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

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

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

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

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

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

Example of execution:

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

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

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

Example of execution:

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

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

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

Example of execution:

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

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

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

Example of execution:

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

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

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

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

Example of execution:

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

$$\mathtt{len}([\mathrm{d}],\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0})))) \rightarrow \mathbf{len([\,],s(s(s(s(0)))))}$$

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

$$\begin{array}{ll} r_0: & \texttt{nat}(\texttt{0}). \\ r_1: & \texttt{nat}(\mathbf{s}(\texttt{X})) \leftarrow \texttt{nat}(\texttt{X}). \end{array}$$

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

$$\begin{array}{ll} r_0: & \texttt{nat(0)}. \\ r_1: & \texttt{nat}(\mathbf{s}(\texttt{X})) \leftarrow \texttt{nat(X)}. \end{array}$$

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

- 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

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

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

- 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

No upper bound for `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

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

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

No upper bound for `nat[1]`.

Example

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

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

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

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

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

No upper bound for 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

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

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

No upper bound for 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

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

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

No upper bound for 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

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

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

No upper bound for 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

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

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

No upper bound for nat[1].

Example

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

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

q[1] → 0, p[1] → 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;
  - Very **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

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

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

The idea

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

Example

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

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

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

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

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

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

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

$$r_2 : \ \mathtt{len}(\mathtt{Tail}, \mathbf{s}(\overset{1}{\mathtt{N}})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{0}{\mathtt{N}}).$$

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

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

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

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

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

$$r_2: \ \mathtt{len}(\mathtt{Tail}, \mathbf{s}\overset{2}{(}\mathtt{N})) \leftarrow \mathtt{len}(\mathbf{list}(\mathtt{Head}, \mathtt{Tail}), \overset{1}{\mathtt{N}}).$$

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

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

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

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

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

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

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.

The idea

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

Example

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

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.
    - $\mathtt{size}(\mathtt{head}(\mathtt{r}_2)) = 0 + 1$;
    - $\mathtt{size}(\mathtt{body}(\mathtt{r}_2)) = 1 + 0$.

The idea

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

Example

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

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

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

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

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

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

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

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

$$
\begin{array}{rlcl}
r_1: & \mathtt{p(f(X))} & \leftarrow & \mathtt{q(X)}. \\
r_2: & \mathtt{q(f(X))} & \leftarrow & \mathtt{p(g(X))}.
\end{array}
$$

Firing Graph

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

Example

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

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

Firing Graph

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

Example

$$
\begin{aligned}
r_1 :\ & \mathtt{p(f(\overset{1}{X}))} & \leftarrow\ & \mathtt{q(\overset{0}{X})}. \\
r_2 :\ & \mathtt{q(f(X))} & \leftarrow\ & \mathtt{p(g(X))}.
\end{aligned}
$$

Firing Graph

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

Example

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

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

Firing Graph

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

Example

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

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

Firing Graph

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

Example

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

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

Firing Graph

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

Example

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

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

Firing Graph

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

Example

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

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

The program is not argument-restricted.

Firing Graph

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

Example

$$\begin{aligned} r_1 &: \; \mathtt{p(f(\overset{2}{X}))} \;\leftarrow\; \mathtt{q(\overset{1}{X})}. \\ r_2 &: \; \mathtt{q(f(\overset{1}{X}))} \;\leftarrow\; \mathtt{p(g(\overset{1}{X}))}. \end{aligned}$$

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

$$\begin{aligned} r_1 :\ & \mathtt{p(f(\overset{2}{X}))} && \leftarrow && \mathtt{q(\overset{1}{X})}. \\ r_2 :\ & \mathtt{q(f(\overset{1}{X}))} && \leftarrow && \mathtt{p(g(\overset{1}{X}))}. \end{aligned}$$

The program is not argument-restricted.

However, rule r_1 will never “trigger” r_2.

Definition (Firing Graph)

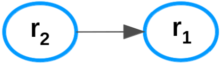
Directed graph whose nodes are the rules of the program and there is an edge from r_1 to r_2 if r_1 fires r_2.

