Termination Proving for Rewrite Systems and Programs

Exercises

Carsten Fuhs

Birkbeck, University of London

15^{th} International School on Rewriting 2026

Advanced Track

Nijmegen, The Netherlands

15 & 16 July 2026

https://www.dcs.bbk.ac.uk/~carsten/isr2026/

Exercise I.1

Example (reverse)

$$\begin{array}{l|l} \mathsf{app}(\mathsf{nil}, y) \to y & \mathsf{app}(\mathsf{add}(n, x), y) \to \mathsf{add}(n, \mathsf{app}(x, y)) \\ \mathsf{reverse}(\mathsf{nil}) \to \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) \to \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \end{array}$$

Use the following strategy to prove termination of this TRS:

(1) Compute the initial set of DP problems $D = \{(DP(\mathcal{R}), \mathcal{R})\}$ where $DP(\mathcal{R})$ are the Dependency Pairs of $\mathcal{R}$.

(2) As long as there is some $(\mathcal{P}, \mathcal{R}) \in D$:

1. Decompose the dependency graph of $(\mathcal{P}, \mathcal{R})$ into its non-trivial Strongly Connected Components, get $(\mathcal{P}_1, \mathcal{R}), \ldots, (\mathcal{P}_n, \mathcal{R})$
2. If $\{(\mathcal{P}_1, \mathcal{R}), \ldots, (\mathcal{P}_n, \mathcal{R})\} \neq \{(\mathcal{P}, \mathcal{R})\}$, $D := D \setminus \{(\mathcal{P}, \mathcal{R})\} \cup \{(\mathcal{P}_1, \mathcal{R}), \ldots, (\mathcal{P}_n, \mathcal{R})\}$.
3. Otherwise apply the reduction pair processor with a suitable polynomial interpretation $[\cdot]$ and usable rules, get $(\mathcal{P}', \mathcal{R})$, set $D := D \setminus \{(\mathcal{P}, \mathcal{R})\} \cup \{(\mathcal{P}', \mathcal{R})\}$.

Exercise I.2 (Exercise I.1 continued)

Consider the following TRS:

Example (TRS_Standard/AG01/#3.12, TPDB)

$$
\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}
$$

Use the proof strategy from Exercise I.1 to prove termination of this TRS.

Hint: You have analysed a subset of these rules in Exercise I.1. Can you reuse parts of your analysis for this example?

Exercise I.3

We saw polynomial interpretations to $\mathbb{N}$ with the well-founded order $>$.

An extension of this technique are polynomial interpretations to

$$\mathbb{Q}_{\geq 0} = \{q \mid q \in \mathbb{Q},\, q \geq 0\}.$$

with interpretations like $[f](x_1, x_2) = \frac{1}{2}x_1 + 3x_1x_2 + \frac{1}{8}$.

Can we use the usual order $>$ over $\mathbb{Q}_{\geq 0}$ to prove termination?

Why (not)? If we cannot, what could be an alternative?

Exercise I.4

Example (Imperative Program)

ℓ_0: **while** (x < y)
x = x + 2;
y = y + 1;

Example (Equivalent Translation to an Integer Transition System)

$$\ell_0(x, y) \longrightarrow \ell_0(x + 2, y + 1) \quad [x < y]$$
$$\ell_0(x, y) \longrightarrow \ell_1(x, y) \quad [x \geq y]$$

Prove termination using successive application of **ranking functions**:

As long as there is at least one rule remaining:

Find a ranking function $[\cdot]$ such that

- for all rules $l \longrightarrow r\ [\varphi]$, we have $[l] \geq [r]$;
- for at least one rule $l \longrightarrow r\ [\varphi]$, we have $[l] > [r]$ and $[l] \geq 0$; delete these rules!

