Beyond Dependency Graphs[*]

Martin Korp and Aart Middeldorp
Institute of Computer Science
University of Innsbruck, Austria

Abstract

The dependency pair framework is a powerful technique for proving termination of rewrite systems. One of the most frequently used methods within the dependency pair framework is the dependency graph processor. In this note we improve this processor by incorporating right-hand sides of forward closures. In combination with tree automata completion we obtain an efficient processor which can be used instead of the dependency graph approximations that are in common use in termination provers.

1 Introduction

Proving termination of term rewrite systems is a very active research area. Several tools exist that perform this task automatically. The most powerful ones are based on the dependency pair framework. This framework combines a great variety of termination techniques in a modular way by means of dependency pair processors. In this note we are concerned with the dependency graph processor. It is one of the most important processors as it enables the decomposition of termination problems into smaller subproblems. The processor requires the computation of an over-approximation of the dependency graph. In the literature several such approximations are proposed [1, 8, 12, 13]. In this note we return to tree automata techniques. We show that tree automata *completion* is much more effective for approximating dependency graphs than the method based on approximating the underlying rewrite system to ensure regularity preservation proposed in [12]. We further show that by incorporating *right-hand sides of forward closures* [4], a technique that recently became popular in connection with the match-bound technique [6, 11], we can eliminate arcs from the (real) dependency graph.

The remainder of the note is organized as follows. In Section 2 we recall some basic facts about dependency graphs and processors. In Section 3 we employ tree automata completion to approximate dependency graphs and in Section 4 we incorporate right-hand sides of forward closures. Experimental data is presented in Section 5.

2 Preliminaries

Familiarity with term rewriting [2] and tree automata [3] is assumed. Knowledge of the dependency pair framework [7, 14] and the match-bound technique [6, 11] will be helpful. Below we recall important definitions concerning the former needed in the remainder of the note. Throughout this note we assume that TRSs are finite.

Let $\mathcal{R}$ be a term rewrite system (TRS for short). The set of *dependency pairs* of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$. A *DP problem* is a triple $(\mathcal{P},\mathcal{R},\mathcal{G})$ where $\mathcal{P}$ and $\mathcal{R}$ are two TRSs and $\mathcal{G} \subseteq \mathcal{P} \times \mathcal{P}$ is a directed graph. A DP problem $(\mathcal{P},\mathcal{R},\mathcal{G})$ is called *finite* if there are no infinite rewrite sequences of the form $s_1 \xrightarrow{\epsilon}_{\alpha_1} t_1 \to_{\mathcal{R}}^* s_2 \xrightarrow{\epsilon}_{\alpha_2} t_2 \to_{\mathcal{R}}^* \cdots$ such that all terms t_1, t_2, $\ldots$ are terminating with respect to $\mathcal{R}$ and $(\alpha_i, \alpha_{i+1}) \in \mathcal{G}$ for all $i \geqslant 1$. Such an infinite sequence is said to be *minimal*. The main result underlying the dependency pair approach states that a TRS $\mathcal{R}$ is terminating if and only if the DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R}, \mathsf{DP}(\mathcal{R}) \times \mathsf{DP}(\mathcal{R}))$ is finite. The latter is shown by applying functions that take a DP problem as input and return a set of DP problems as output, the so-called *DP processors*. These processors must have the property that a DP problem is finite whenever all DP problems returned by the processor are

[*]A full version of this note will appear in the Proc. of the 22nd International Conference on Automated Deduction, 2009.

finite, which is known as *soundness*. To use DP processors for establishing non-termination, they must additionally be *complete* which means that if one of the DP problems returned by the processor is not finite then the original DP problem is not finite.

Numerous DP processors have been developed. In this note we are concerned with the dependency graph processor. It determines which dependency pairs can follow each other in infinite rewrite sequences.

Definition 1. The *dependency graph processor* maps a DP problem $(\mathcal{P},\mathcal{R},\mathcal{G})$ to the set $\{(\mathcal{P},\mathcal{R},\mathcal{G} \cap \mathsf{DG}(\mathcal{P},\mathcal{R}))\}$. Here $\mathsf{DG}(\mathcal{P},\mathcal{R})$ is the *dependency graph* of $\mathcal{P}$ and $\mathcal{R}$, which has the rules in $\mathcal{P}$ as nodes and there is an arc from $s \to t$ to $u \to v$ if and only if there exist substitutions σ and τ such that $t\sigma \to_{\mathcal{R}}^{*} u\tau$.