Firing Graph

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

Example

$$
\begin{aligned}
r_1 &: \mathtt{p(f(}\overset{2}{\mathtt{X}}\mathtt{))} \leftarrow \mathtt{q(}\overset{1}{\mathtt{X}}\mathtt{).}\\
r_2 &: \mathtt{q(f(}\overset{1}{\mathtt{X}}\mathtt{))} \leftarrow \mathtt{p(g(}\overset{1}{\mathtt{X}}\mathtt{)).}
\end{aligned}
$$



The program is not argument-restricted.

However, rule r_1 will never “trigger” r_2.

Definition (Firing Graph)

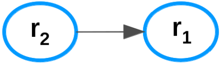
Directed graph whose nodes are the rules of the program and there is an edge from r_1 to r_2 if r_1 fires r_2.

Firing Graph

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

Example

$$
\begin{aligned}
r_1 &: \mathtt{p}(\overset{2}{\mathtt{f(X)}}) &\leftarrow&\quad \mathtt{q}(\overset{1}{\mathtt{X}}).\\
r_2 &: \mathtt{q}(\overset{1}{\mathtt{f(X)}}) &\leftarrow&\quad \mathtt{p}(\overset{1}{\mathtt{g(X)}}).
\end{aligned}
$$



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.

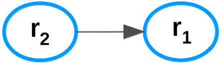
Firing Graph

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

Example

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

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



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.

Consider the rule $\mathtt{p(X, c)} \leftarrow \mathtt{p(Y, f(X, X, c))}$.

Term size

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

Consider the rule $\mathtt{p(X, c) \leftarrow p(Y, f(X, X, c))}$.

- Since terms c and Y do not partecipate to the propagation of values, their 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.

Consider the rule $\mathtt{p}(\mathtt{X},\mathtt{c}) \leftarrow \mathtt{p}(\mathtt{Y},\mathtt{f}(\mathtt{X},\mathtt{X},\mathtt{c}))$.

- Since terms c and Y do not partecipate to the propagation of values, their size is 0;
- Since variable X is propagated, its size is represented by the mathematical variable x;

Term size

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

Consider the rule $\mathtt{p}(\mathtt{X}, \mathtt{c}) \leftarrow \mathtt{p}(\mathtt{Y}, \mathtt{f}(\mathtt{X}, \mathtt{X}, \mathtt{c}))$.

- Since terms c and Y do not partecipate to the propagation of values, their size is 0;
- Since variable X is propagated, its size is represented by the mathematical variable x;
- Finally, since the size of the simple term X is x and the size of c is 0, the size of the complex term $\mathtt{f}(\mathtt{X}, \mathtt{X}, \mathtt{c})$ is $(1 + x) + (1 + x) + 0$.

Term size

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

Consider the rule $\mathtt{p}(\mathtt{X},\mathtt{c}) \leftarrow \mathtt{p}(\mathtt{Y},\mathtt{f}(\mathtt{X},\mathtt{X},\mathtt{c}))$.

- Since terms c and Y do not partecipate to the propagation of values, their size is 0;
- Since variable X is propagated, its size is represented by the mathematical variable x;
- Finally, since the size of the simple term X is x and the size of c is 0, the size of the complex term $\mathtt{f}(\mathtt{X},\mathtt{X},\mathtt{c})$ is $(1+x)+(1+x)+0$.

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.

Consider the rule $\mathtt{p(X,c)} \leftarrow \mathtt{p(Y,f(X,X,c))}$.

- Since terms c and Y do not partecipate to the propagation of values, their size is 0;
- Since variable X is propagated, its size is represented by the mathematical variable x;
- Finally, since the size of the simple term X is x and the size of c is 0, the size of the complex term $\mathtt{f(X,X,c)}$ is $(1+x)+(1+x)+0$.

