Proving Termination by Bounded Increase*

Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp

LuFG Informatik 2, RWTH Aachen, Germany,
{giesl,thiemann,swiderski,psk}@informatik.rwth-aachen.de

Abstract. Most methods for termination analysis of term rewrite systems (TRSs) essentially try to find arguments of functions that *decrease* in recursive calls. However, they fail if the reason for termination is that an argument is *increased* in recursive calls repeatedly until it reaches a bound. In this paper, we solve that problem and show how to prove innermost termination of TRSs with bounded increase automatically.

1 Introduction

In programming, one often writes algorithms that terminate because a value is increased until it reaches a bound. Hence, to apply termination techniques of TRSs in practice, they must be able to deal with those algorithms successfully. But unfortunately, all existing methods and tools for automated termination analysis of TRSs fail on such examples. Therefore, proving termination of TRSs with bounded increase was identified as one of the most urgent and challenging problems at the annual *International Competition of Termination Tools* 2006 [16].

Example 1. As an example consider a TRS for subtraction. TRSs of this form often result from the transformation of conditional TRSs or from functional, logic, or imperative programs.

$$\begin{array}{rlr} \mathsf{minus}(x,y) & \to \mathsf{cond}(\mathsf{gt}(x,y),x,y) & (1) \\ \mathsf{cond}(\mathsf{false},x,y) & \to \mathsf{0} & (2) \\ \mathsf{cond}(\mathsf{true},x,y) & \to \mathsf{s}(\mathsf{minus}(x,\mathsf{s}(y))) & (3) \end{array} \qquad \begin{array}{rlr} \mathsf{gt}(\mathsf{0},v) & \to \mathsf{false} & (4) \\ \mathsf{gt}(\mathsf{s}(u),\mathsf{0}) & \to \mathsf{true} & (5) \\ \mathsf{gt}(\mathsf{s}(u),\mathsf{s}(v)) & \to \mathsf{gt}(u,v) & (6) \end{array}$$

To handle TRSs like Ex. 1, we propose to use polynomial interpretations [14]. But instead of classical polynomial interpretations on natural numbers, we use interpretations on *integers*. Such interpretations can measure the difference between the first and second argument of minus. Indeed, minus is terminating since this difference decreases in each recursive call. However, using integer polynomial interpretations is unsound in the existing termination techniques for TRSs.

This is also true for the *dependency pair (DP) method* [1], which is a powerful method for automated termination analysis of TRSs that is implemented in virtually all current automated termination tools. This method relies on the use of *reduction pairs* $(\succsim, \succ)$ to compare terms. Here, $\succsim$ is a stable quasi-order and $\succ$

* *Proc. CADE-21*, LNAI, 2007. Supported by the Deutsche Forschungsgemeinschaft DFG under grant GI 274/5-1 and the DFG Research Training Group 1298 (*AlgoSyn*).

is a stable order, where $\succsim$ and $\succ$ are compatible (i.e., $\succ \circ \succsim \subseteq \succ$ or $\succsim \circ \succ \subseteq \succ$). Moreover, $\succsim$ and $\succ$ have to satisfy the following properties:

(a) $\succsim$ is monotonic (b) $\succ$ is well founded

After recapitulating the DP method in Sect. 2, in Sect. 3 we extend it to *general* reduction pairs (without requirements (a) and (b)). Then one can also use reduction pairs based on integer polynomial interpretations, which violate the requirements (a) and (b).

In Sect. 4 we extend the DP method further to exploit implicit *conditions*. This is needed to prove that an increase is bounded. For instance, the recursive call of minus in Ex. 1 only takes place under the *condition* $\mathsf{gt}(x, y) = \mathsf{true}$.[1] With our extensions, termination provers based on DPs can handle most algorithms with bounded increase that typically occur in practice. In Sect. 5, we discuss the implementation of our method in our termination tool AProVE [9].