It is well-known [1, 9, 14] that the dependency graph processor is sound and complete.

Example 2. Consider the DP problem $(\mathcal{P},\mathcal{R},\mathcal{G})$ with $\mathcal{R}$ consisting of the rewrite rules $\mathsf{f}(\mathsf{g}(x),y) \to \mathsf{g}(\mathsf{h}(x,y))$ and $\mathsf{h}(\mathsf{g}(x),y) \to \mathsf{f}(\mathsf{g}(\mathsf{a}),\mathsf{h}(x,y))$, $\mathcal{P} = \mathsf{DP}(\mathcal{R})$ consisting of

$$1\colon \mathsf{F}(\mathsf{g}(x),y) \to \mathsf{H}(x,y) \qquad 2\colon \mathsf{H}(\mathsf{g}(x),y) \to \mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{h}(x,y)) \qquad 3\colon \mathsf{H}(\mathsf{g}(x),y) \to \mathsf{H}(x,y)$$

and $\mathcal{G} = \mathcal{P} \times \mathcal{P}$. Because $\mathsf{H}(\mathsf{g}(x),y)$ is an instance of $\mathsf{H}(x,y)$ and $\mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{h}(x,y))$ is an instance of $\mathsf{F}(\mathsf{g}(x),y)$, $\mathsf{DG}(\mathcal{P},\mathcal{R})$ has five arcs:

$$\begin{tikzcd} 1 \arrow[r, bend left] \arrow[rr, bend left=40] & 2 \arrow[l, bend left] & 3 \arrow[l] \arrow[loop above] \end{tikzcd}$$

The dependency graph processor returns the new DP problem $(\mathcal{P},\mathcal{R},\mathsf{DG}(\mathcal{P},\mathcal{R}))$.

3 Tree Automata Completion

We start by recalling some basic facts and notions. Let $\mathcal{R}$ be a TRS over $\mathcal{F}$. The set $\{t \in \mathcal{T}(\mathcal{F}) \mid s \to_{\mathcal{R}}^{*} t \text{ for some } s \in L\}$ of descendants of a set $L \subseteq \mathcal{T}(\mathcal{F})$ of ground terms is denoted by $\to_{\mathcal{R}}^{*}(L)$. We say that a tree automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ is *compatible* with $\mathcal{R}$ and L if $L \subseteq \mathcal{L}(\mathcal{A})$ and for each rewrite rule $l \to r \in \mathcal{R}$ and state substitution $\sigma\colon \mathcal{V}\mathsf{ar}(l) \to Q$ such that $l\sigma \to_{\Delta}^{*} q$ it holds that $r\sigma \to_{\Delta}^{*} q$. For left-linear $\mathcal{R}$ it is known that $\to_{\mathcal{R}}^{*}(L) \subseteq \mathcal{L}(\mathcal{A})$ whenever $\mathcal{A}$ is compatible with $\mathcal{R}$ and L [5].

For two TRSs $\mathcal{P}$ and $\mathcal{R}$ the dependency graph $\mathsf{DG}(\mathcal{P},\mathcal{R})$ contains an arc from a dependency pair α to a dependency pair β if and only if there exist substitutions σ and τ such that $\mathsf{rhs}(\alpha)\sigma \to_{\mathcal{R}}^{*} \mathsf{lhs}(\beta)\tau$. Without loss of generality we may assume that $\mathsf{rhs}(\alpha)\sigma$ and $\mathsf{lhs}(\beta)\tau$ are ground terms. Hence there is no arc from α to β if and only if $\Sigma(\mathsf{lhs}(\beta)) \cap \to_{\mathcal{R}}^{*}(\Sigma(\mathsf{rhs}(\alpha))) = \varnothing$. Here $\Sigma(t)$ denotes the set of ground instances of t with respect to the signature consisting of a fresh constant # together with all function symbols that appear in $\mathcal{P} \cup \mathcal{R}$ minus the root symbols of the left- and right-hand sides of $\mathcal{P}$ that do neither occur on positions below the root in $\mathcal{P}$ nor in $\mathcal{R}$. Since $\to_{\mathcal{R}}^{*}(\Sigma(\mathsf{rhs}(\alpha)))$ is in general not regular, we compute an over-approximation with the help of tree automata completion [5, 10] starting from an automaton that accepts $\Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$. Here ren is the function that linearizes its argument by replacing all occurrences of variables with fresh variables, which is needed to ensure the regularity of $\Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$.

