The Dependency Triple Framework for Termination of Logic Programs[⋆]

Peter Schneider-Kamp[1], Jürgen Giesl[2], and Manh Thang Nguyen[3]

[1] IMADA, University of Southern Denmark, Denmark
[2] LuFG Informatik 2, RWTH Aachen University, Germany
[3] Department of Computer Science, K. U. Leuven, Belgium

Abstract. We show how to combine the two most powerful approaches for automated termination analysis of logic programs (LPs): the *direct* approach which operates directly on LPs and the *transformational* approach which transforms LPs to term rewrite systems (TRSs) and tries to prove termination of the resulting TRSs. To this end, we adapt the well-known *dependency pair framework* from TRSs to LPs. With the resulting method, one can combine arbitrary termination techniques for LPs in a completely modular way and one can use both direct and transformational techniques for different parts of the same LP.

1 Introduction

When comparing the direct and the transformational approach for termination of LPs, there are the following advantages and disadvantages. The *direct* approach is more efficient (since it avoids the transformation to TRSs) and in addition to the TRS techniques that have been adapted to LPs [13, 15], it can also use numerous other techniques that are specific to LPs. The *transformational* approach has the advantage that it can use *all* existing termination techniques for TRSs, not just the ones that have already been adapted to LPs.

Two of the leading tools for termination of LPs are Polytool [14] (implementing the direct approach and including the adapted TRS techniques from [13, 15]) and AProVE [7] (implementing the transformational approach of [17]). In the annual *International Termination Competition*,[4] AProVE was the most powerful tool for termination analysis of LPs (it solved 246 out of 349 examples), but Polytool obtained a close second place (solving 238 examples). Nevertheless, there are several examples where one tool succeeds, whereas the other does not.

This shows that both the direct and the transformational approach have their benefits. Thus, one should combine these approaches *in a modular way.* In other words, for one and the same LP, it should be possible to prove termination of some parts with the direct approach and of other parts with the transformational

⋆ Supported by *FWO/2006/09: Termination analysis: Crossing paradigm borders* and by the *Deutsche Forschungsgemeinschaft (DFG)*, grant *GI 274/5-2.*

[4] `http://www.termination-portal.org/wiki/Termination_Competition`

approach. The resulting method would improve over both approaches and can also prove termination of LPs that cannot be handled by one approach alone.

In this paper, we solve that problem. We build upon [15], where the well-known *dependency pair* (DP) method from term rewriting [2] was adapted in order to apply it to LPs directly. However, [15] only adapted the most basic parts of the method and moreover, it only adapted the classical variant of the DP method instead of the more powerful recent *DP framework* [6, 8, 9] which can combine different TRS termination techniques in a completely flexible way.

After providing the necessary preliminaries on LPs in Sect. 2, in Sect. 3 we adapt the DP framework to the LP setting which results in the new *dependency triple (DT) framework*. Compared to [15], the advantage is that now arbitrary termination techniques based on DTs can be applied in any combination and any order. In Sect. 4, we present three termination techniques within the DT framework. In particular, we also develop a new technique which can transform *parts* of the original LP termination problem into TRS termination problems. Then one can apply TRS techniques and tools to solve these subproblems.

We implemented our contributions in the tool Polytool and coupled it with AProVE which is called on those subproblems which were converted to TRSs. Our experimental evaluation in Sect. 5 shows that this combination clearly improves over both Polytool or AProVE alone, both concerning efficiency and power.

2 Preliminaries on Logic Programming

We briefly recapitulate needed notations. More details on logic programming can be found in [1], for example. A *signature* is a pair (Σ, Δ) where Σ and Δ are finite sets of function and predicate symbols and $\mathcal{T}(\Sigma, \mathcal{V})$ resp. $\mathcal{A}(\Sigma, \Delta, \mathcal{V})$ denote the sets of all terms resp. atoms over the signature (Σ, Δ) and the variables $\mathcal{V}$. We always assume that Σ contains at least one constant of arity 0. A *clause* c is a formula $H \leftarrow B_1, \ldots, B_k$ with $k \geq 0$ and $H, B_i \in \mathcal{A}(\Sigma, \Delta, \mathcal{V})$. A finite set of clauses $\mathcal{P}$ is a (definite) *logic program.* A clause with empty body is a *fact* and a clause with empty head is a *query.* We usually omit "$\leftarrow$" in queries and just write "$B_1, \ldots, B_k$". The empty query is denoted $\Box$.

For a *substitution* $\delta : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$, we often write $t\delta$ instead of $\delta(t)$, where t can be any expression (e.g., a term, atom, clause, etc.). If δ is a variable renaming (i.e., a one-to-one correspondence on $\mathcal{V}$), then $t\delta$ is a *variant* of t. We write $\delta\sigma$ to denote that the application of δ is followed by the application of σ. A substitution δ is a *unifier* of two expressions s and t iff $s\delta = t\delta$. To simplify the presentation, in this paper we restrict ourselves to ordinary unification with occur check. We call δ the *most general unifier (mgu)* of s and t iff δ is a unifier of s and t and for all unifiers σ of s and t, there is a substitution μ such that $\sigma = \delta\mu$.

Let Q be a query $A_1, \ldots, A_m$, let c be a clause $H \leftarrow B_1, \ldots, B_k$. Then Q' is a *resolvent* of Q and c using δ (denoted $Q \vdash_{c,\delta} Q'$) if $\delta = mgu(A_1, H)$, and $Q' = (B_1, \ldots, B_k, A_2, \ldots, A_m)\delta$. A *derivation* of a program $\mathcal{P}$ and a query Q is a possibly infinite sequence $Q_0, Q_1, \ldots$ of queries with $Q_0 = Q$ where for all i, we have $Q_i \vdash_{c_i,\delta_i} Q_{i+1}$ for some substitution δ_i and some renamed-apart variant c_i of a clause of $\mathcal{P}$. For a derivation $Q_0, \ldots, Q_n$ as above, we also write $Q_0 \vdash^n_{\mathcal{P},\delta_0\ldots\delta_{n-1}} Q_n$ or $Q_0 \vdash^n_{\mathcal{P}} Q_n$, and we also write $Q_i \vdash_{\mathcal{P}} Q_{i+1}$ for $Q_i \vdash_{c_i,\delta_i} Q_{i+1}$. A LP $\mathcal{P}$ is *terminating* for the query Q if all derivations of $\mathcal{P}$ and Q are finite. The *answer set* $\mathit{Answer}(\mathcal{P}, Q)$ for a LP $\mathcal{P}$ and a query Q is the set of all substitutions δ such that $Q \vdash^n_{\mathcal{P},\delta} \Box$ for some $n \in \mathbb{N}$. For a set of atomic queries $\mathcal{S} \subseteq \mathcal{A}(\Sigma, \Delta, \mathcal{V})$, we define the *call set* $\mathit{Call}(\mathcal{P}, \mathcal{S}) = \{A_1 \mid Q \vdash^n_{\mathcal{P}} A_1, \ldots, A_m,\ Q \in \mathcal{S},\ n \in \mathbb{N}\}$.

Example 1. The following LP $\mathcal{P}$ uses "$\mathsf{s2m}$*" to create a matrix M of variables for fixed numbers X and Y of rows and columns. Afterwards, it uses "*$\mathsf{subs\_mat}$*" to replace each variable in the matrix by the constant "*a*".*

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{goal}(X, Y, Msu) \leftarrow \mathsf{s2m}(X, Y, M), \mathsf{subs\_mat}(M, Msu).} \\
\mathsf{s2m}(\mathsf{0}, Y, [\,]). & \mathsf{s2m}(\mathsf{s}(X), Y, [R|Rs]) \leftarrow \mathsf{s2\ell}(Y, R), \mathsf{s2m}(X, Y, Rs). \\
\mathsf{s2\ell}(\mathsf{0}, [\,]). & \mathsf{s2\ell}(\mathsf{s}(Y), [C|Cs]) \leftarrow \mathsf{s2\ell}(Y, Cs). \\
\mathsf{subs\_mat}([\,], [\,]). & \mathsf{subs\_mat}([R|Rs], [SR|SRs]) \leftarrow \mathsf{subs\_row}(R, SR), \mathsf{subs\_mat}(Rs, SRs). \\
\mathsf{subs\_row}([\,], [\,]). & \mathsf{subs\_row}([E|R], [\mathsf{a}|SR]) \leftarrow \mathsf{subs\_row}(R, SR).
\end{array}
$$

For example, for suitable substitutions δ_0 and δ_1 we have $\mathsf{goal}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}), Msu) \vdash_{\delta_0,\mathcal{P}} \mathsf{s2m}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}), M), \mathsf{subs\_mat}(M, Msu) \vdash^8_{\delta_1,\mathcal{P}} \Box$*. So* $\mathit{Answer}(\mathcal{P}, \mathsf{goal}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}), Msu))$ *contains $\delta = \delta_0\delta_1$, where $\delta(Msu) = [[\mathsf{a}]]$.*

We want to prove termination of this program for the set of queries $\mathcal{S} = \{\mathsf{goal}(t_1, t_2, t_3) \mid t_1$ *and* t_2 *are ground terms* $\}$*. Here, we obtain*

$$
\begin{aligned}
\mathit{Call}(\mathcal{P}, \mathcal{S}) \subseteq \mathcal{S} \cup \{&\{\mathsf{s2m}(t_1, t_2, t_3) \mid t_1 \text{ and } t_2 \text{ ground}\} \cup \{\mathsf{s2\ell}(t_1, t_2) \mid t_1 \text{ ground}\} \\
&\cup \{\mathsf{subs\_row}(t_1, t_2) \mid t_1 \in \mathit{List}\} \cup \{\mathsf{subs\_mat}(t_1, t_2) \mid t_1 \in \mathit{List}\}
\end{aligned}
$$

where List is the smallest set with $[\,] \in \mathit{List}$ and $[t_1 \mid t_2] \in \mathit{List}$ if $t_2 \in \mathit{List}$.