2 Dependency Pairs

We assume familiarity with term rewriting [2] and briefly recapitulate the DP method. See [1, 8, 10, 12, 13] for further motivations and extensions.

Definition 2 (Dependency Pairs). *For a TRS $\mathcal{R}$, the* defined *symbols $\mathcal{D}$ are the root symbols of left-hand sides of rules. All other function symbols are called* constructors. *For every defined symbol $f \in \mathcal{D}$, 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 $f \in \mathcal{D}$, 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 $DP(\mathcal{R})$.*

Ex. 1 has the following DPs, where MINUS is the tuple symbol for minus, etc.

$$\mathsf{MINUS}(x, y) \to \mathsf{COND}(\mathsf{gt}(x, y), x, y) \quad (7) \qquad \mathsf{COND}(\mathsf{true}, x, y) \to \mathsf{MINUS}(x, \mathsf{s}(y)) \quad (9)$$
$$\mathsf{MINUS}(x, y) \to \mathsf{GT}(x, y) \quad (8) \qquad \mathsf{GT}(\mathsf{s}(u), \mathsf{s}(v)) \to \mathsf{GT}(u, v) \quad (10)$$

In this paper, we only focus on *innermost* termination, i.e., we only regard the innermost rewrite relation $\xrightarrow{\mathrm{i}}$. The reason is that proving innermost termination is considerably easier than proving full termination and there are large classes of TRSs where innermost termination is already sufficient for termination. In particular, this holds for non-overlapping TRSs like Ex. 1.

[1] Proving termination of TRSs like Ex. 1 is far more difficult than proving termination of programs in a language where one uses a *predefined* function gt. (For such languages, there already exist termination techniques that can handle certain forms of bounded increase [5, 15].) However, if a function like gt is not predefined but written by the "user", then the termination technique cannot presuppose any knowledge about gt's semantics. In contrast, the termination technique has to deduce any needed informations about gt from the user-defined gt-rules.

The main result of the DP method for innermost termination states that a TRS $\mathcal{R}$ is innermost terminating iff there is no infinite minimal innermost $(DP(\mathcal{R}), \mathcal{R})$-*chain*. For any TRSs $\mathcal{P}$ and $\mathcal{R}$, a minimal 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 σ (with possibly infinite domain) where $t_i\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} s_{i+1}\sigma$, all $s_i\sigma$ are in normal form, and all $t_i\sigma$ are innermost terminating w.r.t. $\mathcal{R}$.

Termination techniques are now called *DP processors* and they operate on sets of dependency pairs (which are called *DP problems*).[2] 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 $\mathcal{P}$ with infinite minimal innermost $(\mathcal{P}, \mathcal{R})$-chain there is also a $\mathcal{P}' \in \mathit{Proc}(\mathcal{P})$ with infinite minimal 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 minimal innermost $(\mathcal{P}, \mathcal{R})$-chain if $\mathit{Proc}(\mathcal{P}) = \{\varnothing\}$.

So innermost termination proofs in the DP framework start with the initial DP problem $DP(\mathcal{R})$. Then the DP problem is simplified repeatedly by sound DP processors. If all resulting DP problems have been simplified to $\varnothing$, then innermost termination is proved. In Thm. 3, we recapitulate one of the most important processors of the framework, the so-called *reduction pair processor*.

For a DP problem $\mathcal{P}$, the reduction pair processor generates inequality constraints which should be satisfied by a reduction pair $(\succsim, \succ)$. The constraints require that all DPs in $\mathcal{P}$ are strictly or weakly decreasing and all *usable rules* $\mathcal{U}(\mathcal{P})$ are weakly decreasing. Then one can delete all strictly decreasing DPs.