Exercise I.5

Example (Imperative Program)

ℓ_0: **while** $(x > 0)$
  $y = x * x;$
ℓ_1: **while** $(y > 0)$
    $y = y - x;$
ℓ_2: $x = x - 1;$

Example (Equivalent Translation to an Integer Transition System)

$$
\begin{array}{rcll}
\ell_0(x) & \longrightarrow & \ell_1(x, x*x) & [x > 0] \\
\ell_1(x,y) & \longrightarrow & \ell_1(x, y-x) & [y > 0] \\
\ell_1(x,y) & \longrightarrow & \ell_2(x,y) & [y \leq 0] \\
\ell_2(x,y) & \longrightarrow & \ell_0(x-1) & \\
\ell_0(x) & \longrightarrow & \ell_3(x) & [x \leq 0]
\end{array}
$$

1. Show that there exists a non-terminating term for the Integer Transition System.
2. Prove termination of the imperative program: show that from ℓ_0, all evaluations terminate.

   Hint: You will first need to add suitable invariants to the rules that hold for all evaluations from ℓ_0. Then proceed as in Exercise I.4.

Exercise II.1

Recall the imperative program fragment on slide 29 (the `while` loop counting down). In the lecture we added the invariant $x \geq 0$ to the constrained rewrite system, but did not prove that it was an invariant.

Construct a symbolic execution graph for the program that also finds this invariant and extract a constrained rewrite system with the invariant from this symbolic execution graph.

The Java version of the method:

```
public static void foo(int x) {
  if (x >= 0) {          // 1
    while (x != 0) {     // 2
      x = x - 1;         // 3
    }
  }                      // 4
}
```

Exercise II.2

```
public class List {
  private List next;

  public static int length(List xs) {
    int res = 0;                          // 1
    while (xs != null) {                  // 2
      xs = xs.next;                       // 3
      res++;                              // 4
    }
    return res;                           // 5
  }
}
```

Construct a symbolic execution graph for `length`, then extract the corresponding constrained rewrite system for the SCCs in the graph.

Can you prove termination of the resulting constrained rewrite system?

Exercise II.3

Our translation from Java to constrained rewriting pretends that `int` corresponds to $\mathbb{Z}$. But Java's `int` $= \{-2^{31}, \ldots, 2^{31} - 1\}$ values are **32-bit machine integers** with "wrap-around" semantics for built-in arithmetic operations (e.g., `maxint + 1 = minint`).

1. Can you think of an imperative program that terminates on Java's 32-bit machine integers, but not on $\mathbb{Z}$?
2. Can you think of an imperative program that terminates on $\mathbb{Z}$, but not on Java's 32-bit machine integers?
3. How could we repair the translation so that it reflects Java's actual `int` semantics?

Exercise II.4

Example

$$\mathsf{a}(\mathsf{a}(x)) \longrightarrow \mathsf{a}(\mathsf{b}(\mathsf{a}(x)))$$

Find an upper bound for the innermost runtime complexity of this TRS.

Exercise II.4

Example

$$\mathsf{a}(\mathsf{a}(x)) \longrightarrow \mathsf{a}(\mathsf{b}(\mathsf{a}(x)))$$

Find an upper bound for the innermost runtime complexity of this TRS.

Hint: The techniques that we have seen for the DT framework will *not* help here. What terms must we consider for this example?

Exercise II.5

Example (Runtime_Complexity_Innermost_Rewriting/AG01/#3.7, TPDB)

$$\begin{array}{rcl|rcl} \mathsf{half}(0) & \to & 0 & \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{half}(x)) \\ \mathsf{log}(\mathsf{s}(0)) & \to & 0 & \mathsf{log}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{log}(\mathsf{s}(\mathsf{half}(x)))) \end{array}$$

Apply the (simplified) DT framework to find an upper bound for innermost runtime complexity of this TRS:

$\mathcal{S} := DT(\mathcal{R})$ // must still be counted
while $\mathcal{S} \neq \emptyset$:

- Find (at most) restricted interpretation $[\cdot]$ with
  - $[\mathsf{Com}_k](x_1, \ldots, x_k) = x_1 + \cdots + x_k$
  - for all $s \to t \in DT(\mathcal{R}) \cup UR(DT(\mathcal{R}), \mathcal{R})$: $[s] \geq [t]$
- Delete DTs $s \to t$ with $[s] \geq [t] + 1$ from $\mathcal{S}$

If k is the maximum degree of any $[f^\sharp]$, $\mathrm{irc}_\mathcal{R}(n) \in \mathcal{O}(n^k)$!