Beyond Dependency Graphs

Martin Korp and Aart Middeldorp

Institute of Computer Science
University of Innsbruck
Austria
{martin.korp,aart.middeldorp}@uibk.ac.at

Abstract. The dependency pair framework is a powerful technique for proving termination of rewrite systems. One of the most frequently used methods within this framework is the dependency graph processor. In this paper 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 paper 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. Arts and Giesl [1] gave an effective algorithm based on abstraction and unification. Kusakari and Toyama [20, 21] employed Huet and Lévy's notion of ω-reduction to approximate dependency graphs for AC-termination. Middeldorp [22] advocated the use of tree automata techniques and in [23] improved the approximation of [1] by taking symmetry into account. Giesl, Thiemann, and Schneider-Kamp [12] tightly coupled abstraction and unification, resulting in an improvement of [1] which is especially suited for applicative systems.

In this paper 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 [22]. The dependency graph determines whether dependency pairs can follow each other. It does not determine whether dependency pairs follow each other *infinitely often*. We further show that by incorporating *right-hand sides of forward closures* [4, 9], a technique that recently became popular in connection with the match-bound technique [8, 18], we can

eliminate arcs from the (real) dependency graph. Experimental results confirm the competitiveness of the resulting improved dependency graph processor.

The remainder of the paper 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. Incorporating right-hand sides of forward closures is the topic of Section 4. In Section 5 we compare our approach with existing approximations of dependency graphs. Dependency graphs for innermost termination are briefly addressed in Section 6. In Section 7 we report on the extensive experiments that we conducted.

2 Preliminaries

Familiarity with term rewriting [2] and tree automata [3] is assumed. Knowledge of the dependency pair framework [11, 25] and the match-bound technique [8, 18] will be helpful. Below we recall important definitions concerning the former needed in the remainder of the paper. Throughout this paper we assume that term rewrite systems are finite.

Let $\mathcal{R}$ be a term rewrite system (TRS for short) over a signature $\mathcal{F}$. If $l \to r \in \mathcal{R}$ and t is a subterm of r with a defined root symbol that is not a proper subterm of l then $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. Here $l^\sharp$ and $t^\sharp$ denote the terms that are obtained by marking the root symbols of l and t. In examples we use capital letters to represent marked function symbols. 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 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 paper we are concerned with the dependency graph processor. This is one of the most important processors as it enables to decompose a DP problem into smaller subproblems. The dependency graph processor determines which dependency pairs can follow each other in infinite rewrite sequences. Following [25], we find it convenient to split the processor into one which computes the graph and one which computes the strongly connected components (SCCs). This separates the part that needs to be approximated from the computable part and is important to properly describe

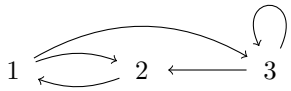
the experiments in Section 7 where we combine several graph approximations before computing SCCs.

Definition 1. *The* dependency graph processor *maps a DP problem* $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ *to* $\{(\mathcal{P}, \mathcal{R}, \mathcal{G} \cap \mathsf{DG}(\mathcal{P}, \mathcal{R}))\}$. *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* $t\sigma \to_{\mathcal{R}}^* u\tau$.

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

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

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:



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

Definition 3. *The* SCC processor *transforms a DP problem* $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ *into* $\{(\mathcal{P}_1, \mathcal{R}, \mathcal{G}_1), \ldots, (\mathcal{P}_n, \mathcal{R}, \mathcal{G}_n)\}$. *Here* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *are the strongly connected components of* $\mathcal{G}$ *and* $\mathcal{G}_i = \mathcal{G} \cap (\mathcal{P}_i \times \mathcal{P}_i)$ *for every* $1 \leqslant i \leqslant n$.

The following result is well-known [1, 11, 14, 25].

Theorem 4. *The dependency graph and SCC processors are sound and complete.* □

We continue the previous example.

Example 5. The SCC processor does not make progress on the DP problem $(\mathcal{P}, \mathcal{R}, \mathsf{DG}(\mathcal{P}, \mathcal{R}))$ since the three nodes form a single SCC in the graph.

3 Tree Automata Completion

We start by recalling some basic facts and notation. A *tree automaton* $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ consists of a signature $\mathcal{F}$, a finite set of states Q, a set of final states $Q_f \subseteq Q$, and a set of transitions Δ of the form $f(q_1, \ldots, q_n) \to q$ with f an n-ary function symbol in $\mathcal{F}$ and $q, q_1, \ldots, q_n \in Q$. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of ground terms $t \in \mathcal{T}(\mathcal{F})$ such that $t \to_\Delta^* q$ for some $q \in Q_f$. 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 $\mathcal{A}$ 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 [6]. To obtain a similar result for non-left-linear TRSs, in [16] quasi-deterministic automata were introduced. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton. We say that a state p *subsumes* a state q if p is final when q is final and for all transitions $f(u_1, \ldots, q, \ldots, u_n) \to u \in \Delta$, the transition $f(u_1, \ldots, p, \ldots, u_n) \to u$ belongs to Δ. For a left-hand side $l \in \mathsf{lhs}(\Delta)$ of a transition, the set $\{q \mid l \to q \in \Delta\}$ of possible right-hand sides is denoted by $Q(l)$. The automaton $\mathcal{A}$ is said to be *quasi-deterministic* if for every $l \in \mathsf{lhs}(\Delta)$ there exists a state $p \in Q(l)$ which subsumes every other state in $Q(l)$. In general, $Q(l)$ may contain more than one state that satisfies the above property. In the following we assume that there is a unique designated state in $Q(l)$, which we denote by p_l. The set of all designated states is denoted by Q_d and the restriction of Δ to transition rules $l \to q$ that satisfy $q = p_l$ is denoted by Δ_d. In [16] we showed that the tree automaton induced by Δ_d is deterministic and accepts the same language as $\mathcal{A}$. For non-left-linear TRSs $\mathcal{R}$ we modify the above definition of compatibility by demanding that the tree automaton $\mathcal{A}$ is quasi-deterministic and for each rewrite rule $l \to r \in \mathcal{R}$ and state substitution $\sigma\colon \mathcal{V}\mathsf{ar}(l) \to Q_d$ with $l\sigma \to_{\Delta_d}^* q$ it holds that $r\sigma \to_{\Delta}^* q$.