The *usable rules* include all rules that can reduce the terms in right-hand sides of $\mathcal{P}$ when their variables are instantiated with normal forms. More precisely, for a term containing a defined symbol f, all f-rules are usable. Moreover, if the f-rules are usable and g occurs in the right-hand side of an f-rule, then the g-rules are usable as well. In Thm. 3, 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 3 (Reduction Pair Processor and Usable Rules). *Let $(\succsim, \succ)$ be a reduction pair. Then the following DP processor Proc is sound.*

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

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

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

For a set of dependency pairs $\mathcal{P}$, its usable rules *are $\mathcal{U}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t)$.*

[2] To ease readability we use a simpler definition of *DP problems* than [8], since this simple definition suffices for the presentation of the new results of this paper.

For the TRS of Ex. 1, according to Thm. 3 we search for a reduction pair with $s \underset{(\sim)}{\succ} t$ for all dependency pairs $s \to t \in DP(\mathcal{R}) = \{(7), \ldots, (10)\}$ and with $\ell \succsim r$ for all usable rules $\ell \to r \in \mathcal{U}(DP(\mathcal{R})) = \{(4), (5), (6)\}$.

A popular method to search for suitable relations $\succsim$ and $\succ$ automatically is the use of *polynomial interpretations* [14]. A polynomial interpretation $\mathcal{P}ol$ maps every n-ary function symbol f to a polynomial $f_{\mathcal{P}ol}$ over n variables $x_1, \ldots, x_n$. Traditionally, one uses polynomials with coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$. This mapping is then extended to terms by defining $[x]_{\mathcal{P}ol} = x$ for all variables x and by defining $[f(t_1, \ldots, t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}([t_1]_{\mathcal{P}ol}, \ldots, [t_n]_{\mathcal{P}ol})$. If $\mathcal{P}ol$ is clear from the context, we also write $[t]$ instead of $[t]_{\mathcal{P}ol}$. Now one defines $s \succ_{\mathcal{P}ol} t$ (resp. $s \succsim_{\mathcal{P}ol} t$) iff $[s] > [t]$ (resp. $[s] \geq [t]$) holds for all instantiations of the variables with natural numbers. It is easy to see that $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is a reduction pair.

As an example, consider the polynomial interpretation $\mathcal{P}ol_1$ with $\mathsf{GT}_{\mathcal{P}ol_1} = x_1$, $\mathsf{MINUS}_{\mathcal{P}ol_1} = x_1 + 1$, $\mathsf{COND}_{\mathcal{P}ol_1} = x_2 + 1$, $\mathsf{s}_{\mathcal{P}ol_1} = x_1 + 1$, and $f_{\mathcal{P}ol_1} = 0$ for all other function symbols f. Then the DPs (8) and (10) are strictly decreasing. The reason for $\mathsf{GT}(\mathsf{s}(x), \mathsf{s}(y)) \succ_{\mathcal{P}ol_1} \mathsf{GT}(x, y)$ is that $[\mathsf{GT}(\mathsf{s}(x), \mathsf{s}(y))] = x + 1$ is greater than $[\mathsf{GT}(x, y)] = x$ for all natural numbers x. Moreover, all other DPs and the usable rules are weakly decreasing w.r.t. $\succsim_{\mathcal{P}ol_1}$. Thus, the DPs (8) and (10) can be removed and the reduction pair processor transforms the initial DP problem $DP(\mathcal{R})$ into $\{(7), (9)\}$. We refer to [4, 7] for efficient algorithms to generate suitable polynomial interpretations automatically. However, it is impossible to transform the problem further into the empty DP problem $\varnothing$. More precisely, there is no reduction pair based on polynomial interpretations (or on any other classical order amenable to automation) where one of the DPs (7) and (9) is strictly decreasing and the other one and the usable rules are weakly decreasing, cf. [11]. Indeed, up to now all implementations of the DP method failed on Ex. 1.

