# Termination Analysis by Dependency Pairs and Inductive Theorem Proving*

S. Swiderski[1], M. Parting[1], J. Giesl[1], C. Fuhs[1], and P. Schneider-Kamp[2]

[1] LuFG Informatik 2, RWTH Aachen University, Germany
[2] Dept. of Mathematics & CS, University of Southern Denmark, Odense, Denmark

**Abstract.** Current techniques and tools for automated termination analysis of term rewrite systems (TRSs) are already very powerful. However, they fail for algorithms whose termination is essentially due to an *inductive* argument. Therefore, we show how to couple the *dependency pair* method for TRS termination with inductive theorem proving. As confirmed by the implementation of our new approach in the tool AProVE, now TRS termination techniques are also successful on this important class of algorithms.

## 1 Introduction

There are many powerful techniques and tools to prove termination of TRSs automatically. Moreover, TRS tools are also very successful in termination analysis of real programming languages like, e.g., Haskell and Prolog [12, 31]. To measure their performance, there is an annual *International Competition of Termination Provers*,[3] where the tools compete on a large data base of TRSs. Nevertheless, there exist natural algorithms like the following one where all these tools fail.

*Example 1. Consider the following TRS* $\mathcal{R}_{sort}$.

$$\begin{aligned}
\mathsf{ge}(x,\mathsf{0}) &\to \mathsf{true} & \mathsf{eq}(\mathsf{0},\mathsf{0}) &\to \mathsf{true}\\
\mathsf{ge}(\mathsf{0},\mathsf{s}(y)) &\to \mathsf{false} & \mathsf{eq}(\mathsf{s}(x),\mathsf{0}) &\to \mathsf{false}\\
\mathsf{ge}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{ge}(x,y) & \mathsf{eq}(\mathsf{0},\mathsf{s}(y)) &\to \mathsf{false}\\
& & \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{eq}(x,y)\\
\mathsf{max}(\mathsf{nil}) &\to \mathsf{0} & & \\
\mathsf{max}(\mathsf{co}(x,\mathsf{nil})) &\to x & \mathsf{if}_1(\mathsf{true},x,y,xs) &\to \mathsf{max}(\mathsf{co}(x,xs))\\
\mathsf{max}(\mathsf{co}(x,\mathsf{co}(y,xs))) &\to \mathsf{if}_1(\mathsf{ge}(x,y),x,y,xs) & \mathsf{if}_1(\mathsf{false},x,y,xs) &\to \mathsf{max}(\mathsf{co}(y,xs))\\
\mathsf{del}(x,\mathsf{nil}) &\to \mathsf{nil} & \mathsf{if}_2(\mathsf{true},x,y,xs) &\to xs\\
\mathsf{del}(x,\mathsf{co}(y,xs)) &\to \mathsf{if}_2(\mathsf{eq}(x,y),x,y,xs) & \mathsf{if}_2(\mathsf{false},x,y,xs) &\to \mathsf{co}(y,\mathsf{del}(x,xs))\\
\mathsf{sort}(\mathsf{nil}) &\to \mathsf{nil} & & \\
\mathsf{sort}(\mathsf{co}(x,xs)) &\to \mathsf{co}(\,\mathsf{max}(\mathsf{co}(x,xs)),\ \mathsf{sort}(\mathsf{del}(\,\mathsf{max}(\mathsf{co}(x,xs)),\ \mathsf{co}(x,xs)\,))\,) & &
\end{aligned}$$

*Here, numbers are represented with* 0 *and* s *(for the successor function) and lists are represented with* nil *(for the empty list) and* co *(for list insertion). For any list* $xs$, max($xs$) *computes its maximum (where* max(nil) *is* 0*), and* del($n, xs$) *deletes*

* In *Proc. CADE'09*, LNAI, 2009. Supported by the DFG Research Training Group 1298 (*AlgoSyn*), the DFG grant GI 274/5-2, and the G.I.F. grant 966-116.6.

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

*the first occurrence of $n$ from the list $xs$. If $n$ does not occur in $xs$, then* $\mathsf{del}(n, xs)$ *returns $xs$. Algorithms like* max *and* del *are often expressed with conditions. Such conditional rules can be automatically transformed into unconditional ones (cf. e.g. [27]) and we already did this transformation in our example. To sort a non-empty list $ys$ (i.e., a list of the form "*$\mathsf{co}(x, xs)$*"),* $\mathsf{sort}(ys)$ *reduces to "*$\mathsf{co}(\mathsf{max}(ys), \mathsf{sort}(\mathsf{del}(\mathsf{max}(ys), ys))$*". So* $\mathsf{sort}(ys)$ *starts with the maximum of $ys$ and then* sort *is called recursively on the list that results from $ys$ by deleting the first occurrence of its maximum. Note that*

$$\textit{every non-empty list contains its maximum.} \tag{1}$$

*Hence, the list* $\mathsf{del}(\mathsf{max}(ys), ys)$ *is shorter than $ys$ and thus, $\mathcal{R}_{sort}$ is terminating.*

So (1) is the main argument needed for termination of $\mathcal{R}_{sort}$. Thus, when trying to prove termination of TRSs like $\mathcal{R}_{sort}$ automatically, one faces 2 problems:

(a) One has to detect the main argument needed for termination and one has to find out that the TRS is terminating provided that this argument is valid.
(b) One has to prove that the argument detected in (a) is valid.

In our example, (1) requires a non-trivial induction proof that relies on the max- and del-rules. Such proofs cannot be done by TRS termination techniques, but they could be performed by state-of-the-art inductive theorem provers [4, 5, 7, 8, 20, 21, 33, 34, 36]. So to solve Problem (b), we would like to couple termination techniques for TRSs (like the *dependency pair* (DP) method which is implemented in virtually every current TRS termination tool) with an inductive theorem prover. Ideally, this prover should perform the validity proof in (b) fully automatically, but of course it is also possible to have user interaction here. However, it still remains to solve Problem (a). Thus, one has to extend the TRS termination techniques such that they can automatically synthesize an argument like (1) and find out that this argument is sufficient in order to complete the termination proof. This is the subject of the current paper.

There is already work on applying inductive reasoning in termination proofs. Some approaches like [6, 15, 16, 28] integrate special forms of inductive reasoning into the termination method itself. These approaches are successful on certain forms of algorithms, but they cannot handle examples like Ex. 1 where one needs more general forms of inductive reasoning. Therefore, in this paper our goal is to couple the termination method with an arbitrary (black-box) inductive theorem prover which may use any kind of proof techniques.

There exist also approaches like [5, 10, 22, 25, 32] where a full inductive theorem prover is used to perform the whole termination proof of a functional program. Such approaches could potentially handle algorithms like Ex. 1 and indeed, Ex. 1 is similar to an algorithm from [10, 32]. In general, to prove termination one has to solve two tasks: (i) one has to synthesize suitable well-founded orders and (ii) one has to prove that recursive calls decrease w.r.t. these orders. If there is just an inductive theorem prover available for the termination proof, then for Task (i) one can only use a fixed small set of orders or otherwise, ask

the user to provide suitable well-founded orders manually. Moreover, then Task (ii) has to be tackled by the full theorem prover which may often pose problems for automation. In contrast, there are many TRS techniques and tools available that are extremely powerful for Task (i) and that offer several specialized methods to perform Task (ii) fully automatically in a very efficient way. So in most cases, no inductive theorem prover is needed for Task (ii). Nevertheless, there exist important algorithms (like $\mathcal{R}_{sort}$) where Task (ii) indeed requires inductive theorem proving. Thus, we propose to use the "best of both worlds", i.e., to apply TRS techniques whenever possible, but to use an inductive theorem prover for those parts where it is needed.

After recapitulating the DP method in Sect. 2, in Sect. 3 we present the main idea for our improvement. To make this improvement powerful in practice, we need the new result that innermost termination of many-sorted term rewriting and of unsorted term rewriting is equivalent. We expect that this observation will be useful also for other applications in term rewriting, since TRSs are usually considered to be unsorted. We use this result in Sect. 4 where we show how the DP method can be coupled with inductive theorem proving in order to prove termination of TRSs like $\mathcal{R}_{sort}$ automatically.

We implemented our new technique in the termination prover AProVE [13]. Here, we used a small inductive theorem prover inspired by [5, 7, 21, 33, 34, 36] which had already been implemented in AProVE before. Although this inductive theorem prover is less powerful than the more elaborated full theorem provers in the literature, it suffices for many of those inductive arguments that typically arise in termination proofs. This is confirmed by the experimental evaluation of our contributions in Sect. 5. Note that the results of this paper allow to couple *any* termination prover implementing DPs with *any* inductive theorem prover. Thus, by using a more powerful inductive theorem prover than the one integrated in AProVE, the power of the resulting tool could even be increased further.

## 2 Dependency Pairs

We assume familiarity with term rewriting [3] and briefly recapitulate the DP method. See e.g. [2, 11, 14, 18, 19] for further motivations and extensions.