3 Dependency Triple Framework

As mentioned before, we already adapted the basic DP method to the LP setting in [15]. The advantage of [15] over previous direct approaches for LP termination is that (a) it can use different well-founded orders for different "loops" of the LP and (b) it uses a constraint-based approach to search for arbitrary suitable well-founded orders (instead of only choosing from a fixed set of orders based on a given small set of norms). Most other direct approaches have only one of the features (a) or (b). Nevertheless, [15] has the disadvantage that it does not permit the combination of arbitrary termination techniques in a flexible and modular way. Therefore, we now adapt the recent DP framework [6, 8, 9] to the LP setting. Def. 2 adapts the notion of *dependency pairs* [2] from TRSs to LPs.[5]

Definition 2 (Dependency Triple). *A* dependency triple *(DT) is a clause $H \leftarrow I, B$ where H and B are atoms and I is a list of atoms. For a LP $\mathcal{P}$, the set of its dependency triples is $DT(\mathcal{P}) = \{H \leftarrow I, B \mid H \leftarrow I, B, \ldots \in \mathcal{P}\}$.*

[5] While Def. 2 is essentially from [15], the rest of this section contains new concepts that are needed for a flexible and general framework.

Example 3. The dependency triples $DT(\mathcal{P})$ of the program in Ex. 1 are:

$$\mathsf{goal}(X, Y, Msu) \leftarrow \mathsf{s2m}(X, Y, M). \tag{1}$$
$$\mathsf{goal}(X, Y, Msu) \leftarrow \mathsf{s2m}(X, Y, M), \mathsf{subs\_mat}(M, Msu). \tag{2}$$
$$\mathsf{s2m}(\mathsf{s}(X), Y, [R|Rs]) \leftarrow \mathsf{s2}\ell(Y, R). \tag{3}$$
$$\mathsf{s2m}(\mathsf{s}(X), Y, [R|Rs]) \leftarrow \mathsf{s2}\ell(Y, R), \mathsf{s2m}(X, Y, Rs). \tag{4}$$
$$\mathsf{s2}\ell(\mathsf{s}(Y), [C|Cs]) \leftarrow \mathsf{s2}\ell(Y, Cs). \tag{5}$$
$$\mathsf{subs\_mat}([R|Rs], [SR|SRs]) \leftarrow \mathsf{subs\_row}(R, SR). \tag{6}$$
$$\mathsf{subs\_mat}([R|Rs], [SR|SRs]) \leftarrow \mathsf{subs\_row}(R, SR), \mathsf{subs\_mat}(Rs, SRs). \tag{7}$$
$$\mathsf{subs\_row}([E|R], [\mathsf{a}|SR]) \leftarrow \mathsf{subs\_row}(R, SR). \tag{8}$$

Intuitively, a dependency triple $H \leftarrow I, B$ states that a call that is an instance of H can be followed by a call that is an instance of B if the corresponding instance of I can be proven. To use DTs for termination analysis, one has to show that there are no infinite "chains" of such calls. The following definition corresponds to the standard definition of *chains* from the TRS setting [2]. Usually, $\mathcal{D}$ stands for the set of DTs, $\mathcal{P}$ is the program under consideration, and $\mathcal{C}$ stands for $\mathit{Call}(\mathcal{P}, \mathcal{S})$ where $\mathcal{S}$ is the set of queries to be analyzed for termination.

Definition 4 (Chain). *Let $\mathcal{D}$ and $\mathcal{P}$ be sets of clauses and let $\mathcal{C}$ be a set of atoms. A (possibly infinite) list $(H_0 \leftarrow I_0, B_0), (H_1 \leftarrow I_1, B_1), \ldots$ of variants from $\mathcal{D}$ is a $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-*chain *iff there are substitutions θ_i, σ_i and an $A \in \mathcal{C}$ such that $\theta_0 = mgu(A, H_0)$ and for all i, we have $\sigma_i \in \mathit{Answer}(\mathcal{P}, I_i\theta_i)$, $\theta_{i+1} = mgu(B_i\theta_i\sigma_i, H_{i+1})$, and $B_i\theta_i\sigma_i \in \mathcal{C}$.*[6]

Example 5. For $\mathcal{P}$ and $\mathcal{S}$ from Ex. 1, the list (2), (7) is a $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P}, \mathcal{S}), \mathcal{P})$-chain. To see this, consider $\theta_0 = \{X/\mathsf{s}(0), Y/\mathsf{s}(0)\}$, $\sigma_0 = \{M/[[C]]\}$, and $\theta_1 = \{R/[C], Rs/[\,], Msu/[SR, SRs]\}$. Then, for $A = \mathsf{goal}(\mathsf{s}(0), \mathsf{s}(0), Msu) \in \mathcal{S}$, we have $H_0\theta_0 = \mathsf{goal}(X, Y, Msu)\theta_0 = A\theta_0$. Furthermore, we have $\sigma_0 \in \mathit{Answer}(\mathcal{P}, \mathsf{s2m}(X, Y, M)\theta_0) = \mathit{Answer}(\mathcal{P}, \mathsf{s2m}(\mathsf{s}(0), \mathsf{s}(0), M))$ and $\theta_1 = mgu(B_0\theta_0\sigma_0, H_1) = mgu(\mathsf{subs\_mat}([[C]], Msu), \mathsf{subs\_mat}([R|Rs], [SR|SRs]))$.

Thm. 6 shows that termination is equivalent to absence of infinite chains.

Theorem 6 (Termination Criterion). *A LP $\mathcal{P}$ is terminating for a set of atomic queries $\mathcal{S}$ iff there is no infinite $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P}, \mathcal{S}), \mathcal{P})$-chain.*

Proof. For the "if"-direction, let there be an infinite derivation $Q_0, Q_1, \ldots$ with $Q_0 \in \mathcal{S}$ and $Q_i \vdash_{c_i, \delta_i} Q_{i+1}$. The clause $c_i \in \mathcal{P}$ has the form $H_i \leftarrow A_i^1, \ldots, A_i^{k_i}$. Let $j_1 > 0$ be the minimal index such that the first atom A'_{j_1} in Q_{j_1} starts an infinite derivation. Such a j_1 always exists as shown in [17, Lemma 3.5]. As we started from an atomic query, there must be some m_0 such that $A'_{j_1} =$

[6] If $\mathcal{C} = \mathit{Call}(\mathcal{P}, \mathcal{S})$, then the condition "$B_i\theta_i\sigma_i \in \mathcal{C}$" is always satisfied due to the definition of "Call". But our goal is to formulate the concept of "chains" as general as possible (i.e., also for cases where $\mathcal{C}$ is an arbitrary set). Then this condition can be helpful in order to obtain as few chains as possible.

$A_0^{m_0}\delta_0\delta_1\ldots\delta_{j_1-1}$. Then "$H_0 \leftarrow A_0^1,\ldots,A_0^{m_0-1},A_0^{m_0}$" is the first DT in our $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P},\mathcal{S}),\mathcal{P})$-chain where $\theta_0=\delta_0$ and $\sigma_0=\delta_1\ldots\delta_{j_1-1}$. As $Q_0 \vdash_{\mathcal{P}}^{j_1} Q_{j_1}$ and $A_0^{m_0}\theta_0\sigma_0 = A'_{j_1}$ is the first atom in Q_{j_1}, we have $A_0^{m_0}\theta_0\sigma_0 \in \mathit{Call}(\mathcal{P},\mathcal{S})$.