3 General Reduction Pairs

Our aim is to handle *integer* polynomial interpretations. More precisely, we want to use polynomial interpretations where all function symbols except tuple symbols are still mapped to polynomials with natural coefficients, but where tuple symbols may be mapped to polynomials with arbitrary integer coefficients. For such integer polynomial interpretations, we still define $s \succ_{\mathcal{P}ol} t$ (resp. $s \succsim_{\mathcal{P}ol} t$) iff $[s] > [t]$ (resp. $[s] \geq [t]$) holds for all instantiations of the variables with *natural* (not with *integer*) numbers. If $\mathcal{F}$ is the original signature without tuple symbols, then the relations $\succ_{\mathcal{P}ol}$ and $\succsim_{\mathcal{P}ol}$ are $\mathcal{F}$-stable, i.e., $s \underset{(\sim)}{\succ}_{\mathcal{P}ol} t$ implies $s\sigma \underset{(\sim)}{\succ}_{\mathcal{P}ol} t\sigma$ for all substitutions σ with terms over $\mathcal{F}$. It is easy to show that $\mathcal{F}$-stability is sufficient for the reduction pairs used in the reduction pair processor.

To solve the remaining DP problem $\{(7), (9)\}$, we want to use the integer polynomial interpretation $\mathcal{P}ol_2$ where $\mathsf{MINUS}_{\mathcal{P}ol_2} = x_1 - x_2$, $\mathsf{COND}_{\mathcal{P}ol_2} = x_2 - x_3$, $\mathsf{s}_{\mathcal{P}ol_2} = x_1 + 1$, and $f_{\mathcal{P}ol_2} = 0$ for all other symbols f. Then DP (9) would be strictly decreasing and could be removed. The resulting DP problem $\{(7)\}$ is easy to solve by $\mathcal{P}ol_3$ with $\mathsf{MINUS}_{\mathcal{P}ol_3} = 1$ and $f_{\mathcal{P}ol_3} = 0$ for all other symbols f.

But such integer interpretations may not be used, since $(\succsim_{\mathcal{P}ol_2}, \succ_{\mathcal{P}ol_2})$ is no reduction pair: $\succsim_{\mathcal{P}ol_2}$ is not monotonic (e.g., $\mathsf{s}(0) \succsim_{\mathcal{P}ol_2} 0$, but $\mathsf{MINUS}(\mathsf{s}(0), \mathsf{s}(0)) \not\succsim_{\mathcal{P}ol_2} \mathsf{MINUS}(\mathsf{s}(0), 0)$). Moreover, $\succ_{\mathcal{P}ol_2}$ is not well founded (e.g., $\mathsf{MINUS}(0,0) \succ_{\mathcal{P}ol_2} \mathsf{MINUS}(0, \mathsf{s}(0)) \succ_{\mathcal{P}ol_2} \mathsf{MINUS}(0, \mathsf{s}(\mathsf{s}(0))) \succ_{\mathcal{P}ol_2} \ldots$). So integer interpretations violate both requirements (a) and (b) for reduction pairs, cf. Sect. 1.

Indeed, using such polynomial interpretations in Thm. 3 is unsound. As $\succ_{\mathcal{P}ol_2}$ is not well founded (i.e., as it violates requirement (b)), $\mathcal{P}ol_2$ could be used for a wrong innermost termination proof of the TRS $\{\mathsf{minus}(x, y) \to \mathsf{minus}(x, \mathsf{s}(y))\}$. But even if requirement (b) were not violated, a violation of requirement (a) would still render Thm. 3 unsound. We demonstrate this in Ex. 4.