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:

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

Rule-bounded programs

Example

Consider the length program:

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

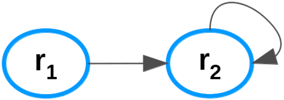


Figure: Firing Graph

Rule-bounded programs

Example

Consider the length program:

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

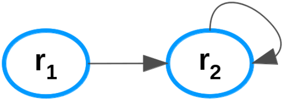


Figure: Firing Graph

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:

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

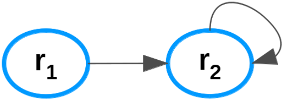


Figure: Firing Graph

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

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

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

Rule-bounded programs

Example

Consider the length program:

$r_1: \ \mathtt{len}([\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}],\mathtt{0}).$

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

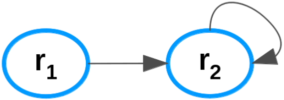


Figure: Firing Graph

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:

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

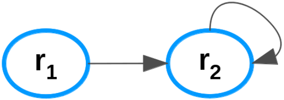


Figure: Firing Graph

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:

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

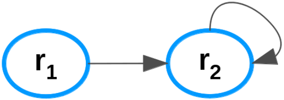


Figure: Firing Graph

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:

$\mathtt{r}_1 :\ \mathtt{len}([\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}], 0).$

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

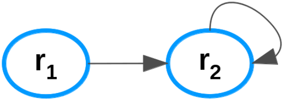


Figure: Firing Graph

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

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

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

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

Rule-bounded programs

Example

Consider the length program:

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

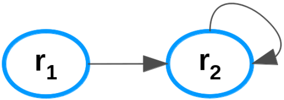


Figure: Firing Graph

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:

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

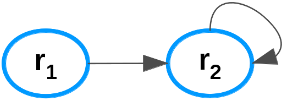


Figure: Firing Graph

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

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

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

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

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

Rule-bounded programs

Example

Consider the length program:

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

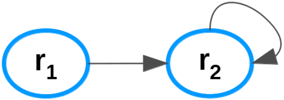


Figure: Firing Graph

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

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

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

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

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)

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

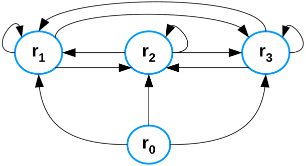


Figure: Bubble sort Firing Graph

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

Examples of rule-bounded programs

Example (Bubble sort)

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

$$\begin{cases}
\alpha_1 \cdot (4+x+y+t) + \alpha_2 \cdot temp + \alpha_3 \cdot sorted \ge \\
\qquad \alpha_1 \cdot (2+y+t) + \alpha_2 \cdot (2+x+temp) + \alpha_3 \cdot sorted \\
\alpha_1 \cdot (4+x+y+t) + \alpha_2 \cdot temp + \alpha_3 \cdot sorted \ge \\
\qquad \alpha_1 \cdot (2+x+t) + \alpha_2 \cdot (2+y+temp) + \alpha_3 \cdot sorted \\
\alpha_1 \cdot (1+x) + \alpha_2 \cdot temp + \alpha_3 \cdot sorted \ge \\
\qquad \alpha_1 \cdot temp + \alpha_2 \cdot 0 + \alpha_3 \cdot (2+x+sorted)
\end{cases}$$

Examples of rule-bounded programs

Example (Bubble sort)

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

$$
\begin{cases}
\alpha_1 \cdot (4+x+y+t) + \alpha_2 \cdot temp + \alpha_3 \cdot sorted \geq \\
\qquad \alpha_1 \cdot (2+y+t) + \alpha_2 \cdot (2+x+temp) + \alpha_3 \cdot sorted \\
\alpha_1 \cdot (4+x+y+t) + \alpha_2 \cdot temp + \alpha_3 \cdot sorted \geq \\
\qquad \alpha_1 \cdot (2+x+t) + \alpha_2 \cdot (2+y+temp) + \alpha_3 \cdot sorted \\
\alpha_1 \cdot (1+x) + \alpha_2 \cdot temp + \alpha_3 \cdot sorted \geq \\
\qquad \alpha_1 \cdot temp + \alpha_2 \cdot 0 + \alpha_3 \cdot (2+x+sorted)
\end{cases}
$$