We repeat this construction and let j_2 be the minimal index with $j_2 > j_1$ such that the first atom A'_{j_2} in Q_{j_2} starts an infinite derivation. As the first atom of Q_{j_1} already started an infinite derivation, there must be some m_{j_1} such that $A'_{j_2} = A_{j_1}^{m_{j_1}}\delta_{j_1}\ldots\delta_{j_2-1}$. Then "$H_{j_1} \leftarrow A_{j_1}^1,\ldots,A_{j_1}^{m_{j_1}-1},A_{j_1}^{m_{j_1}}$" is the second DT in our $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P},\mathcal{S}),\mathcal{P})$-chain where $\theta_1 = mgu(A_0^{m_0}\theta_0\sigma_0, H_{j_1}) = \delta_{j_1}$ and $\sigma_1 = \delta_{j_1+1}\ldots\delta_{j_2-1}$. As $Q_0 \vdash_{\mathcal{P}}^{j_2} Q_{j_2}$ and $A_{j_1}^{m_{j_1}}\theta_1\sigma_1 = A'_{j_2}$ is the first atom in Q_{j_2}, we have $A_{j_1}^{m_{j_1}}\theta_1\sigma_1 \in \mathit{Call}(\mathcal{P},\mathcal{S})$. By repeating this construction infinitely many times, we obtain an infinite $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P},\mathcal{S}),\mathcal{P})$-chain.

For the "only if"-direction, assume that $(H_0 \leftarrow I_0, B_0), (H_1 \leftarrow I_1, B_1), \ldots$ is an infinite $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P},\mathcal{S}),\mathcal{P})$-chain. Thus, there are substitutions θ_i, σ_i and an $A \in \mathit{Call}(\mathcal{P},\mathcal{S})$ such that $\theta_0 = mgu(A, H_0)$ and for all i, we have $\sigma_i \in \mathit{Answer}(\mathcal{P}, I_i\theta_i)$ and $\theta_{i+1} = mgu(B_i\theta_i\sigma_i, H_{i+1})$. Due to the construction of $DT(\mathcal{P})$, there is a clause $c_0 \in \mathcal{P}$ with $c_0 = H_0 \leftarrow I_0, B_0, R_0$ for a list of atoms R_0 and the first step in our derivation is $A \vdash_{c_0,\theta_0} I_0\theta_0, B_0\theta_0, R_0\theta_0$. From $\sigma_0 \in \mathit{Answer}(\mathcal{P}, I_0\theta_0)$ we obtain the derivation $I_0\theta_0 \vdash_{\mathcal{P},\sigma_0}^{n_0} \Box$ and consequently, $I_0\theta_0, B_0\theta_0, R_0\theta_0 \vdash_{\mathcal{P},\sigma_0}^{n_0} B_0\theta_0\sigma_0, R_0\theta_0\sigma_0$ for some $n_0 \in \mathbb{N}$. Hence, $A \vdash_{\mathcal{P},\theta_0\sigma_0}^{n_0+1} B_0\theta_0\sigma_0, R_0\theta_0\sigma_0$. As $\theta_1 = mgu(B_0\theta_0\sigma_0, H_1)$ and as there is a clause $c_1 = H_1 \leftarrow I_1, B_1, R_1 \in \mathcal{P}$, we continue the derivation with $B_0\theta_0\sigma_0, R_0\theta_0\sigma_0 \vdash_{c_1,\theta_1} I_1\theta_1, B_1\theta_1, R_1\theta_1, R_0\theta_0\sigma_0\theta_1$. Due to $\sigma_1 \in \mathit{Answer}(\mathcal{P}, I_1\theta_1)$ we continue with $I_1\theta_1, B_1\theta_1, R_1\theta_1, R_0\theta_0\sigma_0\theta_1 \vdash_{\mathcal{P},\sigma_1}^{n_1} B_1\theta_1\sigma_1, R_1\theta_1\sigma_1, R_0\theta_0\sigma_0\theta_1\sigma_1$ for some $n_1 \in \mathbb{N}$.

By repeating this, we obtain an infinite derivation $A \vdash_{\mathcal{P},\theta_0\sigma_0}^{n_0+1} B_0\theta_0\sigma_0, R_0\theta_0\sigma_0 \vdash_{\mathcal{P},\theta_1,\sigma_1}^{n_1+1} B_1\theta_1\sigma_1, R_1\theta_1\sigma_1, R_0\theta_0\sigma_0\theta_1\sigma_1 \vdash_{\mathcal{P},\theta_2\sigma_2}^{n_2+1} B_2\theta_2\sigma_2, \ldots \vdash_{\mathcal{P},\theta_3\sigma_3}^{n_2+1} \ldots$ Thus, the LP $\mathcal{P}$ is not terminating for A. From $A \in \mathit{Call}(\mathcal{P},\mathcal{S})$ we know there is a $Q \in \mathcal{S}$ such that $Q \vdash_{\mathcal{P}}^{n} A, \ldots$ Hence, $\mathcal{P}$ is also not terminating for $Q \in \mathcal{S}$. $\Box$

Termination techniques are now called *DT processors* and they operate on so-called *DT problems* and try to prove absence of infinite chains.

Definition 7 (DT Problem). *A* DT problem *is a triple* $(\mathcal{D},\mathcal{C},\mathcal{P})$ *where* $\mathcal{D}$ *and* $\mathcal{P}$ *are finite sets of clauses and* $\mathcal{C}$ *is a set of atoms. A DT problem* $(\mathcal{D},\mathcal{C},\mathcal{P})$ *is* terminating *iff there is no infinite* $(\mathcal{D},\mathcal{C},\mathcal{P})$*-chain.*

A DT processor *Proc takes a DT problem as input and returns a set of DT problems which have to be solved instead. Proc is* sound *if for all non-terminating DT problems* $(\mathcal{D},\mathcal{C},\mathcal{P})$*, there is also a non-terminating DT problem in Proc(* $(\mathcal{D}, \mathcal{C},\mathcal{P})$ *). So if Proc(* $(\mathcal{D},\mathcal{C},\mathcal{P})$ *)* $= \varnothing$*, then termination of* $(\mathcal{D},\mathcal{C},\mathcal{P})$ *is proved.*

Termination proofs now start with the *initial* DT problem $(DT(\mathcal{P}), \mathit{Call}(\mathcal{P}, \mathcal{S}),\mathcal{P})$ whose termination is equivalent to the termination of the LP $\mathcal{P}$ for the queries $\mathcal{S}$, cf. Thm. 6. Then sound DT processors are applied repeatedly until all DT problems have been simplified to $\varnothing$.

4 Dependency Triple Processors

In Sect. 4.1 and 4.2, we adapt two of the most important DP processors from term rewriting [2, 6, 8, 9] to the LP setting. In Sect. 4.3 we present a new DT processor to convert DT problems to DP problems.

4.1 Dependency Graph Processor

The first processor decomposes a DT problem into subproblems. Here, one constructs a *dependency graph* to determine which DTs follow each other in chains.

Definition 8 (Dependency Graph). *For a DT problem $(\mathcal{D}, \mathcal{C}, \mathcal{P})$, the nodes of the $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-*dependency graph *are the clauses of $\mathcal{D}$ and there is an arc from a clause c to a clause d iff "c, d" is a $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-chain.*

Example 9. For the initial DT problem $(DT(\mathcal{P}), Call(\mathcal{P}, \mathcal{S}), \mathcal{P})$ of the program in Ex. 1, we obtain the following dependency graph.

$$\begin{tikzcd}
(1) \arrow[r] \arrow[dr] & (3) \arrow[rr] & & (5) \arrow[loop right] & (2) \arrow[r] \arrow[dr] & (6) \arrow[rr] & & (8) \arrow[loop right] \\
& & (4) \arrow[ul] \arrow[loop right] & & & & (7) \arrow[ul] \arrow[loop right] &
\end{tikzcd}$$

As in the TRS setting, the dependency graph is not computable in general. For TRSs, several techniques were developed to over-approximate dependency graphs automatically, cf. e.g. [2, 9]. Def. 10 adapts the estimation of [2].[7] This estimation ignores the intermediate atoms I in a DT $H \leftarrow I, B$.

Definition 10 (Estimated Dependency Graph). *For a DT problem $(\mathcal{D}, \mathcal{C}, \mathcal{P})$, the nodes of the* estimated $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-dependency graph *are the clauses of $\mathcal{D}$ and there is an arc from $H_i \leftarrow I_i, B_i$ to $H_j \leftarrow I_j, B_j$, iff B_i unifies with a variant of H_j and there are atoms $A_i, A_j \in \mathcal{C}$ such that A_i unifies with a variant of H_i and A_j unifies with a variant of H_j.*

For the program of Ex. 1, the estimated dependency graph is identical to the real dependency graph in Ex. 9.

Example 11. To illustrate their difference, consider the LP $\mathcal{P}'$ with the clauses $\mathsf{p} \leftarrow \mathsf{q}(\mathsf{a}), \mathsf{p}$ and $\mathsf{q}(\mathsf{b})$. We consider the set of queries $\mathcal{S}' = \{\mathsf{p}\}$ and obtain $Call(\mathcal{P}', \mathcal{S}') = \{\mathsf{p}, \mathsf{q}(\mathsf{a})\}$. There are two DTs $\mathsf{p} \leftarrow \mathsf{q}(\mathsf{a})$ and $\mathsf{p} \leftarrow \mathsf{q}(\mathsf{a}), \mathsf{p}$. In the estimated dependency graph for the initial DT problem $(DT(\mathcal{P}'), Call(\mathcal{P}', \mathcal{S}'), \mathcal{P}')$, there is an arc from the second DT to itself. But this arc is missing in the real dependency graph because of the unsatisfiable body atom $\mathsf{q}(\mathsf{a})$.