**Definition 2 (Dependency Pairs).** *For a TRS $\mathcal{R}$, the* defined *symbols $\mathcal{D}_\mathcal{R}$ are the root symbols of left-hand sides of rules. All other function symbols are called* constructors*. For every defined symbol $f \in \mathcal{D}_\mathcal{R}$, we introduce a fresh* tuple symbol *$f^\sharp$ with the same arity. To ease readability, we often write $F$ instead of $f^\sharp$, etc. If $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{D}_\mathcal{R}$, we write $t^\sharp$ for $f^\sharp(t_1, \ldots, t_n)$. If $\ell \to r \in \mathcal{R}$ and $t$ is a subterm of $r$ with defined root symbol, then the rule $\ell^\sharp \to t^\sharp$ is a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted $DP(\mathcal{R})$.*

We get the following set $DP(\mathcal{R}_{sort})$, where GE is ge's tuple symbol, etc.

$$\mathsf{GE}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{GE}(x, y) \tag{2}$$
$$\mathsf{EQ}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{EQ}(x, y) \tag{3}$$
$$\mathsf{MAX}(\mathsf{co}(x, \mathsf{co}(y, xs))) \to \mathsf{IF}_1(\mathsf{ge}(x, y), x, y, xs) \tag{4}$$
$$\mathsf{MAX}(\mathsf{co}(x, \mathsf{co}(y, xs))) \to \mathsf{GE}(x, y) \tag{5}$$

$$\mathsf{IF}_1(\mathsf{true}, x, y, xs) \to \mathsf{MAX}(\mathsf{co}(x, xs)) \quad (6)$$
$$\mathsf{IF}_1(\mathsf{false}, x, y, xs) \to \mathsf{MAX}(\mathsf{co}(y, xs)) \quad (7)$$
$$\mathsf{DEL}(x, \mathsf{co}(y, xs)) \to \mathsf{IF}_2(\mathsf{eq}(x, y), x, y, xs) \quad (8)$$
$$\mathsf{DEL}(x, \mathsf{co}(y, xs)) \to \mathsf{EQ}(x, y) \quad (9)$$
$$\mathsf{IF}_2(\mathsf{false}, x, y, xs) \to \mathsf{DEL}(x, xs) \quad (10)$$
$$\mathsf{SORT}(\mathsf{co}(x, xs)) \to \mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x, xs)), \mathsf{co}(x, xs))) \quad (11)$$
$$\mathsf{SORT}(\mathsf{co}(x, xs)) \to \mathsf{DEL}(\mathsf{max}(\mathsf{co}(x, xs)), \mathsf{co}(x, xs)) \quad (12)$$
$$\mathsf{SORT}(\mathsf{co}(x, xs)) \to \mathsf{MAX}(\mathsf{co}(x, xs)) \quad (13)$$

In this paper, we only regard the *innermost* rewrite relation $\xrightarrow{\mathsf{i}}$ and prove innermost termination, since techniques for innermost termination are considerably more powerful than those for full termination. For large classes of TRSs (e.g., TRSs resulting from programming languages [12, 31] or non-overlapping TRSs like Ex. 1), innermost termination is sufficient for termination.

For 2 TRSs $\mathcal{P}$ and $\mathcal{R}$ (where $\mathcal{P}$ usually consists of DPs), an *innermost* $(\mathcal{P}, \mathcal{R})$-*chain* is a sequence of (variable-renamed) pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ such that there is a substitution $\sigma$ (with possibly infinite domain) where $t_i\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{*} s_{i+1}\sigma$ and $s_i\sigma$ is in normal form w.r.t. $\mathcal{R}$, for all $i$.[4] The main result on DPs states that $\mathcal{R}$ is innermost terminating iff there is no infinite innermost $(DP(\mathcal{R}), \mathcal{R})$-chain.

As an example for a chain, consider "(11), (11)", i.e.,

$$\mathsf{SORT}(\mathsf{co}(x\ , xs\ )) \to \mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x\ , xs\ )), \mathsf{co}(x\ , xs\ ))),$$
$$\mathsf{SORT}(\mathsf{co}(x', xs')) \to \mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x', xs')), \mathsf{co}(x', xs'))).$$

Indeed, if $\sigma(x) = \sigma(x') = \mathsf{0}$, $\sigma(xs) = \mathsf{co}(\mathsf{s}(\mathsf{0}), \mathsf{nil})$, and $\sigma(xs') = \mathsf{nil}$, then

$$\mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x, xs)), \mathsf{co}(x, xs)))\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}_{sort}}^{*} \mathsf{SORT}(\mathsf{co}(x', xs'))\sigma.$$

Termination techniques are now called *DP processors* and they operate on pairs of TRSs $(\mathcal{P}, \mathcal{R})$ (which are called *DP problems*).[5] Formally, a DP processor *Proc* takes a DP problem as input and returns a set of new DP problems which then have to be solved instead. A processor *Proc* is *sound* if for all DP problems $(\mathcal{P}, \mathcal{R})$ with an infinite innermost $(\mathcal{P}, \mathcal{R})$-chain there is also a $(\mathcal{P}', \mathcal{R}') \in \mathit{Proc}(\,(\mathcal{P}, \mathcal{R})\,)$ with an infinite innermost $(\mathcal{P}', \mathcal{R}')$-chain. Soundness of a DP processor is required to prove innermost termination and in particular, to conclude that there is no infinite innermost $(\mathcal{P}, \mathcal{R})$-chain if $\mathit{Proc}(\,(\mathcal{P}, \mathcal{R})\,) = \varnothing$.

So innermost termination proofs in the DP framework start with the initial problem $(DP(\mathcal{R}), \mathcal{R})$. Then the problem is simplified repeatedly by sound DP processors. If all DP problems have been simplified to $\varnothing$, then innermost termination is proved. Thm. 3-5 recapitulate three of the most important processors.

Thm. 3 allows us to replace the TRS $\mathcal{R}$ in a DP problem $(\mathcal{P}, \mathcal{R})$ by the *usable rules*. These include all rules that can be used to reduce the terms in right-hand sides of $\mathcal{P}$ when their variables are instantiated with normal forms.

**Theorem 3 (Usable Rule Processor [2, 11]).** *Let $\mathcal{R}$ be a TRS. For any function symbol $f$, let $\mathit{Rls}(f) = \{\ell \to r \in \mathcal{R} \mid \mathrm{root}(\ell) = f\}$. For any term $t$, the*

[4] All results of the present paper also hold if one regards *minimal* instead of ordinary innermost chains, i.e., chains where all $t_i\sigma$ are innermost terminating.

[5] To ease readability we use a simpler definition of *DP problems* than [11], since this simple definition suffices for the presentation of the new results of this paper.

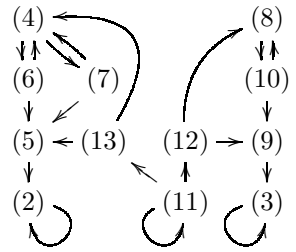
usable rules $\mathcal{U}(t)$ *are the smallest set such that*

- $\mathcal{U}(x) = \varnothing$ *for every variable* $x$ *and*
- $\mathcal{U}(f(t_1, \ldots, t_n)) = Rls(f) \cup \bigcup_{\ell \to r \in Rls(f)} \mathcal{U}(r) \cup \bigcup_{i=1}^{n} \mathcal{U}(t_i)$

*For a TRS* $\mathcal{P}$*, its* usable rules *are* $\mathcal{U}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t)$*. Then the following DP processor Proc is sound:* $Proc(\,(\mathcal{P}, \mathcal{R})\,) = \{\,(\mathcal{P}, \mathcal{U}(\mathcal{P}))\,\}$.

In Ex. 1, this processor transforms the initial DP problem $(DP(\mathcal{R}_{sort}), \mathcal{R}_{sort})$ into $(DP(\mathcal{R}_{sort}), \mathcal{R}'_{sort})$. $\mathcal{R}'_{sort}$ is $\mathcal{R}_{sort}$ without the two sort-rules, since sort does not occur in the right-hand side of any DP and thus, its rules are not usable.

The next processor decomposes a DP problem into several sub-problems. To this end, one determines which pairs can follow each other in innermost chains by constructing an *innermost dependency graph*. For a DP problem $(\mathcal{P}, \mathcal{R})$, the nodes of the innermost dependency graph are the pairs of $\mathcal{P}$, and there is an arc from $s \to t$ to $v \to w$ iff $s \to t, v \to w$ is an innermost $(\mathcal{P}, \mathcal{R})$-chain. The graph obtained in our example is depicted on the side.




In general, the innermost dependency graph is not computable, but there exist many techniques to over-approximate this graph automatically, cf. e.g. [2, 18]. In our example, these estimations would even yield the exact innermost dependency graph.

A set $\mathcal{P}' \neq \varnothing$ of DPs is a *cycle* if for every $s \to t$, $v \to w \in \mathcal{P}'$, there is a non-empty path from $s \to t$ to $v \to w$ traversing only pairs of $\mathcal{P}'$. A cycle $\mathcal{P}'$ is a (non-trivial) *strongly connected component (SCC)* if $\mathcal{P}'$ is not a proper subset of another cycle. The next processor allows us to prove termination separately for each SCC.

