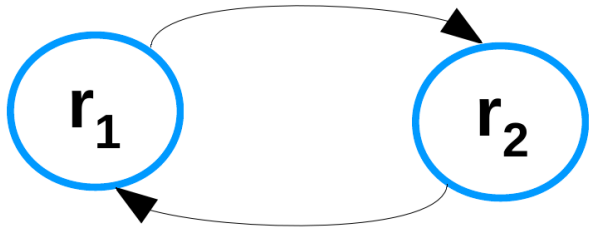


Figure: Firing Graph

We can rewrite the cycle r_1, r_2 as a single rule:

$$\underbrace{\mathtt{q(f(X),Y)} \rightarrow \mathtt{p(X,Y)}}_{r_1} \Rightarrow \underbrace{\mathtt{p(W,Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_2}$$

$$\mathtt{X/W,\ Y/Z}$$

$$\underbrace{\mathtt{q(f(W),Z)} \rightarrow \mathtt{q(W,f(Z))}}_{r_{12}}$$

$$\alpha_{q_1} \cdot (1 + w) + \alpha_{q_2} \cdot z \geq \alpha_{q_1} \cdot w + \alpha_{q_2} \cdot (1 + z)$$

$$\alpha_{q_1} \geq \alpha_{q_2} \qquad \alpha_{q_1} = \alpha_{q_2} = 1$$

Conclusion

Contributions:

- Using linear constraints for checking bottom-up termination;
- The technique is complementary to the other techniques that analyze single arguments;
- The technique recognizes several practical logic programs;

Future work:

- Combine rule-bounded programs with other techniques in the literature;
- Complexity analysis of the proposed technique (there may be many tractable cases);
- Study the termination problem for programs with interpreted function symbols (none of the currently known techniques support them).

Some references

1. Bounded programs: a new decidable class of logic programs with function symbols, *Greco et al.*, IJCAI '13.
2. Logic programming with function symbols: checking termination of bottom-up evaluation through program adornments, *Greco et al.*, ICLP '13 (to appear in TPLP journal).
3. On the termination of logic programs with funtction symbols, *Greco et al.*, ICLP '12 (TC).
4. Incomplete data and data dependencies in relational databases, *Greco et al.*, Syntesis lectures on data management. Morgan and Claypool Publishers '12.
5. One more decidable class of finitely ground programs, *Yuliya Lierler and Vladimir Lifschitz*, ICLP '09.
6. Computable functions in ASP: Theory and Implementation, *Calimeri et al.*, ICLP '08.
7. Termination detection in logic programs using argument sizes, *Kirack Sohn and Allen Van Gelder*, PODS '91.

Thank you for your attention!