The following lemma proves the "soundness" of estimated dependency graphs.

[7] The advantage of a general concept of dependency graphs like Def. 8 is that this permits the introduction of better estimations in the future without having to change the rest of the framework. However, a general concept like Def. 8 was missing in [15], which only featured a variant of the estimated dependency graph from Def. 10.

Lemma 12. *The estimated $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-dependency graph over-approximates the real $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-dependency graph, i.e., whenever there is an arc from c to d in the real graph, then there is also such an arc in the estimated graph.*

Proof. Assume that there is an arc from the clause $H_i \leftarrow I_i, B_i$ to $H_j \leftarrow I_j, B_j$ in the real dependency graph. Then by Def. 4, there are substitutions σ_i and θ_i such that θ_{i+1} is a unifier of $B_i\theta_i\sigma_i$ and H_j. As we can assume H_j and B_i to be variable disjoint, $\theta_i\sigma_i\theta_{i+1}$ is a unifier of B_i and H_j. Def. 4 also implies that for all DTs $H \leftarrow I, B$ in a $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-chain, there is an atom from $\mathcal{C}$ unifying with H. Hence, this also holds for H_i and H_j. $\square$

A set $\mathcal{D}' \neq \varnothing$ of DTs is a *cycle* if for all $c, d \in \mathcal{D}'$, there is a non-empty path from c to d traversing only DTs of $\mathcal{D}'$. A cycle $\mathcal{D}'$ is a *strongly connected component (SCC)* if $\mathcal{D}'$ is not a proper subset of another cycle. So the dependency graph in Ex. 9 has the SCCs $\mathcal{D}_1 = \{(4)\}$, $\mathcal{D}_2 = \{(5)\}$, $\mathcal{D}_3 = \{(7)\}$, $\mathcal{D}_4 = \{(8)\}$. The following processor allows us to prove termination separately for each SCC.

Theorem 13 (Dependency Graph Processor). *We define Proc($(\mathcal{D}, \mathcal{C}, \mathcal{P})$) $= \{(\mathcal{D}_1, \mathcal{C}, \mathcal{P}), \ldots, (\mathcal{D}_n, \mathcal{C}, \mathcal{P})\}$, where $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are the SCCs of the (estimated) $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-dependency graph. Then Proc is sound.*

Proof. Let there be an infinite $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-chain. This infinite chain corresponds to an infinite path in the dependency graph (resp. in the estimated graph, by Lemma 12). Since $\mathcal{D}$ is finite, the path must be contained entirely in some SCC $\mathcal{D}_i$. Thus, $(\mathcal{D}_i, \mathcal{C}, \mathcal{P})$ is non-terminating. $\square$

Example 14. For the program of Ex. 1, the above processor transforms the initial DT problem $(DT(\mathcal{P}), Call(\mathcal{P}, \mathcal{S}), \mathcal{P})$ to $(\mathcal{D}_1, Call(\mathcal{P}, \mathcal{S}), \mathcal{P})$, $(\mathcal{D}_2, Call(\mathcal{P}, \mathcal{S}), \mathcal{P})$, $(\mathcal{D}_3, Call(\mathcal{P}, \mathcal{S}), \mathcal{P})$, and $(\mathcal{D}_4, Call(\mathcal{P}, \mathcal{S}), \mathcal{P})$. So the original termination problem is split up into four subproblems which can now be solved independently.

4.2 Reduction Pair Processor

The next processor uses a *reduction pair* $(\succsim, \succ)$ and requires that all DTs are weakly or strictly decreasing. Then the strictly decreasing DTs can be removed from the current DT problem. A *reduction pair* $(\succsim, \succ)$ consists of a quasi-order $\succsim$ on atoms and terms (i.e., a reflexive and transitive relation) and a well-founded order $\succ$ (i.e., there is no infinite sequence $t_0 \succ t_1 \succ \ldots$). Moreover, $\succsim$ and $\succ$ have to be *compatible* (i.e., $t_1 \succsim t_2 \succ t_3$ implies $t_1 \succ t_3$).[8]

Example 15. We often use reduction pairs built from norms and level mappings [3]. A norm *is a mapping $\| \cdot \| : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathbb{N}$. A* level mapping *is a mapping $| \cdot | : \mathcal{A}(\Sigma, \Delta, \mathcal{V}) \to \mathbb{N}$. Consider the reduction pair $(\succsim, \succ)$ induced*[9]

[8] In contrast to "reduction pairs" in rewriting, we do not require $\succsim$ and $\succ$ to be closed under substitutions. But for automation, we usually choose relations $\succsim$ and $\succ$ that result from polynomial interpretations which are closed under substitutions.

[9] So for terms t_1, t_2 we define $t_1 \;{}_{(}\!\succsim\!{}_{)}\; t_2$ iff $\|t_1\| \;{}_{(}\!\geq\!{}_{)}\; \|t_2\|$ and for atoms A_1, A_2 we define $A_1 \;{}_{(}\!\succsim\!{}_{)}\; A_2$ iff $|A_1| \;{}_{(}\!\geq\!{}_{)}\; |A_2|$.

by the norm $\|X\| = 0$ *for all variables* X*,* $\|\,[\,]\,\| = 0$*,* $\|\mathsf{s}(t)\| = \|\,[s \mid t]\,\| = 1 + \|t\|$ *and the level mapping* $|\mathsf{s2m}(t_1,t_2,t_3)| = |\mathsf{s2\ell}(t_1,t_2)| = |\mathsf{subs\_mat}(t_1,t_2)| = |\mathsf{subs\_row}(t_1,t_2)| = \|t_1\|$*. Then* $\mathsf{subs\_mat}([[C]],[SR \mid SRs]) \succ \mathsf{subs\_mat}([\,],SRs)$*, as* $|\mathsf{subs\_mat}([[C]],[SR \mid SRs])| = \|[[C]]\| = 1$ *and* $|\mathsf{subs\_mat}([\,],SRs)| = \|\,[\,]\,\| = 0$*.*

Now we can define when a DT $H \leftarrow I, B$ is decreasing. Roughly, we require that $H\sigma \succ B\sigma$ must hold for every substitution σ. However, we do not have to regard *all* substitutions, but we may restrict ourselves to such substitutions where all variables of H and B on positions that are "taken into account" by $\succsim$ and $\succ$ are instantiated by ground terms.[10] Formally, a reduction pair $(\succsim, \succ)$ is *rigid* on a term or atom t if we have $t \approx t\delta$ for all substitutions δ. Here, we define $s \approx t$ iff $s \succsim t$ and $t \succsim s$. A reduction pair $(\succsim, \succ)$ is rigid on a set of terms or atoms if it is rigid on all its elements. Now for a DT $H \leftarrow I, B$ to be decreasing, we only require that $H\sigma \succ B\sigma$ holds for all σ where $(\succsim, \succ)$ is rigid on $H\sigma$.

Example 16. The reduction pair from Ex. 15 is rigid on the atom $A = \mathsf{s2m}([[C]], [SR \mid SRs])$*, since* $|A\delta| = 1$ *holds for every substitution* δ*. Moreover, if* $\sigma(Rs) \in$ *List, then the reduction pair is also rigid on* $\mathsf{subs\_mat}([R \mid Rs],[SR \mid SRs])\sigma$*. For every such* σ*, we have* $\mathsf{subs\_mat}([R \mid Rs],[SR \mid SRs])\sigma \succ \mathsf{subs\_mat}(Rs, SRs)\sigma$*.*

We refine the notion of "decreasing" DTs $H \leftarrow I, B$ further. Instead of only considering H and B, one should also take the intermediate body atoms I into account. To approximate their semantics, we use *interargument relations*. An *interargument relation* for a predicate p is a relation $IR_p = \{p(t_1,\ldots,t_n) \mid t_i \in \mathcal{T}(\Sigma,\mathcal{V}) \wedge \varphi_p(t_1,\ldots,t_n)\}$, where (1) $\varphi_p(t_1,\ldots,t_n)$ is a formula of an arbitrary Boolean combination of inequalities, and (2) each inequality in φ_p is either $s_i \succsim s_j$ or $s_i \succ s_j$, where s_i, s_j are constructed from $t_1,\ldots,t_n$ by applying function symbols of $\mathcal{P}$. IR_p is *valid* iff $p(t_1,\ldots,t_n) \vdash^m_{\mathcal{P}} \Box$ implies $p(t_1,\ldots,t_n) \in IR_p$ for every $p(t_1,\ldots,t_n) \in \mathcal{A}(\Sigma,\Delta,\mathcal{V})$.