**Theorem 4 (Dependency Graph Processor [2, 11]).** *The following DP processor Proc is sound:* $Proc(\,(\mathcal{P}, \mathcal{R})\,) = \{(\mathcal{P}_1, \mathcal{R}), \ldots, (\mathcal{P}_n, \mathcal{R})\}$*, where* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *are the SCCs of the innermost dependency graph.*

Our graph has the SCCs $\mathcal{P}_1 = \{(2)\}$, $\mathcal{P}_2 = \{(3)\}$, $\mathcal{P}_3 = \{(4), (6), (7)\}$, $\mathcal{P}_4 = \{(8), (10)\}$, $\mathcal{P}_5 = \{(11)\}$. Thus, $(DP(\mathcal{R}_{sort}), \mathcal{R}'_{sort})$ is transformed into the 5 new DP problems $(\mathcal{P}_i, \mathcal{R}'_{sort})$ for $1 \leq i \leq 5$ that have to be solved instead. For all problems except $(\{(11)\}, \mathcal{R}'_{sort})$ this is easily possible by the DP processors of this section (and this can also be done automatically by current termination tools). Therefore, we now concentrate on the remaining DP problem $(\{(11)\}, \mathcal{R}'_{sort})$.

A *reduction pair* $(\succsim, \succ)$ consists of a stable monotonic quasi-order $\succsim$ and a stable well-founded order $\succ$, where $\succsim$ and $\succ$ are compatible (i.e., $\succsim \circ \succ \circ \succsim \subseteq \succ$). For a DP problem $(\mathcal{P}, \mathcal{R})$, the following processor requires that all DPs in $\mathcal{P}$ are strictly or weakly decreasing and all rules $\mathcal{R}$ are weakly decreasing. Then one can delete all strictly decreasing DPs. Note that both TRSs and relations can be seen as sets of pairs of terms. Thus, $\mathcal{P} \setminus \succ$ denotes $\{s \to t \in \mathcal{P} \mid s \not\succ t\}$.

**Theorem 5 (Reduction Pair Processor [2, 11, 18]).** *Let* $(\succsim, \succ)$ *be a reduction pair. Then the following DP processor Proc is sound.*

$$Proc(\,(\mathcal{P}, \mathcal{R})\,) = \begin{cases} \{\,(\mathcal{P} \setminus \succ, \mathcal{R})\,\}, & \text{if } \mathcal{P} \subseteq \succsim \cup \succ \text{ and } \mathcal{R} \subseteq \succsim \\ \{\,(\mathcal{P}, \mathcal{R})\,\}, & \text{otherwise} \end{cases}$$

For the problem $(\{(11)\}, \mathcal{R}'_{sort})$, we search for a reduction pair where (11) is strictly decreasing (w.r.t. $\succ$) and the rules in $\mathcal{R}'_{sort}$ are weakly decreasing (w.r.t. $\succsim$). However, this is not satisfied by the orders available in current termination tools. That is not surprising, because termination of this DP problem essentially relies on the argument (1) that every non-empty list contains its maximum.

## 3 Many-Sorted Rewriting

Recall that our goal is to prove the absence of infinite innermost $(\mathcal{P}, \mathcal{R})$-chains. Each such chain would correspond to a reduction of the following form

$$s_1\sigma \;\to_{\mathcal{P}}\; t_1\sigma \;\overset{\mathsf{i}}{\to}^!_{\mathcal{R}}\; s_2\sigma \;\to_{\mathcal{P}}\; t_2\sigma \;\overset{\mathsf{i}}{\to}^!_{\mathcal{R}}\; s_3\sigma \;\to_{\mathcal{P}}\; t_3\sigma \;\overset{\mathsf{i}}{\to}^!_{\mathcal{R}}\; \dots$$

where $s_i \to t_i$ are variable-renamed DPs from $\mathcal{P}$ and "$\overset{\mathsf{i}}{\to}^!_{\mathcal{R}}$" denotes zero or more reduction steps to a normal form. The reduction pair processor ensures

$$s_1\sigma \;{}_{(}\!\succsim\!{}_{)}\; t_1\sigma \;\succsim\; s_2\sigma \;{}_{(}\!\succsim\!{}_{)}\; t_2\sigma \;\succsim\; s_3\sigma \;{}_{(}\!\succsim\!{}_{)}\; t_3\sigma \;\succsim\; \dots$$

Hence, strictly decreasing DPs (i.e., where $s_i\sigma \succ t_i\sigma$) cannot occur infinitely often in innermost chains and thus, they can be removed from the DP problem.

However, instead of requiring a strict decrease when going from the left-hand side $s_i\sigma$ of a DP to the right-hand side $t_i\sigma$, it would also be sufficient to require a strict decrease when going from the right-hand side $t_i\sigma$ to the *next* left-hand side $s_{i+1}\sigma$. In other words, if every reduction of $t_i\sigma$ to normal form makes the term strictly smaller w.r.t. $\succ$, then we would have $t_i\sigma \succ s_{i+1}\sigma$. Hence, then the DP $s_i \to t_i$ cannot occur infinitely often and could be removed from the DP problem. Our goal is to formulate a new processor based on this idea.

So essentially, we can remove a DP $s \to t$ from the DP problem, if

$$\text{for every normal substitution } \sigma,\ t\sigma \overset{\mathsf{i}}{\to}^!_{\mathcal{R}} q \text{ implies } t\sigma \succ q. \tag{14}$$

In addition, all DPs and rules still have to be weakly decreasing. A substitution $\sigma$ is called *normal* iff $\sigma(x)$ is in normal form w.r.t. $\mathcal{R}$ for all variables $x$.

So to remove (11) from the remaining DP problem $(\{(11)\}, \mathcal{R}'_{sort})$ of Ex. 1 with the criterion above, we have to use a reduction pair satisfying (14). Here, $t$ is the right-hand side of (11), i.e., $t = \mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x, xs)), \mathsf{co}(x, xs)))$.

Now we will weaken the requirement (14) step by step to obtain a condition amenable to automation. The current requirement (14) is still unnecessarily hard. For instance, in our example we also have to regard substitutions like $\sigma(x) = \sigma(xs) = \mathsf{true}$ and require that $t\sigma \succ q$ holds, although intuitively, here $x$ stands for a natural number and $xs$ stands for a list (and not a Boolean value). We will show that one does not have to require (14) for *all* normal substitutions, but only for "well-typed" ones. The reason is that if there is an infinite innermost reduction, then there is also an infinite innermost reduction of "well-typed" terms.

First, we make precise what we mean by "well-typed". Recall that up to now we regarded ordinary TRSs over untyped signatures $\mathcal{F}$. The following definition shows how to extend such signatures by (monomorphic) types, cf. e.g. [35].

**Definition 6 (Typing).** *Let $\mathcal{F}$ be an (untyped) signature. A many-sorted signature $\mathcal{F}'$ is a* typed variant *of $\mathcal{F}$ if it contains the same function symbols as $\mathcal{F}$,*

*with the same arities. So $f$ is a symbol of $\mathcal{F}$ with arity $n$ iff $f$ is a symbol of $\mathcal{F}'$ with a type of the form $\tau_1 \times \ldots \times \tau_n \to \tau$. Similarly, a typed variant $\mathcal{V}'$ of the set of variables $\mathcal{V}$ contains the same variables as $\mathcal{V}$, but now every variable has a sort $\tau$. We always assume that for every sort $\tau$, $\mathcal{V}'$ contains infinitely many variables of sort $\tau$. A term over $\mathcal{F}$ and $\mathcal{V}$ is* well typed *w.r.t. $\mathcal{F}'$ and $\mathcal{V}'$ iff*

- *$t$ is a variable (of some type $\tau$ in $\mathcal{V}'$) or*
- *$t = f(t_1, \ldots, t_n)$ with $n \geq 0$, where all $t_i$ are well typed and have some type $\tau_i$, and where $f$ has type $\tau_1 \times \ldots \times \tau_n \to \tau$ in $\mathcal{F}'$. Then $t$ has type $\tau$.*

*We only permit typed variants $\mathcal{F}'$ where there exist well-typed ground terms of types $\tau_1, \ldots, \tau_n$ over $\mathcal{F}'$, whenever some $f \in \mathcal{F}'$ has type $\tau_1 \times \ldots \times \tau_n \to \tau$.*[6]

*A TRS $\mathcal{R}$ over*[7] *$\mathcal{F}$ and $\mathcal{V}$ is* well typed *w.r.t. $\mathcal{F}'$ and $\mathcal{V}'$ if for all $\ell \to r \in \mathcal{R}$, we have that $\ell$ and $r$ are well typed and that they have the same type.*[8]

For any TRS $\mathcal{R}$ over a signature $\mathcal{F}$, one can use a standard type inference algorithm to compute a typed variant $\mathcal{F}'$ of $\mathcal{F}$ automatically such that $\mathcal{R}$ is well typed. Of course, a trivial solution is to use a many-sorted signature with just one sort (then every term and every TRS are trivially well typed). But to make our approach more powerful, it is advantageous to use the most general typed variant where $\mathcal{R}$ is well typed instead. Here, the set of terms is decomposed into as many sorts as possible. Then fewer terms are considered to be "well typed" and hence, the condition (14) has to be required for fewer substitutions $\sigma$.

For example, let $\mathcal{F} = \{\mathsf{0}, \mathsf{s}, \mathsf{true}, \mathsf{false}, \mathsf{nil}, \mathsf{co}, \mathsf{ge}, \mathsf{eq}, \mathsf{max}, \mathsf{if}_1, \mathsf{del}, \mathsf{if}_2, \mathsf{SORT}\}$. To make $\{(11)\} \cup \mathcal{R}'_{sort}$ well typed, we obtain the typed variant $\mathcal{F}'$ of $\mathcal{F}$ with the sorts $\mathsf{nat}$, $\mathsf{bool}$, $\mathsf{list}$, and $\mathsf{tuple}$. Here the function symbols have the following types.