Example 4. Consider the following TRS which is not innermost terminating. Here, $\mathsf{round}(x) = x$ *if* x *is even and* $\mathsf{round}(x) = \mathsf{s}(x)$ *if* x *is odd.*

$$\mathsf{minus}(\mathsf{s}(x), x) \to \mathsf{minus}(\mathsf{s}(x), \mathsf{round}(x)) \quad (11)$$

$$\mathsf{round}(0) \to 0 \quad (12)$$

$$\mathsf{round}(\mathsf{s}(0)) \to \mathsf{s}(\mathsf{s}(0)) \quad (13)$$

$$\mathsf{round}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{s}(\mathsf{round}(x))) \quad (14)$$

We use a modification $\mathcal{P}ol_2'$ *of* $\mathcal{P}ol_2$*, where* $\mathsf{MINUS}_{\mathcal{P}ol_2'} = (x_1 - x_2)^2$, $\mathsf{round}_{\mathcal{P}ol_2'} = x_1 + 1$, *and* $\mathsf{ROUND}_{\mathcal{P}ol_2'} = 0$. *Now requirement (b) is satisfied. The* MINUS*-DPs are strictly decreasing (i.e.,* $\mathsf{MINUS}(\mathsf{s}(x), x) \succ_{\mathcal{P}ol_2'} \mathsf{MINUS}(\mathsf{s}(x), \mathsf{round}(x))$ *and* $\mathsf{MINUS}(\mathsf{s}(x), x) \succ_{\mathcal{P}ol_2'} \mathsf{ROUND}(x)$*) and the* ROUND*-DP and the usable rules are weakly decreasing. Thus, if we were allowed to use* $\mathcal{P}ol_2'$ *in Thm. 3, then we could remove the* MINUS*-DPs. The remaining DP problem is easily solved and thus, we would falsely prove innermost termination of this TRS.*

Ex. 4 shows the reason for the unsoundness when dropping requirement (a). Thm. 3 requires $\ell \succsim r$ for all usable rules $\ell \to r$. This is meant to ensure that all reductions with usable rules will weakly decrease the reduced term (w.r.t. $\succsim$). However, this only holds if the quasi-order $\succsim$ is monotonic. In Ex. 4, we have $\mathsf{round}(x) \succsim_{\mathcal{P}ol_2'} x$, but $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{round}(x)) \not\succsim_{\mathcal{P}ol_2'} \mathsf{MINUS}(\mathsf{s}(x), x)$.

Therefore, one should take into account on which positions the used quasi-order $\succsim$ is monotonically *increasing* and on which positions it is monotonically *decreasing*. If a defined function symbol f occurs at a monotonically *increasing* position in the right-hand side of a dependency pair, then one should require $\ell \succsim r$ for all f-rules. If f occurs at a monotonically *decreasing* position, then one should require $r \succsim \ell$. Finally, if f occurs at a position which is neither monotonically increasing nor decreasing, one should require $\ell \approx r$. Here, $\approx$ is the equivalence relation associated with $\succsim$, i.e., $\approx \; = \; \succsim \cap \precsim$.

So we modify our definition of usable rules.[3] When computing $\mathcal{U}(f(t_1, ..., t_n))$, for any $i \in \{1, ..., n\}$ we first check how the quasi-order $\succsim$ treats f's i-th argument. We say that f is $\succsim$*-dependent* on i iff there exist terms $t_1, ..., t_n, t_i'$ where $f(t_1, \ldots, t_i, ..., t_n) \not\approx f(t_1, ..., t_i', \ldots, t_n)$. Moreover, f is $\succsim$*-monotonically increasing* (resp. *decreasing*) on i iff $t_i \succsim t_i'$ implies $f(t_1, ..., t_i, ..., t_n) \succsim f(t_1, ..., t_i', ..., t_n)$ (resp. $f(t_1, ..., t_i, ..., t_n) \precsim f(t_1, ..., t_i', ..., t_n)$) for all terms $t_1, ..., t_n$ and t_i'.

[3] Now $\mathcal{U}(t)$ is no longer a subset of $\mathcal{R}$. We nevertheless refer to $\mathcal{U}(t)$ as "usable" rules in order to keep the similarity to Thm. 3.