Definition 17 (Decreasing DTs). *Let* $(\succsim, \succ)$ *be a reduction pair, and* $\mathfrak{R} = \{IR_{p_1},\ldots,IR_{p_k}\}$ *be a set of valid interargument relations based on* $(\succsim, \succ)$*. Let* $c = H \leftarrow p_1(\boldsymbol{t}_1),\ldots,p_k(\boldsymbol{t}_k), B$ *be a DT. Here, the* $\boldsymbol{t}_i$ *are tuples of terms.*

The DT c *is* weakly decreasing *(denoted* $(\succsim,\mathfrak{R}) \models c$*) if* $H\sigma \succsim B\sigma$ *holds for any substitution* σ *where* $(\succsim, \succ)$ *is rigid on* $H\sigma$ *and where* $p_1(\boldsymbol{t}_1)\sigma \in IR_{p_1},\ldots, p_k(\boldsymbol{t}_k)\sigma \in IR_{p_k}$*. Analogously, c is* strictly decreasing *(denoted* $(\succ,\mathfrak{R}) \models c$*) if* $H\sigma \succ B\sigma$ *holds for any such* σ*.*

Example 18. Recall the reduction pair from Ex. 15 and the remarks about its rigidity in Ex. 16. When considering a set $\mathfrak{R}$ *of trivial valid interargument relations like* $IR_{\mathsf{subs\_row}} = \{\mathsf{subs\_row}(t_1,t_2) \mid t_1,t_2 \in \mathcal{T}(\Sigma,\mathcal{V})\}$*, then the DT (7) is strictly decreasing. Similarly,* $(\succ,\mathfrak{R}) \models (4)$*,* $(\succ,\mathfrak{R}) \models (5)$*, and* $(\succ,\mathfrak{R}) \models (8)$*.*

We can now formulate our second DT processor. To automate it, we refer to [15] for a description of how to synthesize valid interargument relations and how to find reduction pairs automatically that make DTs decreasing.

[10] This suffices, because we require $(\succsim, \succ)$ to be *rigid on* $\mathcal{C}$ in Thm. 19. Thus, $\succsim$ and $\succ$ do not take positions into account where atoms from $Call(\mathcal{P},\mathcal{S})$ have variables.

Theorem 19 (Reduction Pair Processor). *Let $(\succsim, \succ)$ be a reduction pair and let $\mathfrak{R}$ be a set of valid interargument relations. Then Proc is sound.*

$$Proc(\,(\mathcal{D},\mathcal{C},\mathcal{P})\,) = \begin{cases} \{(\mathcal{D}\setminus\mathcal{D}_\succ,\mathcal{C},\mathcal{P})\},\ \textit{if} \\ \ \bullet\ (\succsim,\succ)\ \textit{is rigid on}\ \mathcal{C}\ \textit{and} \\ \ \bullet\ \textit{there is}\ \mathcal{D}_\succ \subseteq \mathcal{D}\ \textit{with}\ \mathcal{D}_\succ \neq \varnothing\ \textit{such that}\ (\succ,\mathfrak{R}) \models c \\ \quad \textit{for all}\ c \in \mathcal{D}_\succ\ \textit{and}\ (\succsim,\mathfrak{R}) \models c\ \textit{for all}\ c \in \mathcal{D}\setminus\mathcal{D}_\succ \\ \{(\mathcal{D},\mathcal{C},\mathcal{P})\},\ \textit{otherwise} \end{cases}$$

Proof. If $Proc(\,(\mathcal{D},\mathcal{C},\mathcal{P})\,) = \{(\mathcal{D},\mathcal{C},\mathcal{P})\}$, then *Proc* is trivially sound. Now we consider the case $Proc(\,(\mathcal{D},\mathcal{C},\mathcal{P})\,) = \{(\mathcal{D}\setminus\mathcal{D}_\succ,\mathcal{C},\mathcal{P})\}$. Assume that $(\mathcal{D}\setminus\mathcal{D}_\succ,\mathcal{C},\mathcal{P})$ is terminating while $(\mathcal{D},\mathcal{C},\mathcal{P})$ is non-terminating. Then there is an infinite $(\mathcal{D},\mathcal{C},\mathcal{P})$-chain $(H_0 \leftarrow I_0, B_0), (H_1 \leftarrow I_1, B_1), \ldots$ where at least one clause from $\mathcal{D}_\succ$ appears infinitely often. There are $A \in \mathcal{C}$ and substitutions θ_i, σ_i such that $\theta_0 = mgu(A, H_0)$ and for all i, we have $\sigma_i \in Answer(\mathcal{P}, I_i\theta_i)$, $\theta_{i+1} = mgu(B_i\theta_i\sigma_i, H_{i+1})$, and $B_i\theta_i\sigma_i \in \mathcal{C}$. We obtain

$$\begin{array}{ll} H_i\theta_i & \\ \approx H_i\theta_i\sigma_i\theta_{i+1} & \text{(by rigidity, as } H_i\theta_i = B_{i-1}\theta_{i-1}\sigma_{i-1}\theta_i \\ & \text{and } B_{i-1}\theta_{i-1}\sigma_{i-1} \in \mathcal{C}) \\ \succsim B_i\theta_i\sigma_i\theta_{i+1} & \text{(since } (\succsim,\mathfrak{R}) \models c_i \text{ where } c_i \text{ is } H_i \leftarrow I_i, B_i, \\ & \text{as } (\succsim,\succ) \text{ is also rigid on any instance of } H_i\theta_i, \\ & \text{and since } \sigma_i \in Answer(\mathcal{P}, I_i\theta_i) \text{ implies } I_i\theta_i\sigma_i\theta_{i+1} \vdash_{\mathcal{P}}^{n} \Box \\ & \text{and } \mathfrak{R} \text{ are valid interargument relations)} \\ = H_{i+1}\theta_{i+1} & \text{(since } \theta_{i+1} = mgu(B_i\theta_i\sigma_i, H_{i+1})) \\ \approx H_{i+1}\theta_{i+1}\sigma_{i+1}\theta_{i+2} & \text{(by rigidity, as } H_{i+1}\theta_{i+1} = B_i\theta_i\sigma_i\theta_{i+1} \text{ and } B_i\theta_i\sigma_i \in \mathcal{C}) \\ \succsim B_{i+1}\theta_{i+1}\sigma_{i+1}\theta_{i+2} & \text{(since } (\succsim,\mathfrak{R}) \models c_{i+1} \text{ where } c_{i+1} \text{ is } H_{i+1} \leftarrow I_{i+1}, B_{i+1}) \\ = \ldots & \end{array}$$

Here, infinitely many $\succsim$-steps are "strict" (i.e., we can replace infinitely many $\succsim$-steps by $\succ$-steps). This contradicts the well-foundedness of $\succ$. ☐

So in our example, we apply the reduction pair processor to all 4 DT problems in Ex. 14. While we could use different reduction pairs for the different DT problems,[11] Ex. 18 showed that all their DTs are strictly decreasing for the reduction pair from Ex. 15. This reduction pair is indeed rigid on $Call(\mathcal{P},\mathcal{S})$. Hence, the reduction pair processor transforms all 4 remaining DT problems to $(\varnothing, Call(\mathcal{P},\mathcal{S}), \mathcal{P})$, which in turn is transformed to $\varnothing$ by the dependency graph processor. Thus, termination of the LP in Ex. 1 is proved.

4.3 Modular Transformation Processor to Term Rewriting

The previous two DT processors considerably improve over [15] due to their increased modularity.[12] In addition, one could easily adapt more techniques from

[11] Using different reduction pairs for different DT problems resulting from one and the same LP is for instance necessary for programs like the *Ackermann* function, cf. [15].

[12] In [15] these two processors were part of a fixed procedure, whereas now they can be applied to any DT problem at any time during the termination proof.

the DP framework (i.e., from the TRS setting) to the DT framework (i.e., to the LP setting). However, we now introduce a new DT processor which allows us to apply any TRS termination technique immediately to LPs (i.e., without having to adapt the TRS technique). It transforms a DT problem for LPs into a DP problem for TRSs.