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{0} & : & \mathsf{nat} & \mathsf{ge}, \mathsf{eq} & : & \mathsf{nat} \times \mathsf{nat} \to \mathsf{bool} \\
\mathsf{s} & : & \mathsf{nat} \to \mathsf{nat} & \mathsf{max} & : & \mathsf{list} \to \mathsf{nat} \\
\mathsf{true}, \mathsf{false} & : & \mathsf{bool} & \mathsf{if}_1, \mathsf{if}_2 & : & \mathsf{bool} \times \mathsf{nat} \times \mathsf{nat} \times \mathsf{list} \to \mathsf{list} \\
\mathsf{nil} & : & \mathsf{list} & \mathsf{SORT} & : & \mathsf{list} \to \mathsf{tuple} \\
\mathsf{co}, \mathsf{del} & : & \mathsf{nat} \times \mathsf{list} \to \mathsf{list} & & &
\end{array}$$

Now we show that innermost termination is a *persistent* property, i.e., a TRS is innermost terminating iff it is innermost terminating on well-typed terms. Here, one can use any typed variant where the TRS is well typed. As noted by [26], persistence of innermost termination follows from results of [30], but to our knowledge, it has never been explicitly stated or applied in the literature before. Note that in contrast to innermost termination, full termination is only persistent for very restricted classes of TRSs, cf. [35].

**Theorem 7 (Persistence).** *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{V}$ and let $\mathcal{R}$ be well typed w.r.t. the typed variants $\mathcal{F}'$ and $\mathcal{V}'$. $\mathcal{R}$ is innermost terminating for all well-typed terms w.r.t. $\mathcal{F}'$ and $\mathcal{V}'$ iff $\mathcal{R}$ is innermost terminating (for* all *terms).*

[6] This is not a restriction, as one can simply add new constants to $\mathcal{F}$ and $\mathcal{F}'$.

[7] Note that $\mathcal{F}$ may well contain function symbols that do not occur in $\mathcal{R}$.

[8] W.l.o.g., here one may rename the variables in every rule. Then it is not a problem if the variable $x$ is used with type $\tau_1$ in one rule and with type $\tau_2$ in another rule.

*Proof.* For persistence, it suffices to show *component closedness* and *sorted modularity* [30]. A property is *component closed* if (a) $\Leftrightarrow$ (b) holds for all TRSs $\mathcal{R}$.

(a) $\to_{\mathcal{R}}$ has the property for all terms
(b) for every equivalence class $Cl$ w.r.t. $\leftrightarrow^*_{\mathcal{R}}$, the restriction of $\to_{\mathcal{R}}$ to $Cl$ has the property

Innermost termination is clearly component closed, since all terms occurring in an innermost reduction are from the same equivalence class.

A property is *sorted modular* if (c) and (d) are equivalent for all TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ forming a disjoint combination. So each $\mathcal{R}_i$ is a TRS over $\mathcal{F}_i$ and $\mathcal{V}$, $\mathcal{F}'_i$ and $\mathcal{V}'$ are typed variants of $\mathcal{F}_i$ and $\mathcal{V}$ where $\mathcal{R}_i$ is well typed, and $\mathcal{F}_1 \cap \mathcal{F}_2 = \varnothing$.

(c) for both $i$, $\mathcal{R}_i$ has the property for all well-typed terms w.r.t. $\mathcal{F}'_i$ and $\mathcal{V}'$
(d) $\mathcal{R}_1 \cup \mathcal{R}_2$ has the property for all well-typed terms w.r.t. $\mathcal{F}'_1 \cup \mathcal{F}'_2$ and $\mathcal{V}'$

For innermost termination, (d) $\Rightarrow$ (c) is trivial. To show (c) $\Rightarrow$ (d), we adapt the proof for (unsorted) modularity of innermost termination in [17]. Assume there is a well-typed term $t$ over $\mathcal{F}'_1 \cup \mathcal{F}'_2$ and $\mathcal{V}'$ with infinite innermost $\mathcal{R}_1 \cup \mathcal{R}_2$-reduction. Then there is also a minimal such term (its proper subterms are innermost terminating w.r.t. $\mathcal{R}_1 \cup \mathcal{R}_2$). The reduction has the form $t \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}_1 \cup \mathcal{R}_2} t_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}_1 \cup \mathcal{R}_2} t_2 \xrightarrow{\mathrm{i}}_{\mathcal{R}_1 \cup \mathcal{R}_2} \dots$ where the step from $t_1$ to $t_2$ is the first root step. Such a root step must exist since $t$ is minimal. Due to the innermost strategy, all proper subterms of $t_1$ are in $\mathcal{R}_1 \cup \mathcal{R}_2$-normal form. W.l.o.g., let $\mathrm{root}(t_1) \in \mathcal{F}_1$. Then $t_1 = C[s_1, \dots, s_m]$ with $m \geq 0$, where $C$ is a context without symbols from $\mathcal{F}_2$ and the roots of $s_1, \dots, s_m$ are from $\mathcal{F}_2$. Since $s_1, \dots, s_m$ are irreducible, the reduction from $t_1$ onwards is an $\mathcal{R}_1$-reduction, i.e., $t_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}_1} t_2 \xrightarrow{\mathrm{i}}_{\mathcal{R}_1} \dots$ Let $\overline{t_j}$ result from $t_j$ by replacing $s_1, \dots, s_m$ by fresh variables[9] $x_1, \dots, x_m$. Thus, the $\overline{t_j}$ are well-typed terms over $\mathcal{F}'_1$ and $\mathcal{V}'$ with $\overline{t_1} \xrightarrow{\mathrm{i}}_{\mathcal{R}_1} \overline{t_2} \xrightarrow{\mathrm{i}}_{\mathcal{R}_1} \dots$ which shows that $\overline{t_1}$ starts an infinite innermost $\mathcal{R}_1$-reduction. □

We expect that there exist several points where Thm. 7 could simplify innermost termination proofs.[10] In this paper, we use Thm. 7 to weaken the condition (14) required to remove a DP from a DP problem $(\mathcal{P}, \mathcal{R})$. Now one can use any typed variant where $\mathcal{P} \cup \mathcal{R}$ is well typed. To remove $s \to t$ from $\mathcal{P}$, it suffices if

$$\text{for every normal } \sigma \textit{ where } t\sigma \textit{ is well typed, } t\sigma \xrightarrow{\mathrm{i}}{}^!_{\mathcal{R}} q \text{ implies } t\sigma \succ q. \tag{15}$$

## 4 Coupling DPs and Inductive Theorem Proving

Condition (15) is still too hard, because up to now, $t\sigma$ does not have to be ground. We show (in Thm. 12) that for DP problems $(\mathcal{P}, \mathcal{R})$ satisfying suitable non-overlappingness requirements and where $\mathcal{R}$ is already innermost terminating, (15) can be relaxed to ground substitutions $\sigma$. Then $s \to t$ can be removed from $\mathcal{P}$ if

$$\begin{array}{l}\text{for every normal substitution } \sigma \text{ where } t\sigma \text{ is a well-typed } \textit{ground} \text{ term,}\\ t\sigma \xrightarrow{\mathrm{i}}{}^!_{\mathcal{R}} q \text{ implies } t\sigma \succ q.\end{array} \tag{16}$$

[9] Recall that $\mathcal{V}'$ has infinitely many variables for every sort.
[10] E.g., by Thm. 7 one could switch to termination methods like [24] exploiting sorts.

*Example 8. Innermost termination of $\mathcal{R}$ is really needed to replace (15) by (16). To see this, consider the DP problem $(\mathcal{P},\mathcal{R})$ with $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(x)\}$ and the non-innermost terminating TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{a}\}$.*[11] *Let $\mathcal{F} = \{\mathsf{F}, \mathsf{a}\}$. We use a typed variant $\mathcal{F}'$ where $\mathsf{F} : \tau_1 \to \tau_2$ and $\mathsf{a} : \tau_1$. For the right-hand side $t = \mathsf{F}(x)$ of the DP, the only well-typed ground instantiation is $\mathsf{F}(\mathsf{a})$. Since this term has no normal form $q$, the condition (16) holds. Nevertheless, it is not sound to remove the only DP from $\mathcal{P}$, since $\mathsf{F}(x_1) \to \mathsf{F}(x_1)$, $\mathsf{F}(x_2) \to \mathsf{F}(x_2), \ldots$ is an infinite innermost $(\mathcal{P},\mathcal{R})$-chain (but there is no infinite innermost* ground *chain).*

*To see the reason for the non-overlappingness requirement, consider $(\mathcal{P},\mathcal{R})$ with $\mathcal{P} = \{\mathsf{F}(\mathsf{f}(x)) \to \mathsf{F}(\mathsf{f}(x))\}$ and $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{a}\}$. Now $\mathcal{F} = \{\mathsf{F}, \mathsf{f}, \mathsf{a}\}$ and in the typed variant we have $\mathsf{F} : \tau_1 \to \tau_2$, $\mathsf{f} : \tau_1 \to \tau_1$, and $\mathsf{a} : \tau_1$. For the right-hand side $t = \mathsf{F}(\mathsf{f}(x))$ of the DP, the only well-typed ground instantiations are $\mathsf{F}(\mathsf{f}^n(\mathsf{a}))$ with $n \geq 1$. If we take the embedding order $\succ_{emb}$, then all well-typed ground instantiations of $t$ are $\succ_{emb}$-greater than their normal form $\mathsf{F}(\mathsf{a})$. So Condition (16) would allow us to remove the only DP from $\mathcal{P}$. But again, this is unsound, since there is an infinite innermost $(\mathcal{P},\mathcal{R})$-chain (but no such* ground *chain).*