Now if f is not $\succsim$-dependent on i, then $\mathcal{U}(t_i)$ does not have to be included in $\mathcal{U}(f(t_1,\ldots,t_n))$ at all. (This idea was already used in recent refined definitions of the "usable rules", cf. [10].) Otherwise, we include the usable rules $\mathcal{U}(t_i)$ if f is $\succsim$-monotonically increasing on i. If it is $\succsim$-monotonically decreasing, we include the reversed rules $\mathcal{U}^{-1}(t_i)$ instead. Finally, if f is $\succsim$-dependent on i, but neither $\succsim$-monotonically increasing nor decreasing, then we include the usable rules of t_i in both directions, i.e., we include $\mathcal{U}^2(t_i)$ which is defined to be $\mathcal{U}(t_i) \cup \mathcal{U}^{-1}(t_i)$.

Definition 5 (General Usable Rules). *For any function symbol f and any $i \in \{1,\ldots,\text{arity}(f)\}$, we define*

$$\text{ord}(f,i) = \begin{cases} 0, \textit{ if } f \textit{ is not } \succsim\textit{-dependent on } i \\ 1, \textit{ otherwise, if } f \textit{ is } \succsim\textit{-monotonically increasing on } i \\ -1, \textit{ otherwise, if } f \textit{ is } \succsim\textit{-monotonically decreasing on } i \\ 2, \textit{ otherwise} \end{cases}$$

For any TRS U, we define $U^0 = \varnothing$, $U^1 = U$, $U^{-1} = \{r \to \ell \mid \ell \to r \in U\}$, and $U^2 = U \cup U^{-1}$. For any term t, we define $\mathcal{U}(t)$ as the smallest set such that[4]

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

For a set of dependency pairs $\mathcal{P}$, we again define $\mathcal{U}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t)$.

So in Ex. 4, if $\mathsf{MINUS}_{\mathcal{P}ol'_2} = (x_1 - x_2)^2$ then MINUS is $\succsim_{\mathcal{P}ol'_2}$-dependent on 2, but neither $\succsim_{\mathcal{P}ol'_2}$-monotonically increasing nor decreasing. Hence, the usable rules include $\ell \to r$ and $r \to \ell$ for all round-rules $\ell \to r \in \{(12),(13),(14)\}$. Thus, we cannot falsely prove innermost termination with $\mathcal{P}ol'_2$ anymore. Indeed, with the modified definition of usable rules above, Thm. 3 can also be used for reduction pairs where $\succsim$ is not monotonic, i.e., where requirement (a) is violated.

We now also show how to omit the requirement (b) that the order $\succ$ in a reduction pair has to be well founded. Instead, we replace well-foundedness by the weaker requirement of *non-infinitesimality*.

Definition 6 (Non-Infinitesimal). *A relation $\succ$ is* non-infinitesimal *if there do not exist any $t, s_0, s_1, \ldots$ with $s_i \succ s_{i+1}$ and $s_i \succ t$ for all $i \in \mathbb{N}$.*

Any well-founded relation is non-infinitesimal. Thm. 7 shows that integer polynomial orders (which are not well founded) are non-infinitesimal as well.[5]

Theorem 7 (Non-Infinitesimality of Integer Polynomial Orders). *Let $\mathcal{P}ol$ be an integer polynomial interpretation. Then $\succ_{\mathcal{P}ol}$ is non-infinitesimal.*

Note that non-infinitesimality of $\succ_{\mathcal{P}ol}$ does not hold for polynomial interpretations on *rational* numbers. To see this, let $\mathsf{a}_{\mathcal{P}ol} = 1$, $\mathsf{b}_{\mathcal{P}ol} = 0$, and $\mathsf{f}_{\mathcal{P}ol} = \frac{x_1}{2}$.

[4] To ease readability, for $k \in \{-1,0,1,2\}$ we write "$\mathcal{U}^k(t)$" instead of "$(\mathcal{U}(t))^k$".

[5] All proofs can be found in [11].