Theorem 6 ([6, 16]). *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ a tree automaton, and L a set of ground terms. If $\mathcal{A}$ is compatible with $\mathcal{R}$ and L then $\to_{\mathcal{R}}^*(L) \subseteq \mathcal{L}(\mathcal{A})$.* □

For two TRSs $\mathcal{P}$ and $\mathcal{R}$ the dependency graph $\mathsf{DG}(\mathcal{P}, \mathcal{R})$ contains an arc from dependency pair α to 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 $\mathcal{R}$.[1] For an arbitrary term t and regular language L it is decidable whether $\Sigma(t) \cap L = \varnothing$—a result of Tison (see [22])—and hence we can check the above condition by constructing a tree automaton that accepts $\to_{\mathcal{R}}^*(\Sigma(\mathsf{rhs}(\alpha)))$. Since this set is in general not regular, we compute an over-approximation with the help of tree automata completion 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 7. *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 language L in the above definition allows us to refine the set of starting terms $\Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$ which are considered in the computation of an arc from α

[1] The fresh constant $\#$ is added to the signature to ensure that $\Sigma(t)$ cannot be empty.

to β. In Section 4 we make use of L to remove arcs from the (real) dependency graph. In the remainder of this section we always have $L = \Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$.

Definition 8. *The nodes of the* c-dependency graph $\mathsf{DG}_\mathsf{c}(\mathcal{P},\mathcal{R})$ *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* $\Sigma(\mathsf{ren}(\mathsf{rhs}(\alpha)))$.

The c in the above definition refers to the fact that a compatible tree automaton is constructed by tree automata completion.

Lemma 9. *Let* $\mathcal{P}$ *and* $\mathcal{R}$ *be two TRSs. Then* $\mathsf{DG}_\mathsf{c}(\mathcal{P},\mathcal{R}) \supseteq \mathsf{DG}(\mathcal{P},\mathcal{R})$.

Proof. Easy consequence of Theorem 6. □

The general idea of tree automata completion [6, 7, 16] is to look for violations of the compatibility requirement: $l\sigma \to_\Delta^* q$ ($l\sigma \to_{\Delta_d}^* q$) but not $r\sigma \to_\Delta^* q$ for some rewrite rule $l \to r \in \mathcal{R}$, state substitution $\sigma\colon \mathcal{V}\mathsf{ar}(l) \to Q$ ($\sigma\colon \mathcal{V}\mathsf{ar}(l) \to Q_d$), and state q. This triggers the addition of new states and transitions to the current automaton to ensure $r\sigma \to_\Delta^* q$. There are several ways to do this, ranging from establishing a completely new path $r\sigma \to_\Delta^* q$ to adding as few as possible new transitions by reusing transitions from the current automaton. After $r\sigma \to_\Delta^* q$ has been established, we look for further compatibility violations. This process is repeated until a compatible tree automaton is obtained, which may never happen if new states are kept being added.

Example 10. We continue with our example. A tree automaton $\mathcal{A}$, with final state 2, that accepts $\Sigma(\mathsf{H}(x,y))$ is easily constructed:

$$\mathsf{a} \to 1 \qquad \mathsf{f}(1,1) \to 1 \qquad \mathsf{g}(1) \to 1 \qquad \mathsf{h}(1,1) \to 1 \qquad \mathsf{H}(1,1) \to 2$$

Because $\to_\mathcal{R}^*(\Sigma(\mathsf{H}(x,y))) = \Sigma(\mathsf{H}(x,y))$, the automaton is already compatible with $\mathcal{R}$ and $\Sigma(\mathsf{H}(x,y))$, so completion is trivial here. As $\mathsf{H}(\mathsf{g}(\mathsf{a}),\mathsf{a})$ is accepted by $\mathcal{A}$, $\mathsf{DG}_\mathsf{c}(\mathcal{P},\mathcal{R})$ contains arcs from 1 and 3 to 2 and 3. Similarly, we can construct a tree automaton $\mathcal{B}$ with final state 2 that is compatible with $\mathcal{R}$ and $\Sigma(\mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{h}(x,y)))$:

$$\begin{array}{lllll} \mathsf{a} \to 1 & \mathsf{f}(1,1) \to 1 & \mathsf{F}(3,4) \to 2 & \mathsf{g}(1) \to 4 & \mathsf{h}(1,1) \to 1 \\ \mathsf{a} \to 2 & \mathsf{f}(1,1) \to 4 & \mathsf{g}(1) \to 1 & \mathsf{g}(2) \to 3 & \mathsf{h}(1,1) \to 4 \end{array}$$

Because $\mathsf{F}(\mathsf{g}(\mathsf{a}),\mathsf{h}(\mathsf{a},\mathsf{a}))$ is accepted by $\mathcal{B}$, we obtain an arc from 2 to 1. Further arcs do not exist.

It can be argued that the use of tree automata techniques for the DP problem of Example 2 is a waste of resources because the dependency graph can also be computed by just taking the root symbols of the dependency pairs into consideration. However, in the next section we show that this radically changes when taking right-hand sides of forward closures into account.

4 Incorporating Forward Closures

Given a DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ and $\alpha, \beta \in \mathcal{P}$, an arc from α to β in the dependency graph $\mathsf{DG}(\mathcal{P}, \mathcal{R})$ is an indication that β may follow α in an *infinite* sequence in $\mathcal{P} \cup \mathcal{R}$ but not a sufficient condition because if the problem is finite there are no infinite sequences whatsoever. What we would like to determine is whether β can follow α infinitely many times in a minimal sequence. With the existing approximations of the dependency graph, only local connections can be tested. In this section we show that by using right-hand sides of forward closures, we can sometimes delete arcs from the dependency graph which cannot occur infinitely many times in minimal sequences.

When proving the termination of a TRS $\mathcal{R}$ that is non-overlapping or right-linear it is sufficient to restrict attention to the set of right-hand sides of forward closures [4, 9]. 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 $\mathsf{RFC}(\{t\}, \mathcal{R})$. Furthermore, we restrict our attention to right-linear TRSs because we cannot cope with non-right-linear TRSs during the construction of the set of right-hand sides of forward closures [8].

Dershowitz [4] showed that a right-linear TRS $\mathcal{R}$ is terminating if and only if $\mathcal{R}$ is terminating on $\mathsf{RFC}(\mathsf{rhs}(\mathcal{R}), \mathcal{R})$. In [17, 18] we showed how to extend this result to the dependency pairs setting.