Definition 3. Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs, L a language, and $\alpha, \beta \in \mathcal{P}$. We say that β is *unreachable* from α with respect to L if there is a tree automaton $\mathcal{A}$ compatible with $\mathcal{R}$ and $L \cap \Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$ such that $\Sigma(\mathsf{lhs}(\beta)) \cap \mathcal{L}(\mathcal{A}) = \varnothing$. The nodes of the c-*dependency graph* $\mathsf{DG_c}(\mathcal{P},\mathcal{R})$ are the rewrite rules of $\mathcal{P}$ and there is no arc from α to β if and only if β is unreachable from α with respect to $\Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$.

Lemma 4. *For two left-linear TRSs $\mathcal{P}$ and $\mathcal{R}$, $\mathsf{DG_c}(\mathcal{P},\mathcal{R}) \supseteq \mathsf{DG}(\mathcal{P},\mathcal{R})$.* □

One can extend the above lemma to *arbitrary* TRSs by following the approach in [10], as described in the full version of this note.

4 Incorporating Forward Closures

When proving the termination of a right-linear TRS $\mathcal{R}$ it is sufficient to restrict attention to the set of right-hand sides of forward closures [4]. This set is defined as the closure of the right-hand sides of the rules in $\mathcal{R}$ under narrowing. Formally, given a set L of terms, $\mathsf{RFC}(L,\mathcal{R})$ is the least extension of L such that $t[r]_p\sigma \in \mathsf{RFC}(L,\mathcal{R})$ whenever $t \in \mathsf{RFC}(L,\mathcal{R})$ and there exist a non-variable position p and a fresh variant $l \to r$ of a rewrite rule in $\mathcal{R}$ with σ a most general unifier of $t|_p$ and l. In the sequel we write $\mathsf{RFC}(t,\mathcal{R})$ for $\mathsf{RFC}(\{t\},\mathcal{R})$.

Definition 5. Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The *improved dependency graph* of $\mathcal{P}$ and $\mathcal{R}$, denoted by $\mathsf{IDG}(\mathcal{P},\mathcal{R})$, has the rules in $\mathcal{P}$ as nodes and there is an arc from $s \to t$ to $u \to v$ if and only if there exist substitutions σ and τ such that $t\sigma \to_{\mathcal{R}}^* u\tau$ and $t\sigma \in \Sigma_\#(\mathsf{RFC}(t,\mathcal{P}\cup\mathcal{R}))$. Here $\Sigma_\#$ is the operation that replaces all variables by the fresh constant #.

Theorem 6. *The* improved dependency graph processor *which maps a DP problem* $(\mathcal{P},\mathcal{R},\mathcal{G})$ *to* $\{(\mathcal{P},\mathcal{R},\mathcal{G}\cap\mathsf{IDG}(\mathcal{P},\mathcal{R}))\}$ *if* $\mathcal{P}\cup\mathcal{R}$ *is right-linear and* $\{(\mathcal{P},\mathcal{R},\mathcal{G}\cap\mathsf{DG}(\mathcal{P},\mathcal{R}))\}$ *otherwise is sound and complete.* $\square$

Example 7. We consider again the DP problem $(\mathcal{P},\mathcal{R},\mathcal{G})$ of Example 2. Let $s = \mathsf{H}(x,y)$ and $t = \mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{h}(x,y))$. Because each term in $\Sigma_\#(\mathsf{RFC}(s,\mathcal{P}\cup\mathcal{R}))$ is a ground instance of $\mathsf{F}(\mathsf{g}(\mathsf{a}),x)$ or $\mathsf{H}(\mathsf{a},x)$, or equal to $\mathsf{H}(\#,\#)$ and each term in $\Sigma_\#(\mathsf{RFC}(t,\mathcal{P}\cup\mathcal{R}))$ is a ground instance of $\mathsf{F}(\mathsf{g}(\mathsf{a}),x)$ or $\mathsf{H}(\mathsf{a},x)$, $\mathsf{IDG}(\mathcal{P},\mathcal{R})$ contains an arc from 2 to 1. Further arcs do not exist. So $\mathsf{IDG}(\mathcal{P},\mathcal{R})$ looks as follows:

$$\begin{tikzcd} 1 & 2 \arrow[l] & 3 \end{tikzcd}$$

The resulting DP problem $(\mathcal{P},\mathcal{R},\mathsf{IDG}(\mathcal{P},\mathcal{R}))$ can be easily show to be finite.