For $s_i = \mathsf{f}^i(\mathsf{a})$ and $t = \mathsf{b}$, we get the infinite sequence $\mathsf{a} \succ_{\mathcal{P}ol} \mathsf{f}(\mathsf{a}) \succ_{\mathcal{P}ol} \mathsf{f}(\mathsf{f}(\mathsf{a})) \succ_{\mathcal{P}ol} \dots$ (i.e., $s_i \succ_{\mathcal{P}ol} s_{i+1}$ for all i) and $\mathsf{f}^i(\mathsf{a}) \succ_{\mathcal{P}ol} \mathsf{b}$ (i.e., $s_i \succ_{\mathcal{P}ol} t$) for all i.

We now extend the reduction pair processor from Thm. 3 to *general* reduction pairs. A *general* reduction pair $(\succsim, \succ)$ consists of an $\mathcal{F}$-stable quasi-order $\succsim$ and a compatible $\mathcal{F}$-stable non-infinitesimal order $\succ$, where $\mathcal{F}$ is the original signature of the TRS, i.e., without tuple symbols. Moreover, the equivalence relation $\approx$ associated with $\succsim$ must be monotonic (i.e., $s \approx t$ implies $u[s]_\pi \approx u[t]_\pi$ for any position π of any term u). But we do not require monotonicity of $\succsim$ or well-foundedness of $\succ$, i.e., both requirements (a) and (b) are dropped. So for any integer polynomial interpretation $\mathcal{P}ol$, $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is a general reduction pair.

In contrast to the reduction pair processor from Thm. 3, the new processor transforms a DP problem into *two* new problems. As before, the first problem results from removing all strictly decreasing dependency pairs. The second DP problem results from removing all DPs $s \to t$ from $\mathcal{P}$ that are *bounded from below*, i.e., DPs which satisfy the inequality $s \succsim \mathsf{c}$ for a fresh constant c.

Theorem 8 (General Reduction Pair Processor). *Let $(\succsim, \succ)$ be a general reduction pair. Let c be a fresh constant not occurring in the signature and let $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim \mathsf{c}\}$. Then the following DP processor Proc is sound. Here, $\mathcal{U}(\mathcal{P})$ is defined as in Def. 5.*

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

Example 9. *To modify Ex. 4 into an innermost terminating TRS, we replace rule (11) by* $\mathsf{minus}(\mathsf{s}(x), x) \to \mathsf{minus}(\mathsf{s}(x), \mathsf{round}(\mathsf{s}(x)))$. *We regard the interpretation $\mathcal{P}ol_2''$ with $\mathsf{MINUS}_{\mathcal{P}ol_2''} = x_1 - x_2$, $\mathsf{s}_{\mathcal{P}ol_2''} = x_1 + 1$, $\mathsf{0}_{\mathcal{P}ol_2''} = 0$, $\mathsf{round}_{\mathcal{P}ol_2''} = x_1$, $\mathsf{ROUND}_{\mathcal{P}ol_2''} = 0$, and $\mathsf{c}_{\mathcal{P}ol_2''} = 0$. Then the MINUS-DPs are strictly decreasing and the ROUND-DP and the usable rules are weakly decreasing. Here, the usable rules are the* reversed round*-rules, since MINUS is $\succsim$-monotonically decreasing on 2. Moreover, all dependency pairs are bounded from below (i.e., $\mathsf{MINUS}(\mathsf{s}(x), x) \succsim_{\mathcal{P}ol_2''} \mathsf{c}$ and $\mathsf{ROUND}(\mathsf{s}(\mathsf{s}(x))) \succsim_{\mathcal{P}ol_2''} \mathsf{c}$). Thus, we can transform the initial DP problem $\mathcal{P} = DP(\mathcal{R})$ into $\mathcal{P} \setminus \mathcal{P}_{bound} = \varnothing$ and into $\mathcal{P} \setminus \succ$, which only contains the ROUND-DP. This remaining DP problem is easily solved and thus, we can prove innermost termination of the TRS.*