Example 20. The following program $\mathcal{P}$ from [11] is part of the Termination Problem Data Base (TPDB) *used in the* International Termination Competition. *Typically,* cnf*'s first argument is a Boolean formula (where the function symbols* n, a, o *stand for the Boolean connectives) and the second is a variable which will be instantiated to an equivalent formula in conjunctive normal form. To this end,* cnf *uses the predicate* tr *which holds if its second argument results from its first one by a standard transformation step towards conjunctive normal form.*

$$
\begin{array}{ll}
\mathsf{cnf}(X,Y) \leftarrow \mathsf{tr}(X,Z), \mathsf{cnf}(Z,Y). & \mathsf{cnf}(X,X). \\
\mathsf{tr}(\mathsf{n}(\mathsf{n}(X)), X). & \mathsf{tr}(\mathsf{o}(X_1,Y), \mathsf{o}(X_2,Y)) \leftarrow \mathsf{tr}(X_1,X_2). \\
\mathsf{tr}(\mathsf{n}(\mathsf{a}(X,Y)), \mathsf{o}(\mathsf{n}(X), \mathsf{n}(Y))). & \mathsf{tr}(\mathsf{o}(X,Y1), \mathsf{o}(X,Y2)) \leftarrow \mathsf{tr}(Y1,Y2). \\
\mathsf{tr}(\mathsf{n}(\mathsf{o}(X,Y)), \mathsf{a}(\mathsf{n}(X), \mathsf{n}(Y))). & \mathsf{tr}(\mathsf{a}(X_1,Y), \mathsf{a}(X_2,Y)) \leftarrow \mathsf{tr}(X_1,X_2). \\
\mathsf{tr}(\mathsf{o}(X, \mathsf{a}(Y,Z)), \mathsf{a}(\mathsf{o}(X,Y), \mathsf{o}(X,Z))). & \mathsf{tr}(\mathsf{a}(X,Y1), \mathsf{a}(X,Y2)) \leftarrow \mathsf{tr}(Y1,Y2). \\
\mathsf{tr}(\mathsf{o}(\mathsf{a}(X,Y),Z), \mathsf{a}(\mathsf{o}(X,Z), \mathsf{o}(Y,Z))). & \mathsf{tr}(\mathsf{n}(X_1), \mathsf{n}(X_2)) \leftarrow \mathsf{tr}(X_1,X_2).
\end{array}
$$

Consider the queries $\mathcal{S} = \{\mathsf{cnf}(t_1,t_2) \mid t_1 \textit{ is ground}\} \cup \{\mathsf{tr}(t_1,t_2) \mid t_1 \textit{ is ground}\}$. *By applying the dependency graph processor to the initial DT problem, we obtain two new DT problems. The first is* $(\mathcal{D}_1, \mathit{Call}(\mathcal{P},\mathcal{S}), \mathcal{P})$ *where* $\mathcal{D}_1$ *contains all recursive* tr*-clauses. This DT problem can easily be solved by the reduction pair processor. The other resulting DT problem is*

$$(\{\mathsf{cnf}(X,Y) \leftarrow \mathsf{tr}(X,Z), \mathsf{cnf}(Z,Y)\}, \mathit{Call}(\mathcal{P},\mathcal{S}), \mathcal{P}). \tag{9}$$

To make this DT strictly decreasing, one needs a reduction pair $(\succsim, \succ)$ *where* $t_1 \succ t_2$ *holds whenever* $\mathsf{tr}(t_1,t_2)$ *is satisfied. This is impossible with the orders* $\succ$ *in current direct LP termination tools. In contrast, it would easily be possible if one uses other orders like the* recursive path order *[5] which is well established in term rewriting. This motivates the new processor presented in this section.*

To transform DT to DP problems, we adapt the existing transformation from logic programs $\mathcal{P}$ to TRSs $\mathcal{R}_\mathcal{P}$ from [17]. Here, two new n-ary function symbols p_{in} and p_{out} are introduced for each n-ary predicate p:

- Each fact $p(\boldsymbol{s})$ of the LP is transformed to the rewrite rule $p_{in}(\boldsymbol{s}) \to p_{out}(\boldsymbol{s})$.
- Each clause c of the form $p(\boldsymbol{s}) \leftarrow p_1(\boldsymbol{s}_1), \ldots, p_k(\boldsymbol{s}_k)$ is transformed into the following rewrite rules:

$$
\begin{aligned}
p_{in}(\boldsymbol{s}) &\to u_{c,1}(p_{1_{in}}(\boldsymbol{s}_1), \mathcal{V}(\boldsymbol{s})) \\
u_{c,1}(p_{1_{out}}(\boldsymbol{s}_1), \mathcal{V}(\boldsymbol{s})) &\to u_{c,2}(p_{2_{in}}(\boldsymbol{s}_2), \mathcal{V}(\boldsymbol{s}) \cup \mathcal{V}(\boldsymbol{s}_1)) \\
&\ldots \\
u_{c,k}(p_{k_{out}}(\boldsymbol{s}_k), \mathcal{V}(\boldsymbol{s}) \cup \mathcal{V}(\boldsymbol{s}_1) \cup \ldots \cup \mathcal{V}(\boldsymbol{s}_{k-1})) &\to p_{out}(\boldsymbol{s})
\end{aligned}
$$

  Here, the $u_{c,i}$ are new function symbols and $\mathcal{V}(\boldsymbol{s})$ are the variables in $\boldsymbol{s}$. Moreover, if $\mathcal{V}(\boldsymbol{s}) = \{x_1, \ldots, x_n\}$, then "$u_{c,1}(p_{1_{in}}(\boldsymbol{s}_1), \mathcal{V}(\boldsymbol{s}))$" abbreviates the term $u_{c,1}(p_{1_{in}}(\boldsymbol{s}_1), x_1, \ldots, x_n)$, etc.

So the fact $\mathsf{tr}(\mathsf{n}(\mathsf{n}(X)), X)$ is transformed to $\mathsf{tr}_{in}(\mathsf{n}(\mathsf{n}(X)), X) \to \mathsf{tr}_{out}(\mathsf{n}(\mathsf{n}(X)), X)$ and the clause $\mathsf{cnf}(X, Y) \leftarrow \mathsf{tr}(X, Z), \mathsf{cnf}(Z, Y)$ is transformed to

$$\mathsf{cnf}_{in}(X, Y) \to \mathsf{u}_1(\mathsf{tr}_{in}(X, Z), X, Y) \tag{10}$$

$$\mathsf{u}_1(\mathsf{tr}_{out}(X, Z), X, Y) \to \mathsf{u}_2(\mathsf{cnf}_{in}(Z, Y), X, Y, Z) \tag{11}$$

$$\mathsf{u}_2(\mathsf{cnf}_{out}(Z, Y), X, Y, Z) \to \mathsf{cnf}_{out}(X, Y) \tag{12}$$

To formulate the connection between a LP and its corresponding TRS, the sets of queries that should be analyzed for termination have to be represented by an *argument filter* π where $\pi(f) \subseteq \{1, \ldots, n\}$ for every n-ary $f \in \Sigma \cup \Delta$. We extend π to terms and atoms by defining $\pi(x) = x$ if x is a variable and $\pi(f(t_1, \ldots, t_n)) = f(\pi(t_{i_1}), \ldots, \pi(t_{i_k}))$ if $\pi(f) = \{i_1, \ldots, i_k\}$ with $i_1 < \ldots < i_k$.

Argument filters specify those positions which have to be instantiated with ground terms. In Ex. 20, we wanted to prove termination for the set $\mathcal{S}$ of all queries $\mathsf{cnf}(t_1, t_2)$ or $\mathsf{tr}(t_1, t_2)$ where t_1 is ground. These queries are described by the filter with $\pi(\mathsf{cnf}) = \pi(\mathsf{tr}) = \{1\}$. Hence, we can also represent $\mathcal{S}$ as $\mathcal{S} = \{A \mid A \in \mathcal{A}(\Sigma, \Delta, \mathcal{V}), \pi(A) \text{ is ground}\}$. Thm. 21 shows that instead of proving termination of a LP $\mathcal{P}$ for a set of queries $\mathcal{S}$, it suffices to prove termination of the corresponding TRS $\mathcal{R}_\mathcal{P}$ for a corresponding set of terms $\mathcal{S}'$. As shown in [17], here we have to regard a variant of term rewriting called *infinitary constructor rewriting*, where variables in rewrite rules may only be instantiated by *constructor terms*,[13] which however may be *infinite*. This is needed since LPs use unification, whereas TRSs use matching for their evaluation.

Theorem 21 (Soundness of the Transformation [17]). *Let $\mathcal{R}_\mathcal{P}$ be the TRS resulting from transforming a LP $\mathcal{P}$ over a signature (Σ, Δ). Let π be an argument filter with $\pi(p_{in}) = \pi(p)$ for all $p \in \Delta$. Let $\mathcal{S} = \{A \mid A \in \mathcal{A}(\Sigma, \Delta, \mathcal{V}),$ $\pi(A)$ is finite and ground$\}$ and $\mathcal{S}' = \{p_{in}(\boldsymbol{t}) \mid p(\boldsymbol{t}) \in \mathcal{S}\}$. If the TRS $\mathcal{R}_\mathcal{P}$ terminates for all terms in $\mathcal{S}'$, then the LP $\mathcal{P}$ terminates for all queries in $\mathcal{S}$.*

The DP framework for termination of term rewriting can also be used for infinitary constructor rewriting, cf. [17]. To this end, for each defined symbol f, one introduces a fresh *tuple symbol* $f^\sharp$ of the same arity. For a term $t = g(\boldsymbol{t})$ with defined root symbol g, let $t^\sharp$ denote $g^\sharp(\boldsymbol{t})$. Then the set of *dependency pairs* for a TRS $\mathcal{R}$ is $DP(\mathcal{R}) = \{\ell^\sharp \to t^\sharp \mid \ell \to r \in \mathcal{R}, t \text{ is a subterm of } r \text{ with defined root symbol}\}$. For instance, the rules (10) - (12) give rise to the following DPs.