To prove a condition like (16), we replace (16) by the following condition (17), which is easier to check. Here, we require that for all instantiations $t\sigma$ as above, *every* reduction of $t\sigma$ to its normal form uses a strictly decreasing rule $\ell \to r$ (i.e., a rule with $\ell \succ r$) on a strongly monotonic position $\pi$. A position $\pi$ in a term $u$ is *strongly monotonic* w.r.t. $\succ$ iff $t_1 \succ t_2$ implies $u[t_1]_\pi \succ u[t_2]_\pi$ for all terms $t_1$ and $t_2$. So to remove $s \to t$ from $\mathcal{P}$, now it suffices if

> for every normal substitution $\sigma$ where $t\sigma$ is a well-typed ground term,
> every reduction "$t\sigma \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} q$" has the form
>
> $$t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s[\ell\delta]_\pi \xrightarrow{\mathsf{i}}_{\mathcal{R}} s[r\delta]_\pi \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} q \qquad (17)$$
>
> for a rule $\ell \to r \in \mathcal{R}$ where $\ell \succ r$
> and where the position $\pi$ in $s$ is strongly monotonic w.r.t. $\succ$.[12]

For example, for $\mathcal{R}_{sort}$'s termination proof one may use a reduction pair $(\succsim, \succ)$ based on a *polynomial interpretation* [9, 23]. A polynomial interpretation $\mathcal{P}ol$ maps every $n$-ary function symbol $f$ to a polynomial $f_{\mathcal{P}ol}$ over $n$ vari-

[11] One cannot assume that DP problems $(\mathcal{P},\mathcal{R})$ always have a specific form, e.g., that $\mathcal{P}$ includes $\mathsf{A} \to \mathsf{A}$ whenever $\mathcal{R}$ includes $\mathsf{a} \to \mathsf{a}$. The reason is that a DP problem $(\mathcal{P},\mathcal{R})$ can result from arbitrary DP processors that were applied before. Hence, one really has to make sure that processors are sound for *arbitrary* DP problems $(\mathcal{P},\mathcal{R})$.

[12] In special cases, condition (17) can be automated by $k$-times *narrowing* the DP $s \to t$ [14]. However, this only works if for any substitution $\sigma$, the reduction $t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s[\ell\delta]_\pi$ is shorter than a fixed number $k$. So it fails for TRSs like $\mathcal{R}_{sort}$ where termination relies on an inductive property. Here, the reduction

$$\mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x, xs)), \mathsf{co}(x, xs)))\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}_{sort}} \mathsf{SORT}(\mathsf{if}_2(\mathsf{true}, \ldots, \ldots))$$

can be arbitrarily long, depending on $\sigma$. Therefore, narrowing the DP (11) a fixed number of times does not help.

ables $x_1, \ldots, x_n$ with coefficients from $\mathbb{N}$. This mapping is extended to terms by $[x]_{\mathcal{P}ol} = x$ for all variables $x$ and $[f(t_1, \ldots, t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}([t_1]_{\mathcal{P}ol}, \ldots, [t_n]_{\mathcal{P}ol})$. Now $s \succ_{\mathcal{P}ol} t$ (resp. $s \succsim_{\mathcal{P}ol} t$) iff $[s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$ (resp. $[s]_{\mathcal{P}ol} \geq [t]_{\mathcal{P}ol}$) holds for all instantiations of the variables with natural numbers. For instance, consider the interpretation $\mathcal{P}ol_1$ with

$$\begin{array}{ll} 0_{\mathcal{P}ol_1} = \mathsf{nil}_{\mathcal{P}ol_1} = \mathsf{true}_{\mathcal{P}ol_1} = \mathsf{false}_{\mathcal{P}ol_1} = \mathsf{ge}_{\mathcal{P}ol_1} = \mathsf{eq}_{\mathcal{P}ol_1} = 0 & \mathsf{s}_{\mathcal{P}ol_1} = 1 + x_1 \\ \mathsf{co}_{\mathcal{P}ol_1} = 1 + x_1 + x_2 & \mathsf{max}_{\mathcal{P}ol_1} = x_1 \\ \mathsf{if}_{1\mathcal{P}ol_1} = 1 + x_2 + x_3 + x_4 & \mathsf{del}_{\mathcal{P}ol_1} = x_2 \\ \mathsf{if}_{2\mathcal{P}ol_1} = 1 + x_3 + x_4 & \mathsf{SORT}_{\mathcal{P}ol_1} = x_1 \end{array}$$

When using the reduction pair $(\succsim_{\mathcal{P}ol_1}, \succ_{\mathcal{P}ol_1})$, the DP (11) and all rules of $\mathcal{R}'_{sort}$ are weakly decreasing. Moreover, then Condition (17) is indeed satisfied for the right-hand side $t$ of (11). To see this, note that in *every* reduction $t\sigma \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} q$ where $t\sigma$ is a well-typed ground term, eventually one has to apply the rule "$\mathsf{if}_2(\mathsf{true}, x, y, xs) \to xs$" which is strictly decreasing w.r.t. $\succ_{\mathcal{P}ol_1}$. This rule is used by the $\mathsf{del}$-algorithm to delete an element, i.e., to reduce the length of the list. Moreover, the rule is used within a context of the form $\mathsf{SORT}(\mathsf{co}(..., \mathsf{co}(..., \quad \ldots \quad \mathsf{co}(..., \Box))))$. Note that the polynomial $\mathsf{SORT}_{\mathcal{P}ol_1}$ resp. $\mathsf{co}_{\mathcal{P}ol_1}$ is strongly monotonic in its first resp. second argument. Thus, the strictly decreasing rule is indeed used on a strongly monotonic position.

To check automatically whether every reduction of $t\sigma$ to normal form uses a strictly decreasing rule on a strongly monotonic position, we add new rules and function symbols to the TRS $\mathcal{R}$ which results in an extended TRS $\mathcal{R}^{\succ}$. Moreover, for every term $u$ we define a corresponding term $u^{\succ}$. For non-overlapping TRSs $\mathcal{R}$, we have the following property, cf. Lemma 10: if $u^{\succ} \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}^{\succ}} \mathsf{tt}$, then for every reduction $u \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} q$, we have $u \succ q$. We now explain how to construct $\mathcal{R}^{\succ}$.

For every $f \in \mathcal{D}_{\mathcal{R}}$, we introduce a new symbol $f^{\succ}$. Now $f^{\succ}(u_1, ..., u_n)$ should reduce to $\mathsf{tt}$ in the new TRS $\mathcal{R}^{\succ}$ whenever the reduction of $f(u_1, ..., u_n)$ in the original TRS $\mathcal{R}$ uses a strictly decreasing rule on a strongly monotonic position. Thus, if a rule $f(\ell_1, ..., \ell_n) \to r$ of $\mathcal{R}$ was strictly decreasing (i.e., $f(\ell_1, ..., \ell_n) \succ r$), then we add the rule $f^{\succ}(\ell_1, ..., \ell_n) \to \mathsf{tt}$ in $\mathcal{R}^{\succ}$. Otherwise, a strictly decreasing rule will be used on a strongly monotonic position to reduce an instance of $f(\ell_1, \ldots, \ell_n)$ if this holds for the corresponding instance of the right-hand side $r$. Hence, then we add the rule $f^{\succ}(\ell_1, ..., \ell_n) \to r^{\succ}$ in $\mathcal{R}^{\succ}$ instead. It remains to define $u^{\succ}$ for any term $u$ over the signature of $\mathcal{R}$. If $u = f(u_1, ..., u_n)$, then we regard the subterms on the strongly monotonic positions of $u$ and check whether their reduction uses a strictly decreasing rule. For any $n$-ary symbol $f$, let $mon_{\succ}(f)$ contain those positions from $\{1, \ldots, n\}$ where the term $f(x_1, ..., x_n)$ is strongly monotonic. If $mon_{\succ}(f) = \{i_1, \ldots, i_m\}$, then for $u = f(u_1, ..., u_n)$ we obtain $u^{\succ} = u^{\succ}_{i_1} \vee \ldots \vee u^{\succ}_{i_m}$, if $f$ is a constructor. If $f$ is defined, then a strictly decreasing rule could also be applied on the root position of $u$. Hence, then we have $u^{\succ} = u^{\succ}_{i_1} \vee \ldots \vee u^{\succ}_{i_m} \vee f^{\succ}(u_1, ..., u_n)$. Of course, $\mathcal{R}^{\succ}$ also contains appropriate rules for the disjunction "$\vee$".[13] The empty disjunction is represented by $\mathsf{ff}$.

[13] It suffices to include just the rules "$\mathsf{tt} \vee b \to \mathsf{tt}$" and "$\mathsf{ff} \vee b \to b$", since $\mathcal{R}^{\succ}$ is only used for inductive proofs and "$b \vee \mathsf{tt} = \mathsf{tt}$" and "$b \vee \mathsf{ff} = b$" are inductive consequences.