Lemma 11. *Let $\mathcal{P}$ and $\mathcal{R}$ be two right-linear TRSs and let $\alpha \in \mathcal{P}$. Then $\mathcal{P} \cup \mathcal{R}$ admits a minimal rewrite sequence with infinitely many α steps if and only if it admits such a sequence starting from a term in $\mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})$.* □

A careful inspection of the complicated proof of Lemma 11 given in [18] reveals that the statement can be adapted to our needs. We say that dependency pair β *directly follows* dependency pair α in a minimal sequence $s_1 \xrightarrow{\epsilon}_{\alpha_1} t_1 \to^*_{\mathcal{R}} s_2 \xrightarrow{\epsilon}_{\alpha_2} t_2 \to^*_{\mathcal{R}} \cdots$ if $\alpha_i = \alpha$ and $\alpha_{i+1} = \beta$ for some $i \geqslant 1$.

Lemma 12. *Let $\mathcal{P}$ and $\mathcal{R}$ be two right-linear TRSs and let $\alpha, \beta \in \mathcal{P}$. The TRS $\mathcal{P} \cup \mathcal{R}$ admits a minimal rewrite sequence in which infinitely many β steps directly follow α steps if and only if it admits such a sequence starting from a term in $\mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})$.* □

Definition 13. *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 $\#$.*

Note that the use of $\Sigma_{\#}$ in the above definition is essential. If we would replace $\Sigma_{\#}$ by Σ then $\Sigma(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R})) \supseteq \Sigma(t)$ because $t \in \mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R})$ and hence $\mathsf{IDG}(\mathcal{P}, \mathcal{R}) = \mathsf{DG}(\mathcal{P}, \mathcal{R})$. According to the following lemma the improved dependency graph can be used whenever the participating TRSs are right-linear.

Lemma 14. *Let $\mathcal{P}$ and $\mathcal{R}$ be right-linear TRSs and $\alpha, \beta \in \mathcal{P}$. If there is a minimal sequence in $\mathcal{P} \cup \mathcal{R}$ in which infinitely many β steps directly follow α steps, then $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$ admits an arc from α to β.*

Proof. Assume that there is a minimal rewrite sequence

$$s_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_1 \to^*_{\mathcal{R}} s_2 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to^*_{\mathcal{R}} s_3 \xrightarrow{\epsilon}_{\mathcal{P}} \cdots$$

in which infinitely many β steps directly follow α steps. According to Lemma 12 we may assume without loss of generality that $s_1 \in \mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})$. Let $i \geqslant 1$ such that $s_i \xrightarrow{\epsilon}_{\alpha} t_i \to^*_{\mathcal{R}} s_{i+1} \xrightarrow{\epsilon}_{\beta} t_{i+1}$. Because $\mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})$ is closed under rewriting with respect to $\mathcal{P}$ and $\mathcal{R}$ we know that $t_i \in \mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})$. We have $t_i = \mathsf{rhs}(\alpha)\sigma$ and $s_{i+1} = \mathsf{lhs}(\beta)\tau$ for some substitutions σ and τ. From $t_i \in \mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})$ we infer that $t_i\theta \in \Sigma_{\#}(\mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R}))$ for the substitution θ that replaces every variable by $\#$. Due to the fact that rewriting is closed under substitutions we have $t_i\theta \to^*_{\mathcal{R}} s_{i+1}\theta$. Hence $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$ contains an arc from α to β. □

The following example shows that it is essential to include $\mathcal{P}$ in the construction of the set $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R}))$ in the definition of $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$.

Example 15. Consider the TRS $\mathcal{R}$ consisting of the rewrite rules $\mathsf{f}(x) \to \mathsf{g}(x)$, $\mathsf{g}(\mathsf{a}) \to \mathsf{h}(\mathsf{b})$, and $\mathsf{h}(x) \to \mathsf{f}(\mathsf{a})$ and the TRS $\mathcal{P} = \mathsf{DP}(\mathcal{R})$ consisting of

$$\mathsf{F}(x) \to \mathsf{G}(x) \qquad \mathsf{G}(\mathsf{a}) \to \mathsf{H}(\mathsf{b}) \qquad \mathsf{H}(x) \to \mathsf{F}(\mathsf{a})$$

The DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{P} \times \mathcal{P})$ is not finite because it admits the loop

$$\mathsf{F}(\mathsf{a}) \xrightarrow{\epsilon}_{\mathcal{P}} \mathsf{G}(\mathsf{a}) \xrightarrow{\epsilon}_{\mathcal{P}} \mathsf{H}(\mathsf{b}) \xrightarrow{\epsilon}_{\mathcal{P}} \mathsf{F}(\mathsf{a})$$

Let $t = \mathsf{G}(x)$. We have $\mathsf{RFC}(t, \mathcal{R}) = \{t\}$ and hence $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{R})) = \{\mathsf{G}(\#)\}$. If we now replace $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R}))$ by $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{R}))$ in Definition 13, we would conclude that $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$ does not contain an arc from $\mathsf{F}(x) \to \mathsf{G}(x)$ to $\mathsf{G}(\mathsf{a}) \to \mathsf{H}(\mathsf{b})$ because $\mathsf{G}(\#)$ is a normal form which is different from $\mathsf{G}(\mathsf{a})$. But this makes the resulting DP problem $(\mathcal{P}, \mathcal{R}, \mathsf{IDG}(\mathcal{P}, \mathcal{R}))$ finite.

Theorem 16. *The* improved dependency graph processor

$$(\mathcal{P}, \mathcal{R}, \mathcal{G}) \mapsto \begin{cases} \{(\mathcal{P}, \mathcal{R}, \mathcal{G} \cap \mathsf{IDG}(\mathcal{P}, \mathcal{R}))\} & \text{if } \mathcal{P} \cup \mathcal{R} \text{ is right-linear} \\ \{(\mathcal{P}, \mathcal{R}, \mathcal{G} \cap \mathsf{DG}(\mathcal{P}, \mathcal{R}))\} & \text{otherwise} \end{cases}$$

is sound and complete.

Proof. Soundness is an easy consequence of Lemma 14 and Theorem 4. Completeness follows from the inclusions $\mathcal{G} \cap \mathsf{DG}(\mathcal{P}, \mathcal{R}) \subseteq \mathcal{G} \supseteq \mathcal{G} \cap \mathsf{IDG}(\mathcal{P}, \mathcal{R})$. □

Example 17. We consider again the DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ of Example 2. Let $s = \mathsf{H}(x, y)$ and $t = \mathsf{F}(\mathsf{g}(\mathsf{a}), \mathsf{h}(x, y))$. We first compute $\mathsf{RFC}(s, \mathcal{P} \cup \mathcal{R})$ and $\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R})$. The former set consists of $\mathsf{H}(x, y)$ together will all terms in $\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R})$.