$$\mathsf{cnf}^\sharp_{in}(X, Y) \to \mathsf{tr}^\sharp_{in}(X, Z) \tag{13}$$

$$\mathsf{cnf}^\sharp_{in}(X, Y) \to \mathsf{u}^\sharp_1(\mathsf{tr}_{in}(X, Z), X, Y) \tag{14}$$

$$\mathsf{u}^\sharp_1(\mathsf{tr}_{out}(X, Z), X, Y) \to \mathsf{cnf}^\sharp_{in}(Z, Y) \tag{15}$$

$$\mathsf{u}^\sharp_1(\mathsf{tr}_{out}(X, Z), X, Y) \to \mathsf{u}^\sharp_2(\mathsf{cnf}_{in}(Z, Y), X, Y, Z) \tag{16}$$

Termination problems are now represented as *DP problems* $(\mathcal{D}, \mathcal{R}, \pi)$ where $\mathcal{D}$ and $\mathcal{R}$ are TRSs (here, $\mathcal{D}$ is usually a set of DPs) and π is an argument filter. A

[13] As usual, the symbols on root positions of left-hand sides of rewrite rules are called *defined* symbols and all remaining function symbols are *constructors*. A *constructor term* is a term built only from constructors and variables.

list $s_1 \to t_1, s_2 \to t_2, \dots$ of variants from $\mathcal{D}$ is a $(\mathcal{D}, \mathcal{R}, \pi)$-*chain* iff for all i, there are substitutions σ_i such that $t_i\sigma_i$ rewrites to $s_{i+1}\sigma_{i+1}$ and such that $\pi(s_i\sigma_i)$, $\pi(t_i\sigma_i)$, and $\pi(q)$ are finite and ground, for all terms q in the reduction from $t_i\sigma_i$ and $s_{i+1}\sigma_{i+1}$. $(\mathcal{D}, \mathcal{R}, \pi)$ is *terminating* iff there is no infinite $(\mathcal{D}, \mathcal{R}, \pi)$-chain.

Example 22. For instance, "(14), (15)" is a chain for the argument filter π with $\pi(\mathsf{cnf}^\sharp_{in}) = \pi(\mathsf{tr}_{in}) = \{1\}$ and $\pi(u^\sharp_1) = \pi(\mathsf{tr}_{out}) = \{1, 2\}$. To see this, consider the substitution $\sigma = \{X/\mathsf{n}(\mathsf{n}(\mathsf{a})),\ Z/\mathsf{a}\}$. Now $\mathsf{u}^\sharp_1(\mathsf{tr}_{in}(X, Z), X, Y)\sigma$ reduces in one step to $\mathsf{u}^\sharp_1(\mathsf{tr}_{out}(X, Z), X, Y)\sigma$ and all instantiated left- and right-hand sides of (14) and (15) are ground after filtering them with π.

To prove termination of a TRS $\mathcal{R}$ for all terms $\mathcal{S}'$ in Thm. 21, now it suffices to show termination of the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, \pi)$. Here, one has to make sure that $\pi(DP(\mathcal{R}_\mathcal{P}))$ and $\pi(\mathcal{R}_\mathcal{P})$ satisfy the *variable condition*, i.e., that $\mathcal{V}(\pi(r)) \subseteq \mathcal{V}(\pi(\ell))$ holds for all $\ell \to r \in DP(\mathcal{R}) \cup \mathcal{R}$. If this does not hold, then π has to be refined (by filtering away more argument positions) until the variable condition is fulfilled. This leads to the following corollary from [17].

Corollary 23 (Transformation Technique [17]). *Let $\mathcal{R}_\mathcal{P}, \mathcal{P}, \pi$ be as in Thm. 21, where $\pi(p_{in}) = \pi(p^\sharp_{in}) = \pi(p)$ for all $p \in \Delta$. Let $\pi(DP(\mathcal{R}_\mathcal{P}))$ and $\pi(\mathcal{R}_\mathcal{P})$ satisfy the variable condition and let $\mathcal{S} = \{A \mid A \in \mathcal{A}(\Sigma, \Delta, \mathcal{V}), \pi(A)$ is finite and ground$\}$. If the DP problem $(DP(\mathcal{R}_\mathcal{P}), \mathcal{R}_\mathcal{P}, \pi)$ is terminating, then the LP $\mathcal{P}$ terminates for all queries in $\mathcal{S}$.*

Note that Thm. 21 and Cor. 23 are applied right at the beginning of the termination proof. So here one immediately transforms the full LP into a TRS (or a DP problem) and performs the whole termination proof on the TRS level. The disadvantage is that LP-specific techniques cannot be used anymore. It would be better to only apply this transformation for those parts of the termination proof where it is necessary and to perform most of the proof on the LP level.

This is achieved by the following new transformation processor within our DT framework. Now one can first apply other DT processors like the ones from Sect. 4.1 and 4.2 (or other LP termination techniques). Only for those subproblems where a solution cannot be found, one uses the following DT processor.

Theorem 24 (DT Transformation Processor). *Let $(\mathcal{D}, \mathcal{C}, \mathcal{P})$ be a DT problem and let π be an argument filter with $\pi(p_{in}) = \pi(p^\sharp_{in}) = \pi(p)$ for all predicates p such that $\mathcal{C} \subseteq \{A \mid A \in \mathcal{A}(\Sigma, \Delta, \mathcal{V}), \pi(A)$ is finite and ground$\}$ and such that $\pi(DP(\mathcal{R}_\mathcal{D}))$ and $\pi(\mathcal{R}_\mathcal{P})$ satisfy the variable condition. Then Proc is sound.*

$$Proc(\,(\mathcal{D}, \mathcal{C}, \mathcal{P})\,) = \begin{cases} \varnothing, & \textit{if } (DP(\mathcal{R}_\mathcal{D}), \mathcal{R}_\mathcal{P}, \pi) \textit{ is a terminating DP problem} \\ \{(\mathcal{D}, \mathcal{C}, \mathcal{P})\}, & \textit{otherwise} \end{cases}$$

Proof. If $Proc(\,(\mathcal{D}, \mathcal{C}, \mathcal{P})\,) = \{(\mathcal{D}, \mathcal{C}, \mathcal{P})\}$, then soundness is trivial. Now let $Proc(\,(\mathcal{D}, \mathcal{C}, \mathcal{P})\,) = \varnothing$. Assume there is an infinite $(\mathcal{D}, \mathcal{C}, \mathcal{P})$-chain $(H_0 \leftarrow I_0, B_0)$, $(H_1 \leftarrow I_1, B_1), \dots$ Similar to the proof of Thm. 6, we have

$$A \vdash_{H_0 \leftarrow I_0, B_0,\, \theta_0} I_0\theta_0, B_0\theta_0 \vdash^{n_0}_{\mathcal{P}, \sigma_0} B_0\theta_0\sigma_0 \vdash_{H_1 \leftarrow I_1, B_1,\, \theta_1} I_1\theta_1, B_1\theta_1 \vdash^{n_1}_{\mathcal{P}, \sigma_1} B_0\theta_1\sigma_1 \dots$$

For every atom $p(t_1, \ldots, t_n)$, let $\overline{p(t_1, \ldots, t_n)}$ be the term $p_{in}(t_1, \ldots, t_n)$. Then by the results on the correspondence between LPs and TRSs from [17] (in particular [17, Lemma 3.4]), we can conclude

$$\overline{A}\theta_0\sigma_0\,(\overset{\varepsilon}{\to}_{\mathcal{R}_\mathcal{D}} \circ \overset{>\varepsilon}{\to}{}^*_{\mathcal{R}_\mathcal{P}})^+\,\overline{B_0}\theta_0\sigma_0,\;\; \overline{B_0}\theta_0\sigma_0\theta_1\sigma_1\,(\overset{\varepsilon}{\to}_{\mathcal{R}_\mathcal{D}} \circ \overset{>\varepsilon}{\to}{}^*_{\mathcal{R}_\mathcal{P}})^+\,\overline{B_1}\theta_0\sigma_0\theta_1\sigma_1, \ldots$$