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

Examples of rule-bounded programs

Example (Tree visit)

r_0 : $\mathtt{visit(Tree,[\,],[\,])} \leftarrow \mathtt{input(Tree)}.$
r_1 : $\mathtt{visit(Left,[Root|Visited],[Right|ToVisit])} \leftarrow$
$\mathtt{visit(tree(Root,Left,Right),Visited,ToVisit)}.$
r_2 : $\mathtt{visit(Next,Visited,ToVisit)} \leftarrow \mathtt{visit(null,Visited,[Next|ToVisit])}.$

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

Examples of rule-bounded programs

Example (Tree visit)

$$
\begin{aligned}
&r_0 : \mathtt{visit(Tree,[\,],[\,])} \leftarrow \mathtt{input(Tree)}.\\
&r_1 : \mathtt{visit(Left,[Root|Visited],[Right|ToVisit])} \leftarrow\\
&\qquad\qquad\qquad \mathtt{visit(tree(Root,Left,Right),Visited,ToVisit)}.\\
&r_2 : \mathtt{visit(Next,Visited,ToVisit)} \leftarrow \mathtt{visit(null,Visited,[Next|ToVisit])}.
\end{aligned}
$$

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

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

Dealing with non-increasing cycles

Example

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

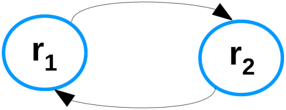


Figure: Firing Graph

The program terminates, but...

Dealing with non-increasing cycles

Example

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

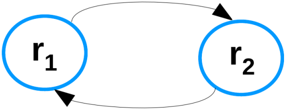


Figure: Firing Graph

The program terminates, but...

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

Dealing with non-increasing cycles

Example

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

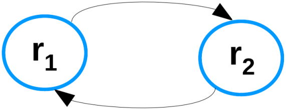


Figure: Firing Graph

The program terminates, but...

Rule r_2 increases its head size ⇒ program is not rule-bounded.

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

Dealing with non-increasing cycles

Example

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

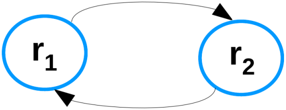


Figure: Firing Graph

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

Dealing with non-increasing cycles

Example

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

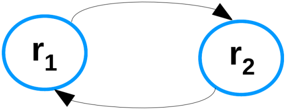


Figure: Firing Graph

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

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

Dealing with non-increasing cycles

Example

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

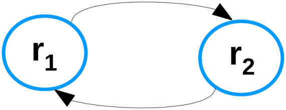


Figure: Firing Graph

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

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

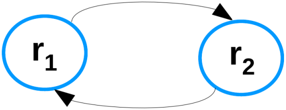


Figure: Firing Graph

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

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

Dealing with non-increasing cycles

Example

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

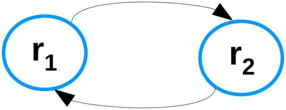


Figure: Firing Graph

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

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

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

Dealing with non-increasing cycles

Example

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

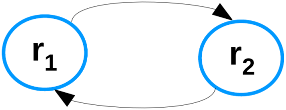


Figure: Firing Graph

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

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

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

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

Dealing with non-increasing cycles

Example

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

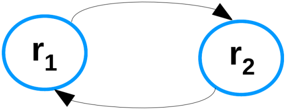


Figure: Firing Graph

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

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

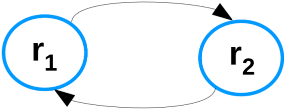


Figure: Firing Graph

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

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

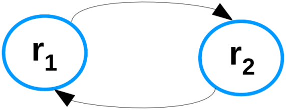


Figure: Firing Graph

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

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

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

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

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

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

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 a good number of practical logic programs;

Future work:

- Combine rule-bounded programs with other techniques in the literature;
- Deep 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 reference

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!