Each term contained in the latter set is an instance of $\mathsf{F}(\mathsf{g}(\mathsf{a}), x)$ or $\mathsf{H}(\mathsf{a}, x)$. It follows that 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 $\mathsf{H}(\#, \#)$. Similarly, 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 $\mathsf{H}(\mathsf{a}, x)$. Hence $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$ contains an arc from 2 to 1. Further arcs do not exist because there are no substitution τ and term $u \in \Sigma_{\#}(\mathsf{RFC}(s, \mathcal{P} \cup \mathcal{R}))$ such that $u \to^*_{\mathcal{R}} \mathsf{H}(\mathsf{g}(x), y)\tau$. So $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$ looks as follows:

$$1 \longleftarrow 2 \qquad 3$$

Therefore the improved dependency graph processor produces the new DP problem $(\mathcal{P}, \mathcal{R}, \mathsf{IDG}(\mathcal{P}, \mathcal{R}))$. Since the above graph does not admit any SCCs, the SCC processor yields the finite DP problem $(\mathcal{P}, \mathcal{R}, \varnothing)$. Consequently, $\mathcal{R}$ is terminating.

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

Definition 18. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The nodes of the* c-improved dependency graph $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ *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 19. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. Then $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R}) \supseteq \mathsf{IDG}(\mathcal{P}, \mathcal{R})$.*

Proof. Assume to the contrary that the claim does not hold. Then there are rules $s \to t$ and $u \to v$ in $\mathcal{P}$ such that there is an arc from $s \to t$ to $u \to v$ in $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$ but not in $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$. By Definition 13 there are substitutions σ and τ such that $t\sigma \in L$ with $L = \Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R}))$ and $t\sigma \to^*_{\mathcal{R}} u\tau$. Since $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ does not admit an arc from $s \to t$ to $u \to v$, there is a tree automaton $\mathcal{A}$ compatible with $\mathcal{R}$ and $L \cap \Sigma(\mathsf{ren}(t))$ such that $\Sigma(u) \cap \mathcal{L}(\mathcal{A}) = \varnothing$. Theorem 6 yields $\to^*_{\mathcal{R}}(L \cap \Sigma(\mathsf{ren}(t))) \subseteq \mathcal{L}(\mathcal{A})$. From $t\sigma \in L \cap \Sigma(\mathsf{ren}(t))$ and $t\sigma \to^*_{\mathcal{R}} u\tau$ we infer that $u\tau \in \mathcal{L}(\mathcal{A})$, contradicting $\Sigma(u) \cap \mathcal{L}(\mathcal{A}) = \varnothing$. □

To compute $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ we have to construct an intermediate 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 [8, 16]. We continue our leading example.

Example 20. We construct $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ for the DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ of Example 17. Let $s = \mathsf{H}(x, y)$. A tree automaton $\mathcal{A}$ that is compatible with $\mathcal{R}$ and $\Sigma_{\#}(\mathsf{RFC}(s, \mathcal{P} \cup \mathcal{R})) \cap \Sigma(s)$ consists of the transitions

$$\begin{array}{lllll} \# \to 1 & \mathsf{g}(3) \to 4 & \mathsf{f}(4,5) \to 5 & \mathsf{h}(1,1) \to 5 & \mathsf{H}(1,1) \to 2 \\ \mathsf{a} \to 3 & \mathsf{g}(6) \to 5 & & \mathsf{h}(3,5) \to 6 & \mathsf{H}(3,5) \to 2 \end{array}$$

with final state 2. Since $\mathcal{A}$ does not accept any ground instance of the term $\mathsf{H}(\mathsf{g}(x), y)$ we conclude that the rules 2 and 3 are unreachable from 1 and 3. It remains to check whether there is any outgoing arc from rule 2. Let $t =$

$\mathsf{F}(\mathsf{g}(\mathsf{a}), \mathsf{h}(x, y))$ be the right-hand side of 2. Similar as before we can construct a tree automaton $\mathcal{B}$ with final state 5 and consisting of the transitions

$$\begin{array}{llll} \# \to 1 & \mathsf{g}(2) \to 3 & \mathsf{f}(3,4) \to 4 & \mathsf{h}(1,1) \to 4 \\ \mathsf{a} \to 2 & \mathsf{g}(6) \to 4 & \mathsf{F}(3,4) \to 5 & \mathsf{h}(2,4) \to 6 \end{array}$$

which is compatible with $\mathcal{R}$ and $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R})) \cap \Sigma(t)$. Since the instance $\mathsf{F}(\mathsf{g}(\mathsf{a}), \mathsf{h}(\#, \#))$ of $\mathsf{F}(\mathsf{g}(x), y)$ is accepted by $\mathcal{B}$, $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ contains an arc from 2 to 1. Further arcs do not exist. Hence $\mathsf{IDG}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ coincides with $\mathsf{IDG}(\mathcal{P}, \mathcal{R})$.

5 Comparison

In the literature several over-approximations of the dependency graph are described [1, 12, 21–23]. In this section we compare the tree automata approach to approximate the processors of Definition 1 and Theorem 16 developed in the preceding sections with the earlier tree automata approach of [22] as well as the approximation used in tools like AProVE [10] and T$_{\mathsf{T}}$T$_2$ [19], which is a combination of ideas of [12] and [23]. We start by formally defining the latter.

Definition 21. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The nodes of the* estimated dependency graph $\mathsf{DG}_{\mathsf{e}}(\mathcal{P}, \mathcal{R})$ *are the rewrite rules of $\mathcal{P}$ and there is an arc from $s \to t$ to $u \to v$ if and only if $\mathsf{tcap}(\mathcal{R}, t)$ and u as well as t and $\mathsf{tcap}(\mathcal{R}^{-1}, u)$ are unifiable. Here $\mathcal{R}^{-1} = \{r \to l \mid l \to r \in \mathcal{R}\}$ and the function $\mathsf{tcap}(\mathcal{R}, t)$ is defined as $f(\mathsf{tcap}(\mathcal{R}, t_1), \ldots, \mathsf{tcap}(\mathcal{R}, t_n))$ if $t = f(t_1, \ldots, t_n)$ and the term $f(\mathsf{tcap}(\mathcal{R}, t_1), \ldots, \mathsf{tcap}(\mathcal{R}, t_n))$ does not unify with any left-hand side of $\mathcal{R}$. Otherwise $\mathsf{tcap}(\mathcal{R}, t)$ is a fresh variable.*

The approach described in [22] to approximate dependency graphs based on tree automata techniques relies on regularity preservation rather than completion. Below we recall the relevant definitions. An *approximation mapping* is a mapping ϕ from TRSs to TRSs such that $\to_{\mathcal{R}} \subseteq \to^*_{\phi(\mathcal{R})}$. We say that ϕ is *regularity preserving* if $\leftarrow^*_{\phi(\mathcal{R})}(L) = \{s \in \mathcal{T}(\mathcal{F}) \mid s \to^*_{\phi(\mathcal{R})} t \text{ for some } t \in L\}$ is regular for all $\mathcal{R}$ and regular L. Here $\mathcal{F}$ is the signature of $\mathcal{R}$.