**Definition 9 ($\mathcal{R}^\succ$).** *Let $\succ$ be an order on terms and let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{V}$. We extend $\mathcal{F}$ to a new signature $\mathcal{F}^\succ = \mathcal{F} \uplus \{f^\succ \mid f \in \mathcal{D}_\mathcal{R}\} \uplus \{\mathsf{tt}, \mathsf{ff}, \vee\}$. For any term $u$ over $\mathcal{F}$ and $\mathcal{V}$, we define the term $u^\succ$ over $\mathcal{F}^\succ$ and $\mathcal{V}$:*

$$u^\succ = \begin{cases} \bigvee_{i \in mon_\succ(f)} u_i^\succ, & \text{if } u = f(u_1, \ldots, u_n) \text{ and } f \notin \mathcal{D}_\mathcal{R} \\ \bigvee_{i \in mon_\succ(f)} u_i^\succ \quad \vee \quad f^\succ(u_1, \ldots, u_n), & \text{if } u = f(u_1, \ldots, u_n) \text{ and } f \in \mathcal{D}_\mathcal{R} \\ \mathsf{ff}, & \text{if } u \in \mathcal{V} \end{cases}$$

$$\begin{aligned} \text{Moreover, we define } \mathcal{R}^\succ = \quad & \{f^\succ(\ell_1, ..., \ell_n) \to \mathsf{tt} \mid f(\ell_1, ..., \ell_n) \to r \in \mathcal{R} \cap \succ\} \\ & \cup \{f^\succ(\ell_1, ..., \ell_n) \to r^\succ \mid f(\ell_1, ..., \ell_n) \to r \in \mathcal{R} \setminus \succ\} \\ & \cup \mathcal{R} \cup \{\mathsf{tt} \vee b \to \mathsf{tt},\ \mathsf{ff} \vee b \to b\}. \end{aligned}$$

In our example, the only rules of $\mathcal{R}'_{sort}$ which are strictly decreasing w.r.t. $\succ_{\mathcal{P}ol_1}$ are the last two $\mathsf{max}$-rules and the rule "$\mathsf{if}_2(\mathsf{true}, x, y, xs) \to xs$". So according to Def. 9, the TRS $\mathcal{R}'^{\succ_{\mathcal{P}ol_1}}_{sort}$ contains $\mathcal{R}'_{sort} \cup \{\mathsf{tt} \vee b \to \mathsf{tt},\ \mathsf{ff} \vee b \to b\}$ and the following rules. Here, we already simplified disjunctions of the form "$\mathsf{ff} \vee t$" or "$t \vee \mathsf{ff}$" to $t$. To ease readability, we wrote "$\mathsf{ge}^\succ$" instead of "$\mathsf{ge}^{\succ_{\mathcal{P}ol_1}}$", etc.

$$\begin{array}{rl} \mathsf{ge}^\succ(x, 0) & \to \mathsf{ff} \\ \mathsf{ge}^\succ(0, \mathsf{s}(y)) & \to \mathsf{ff} \\ \mathsf{ge}^\succ(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{ge}^\succ(x, y) \\ \\ \mathsf{max}^\succ(\mathsf{nil}) & \to \mathsf{ff} \\ \mathsf{max}^\succ(\mathsf{co}(x, \mathsf{nil})) & \to \mathsf{tt} \\ \mathsf{max}^\succ(\mathsf{co}(x, \mathsf{co}(y, xs))) & \to \mathsf{tt} \\ \\ \mathsf{del}^\succ(x, \mathsf{nil}) & \to \mathsf{ff} \\ \mathsf{del}^\succ(x, \mathsf{co}(y, xs)) & \to \mathsf{if}_2^\succ(\mathsf{eq}(x, y), x, y, xs) \end{array} \qquad \begin{array}{rl} \mathsf{eq}^\succ(0, 0) & \to \mathsf{ff} \\ \mathsf{eq}^\succ(\mathsf{s}(x), 0) & \to \mathsf{ff} \\ \mathsf{eq}^\succ(0, \mathsf{s}(y)) & \to \mathsf{ff} \\ \mathsf{eq}^\succ(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{eq}^\succ(x, y) \\ \\ \mathsf{if}_1^\succ(\mathsf{true}, x, y, xs) & \to \mathsf{max}^\succ(\mathsf{co}(x, xs)) \\ \mathsf{if}_1^\succ(\mathsf{false}, x, y, xs) & \to \mathsf{max}^\succ(\mathsf{co}(y, xs)) \\ \\ \mathsf{if}_2^\succ(\mathsf{true}, x, y, xs) & \to \mathsf{tt} \\ \mathsf{if}_2^\succ(\mathsf{false}, x, y, xs) & \to \mathsf{del}^\succ(x, xs) \end{array}$$

**Lemma 10 (Soundness of $\mathcal{R}^\succ$).** *Let $(\succsim, \succ)$ be a reduction pair and let $\mathcal{R}$ be a non-overlapping TRS over $\mathcal{F}$ and $\mathcal{V}$ with $\mathcal{R} \subseteq \succsim$. For any terms $u$ and $q$ over $\mathcal{F}$ and $\mathcal{V}$ with $u^\succ \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ and $u \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} q$, we have $u \succ q$.*

*Proof.* We use induction on the lexicographic combination of the length of the reduction $u \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} q$ and of the structure of $u$.

First let $u$ be a variable. Here, $u^\succ = \mathsf{ff}$ and thus, $u^\succ \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ is impossible.

Now let $u = f(u_1, ..., u_n)$. The reduction $u \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} q$ starts with $u = f(u_1, ..., u_n) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} f(q_1, ..., q_n)$ where the reductions $u_i \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} q_i$ are at most as long as $u \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} q$. If there is a $j \in mon_\succ(f)$ with $u_j^\succ \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$, then $u_j \succ q_j$ by induction hypothesis. So $u = f(u_1, ..., u_j, ..., u_n) \succ f(u_1, ..., q_j, ..., u_n) \succsim f(q_1, ..., q_j, ..., q_n) \succsim q$, as $\mathcal{R} \subseteq \succsim$.

Otherwise, $u^\succ \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ means that $f^\succ(u_1, \ldots, u_n) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$. As $\mathcal{R} \subseteq \mathcal{R}^\succ$, we have $f^\succ(u_1, \ldots, u_n) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} f^\succ(q_1, \ldots, q_n)$. Since $\mathcal{R}$ is non-overlapping, $\mathcal{R}^\succ$ is non-overlapping as well. This implies confluence of $\xrightarrow{\mathsf{i}}_{\mathcal{R}^\succ}$, cf. [29]. Hence, we also get $f^\succ(q_1, \ldots, q_n) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$. There is a rule $f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$ and a normal substitution $\delta$ with $f^\succ(q_1, \ldots, q_n) = f(\ell_1, \ldots, \ell_n)\delta \xrightarrow{\mathsf{i}}_\mathcal{R} r\delta \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} q$. Note that the $q_i$ only contain symbols of $\mathcal{F}$. Thus, as the $q_i$ are normal forms w.r.t. $\mathcal{R}$, they are also normal forms w.r.t. $\mathcal{R}^\succ$. Therefore, as $\mathcal{R}^\succ$ is non-overlapping, the only rule of $\mathcal{R}^\succ$ applicable to $f^\succ(q_1, \ldots, q_n)$ is the one resulting from $f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}$. If $f(\ell_1, \ldots, \ell_n) \succ r$, then that rule would be "$f^\succ(\ell_1, \ldots, \ell_n) \to \mathsf{tt}$" and

$$u = f(u_1, \ldots, u_n) \succsim f(q_1, \ldots, q_n) = f(\ell_1, \ldots, \ell_n)\delta \succ r\delta \succsim q.$$

Otherwise, the rule is "$f^{\succ}(\ell_1,...,\ell_n) \to r^{\succ}$", i.e., $f^{\succ}(q_1,...,q_n) = f^{\succ}(\ell_1,...,\ell_n)\delta \xrightarrow{\mathsf{i}}_{\mathcal{R}^{\succ}} r^{\succ}\delta \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}^{\succ}} \mathsf{tt}$. Since the reduction $r\delta \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} q$ is shorter than the original reduction $u \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} q$, the induction hypothesis implies $r\delta \succ q$. Thus,

$$u = f(u_1,\ldots,u_n) \succsim f(q_1,\ldots,q_n) = f(\ell_1,\ldots,\ell_n)\delta \succsim r\delta \succ q. \qquad \square$$

With Lemma 10, the condition (17) needed to remove a DP from a DP problem can again be reformulated. To remove $s \to t$ from $\mathcal{P}$, now it suffices if

$$\text{for every normal substitution } \sigma \text{ where } t\sigma \text{ is a well-typed ground term, we have } t^{\succ}\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}^{\succ}} \mathsf{tt}. \tag{18}$$