Similar to $\mathsf{DG}(\mathcal{P},\mathcal{R})$, $\mathsf{IDG}(\mathcal{P},\mathcal{R})$ is not computable in general. We can however over-approximate $\mathsf{IDG}(\mathcal{P},\mathcal{R})$ by using tree automata completion as described in Section 3.

Definition 8. Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The nodes of the c-*improved dependency graph* $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$ are the rewrite rules of $\mathcal{P}$ and there is no arc from α to β if and only if β is unreachable from α with respect to $\Sigma_\#(\mathsf{RFC}(\mathsf{rhs}(\alpha),\mathcal{P}\cup\mathcal{R}))$.

Lemma 9. *Let* $\mathcal{P}$ *and* $\mathcal{R}$ *be two linear TRSs. Then* $\mathsf{IDG_c}(\mathcal{P},\mathcal{R}) \supseteq \mathsf{IDG}(\mathcal{P},\mathcal{R})$.

To compute $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$ we have to construct a tree automaton that accepts $\mathsf{RFC}(\mathsf{rhs}(\alpha),\mathcal{P}\cup\mathcal{R})$. This can be done by using tree automata completion as described in in [6, 10]. We remark that the above lemma holds also for *right-linear* TRSs (see the description of the full version of this note).

5 Experimental Results

The techniques described in the preceding sections, extended to non-left-linear TRSs as described in the full version of this note, are integrated in the termination prover TT₂. Below we report on the experiments we performed with $\mathsf{T_TT_2}$ on the 1331 TRSs in version 5.0 of the Termination Problem Data Base[1] that satisfy the variable condition, i.e., $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$ for each rewrite rule $l \to r \in \mathcal{R}$. We used a workstation equipped with an Intel® Pentium™ M processor running at a CPU rate of 2 GHz and 1 GB of system memory. For all experiments we used a 60 seconds time limit. Our results are summarized in Table 1. We list the number of successful termination attempts, the average wall-clock

[1] `http://www.termination-portal.org`

Table 1: Dependency graph approximations

	e	c	r	$*$
# successes	60	67	176	183
average time	190	390	340	279
# timeouts	2	60	78	2

time needed to compute the graphs (measured in milliseconds), and the number of timeouts. Besides the SCC processor [9, 14] we used the dependency graph processor of Definition 1 and the improved dependency graph processor of Theorem 6 with $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$ ($\mathsf{DG_c}(\mathcal{P},\mathcal{R})$) for (non-)right-linear $\mathcal{P}\cup\mathcal{R}$ (r for short). To approximate dependency graphs we used the estimation described in [8] (abbreviated by e) and $\mathsf{DG_c}(\mathcal{P},\mathcal{R})$ of Definition 3 (indicated by c).

The power of the new processors is apparent, although the difference with e decreases when other DP processors are in place. An obvious disadvantage of the new processors is the large number of timeouts. This is mostly due to the unbounded number of new states to resolve compatibility violations during tree automata completion. Since the processors c and r seem to be quite fast when they terminate, an obvious idea to avoid timeouts is to equip each computation of a compatible tree automaton with a small time limit. This is shown in the column labeled $*$, which denotes the composition of e, c and r with a time limit of 500 milliseconds each for the latter two.

References

[1] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1-2):133–178, 2000.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available from `www.grappa.univ-lille3.fr/tata`, 2002.

[4] N. Dershowitz. Termination of linear rewriting systems (preliminary version). In *Proc. 8th ICALP*, volume 115 of *LNCS*, pages 448–458, 1981.

[5] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA*, volume 1379 of *LNCS*, pages 151–165, 1998.

[6] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *I&C*, 205(4):512–534, 2007.

[7] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 11th LPAR*, volume 3425 of *LNAI*, pages 301–331, 2004.

[8] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. 5th FroCoS*, volume 3717 of *LNAI*, pages 216–231, 2005.

[9] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *I&C*, 199(1-2):172–199, 2005.

[10] M. Korp and A. Middeldorp. Proving termination of rewrite systems using bounds. In *Proc. 18th RTA*, volume 4533 of *LNCS*, pages 273–287, 2007.

[11] M. Korp and A. Middeldorp. Match-bounds revisited. *I&C*, 2009. To appear.

[12] A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proc. 1st IJCAR*, volume 2083 of *LNAI*, pages 593–610, 2001.

[13] A. Middeldorp. Approximations for strategies and termination. In *Proc. 2nd WRS*, volume 70 of *ENTCS*, pages 1–20, 2002.

[14] R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting*. PhD thesis, RWTH Aachen, 2007. Available as technical report AIB-2007-17.