The three approximation mappings s, nv, g are defined as follows: $\mathsf{s}(\mathcal{R}) = \{\mathsf{ren}(l) \to x \mid l \to r \in \mathcal{R} \text{ and } x \text{ is a fresh variable}\}$, $\mathsf{nv}(\mathcal{R}) = \{\mathsf{ren}(l) \to \mathsf{ren}(r) \mid l \to r \in \mathcal{R}\}$, and $\mathsf{g}(\mathcal{R})$ is defined as any left-linear TRS that is obtained from $\mathcal{R}$ by linearizing the left-hand sides and renaming the variables in the right-hand sides that occur at a depth greater than 1 in the corresponding left-hand sides. These mappings are known to be regularity preserving [5, 24].

Definition 22. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs and let ϕ be an approximation mapping. The nodes of the* ϕ-approximated dependency graph $\mathsf{DG}_{\phi}(\mathcal{P}, \mathcal{R})$ *are the rewrite rules of $\mathcal{P}$ and there is an arc from $s \to t$ to $u \to v$ if and only if both $\Sigma(t) \cap \leftarrow^*_{\phi(\mathcal{R})}(\Sigma(\mathsf{ren}(u))) \neq \varnothing$ and $\Sigma(u) \cap \leftarrow^*_{\phi(\mathcal{R}^{-1})}(\Sigma(\mathsf{ren}(t))) \neq \varnothing$.*

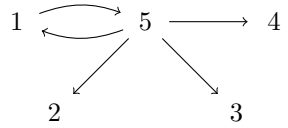
Lemma 23 ([12, 22, 23]). *For TRSs $\mathcal{P}$ and $\mathcal{R}$, $\mathsf{DG}_{\mathsf{e}}(\mathcal{P}, \mathcal{R}) \supseteq \mathsf{DG}(\mathcal{P}, \mathcal{R})$ and $\mathsf{DG}_{\mathsf{s}}(\mathcal{P}, \mathcal{R}) \supseteq \mathsf{DG}_{\mathsf{nv}}(\mathcal{P}, \mathcal{R}) \supseteq \mathsf{DG}_{\mathsf{g}}(\mathcal{P}, \mathcal{R}) \supseteq \mathsf{DG}(\mathcal{P}, \mathcal{R})$.* □

From Examples 2 and 20 it is obvious that neither $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ nor $\mathsf{DG_g}(\mathcal{P},\mathcal{R})$ subsumes $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$. The converse depends very much on the approximation strategy that is used; it can always happen that the completion procedure does not terminate or that the over-approximation is too inexact. Nevertheless, there are problems where $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ and $\mathsf{DG_s}(\mathcal{P},\mathcal{R})$ are properly contained in $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$. An example is provided by the TRS consisting of the rules $\mathsf{f}(x,x) \to \mathsf{f}(\mathsf{a},\mathsf{g}(x,\mathsf{b}))$, $\mathsf{f}(\mathsf{a},\mathsf{g}(x,x)) \to \mathsf{f}(\mathsf{a},\mathsf{a})$, and $\mathsf{g}(\mathsf{a},\mathsf{b}) \to \mathsf{b}$. The following example shows that neither $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ nor $\mathsf{DG_g}(\mathcal{P},\mathcal{R})$ subsumes $\mathsf{DG_c}(\mathcal{P},\mathcal{R})$.

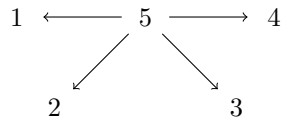
Example 24. Consider the TRSs $\mathcal{R}$ and $\mathcal{P}$ with $\mathcal{R}$ consisting of the rewrite rules $\mathsf{p}(\mathsf{p}(\mathsf{p}(x))) \to \mathsf{p}(\mathsf{p}(x))$, $\mathsf{f}(x) \to \mathsf{g}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a}))))$, and $\mathsf{g}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x)))) \to \mathsf{f}(x)$ and $\mathcal{P} = \mathsf{DP}(\mathcal{R})$ consisting of

$$1\colon \mathsf{F}(x) \to \mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a})))) \qquad 3\colon \mathsf{F}(x) \to \mathsf{P}(\mathsf{p}(\mathsf{a})) \qquad 5\colon \mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x)))) \to \mathsf{F}(x)$$
$$2\colon \mathsf{F}(x) \to \mathsf{P}(\mathsf{p}(\mathsf{p}(\mathsf{a}))) \qquad 4\colon \mathsf{F}(x) \to \mathsf{P}(\mathsf{a})$$

First we compute $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$. It is clear that $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ contains arcs from 5 to 1, 2, 3, and 4. Furthermore, it contains an arc from 1 to 5 because the term $\mathsf{tcap}(\mathcal{R},\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a})))))) = \mathsf{G}(y)$ unifies with $\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x))))$ and $\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a}))))$ unifies with $\mathsf{tcap}(\mathcal{R}^{-1},\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x))))))) = \mathsf{G}(y)$. Further arcs do not exist and hence $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ looks as follows:



Next we compute $\mathsf{DG_g}(\mathcal{P},\mathcal{R})$. Similarly as $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$, $\mathsf{DG_g}(\mathcal{P},\mathcal{R})$ has arcs from 5 to 1, 2, 3, and 4. Furthermore $\mathsf{DG_g}(\mathcal{P},\mathcal{R})$ contains an arc from 1 to 5 because $\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a})))) \to_{\mathsf{g}(\mathcal{R})} \mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(\mathsf{a})))) \in \Sigma(\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x))))))$ by applying the rewrite rule $\mathsf{p}(\mathsf{p}(\mathsf{p}(x))) \to \mathsf{p}(\mathsf{p}(y))$ and $\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x)))) \to_{\mathsf{g}(\mathcal{R}^{-1})} \mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a})))) \in \{\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a})))))\}$ using the rule $\mathsf{p}(\mathsf{p}(x)) \to \mathsf{p}(\mathsf{p}(\mathsf{p}(y)))$. Hence $\mathsf{DG_g}(\mathcal{P},\mathcal{R})$ coincides with $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$. The graph $\mathsf{DG_c}(\mathcal{P},\mathcal{R})$



does not contain an arc from 1 to 5 because 5 is unreachable from 1. This is certified by the following tree automaton $\mathcal{A}$:

$$\mathsf{a} \to 1 \qquad \mathsf{p}(1) \to 2 \qquad \mathsf{p}(2) \to 3 \qquad \mathsf{p}(2) \to 4 \qquad \mathsf{p}(3) \to 4 \qquad \mathsf{G}(4) \to 5$$

with 5 as the only final state. Note that $\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{a})))) \in \mathcal{L}(\mathcal{A})$, $\to_{\mathcal{R}}^{*}(\mathcal{L}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$, and $\Sigma(\mathsf{G}(\mathsf{p}(\mathsf{p}(\mathsf{s}(x))))))) \cap \mathcal{L}(\mathcal{A}) = \varnothing$.

Concerning the converse direction, there are TRSs like $\mathsf{f}(\mathsf{a},\mathsf{b},x) \to \mathsf{f}(x,x,x)$ such that $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ and $\mathsf{DG_{nv}}(\mathcal{P},\mathcal{R})$ are properly contained in $\mathsf{DG_c}(\mathcal{P},\mathcal{R})$. We assume that this also holds for $\mathsf{DG_s}(\mathcal{P},\mathcal{R})$ although we did not succeed in finding an example.

6 Innermost Termination

In this section we sketch how the ideas presented in Sections 3 and 4 can be extended to innermost termination. Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs and $\mathcal{G} \subseteq \mathcal{P} \times \mathcal{P}$ a directed graph. A *minimal innermost* rewrite sequence is an infinite rewrite sequence of the form $s_1 \xrightarrow{\mathsf{i}}_{\mathcal{P}} t_1 \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{*} s_2 \xrightarrow{\mathsf{i}}_{\mathcal{P}} t_2 \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{*} \cdots$ such that $s_i \xrightarrow{\epsilon} t_i$ and $(\alpha_i, \alpha_{i+1}) \in \mathcal{G}$ for all $i \geqslant 1$. Here $\xrightarrow{\mathsf{i}}$ denotes the innermost rewrite relation of $\mathcal{P} \cup \mathcal{R}$. A DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ is called *innermost finite* if there are no minimal innermost rewrite sequences.

Definition 25. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The* innermost dependency graph *of $\mathcal{P}$ and $\mathcal{R}$, denoted by $\mathsf{DG}^{\mathsf{i}}(\mathcal{P}, \mathcal{R})$, 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 $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{*} u\tau$ and $s\sigma$ and $u\tau$ are normal forms with respect to $\mathcal{R}$.*

By incorporating right-hand sides of forward closures, arcs of the innermost dependency graph can sometimes be eliminated. The only complication is that innermost rewriting is not closed under substitutions. To overcome this problem, we add a fresh unary function symbol besides the constant # to the signature and assume that $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R}))$ denotes the set of ground terms that are obtained from terms in $\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R})$ by instantiating the variables by terms built from # and this unary function symbol.

Definition 26. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The* improved innermost dependency graph *of $\mathcal{P}$ and $\mathcal{R}$, denoted by $\mathsf{IDG}^{\mathsf{i}}(\mathcal{P}, \mathcal{R})$, 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 $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{*} u\tau$, $t\sigma \in \Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R}))$, and $s\sigma$ and $u\tau$ are normal forms with respect to $\mathcal{R}$.*

The following result corresponds to Lemma 14.

Lemma 27. *Let $\mathcal{P}$ and $\mathcal{R}$ be right-linear TRSs and $\alpha, \beta \in \mathcal{P}$. If there is a minimal innermost sequence in $\mathcal{P} \cup \mathcal{R}$ in which infinitely many β steps directly follow α steps then $\mathsf{IDG}^{\mathsf{i}}(\mathcal{P}, \mathcal{R})$ admits an arc from α to β.* □

We approximate (improved) innermost dependency graphs as discussed in Section 3. In order to make use of the fact that $s\sigma$ and $u\tau$ are normal forms with respect to $\mathcal{R}$, we restrict $\Sigma(\mathsf{ren}(t))$ and $\Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P} \cup \mathcal{R}))$ to the normalized instances (i.e., ground instances that are obtained by substituting normal forms for the variables) of $\mathsf{ren}(t)$, denoted by $\mathsf{NF}(\mathsf{ren}(t), \mathcal{R})$. This is possible because $s\sigma \to_{\mathcal{P}} t\sigma$ and σ is normalized as $s\sigma$ is a normal form.

Definition 28. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. The nodes of the* c-innermost dependency graph $\mathsf{DG}^{\mathsf{i}}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ *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 $\mathsf{NF}(\mathsf{ren}(\mathsf{rhs}(\alpha)), \mathcal{R})$. The nodes of the* c-improved innermost dependency graph $\mathsf{IDG}^{\mathsf{i}}_{\mathsf{c}}(\mathcal{P}, \mathcal{R})$ *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 $\Sigma_{\#}(\mathsf{RFC}(\mathsf{rhs}(\alpha), \mathcal{P} \cup \mathcal{R})) \cap \mathsf{NF}(\mathsf{ren}(\mathsf{rhs}(\alpha)), \mathcal{R})$.*

Lemma 29. *Let $\mathcal{P}$ and $\mathcal{R}$ be two TRSs. Then* $\mathsf{DG}^{\mathsf{i}}_{\mathsf{c}}(\mathcal{P},\mathcal{R}) \supseteq \mathsf{DG}^{\mathsf{i}}(\mathcal{P},\mathcal{R})$ *and* $\mathsf{IDG}^{\mathsf{i}}_{\mathsf{c}}(\mathcal{P},\mathcal{R}) \supseteq \mathsf{IDG}^{\mathsf{i}}(\mathcal{P},\mathcal{R})$.