So in our example, to remove the DP (11) using the reduction pair $(\succsim_{\mathcal{P}ol_1}, \succ_{\mathcal{P}ol_1})$, we require "$t^{\succ_{\mathcal{P}ol_1}}\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}'^{\succ_{\mathcal{P}ol_1}}_{sort}} \mathsf{tt}$", where $t$ is the right-hand side of (11), i.e., $t = \mathsf{SORT}(\mathsf{del}(\mathsf{max}(\mathsf{co}(x,xs)),\mathsf{co}(x,xs)))$. Since $mon_{\succ_{\mathcal{P}ol_1}}(\mathsf{SORT}) = \{1\}$, $mon_{\succ_{\mathcal{P}ol_1}}(\mathsf{del}) = \{2\}$, $mon_{\succ_{\mathcal{P}ol_1}}(\mathsf{co}) = \{1,2\}$, and $x^{\succ_{\mathcal{P}ol_1}} = xs^{\succ_{\mathcal{P}ol_1}} = \mathsf{ff}$, $t^{\succ_{\mathcal{P}ol_1}}$ is $\mathsf{del}^{\succ_{\mathcal{P}ol_1}}(\mathsf{max}(\mathsf{co}(x,xs)),\mathsf{co}(x,xs))$ when simplifying disjunctions with $\mathsf{ff}$. So to remove (11), we require the following for all normal substitutions $\sigma$ where $t\sigma$ is well typed and ground.[14]

$$\mathsf{del}^{\succ_{\mathcal{P}ol_1}}(\mathsf{max}(\mathsf{co}(x,xs)),\mathsf{co}(x,xs))\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}'^{\succ_{\mathcal{P}ol_1}}_{sort}} \mathsf{tt} \tag{19}$$

Note that the rules for $\mathsf{del}^{\succ_{\mathcal{P}ol_1}}$ (given before Lemma 10) compute the *member*-function. In other words, $\mathsf{del}^{\succ_{\mathcal{P}ol_1}}(x,xs)$ holds iff $x$ occurs in the list $xs$. Thus, (19) is equivalent to the main termination argument (1) of Ex. 1, i.e., to the observation that every non-empty list contains its maximum. Thus, now we can detect and express termination arguments like (1) within the DP framework.

Our goal is to use inductive theorem provers to verify arguments like (1) or, equivalently, to verify conditions like (18). Indeed, (18) corresponds to the question whether a suitable conjecture is *inductively valid* [4, 5, 7, 8, 20, 21, 33, 34, 36].

**Definition 11 (Inductive Validity).** *Let $\mathcal{R}$ be a TRS and let $s,t$ be terms over $\mathcal{F}$ and $\mathcal{V}$. We say that $t = s$ is* inductively valid *in $\mathcal{R}$ (denoted $\mathcal{R} \models_{ind} t = s$) iff there exist typed variants $\mathcal{F}'$ and $\mathcal{V}'$ such that $\mathcal{R}$, $t$, $s$ are well typed where $t$ and $s$ have the same type, and such that $t\sigma \xleftrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s\sigma$ holds for all substitutions $\sigma$ over $\mathcal{F}'$ where $t\sigma$ and $s\sigma$ are well-typed ground terms. To make the specific typed variants explicit, we also write "$\mathcal{R} \models_{ind}^{\mathcal{F}',\mathcal{V}'} t = s$".*

Of course, in general $\mathcal{R} \models_{ind} t = s$ is undecidable, but it can often be proved automatically by *inductive theorem provers*. By reformulating Condition (18), we now obtain that in a DP problem $(\mathcal{P},\mathcal{R})$, $s \to t$ can be removed from $\mathcal{P}$ if

$$\mathcal{R}^{\succ} \models_{ind} t^{\succ} = \mathsf{tt}. \tag{20}$$

Of course, in addition all DPs $\mathcal{P}$ and all rules $\mathcal{R}$ have to be weakly decreasing.

Now we formulate a new DP processor based on Condition (20). Recall that to derive (20) we required a non-overlappingness condition and innermost ter-

[14] Note that the restriction to *well-typed ground terms* is crucial. Indeed, (19) does not hold for non-ground or non-well-typed substitutions like $\sigma(x) = \sigma(xs) = \mathsf{true}$.

mination of $\mathcal{R}$. (These requirements ensure that it suffices to regard only *ground* instantiations when proving that reductions of $t\sigma$ to normal form are strictly decreasing, cf. Ex. 8. Moreover, non-overlappingness is needed for Lemma 10 to make sure that $t^\succ\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ really guarantees that *all* reductions of $t\sigma$ to normal form are strictly decreasing. Non-overlappingness also ensures that $t^\succ\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ in Condition (18) is equivalent to $t^\succ\sigma \xleftrightarrow{\mathsf{i}}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ in Condition (20).)

To ensure innermost termination of $\mathcal{R}$, the following processor transforms $(\mathcal{P},\mathcal{R})$ not only into the new DP problem $(\mathcal{P}\setminus\{s\to t\},\mathcal{R})$, but it also generates the problem $(DP(\mathcal{R}),\mathcal{R})$. Absence of infinite innermost $(DP(\mathcal{R}),\mathcal{R})$-chains is equivalent to innermost termination of $\mathcal{R}$. Note that in practice $\mathcal{R}$ only contains the usable rules of $\mathcal{P}$ (since one should have applied the usable rule processor of Thm. 3 before). Then the DP problem $(DP(\mathcal{R}),\mathcal{R})$ means that the TRS consisting just of the usable rules must be innermost terminating. An application of the dependency graph processor of Thm. 4 will therefore transform $(DP(\mathcal{R}),\mathcal{R})$ into DP problems that have already been generated *before*. So (except for algorithms with nested or mutual recursion), the DP problem $(DP(\mathcal{R}),\mathcal{R})$ obtained by the following processor does not lead to new proof obligations.

In Thm. 12, we restrict ourselves to DP problems $(\mathcal{P},\mathcal{R})$ with the *tuple property*. This means that for all $s\to t\in\mathcal{P}$, $\mathrm{root}(s)$ and $\mathrm{root}(t)$ are tuple symbols and tuple symbols neither occur anywhere else in $s$ or $t$ nor in $\mathcal{R}$. This is always satisfied for the initial DP problem and it is maintained by almost all DP processors in the literature (including all processors of this paper).

**Theorem 12 (Induction Processor).** *Let $(\succsim,\succ)$ be a reduction pair, let $(\mathcal{P},\mathcal{R})$ have the tuple property, let $\mathcal{R}$ be non-overlapping and let there be no critical pairs between $\mathcal{R}$ and $\mathcal{P}$.*[15] *Let $\mathcal{F}'$, $\mathcal{V}'$ be typed variants of $\mathcal{P}\cup\mathcal{R}$'s signature such that $\mathcal{P}\cup\mathcal{R}$ is well typed. Then the following DP processor Proc is sound.*

$$Proc(\,(\mathcal{P},\mathcal{R})\,) = \begin{cases} \{\ (\mathcal{P}\setminus\{s\to t\},\,\mathcal{R}),\ \ (DP(\mathcal{R}),\,\mathcal{R})\ \}, & \text{if } \mathcal{R}^\succ \models^{\mathcal{F}',\mathcal{V}'}_{ind} t^\succ = \mathsf{tt} \\ & \text{and } \mathcal{P}\cup\mathcal{R}\subseteq\succsim \\ \{\,(\mathcal{P},\mathcal{R})\,\}, & \text{otherwise} \end{cases}$$

*Proof.* Suppose there is an infinite innermost $(\mathcal{P},\mathcal{R})$-chain, i.e., $\mathcal{P}\cup\mathcal{R}$ is not innermost terminating. By persistence of innermost termination (Thm. 7), there is a well-typed term that is not innermost terminating w.r.t. $\mathcal{P}\cup\mathcal{R}$. Let $q$ be a minimal such term (i.e., $q$'s proper subterms are innermost terminating). Due to the tuple property, w.l.o.g. $q$ either contains no tuple symbol or $q$ contains a tuple symbol only at the root. In the first case, only $\mathcal{R}$-rules can reduce $q$. Thus, $\mathcal{R}$ is not innermost terminating and there is an infinite innermost $(DP(\mathcal{R}),\,\mathcal{R})$-chain.

Now let $\mathcal{R}$ be innermost terminating. So $\mathrm{root}(q)$ is a tuple symbol and $q$ contains no further tuple symbol. Hence, in $q$'s infinite innermost $\mathcal{P}\cup\mathcal{R}$-reduction, $\mathcal{R}$-rules are only applied below the root and $\mathcal{P}$-rules are only applied on the root position. Moreover, there are infinitely many $\mathcal{P}$-steps. Hence, this infinite reduction corresponds to an infinite innermost $(\mathcal{P},\mathcal{R})$-chain $s_1\to t_1, s_2\to t_2,\ldots$ where $t_i\sigma \xrightarrow{\mathsf{i}}{}^!_{\mathcal{R}} s_{i+1}\sigma$ for all $i$ and all occurring terms are well typed.

[15] More precisely, for all $v\to w$ in $\mathcal{P}$, non-variable subterms of $v$ may not unify with left-hand sides of rules from $\mathcal{R}$ (after variable renaming).

Next we show that due to innermost termination of $\mathcal{R}$, there is even an infinite innermost $(\mathcal{P},\mathcal{R})$-chain on well-typed *ground* terms. Let $\delta$ instantiate all variables in $s_1\sigma$ by ground terms of the corresponding sort. (Recall that in any typed variant there are such ground terms.) We define the normal substitution $\sigma'$ such that $\sigma'(x)$ is the $\mathcal{R}$-normal form of $x\sigma\delta$ for all variables $x$. This normal form must exist since $\mathcal{R}$ is innermost terminating and it is unique since $\mathcal{R}$ is non-overlapping. Clearly, $t_i\sigma \xrightarrow{i}{}^*_{\mathcal{R}} s_{i+1}\sigma$ implies $t_i\sigma\delta \to^*_{\mathcal{R}} s_{i+1}\sigma\delta$, i.e., $t_i\sigma' \to^*_{\mathcal{R}} s_{i+1}\sigma'$. As left-hand sides $s_i$ of DPs do not overlap with rules of $\mathcal{R}$, all $s_i\sigma'$ are in normal form. Due to non-overlappingness of $\mathcal{R}$, $s_{i+1}\sigma'$ is the *only* normal form of $t_i\sigma'$ and thus, it can also be reached by innermost steps, i.e., $t_i\sigma' \xrightarrow{i}{}^!_{\mathcal{R}} s_{i+1}\sigma'$. Hence, there is an infinite innermost $(\mathcal{P},\mathcal{R})$-chain on well-typed *ground* terms.