Here, $\to_\mathcal{R}$ denotes the rewrite relation of a TRS $\mathcal{R}$, $\overset{\varepsilon}{\to}$ resp. $\overset{>\varepsilon}{\to}$ denote reductions on resp. below the root position and $\to^*$ resp. $\to^+$ denote zero or more resp. one or more reduction steps. This implies

$$\overline{A}^\sharp\theta_0\sigma_0\,(\overset{\varepsilon}{\to}_{DP(\mathcal{R}_\mathcal{D})} \circ \overset{>\varepsilon}{\to}{}^*_{\mathcal{R}_\mathcal{P}})^+\,\overline{B_0}^\sharp\theta_0\sigma_0,\; \overline{B_0}^\sharp\theta_0\sigma_0\theta_1\sigma_1\,(\overset{\varepsilon}{\to}_{DP(\mathcal{R}_\mathcal{D})} \circ \overset{>\varepsilon}{\to}{}^*_{\mathcal{R}_\mathcal{P}})^+\,\overline{B_1}^\sharp\theta_0\sigma_0\theta_1\sigma_1,$$

etc. Let σ be the infinite substitution $\theta_0\sigma_0\theta_1\sigma_1\theta_2\sigma_2 \ldots$ where all remaining variables in σ's range can w.l.o.g. be replaced by ground terms. Then we have

$$\overline{A}^\sharp\sigma \quad (\overset{\varepsilon}{\to}_{DP(\mathcal{R}_\mathcal{D})} \circ \overset{>\varepsilon}{\to}{}^*_{\mathcal{R}_\mathcal{P}})^+ \quad \overline{B_0}^\sharp\sigma \quad (\overset{\varepsilon}{\to}_{DP(\mathcal{R}_\mathcal{D})} \circ \overset{>\varepsilon}{\to}{}^*_{\mathcal{R}_\mathcal{P}})^+ \quad \overline{B_1}^\sharp\sigma \ldots, \qquad (17)$$

which gives rise to an infinite $(DP(\mathcal{R}_\mathcal{D}), \mathcal{R}_\mathcal{P}, \pi)$-chain. To see this, note that $\pi(A)$ and all $\pi(B_i\theta_i\sigma_i)$ are finite and ground by the definition of chains of DTs. Hence, this also holds for $\pi(\overline{A}^\sharp\sigma)$ and all $\pi(\overline{B_i}^\sharp\sigma)$. Moreover, since $\pi(DP(\mathcal{R}_\mathcal{D}))$ and $\pi(\mathcal{R}_\mathcal{P})$ satisfy the variable condition, all terms occurring in the reduction (17) are finite and ground when filtering them with π. □

Example 25. We continue the termination proof of Ex. 20. Since the remaining DT problem (9) could not be solved by direct termination tools, we apply the DT processor of Thm. 24. Here, $\mathcal{R}_\mathcal{D} = \{(10), (11), (12)\}$ and hence, we obtain the DP problem $(\{(13), \ldots, (16)\}, \mathcal{R}_\mathcal{P}, \pi)$ where $\pi(\mathsf{cnf}) = \pi(\mathsf{tr}) = \{1\}$. On the other function symbols, π is defined as in Ex. 22 in order to fulfill the variable condition. This DP problem can easily be proved terminating by existing TRS techniques and tools, e.g., by using a recursive path order.

5 Experiments and Conclusion

We have introduced a new DT framework for termination analysis of LPs. It permits to split termination problems into subproblems, to use different orders for the termination proof of different subproblems, and to transform subproblems into termination problems for TRSs in order to apply existing TRS tools. In particular, it subsumes and improves upon recent direct and transformational approaches for LP termination analysis like [15, 17].

To evaluate our contributions, we performed extensive experiments comparing our new approach with the most powerful current direct and transformational tools for LP termination: Polytool [14] and AProVE [7].[14] The *International Termination Competition* showed that direct termination tools like Polytool and

[14] In [17], Polytool and AProVE were compared with three other representative tools for LP termination analysis: TerminWeb [4], cTI [12], and TALP [16]. Here, TerminWeb and cTI use a direct approach whereas TALP uses a transformational approach. In the experiments of [17], it turned out that Polytool and AProVE were considerably more powerful than the other three tools.

transformational tools like AProVE have comparable power, cf. Sect. 1. Nevertheless, there exist examples where one tool is successful, whereas the other fails.

For example, AProVE fails on the LP from Ex. 1. The reason is that by Cor. 23, it has to represent $\mathit{Call}(\mathcal{P}, \mathcal{S})$ by an argument filtering π which satisfies the variable condition. However, in this example there is no such argument filtering π where $(DP(\mathcal{R}_\mathcal{P}), \mathcal{P}, \pi)$ is terminating. In contrast, Polytool represents $\mathit{Call}(\mathcal{P}, \mathcal{S})$ by type graphs [10] and easily shows termination of this example.

On the other hand, Polytool fails on the LP from Ex. 20. Here, one needs orders like the recursive path order that are not available in direct termination tools. Indeed, other powerful direct termination tools such as TerminWeb [4] and cTI [12] fail on this example, too. The transformational tool TALP [16] fails on this program as well, as it does not use recursive path orders. In contrast, AProVE easily proves termination using a suitable recursive path order.

The results of this paper combine the advantages of direct and transformational approaches. We implemented our new approach in a new version of Polytool. Whenever the transformation processor of Thm. 24 is used, it calls AProVE on the resulting DP problem. Thus, we call our implementation "PolyAProVE".

In our experiments, we applied the two existing tools Polytool and AProVE as well as our new tool PolyAProVE to a set of 298 LPs. This set includes all LP examples of the TPDB that is used in the *International Termination Competition*. However, to eliminate the influence of the translation from Prolog to pure logic programs, we removed all examples that use non-trivial built-in predicates or that are not definite logic programs after ignoring the cut operator. This yields the same set of examples that was used in the experimental evaluation of [17]. In addition to this set we considered two more examples: the LP of Ex. 1 and the combination of Examples 1 and 20. For all examples, we used a time limit of 60 seconds corresponding to the standard setting of the competition.

Below, we give the results and the overall time (in seconds) required to run the tools on all 298 examples.

	PolyAProVE	AProVE	Polytool
Successes	**237**	232	218
Failures	**58**	58	73
Timeouts	**3**	8	7
Total Runtime	**762.3**	2227.2	588.8
Avg. Time	**2.6**	7.5	2.0

Our experiments show that PolyAProVE solves all examples that can be solved by Polytool or AProVE (including both LPs from Ex. 1 and 20). PolyAProVE also solves all examples from this collection that can be handled by any of the three other tools TerminWeb, cTI, and TALP. Moreover, it also succeeds on LPs whose termination could not be proved by any tool up to now. For example, it proves termination of the LP consisting of the clauses of both Ex. 1 and 20 together, whereas all other five tools fail. Another main advantage of PolyAProVE compared to powerful purely transformational tools like AProVE is a substantial increase in efficiency. PolyAProVE needs only about one third (34%) of the total runtime of AProVE. The reason is that many examples can already be handled by the direct techniques introduced in this paper. The transformation to term rewriting, which incurs a significant runtime penalty, is only used if the other DT processors fail. Thus, the performance of PolyAProVE is much closer to that of direct tools like Polytool than to that of transformational tools like AProVE.

For details on our experiments and to access our collection of examples, we refer to `http://aprove.informatik.rwth-aachen.de/eval/PolyAProVE/`.

References

1. K. R. Apt. *From Logic Programming to Prolog*. Prentice Hall, London, 1997.
2. T. Arts and J. Giesl. Termination of Term Rewriting using Dependency Pairs. *Theoretical Computer Science*, 236(1,2):133–178, 2000.
3. A. Bossi, N. Cocco, and M. Fabris. Norms on Terms and their use in Proving Universal Termination of a Logic Program. *Th. Comp. Sc.*, 124(2):297–328, 1994.
4. M. Codish and C. Taboch. A Semantic Basis for Termination Analysis of Logic Programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
5. N. Dershowitz. Termination of Rewriting. *J. Symb. Comp.*, 3(1,2):69–116, 1987.
6. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. LPAR '04*, LNAI 3452, pp. 301–331, 2005.
7. J. Giesl, P. Schneider-Kamp, R. Thiemann. AProVE 1.2: Automatic Termination Proofs in the DP Framework. In *Proc. IJCAR '06*, LNAI 4130, pp. 281–286, 2006.
8. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
9. N. Hirokawa and A. Middeldorp. Automating the Dependency Pair Method. *Information and Computation*, 199(1,2):172–199, 2005.
10. G. Janssens and M. Bruynooghe. Deriving Descriptions of Possible Values of Program Variables by Means of Abstract Interpretation. *Journal of Logic Programming*, 13(2,3):205–258, 1992.
11. M. Jurdzinski. LP Course Notes. `http://www.dcs.warwick.ac.uk/~mju/CS205/`.
12. F. Mesnard and R. Bagnara. cTI: A Constraint-Based Termination Inference Tool for ISO-Prolog. *Theory and Practice of Logic Programming*, 5(1, 2):243–257, 2005.
13. M. T. Nguyen and D. De Schreye. Polynomial Interpretations as a Basis for Termination Analysis of Logic Programs. *Proc. ICLP '05*, LNCS 3668, 311–325, 2005.
14. M. T. Nguyen and D. De Schreye. Polytool: Proving Termination Automatically Based on Polynomial Interpretations. In *Proc. LOPSTR '06*, LNCS 4407, pp. 210–218, 2007.
15. M. T. Nguyen, J. Giesl, P. Schneider-Kamp, and D. De Schreye. Termination Analysis of Logic Programs based on Dependency Graphs. In *Proc. LOPSTR '07*, LNCS 4915, pp. 8–22, 2008.
16. E. Ohlebusch, C. Claves, and C. Marché. TALP: A Tool for the Termination Analysis of Logic Programs. In *Proc. RTA '00*, LNCS 1833, pp. 270–273, 2000.
17. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Proofs for Logic Programs by Term Rewriting. *ACM Transactions on Computational Logic*, 11(1), 2009.