Proof. Straightforward adaption of the proofs of Lemmata 9 and 19. □

7 Experimental Results

The techniques described in the preceding sections are integrated in the termination prover T$_{\mathsf{T}}$T$_2$. There are various ways to implement the (improved) dependency graph processors, ranging from checking single arcs to computing SCCs in between in order to reduce the number of arcs that have to be checked. The following procedure turned out to be the most efficient. For every term $t \in \mathsf{rhs}(\mathcal{P})$, T$_{\mathsf{T}}T_2$ constructs a tree automaton $\mathcal{A}_t$ that is compatible with $\mathcal{R}$ and $\ell(t)$. Here $\ell(t) = \Sigma_{\#}(\mathsf{RFC}(t, \mathcal{P}\cup\mathcal{R})) \cap \Sigma(\mathsf{ren}(t))$ if $\mathcal{P}\cup\mathcal{R}$ is right-linear and $\ell(t) = \Sigma(\mathsf{ren}(t))$ otherwise. During that process it is checked if there is a term $u \in \mathsf{lhs}(\mathcal{P})$ and a substitution σ such that $u\sigma \in \mathcal{L}(\mathcal{A}_t)$. As soon as this condition evaluates to true, we add an arc from $s \to t$ to $u \to v$ for all terms s and v such that $s \to t$ and $u \to v$ are rules in $\mathcal{P}$. This procedure is repeated until for all $t \in \mathsf{rhs}(\mathcal{P})$, either $\mathcal{A}_t$ is compatible with $\mathcal{R}$ and $\ell(t)$ or an arc was added from $s \to t$ to $u \to v$ for all terms s and rules $u \to v \in \mathcal{P}$ such that $\mathsf{root}(t) = \mathsf{root}(u)$.

Another important point is the strategy used to solve compatibility violations. In T$_{\mathsf{T}}$T$_2$ we establish paths as described in [16]. A disadvantage of this strategy is that it can happen that the completion procedure does not terminate because new states are kept being added. Hence we have to set a time limit on the involved processors to avoid that the termination proving process does not proceed beyond the calculation of (improved) dependency graphs. Alternatively one could follow the approach described in [6]. However, our experiments showed that the former approach produces better over-approximations.

Below we report on the experiments we performed with T$_{\mathsf{T}}$T$_2$ on the 1331 TRSs in the full termination category in version 5.0 of the Termination Problem Data Base[2] 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.[3]

For the results in Tables 1 and 2 we used the following (improved) dependency graph processors:

- `t` A simple and fast approximation of the dependency graph processor of Theorem 4 using root comparisons to estimate the dependency graph; an arc is added from α to β if the root symbols of $\mathsf{rhs}(\alpha)$ and $\mathsf{lhs}(\beta)$ coincide.

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

[3] Full experimental data, also covering the results on innermost termination in Section 6, can be found at `http://cl-informatik.uibk.ac.at/software/ttt2/experiments/bdg`.

Table 1. Dependency graph approximations I (without poly)

	without usable rules					with usable rules				
	t	e	c	r	*	t	e	c	r	*
arcs removed	55	68	68	72	73	55	67	68	73	74
# SCCs	4416	4529	4218	3786	4161	4416	4532	4179	3680	4114
# rules	24404	22198	20519	19369	21196	24404	22233	20306	19033	21093
# successes	25	60	67	176	183	25	58	67	191	195
average time	105	190	411	365	211	133	224	491	434	242
# timeouts	2	2	60	78	2	2	2	60	81	2

Table 2. Dependency graph approximations I (with poly)

	without usable rules					with usable rules				
	t	e	c	r	*	t	e	c	r	*
# successes	454	494	493	528	548	454	491	492	529	548
average time	265	194	329	139	198	262	196	352	134	191
# timeouts	14	14	71	89	14	14	14	70	91	14

- e The dependency graph processor with the estimation $\mathsf{DG}_\mathsf{e}(\mathcal{P},\mathcal{R})$ described in Section 5. This is the default dependency graph processor in $\mathsf{TTT2}$ and AProVE.
- c The dependency graph processor with $\mathsf{DG}_\mathsf{c}(\mathcal{P},\mathcal{R})$ of Definition 8.
- r The improved dependency graph processor of Theorem 16 with $\mathsf{IDG}_\mathsf{c}(\mathcal{P},\mathcal{R})$ ($\mathsf{DG}_\mathsf{c}(\mathcal{P},\mathcal{R})$) for (non-)right-linear $\mathcal{P}\cup\mathcal{R}$.

After applying the above processors we use the SCC processor. In Table 2 this is additionally followed by the reduction pair processor instantiated by linear polynomial interpretations with 0/1 coefficients (poly for short) [13].

In the top half of Table 1 we list the average number of removed arcs (as percentage of the complete graph), the number of SCCs, and the number of rewrite rules in the computed SCCs. In the bottom half we list the number of successful termination attempts, the average wall-clock time needed to compute the graphs (measured in milliseconds), and the number of timeouts. In Table 2 polynomial interpretations are in effect and the average time now refers to the time to prove termination.

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. As explained earlier, this is mostly due to the unbounded number of new states to resolve compatibility violations during tree automata completion. Modern termination tools use a variety of techniques to prove finiteness of DP problems. So it is in general more important that the graph approximations used in the (improved) dependency graph processor terminate quickly rather than that they are powerful. Since the processors c and r seem to be quite fast when they terminate, an obvious idea is to

Table 3. Dependency graph approximations II (without poly)

	without usable rules				with usable rules			
	c	s	nv	g	c	s	nv	g
arcs removed	68	63	61	48	68	63	62	48
# SCCs	4218	2294	1828	96	4179	2323	1925	270
# rules	20519	6754	5046	140	20306	6631	5133	448
# successes	67	54	67	42	67	57	64	51
average time	411	3640	3463	6734	491	3114	3817	1966
# timeouts	60	263	372	1197	60	251	349	1068

Table 4. Dependency graph approximations II (with poly)

	without usable rules				with usable rules			
	c	s	nv	g	c	s	nv	g
# successes	493	443	427	78	492	446	425	146
average time	329	2603	2143	5745	352	2396	2378	1180
# timeouts	71	264	375	1197	70	252	348	1069

equip each computation of a compatible tree automaton with a small time limit. Another natural idea is to limit the number of allowed compatibility violations. For instance, by reducing this number to 5 we can still prove termination of 141 TRSs with processor r while the number of timeouts is reduced from 78 to 21. Another strategy is to combine different graph approximations. This is shown in the columns of Tables 1 and 2 labeled $*$, which denotes the composition of t, e, c and r with a time limit of 500 milliseconds each for the latter three. We remark that the timeouts in the t and $*$ columns are solely due to the SCC processor.