If this chain does not contain infinitely many variable-renamed copies of the DP $s \to t$, then its tail is an infinite innermost $(\mathcal{P}\backslash\{s \to t\}, \mathcal{R})$-chain. Otherwise, $s_{i_1} \to t_{i_1}, s_{i_2} \to t_{i_2}, \ldots$ are variable-renamed copies of $s \to t$ and thus, $t_{i_1}\sigma'$, $t_{i_2}\sigma'$, $\ldots$ are well-typed ground instantiations of $t$. As $\mathcal{R}^\succ \models_{ind} t^\succ = \mathsf{tt}$, we have $(t_{i_j}\sigma')^\succ = t^\succ_{i_j}\sigma' \xrightarrow{i}{}^*_{\mathcal{R}^\succ} \mathsf{tt}$ for all $j$. Since $t_{i_j}\sigma' \xrightarrow{i}{}^!_{\mathcal{R}} s_{i_j+1}\sigma'$, Lemma 10 implies $t_{i_j}\sigma' \succ s_{i_j+1}\sigma'$ for all (infinitely many) $j$. Moreover, $s_i\sigma' \succsim t_i\sigma'$ and $t_i\sigma' \succsim s_{i+1}\sigma'$ for all $i$, since $\mathcal{P} \cup \mathcal{R} \subseteq \succsim$. This contradicts the well-foundedness of $\succ$. □

In our example, we ended up with the DP problem $(\{(11)\}, \mathcal{R}'_{sort})$. To remove the DP (11) from the DP problem, we use an inductive theorem prover to prove

$$\mathcal{R}'^{\succ_{\mathcal{P}ol_1}}_{sort} \models_{ind} \mathsf{del}^{\succ_{\mathcal{P}ol_1}}(\mathsf{max}(\mathsf{co}(x, xs)), \mathsf{co}(x, xs)) = \mathsf{tt}, \tag{21}$$

i.e., that every non-empty list contains its maximum. The tuple property and the non-overlappingness requirements in Thm. 12 are clearly fulfilled. Moreover, all rules decrease w.r.t. $\succsim_{\mathcal{P}ol_1}$. Hence, the induction processor results in the trivial problem $(\varnothing,\mathcal{R}'_{sort})$ and the problem $(DP(\mathcal{R}'_{sort}), \mathcal{R}'_{sort}) = (\{(2), ..., (10)\}, \mathcal{R}'_{sort})$. The dependency graph processor transforms the latter problem into the problems $(\mathcal{P}_i, \mathcal{R}'_{sort})$ with $1 \le i \le 4$ that had already been solved before, cf. Sect. 2. For example, the induction prover in AProVE proves (21) automatically and thus, it can easily perform the above proof and verify termination of the TRS $\mathcal{R}_{sort}$.

## 5 Experiments and Conclusion

We introduced a new processor in the DP framework which can handle TRSs that terminate because of inductive properties of their algorithms. This processor automatically tries to extract these properties and transforms them into conjectures which are passed to an inductive theorem prover for verification. To obtain a powerful method, we showed that it suffices to prove these conjectures only for well-typed terms, even though the original TRSs under examination are untyped.

We implemented the new processor of Thm. 12 in our termination tool AProVE [13] and coupled it with the small inductive theorem prover that was already available in AProVE. To automate Thm. 12, AProVE selects a DP $s \to t$ and searches for a reduction pair $(\succsim, \succ)$ which orients at least one rule of $\mathcal{U}(t)$ strictly (on a strongly monotonic position). Then AProVE tests if $t^\succ = \mathsf{tt}$ is inductively valid. So in contrast to previous approaches that use inductive the-

orem provers for termination analysis (cf. Sect. 1), our automation can search for arbitrary reduction pairs instead of being restricted to a fixed small set of orders. The search for the reduction pair is guided by the fact that there has to be a strictly decreasing usable rule on a strongly monotonic position.

To demonstrate the power of our method, [1] features a collection of 19 typical TRSs where an inductive argument is needed for the termination proof. This collection contains several TRSs computing classical arithmetical algorithms as well as many TRSs with standard algorithms for list manipulation like sorting, reversing, etc. The previous version of AProVE was the most powerful tool for termination of term rewriting at the *International Competition of Termination Provers*. Nevertheless, this previous AProVE version as well as all other tools in the competition failed on all of these examples. In contrast, with a time limit of 60 seconds per example, our new version of AProVE automatically proves termination of 16 of them. At the same time, the new version of AProVE is as successful as the previous one on the remaining examples of the *Termination Problem Data Base*, which is the collection of examples used in the termination competition. Thus, the present paper is a substantial advance in automated termination proving, since it allows the first combination of powerful TRS termination tools with inductive theorem provers. For details on our experiments and to access our implementation via a web-interface, we refer to [1].

**Acknowledgements.** We are very grateful to Aart Middeldorp and Hans Zantema for suggesting the proof idea of Thm. 7 and for pointing us to [30].

## References

1. AProVE website `http://aprove.informatik.rwth-aachen.de/eval/Induction/`
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133-178, 2000.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge, 1998.
4. A. Bouhoula and M. Rusinowitch. SPIKE: A system for automatic inductive proofs. In *Proc. AMAST'95*, LNCS 936, pp. 576-577, 1995.
5. R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979.
6. J. Brauburger and J. Giesl. Termination analysis by inductive evaluation. In *Proc. CADE'98*, LNAI 1421, pp. 254-269, 1998.
7. A. Bundy. The automation of proof by mathematical induction. In *Handbook of Automated Reasoning*, Vol. 1, pp. 845-911, Robinson and Voronkov (eds.), 2001.
8. H. Comon. Inductionless induction. In *Handbook of Automated Reasoning*, Vol. 1, pp. 913-962, J. A. Robinson and A. Voronkov (eds.), Elsevier & MIT, 2001.
9. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT'07*, LNCS 4501, pp. 340-354, 2007.
10. J. Giesl. Termination analysis for functional programs using term orderings. In *Proc. SAS'95*, LNCS 983, pp. 154-171, 1995.
11. J. Giesl, R. Thiemann, P. Schneider-Kamp. The DP framework: Combining techniques for automated termination proofs. *LPAR'04*, LNAI 3452, p. 301-331, 2005.
12. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. RTA'06*, LNCS 4098, pp. 297-312, 2006.

13. 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.
14. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155-203, 2006.
15. J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. In *Proc. CADE'07*, LNAI 4603, pp. 443-459, 2007.
16. I. Gnaedig and H. Kirchner. Termination of rewriting under strategies. *ACM Transactions on Computational Logic*, 10(3), 2008.
17. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24(1,2):2-23, 1995.
18. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172-199, 2005.
19. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474-511, 2007.
20. D. Kapur and D. R. Musser. Proof by consistency. *Artif. Int.*, 31(2):125-157, 1987.
21. M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer, 2000.
22. A. Krauss. Certified size-change termination. In *Proc. CADE'07*, LNAI 4603, pp. 460-475, 2007.
23. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
24. S. Lucas and J. Meseguer. Order-sorted dependency pairs. In *Proc. PPDP'08*, pp. 108-119, ACM Press, 2008.
25. P. Manolios and D. Vroon. Termination analysis with calling context graphs. In *Proc. CAV'06*, LNCS 4144, pp. 401-414, 2006.
26. A. Middeldorp and H. Zantema. Personal communication, 2008.
27. E. Ohlebusch. Termination of logic programs: Transformational approaches revisited. *Appl. Algebra in Engineering, Comm. and Computing*, 12:73-116, 2001.
28. S. E. Panitz and M. Schmidt-Schauß. TEA: Automatically proving termination of programs in a non-strict higher-order functional language. In *Proc. SAS'97*, LNCS 1302, pp. 345-360, 1997.
29. D. A. Plaisted. Equational reasoning and term rewriting systems. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 1, pp. 273-364, D. M. Gabbay, C. J. Hogger, and J. A. Robinson (eds.), Oxford, 1993.
30. J. van de Pol. Modularity in many-sorted term rewriting. Master's Thesis, Utrecht University, 1992. Available from `http://homepages.cwi.nl/~vdpol/papers/`.
31. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM Transactions on Computational Logic*, 2009. To appear.
32. C. Walther. On proving the termination of algorithms by machine. *Artificial Intelligence*, 71(1): 101-157, 1994.
33. C. Walther. Mathematical induction. In *Handbook of Logic in AI and Logic Programming*, Vol. 2, pp. 127-228, Gabbay, Hogger, Robinson (eds.), Oxford, 1994.
34. C. Walther and S. Schweitzer. About VeriFun. In *Proc. CADE'03*, LNAI 2741, pp. 322-327, 2003.
35. H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation* 17(1): 23-50, 1994.
36. H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In *Proc. CADE'88*, LNAI 310, pp. 162-181, 1988.