A widely used approach to increase the power of DP processors is to consider only those rewrite rules of $\mathcal{R}$ which are usable [13, 15]. When incorporating usable rules into the processors mentioned above, we obtain the results in the second half of Tables 1 and 2. It is interesting to observe that r (and by extension $*$) is the only processor that benefits from usable rules. This is due to the right-linearity condition in Definition 16, which obviates the addition of projection rules to DP problems.

We also implemented the approximations based on tree automata and regularity preservation described in Section 5. The results are summarized in Tables 3 and 4. It is apparent that these approximations are too time-consuming to be of any use in automatic termination provers.

One advantage of more powerful (improved) dependency graph approximations is that termination proofs can get much simpler. This is implicitly illustrated in the experiments when polynomial interpretations are in effect; using r produces the fastest termination proofs. This positive effect is also preserved if more DP processors are in effect. By incorporating the new approximations

into the strategy of $\mathsf{TTT2}$ used in the termination competition of 2008,[4] $\mathsf{TTT2}$ can additionally prove termination of the TRS `TRCSR/ExProp7_Luc06_C.trs`: Using r, the number of arcs in the computed (improved) dependency graph is reduced from 159 to 30, resulting in a decrease of the number of SCCs from 7 to 2. This gives a speedup of about 500 milliseconds. In 2008, $\mathsf{TTT2}$ could not prove termination of this TRS because it exceeded its internal time limit of 5 seconds.

We conclude this section with the following small example.

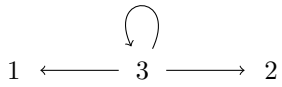
Example 30. The TRS `Endrullis/quadruple1` ($\mathcal{R}$ in the following) consists of the following rewrite rule:

$$\mathsf{p}(\mathsf{p}(\mathsf{b}(\mathsf{a}(x)),y),\mathsf{p}(z,u)) \to \mathsf{p}(\mathsf{p}(\mathsf{b}(z),\mathsf{a}(\mathsf{a}(\mathsf{b}(y)))),\mathsf{p}(u,x))$$

To prove termination of $\mathcal{R}$ using dependency pairs, we transform $\mathcal{R}$ into the initial DP problem $(\mathcal{P},\mathcal{R},\mathcal{G})$ where $\mathcal{P} = \mathsf{DP}(\mathcal{R})$ consists of the rewrite rules

$$\begin{aligned}
&1\colon \mathsf{P}(\mathsf{p}(\mathsf{b}(\mathsf{a}(x)),y),\mathsf{p}(z,u)) \to \mathsf{P}(\mathsf{p}(\mathsf{b}(z),\mathsf{a}(\mathsf{a}(\mathsf{b}(y)))),\mathsf{p}(u,x))\\
&2\colon \mathsf{P}(\mathsf{p}(\mathsf{b}(\mathsf{a}(x)),y),\mathsf{p}(z,u)) \to \mathsf{P}(\mathsf{b}(z),\mathsf{a}(\mathsf{a}(\mathsf{b}(y))))\\
&3\colon \mathsf{P}(\mathsf{p}(\mathsf{b}(\mathsf{a}(x)),y),\mathsf{p}(z,u)) \to \mathsf{P}(u,x)
\end{aligned}$$

and $\mathcal{G} = \mathcal{P} \times \mathcal{P}$. Applying the improved dependency graph processor produces the new DP problem $(\mathcal{P},\mathcal{R},\mathsf{IDG_c}(\mathcal{P},\mathcal{R}))$ where $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$ looks as follows:



After deploying the SCC processor we are left with the single DP problem $(\{3\},\mathcal{R},\{(3,3)\})$ which can easily shown to be finite by various DP processors. Using poly, $\mathsf{TTT2}$ needs about 14 milliseconds to prove termination of $\mathcal{R}$. If we use $\mathsf{DG_e}(\mathcal{P},\mathcal{R})$ instead of $\mathsf{IDG_c}(\mathcal{P},\mathcal{R})$, we do not make any progress by applying the SCC processor and thus termination of $\mathcal{R}$ cannot be shown that easily. This is reflected in the latest edition of the termination competition (2008): AProVE combined a variety of processors to infer termination within 24.31 seconds, Jambox[5] proved termination of $\mathcal{R}$ within 8.11 seconds by using linear matrix interpretations up to dimension 3, and $\mathsf{TTT2}$ used RFC match-bounds [18] to prove termination within 143 milliseconds.

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.

[4] `http://termcomp.uibk.ac.at`

[5] `http://joerg.endrullis.de`

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. I. Durand and A. Middeldorp. Decidable call-by-need computations in term rewriting. *I&C*, 196(2):95–126, 2005.
6. 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.
7. A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. Finding finite automata that certify termination of string rewriting systems. *IJFCS*, 16(3):471–486, 2005.
8. 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.
9. O. Geupel. Overlap closures and termination of term rewriting systems. Report MIP-8922, Universität Passau, 1989.
10. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. 3rd IJCAR*, volume 4130 of *LNAI*, pages 281–286, 2006.
11. 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.
12. 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.
13. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *JAR*, 37(3):155–203, 2006.
14. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *I&C*, 199(1-2):172–199, 2005.
15. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *I&C*, 205(4):474–511, 2007.
16. M. Korp and A. Middeldorp. Proving termination of rewrite systems using bounds. In *Proc. 18th RTA*, volume 4533 of *LNCS*, pages 273–287, 2007.
17. M. Korp and A. Middeldorp. Match-bounds with dependency pairs for proving termination of rewrite systems. In *Proc. 2nd LATA*, volume 5196 of *LNCS*, pages 321–332, 2008.
18. M. Korp and A. Middeldorp. Match-bounds revisited. *I&C*, 2009. To appear.
19. M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean termination tool 2. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 295–304, 2009.
20. K. Kusakari. *Termination, AC-Termination and Dependency Pairs of Term Rewriting Systems*. PhD thesis, JAIST, 2000.
21. K. Kusakari and Y. Toyama. On proving AC-termination by AC-dependency pairs. Research Report IS-RR-98-0026F, School of Information Science, JAIST, 1998.
22. A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proc. 1st IJCAR*, volume 2083 of *LNAI*, pages 593–610, 2001.
23. A. Middeldorp. Approximations for strategies and termination. In *Proc. 2nd WRS*, volume 70 of *ENTCS*, pages 1–20, 2002.
24. T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. *I&C*, 178(2):499–514, 2002.
25. R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting*. PhD thesis, RWTH Aachen, 2007. Available as technical report AIB-2007-17.