Since $\mathcal{U}(\mathcal{P})$ now depends on $\succsim$, the constraints that the reduction pair has to satisfy in Thm. 8 depend on the reduction pair itself. Nevertheless, if one uses reduction pairs based on polynomial interpretations, then the search for suitable reduction pairs can still be mechanized efficiently. More precisely, one can reformulate Thm. 8 in a way where one first generates constraints (that are independent of $\succsim$) and searches for a reduction pair satisfying the constraints afterwards. We showed in [10, Sect. 7.1] how to reformulate "f is $\succsim$-dependent on i" accordingly and "f is $\succsim$-monotonically increasing on i" can be reformulated by requiring that the partial derivative of $f_{\mathcal{P}ol}$ w.r.t. x_i is non-negative, cf. [1, Footnote 11].

There have already been previous approaches to extend the DP method to non-monotonic reduction pairs. Hirokawa and Middeldorp [13] allowed interpretations like $\mathsf{MINUS}_{\mathcal{P}ol} = \max(x_1 - x_2, 0)$.[6] However, instead of detecting $\succsim$-monotonically increasing and decreasing positions, they always require $\ell \approx r$ for the usable rules. Therefore, their technique fails on Ex. 9, since their constraints cannot be fulfilled by the interpretations considered in their approach, cf. [11].

Another approach was presented in [1, Thm. 33] and further extended in [6]. Essentially, here one permits non-monotonic quasi-orders $\succsim$ provided that f is $\succsim$-monotonically increasing on a position i whenever there is a subterm $f(t_1, ..., t_i, ..., t_n)$ in a right-hand side of a dependency pair or of a usable rule where t_i contains a defined symbol. Then Thm. 3 is still sound (this also follows from Def. 5 and Thm. 8). However, this approach would not allow us to handle arbitrary non-monotonic reduction pairs and therefore, it also fails on Ex. 9.

4 Conditions for Bounded Increase

With Thm. 8 we still cannot use our desired integer polynomial interpretation $\mathcal{P}ol_2$ with $\mathsf{MINUS}_{\mathcal{P}ol_2} = x_1 - x_2$, $\mathsf{COND}_{\mathcal{P}ol_2} = x_2 - x_3$, $\mathsf{s}_{\mathcal{P}ol_2} = x_1 + 1$, and $f_{\mathcal{P}ol_2} = 0$ for all other function symbols f to prove innermost termination of Ex. 1. When trying to solve the remaining DP problem $\{(7), (9)\}$, the DP (9) would be strictly decreasing but none of the two DPs would be bounded. The reason is that we have neither $\mathsf{MINUS}(x, y) \succsim_{\mathcal{P}ol_2} \mathsf{c}$ nor $\mathsf{COND}(\mathsf{true}, x, y) \succsim_{\mathcal{P}ol_2} \mathsf{c}$ for any possible value of $\mathsf{c}_{\mathcal{P}ol_2}$. Thus, the reduction pair processor would return the two DP problems $\{(7)\}$ and $\{(7), (9)\}$, i.e., it would not simplify the DP problem. (Of course since $\{(7)\} \subseteq \{(7), (9)\}$, it suffices to regard just the problem $\{(7), (9)\}$.)

The solution is to consider *conditions* when requiring inequalities like $s \;{}_{(}\!\succsim\!{}_{)}\; t$ or $s \succsim \mathsf{c}$. For example, to include the DP (7) in $\mathcal{P}_{bound}$, we do not have to demand $\mathsf{MINUS}(x, y) \succsim \mathsf{c}$ for *all* instantiations of x and y. Instead, it suffices to require the inequality only for those instantiations of x and y which can be used in potentially infinite minimal innermost chains. So we require $\mathsf{MINUS}(x, y) \succsim \mathsf{c}$ only for instantiations σ where (7)'s instantiated right-hand side $\mathsf{COND}(\mathsf{gt}(x, y), x, y)\sigma$ reduces to an instantiated left-hand side $u\sigma$ for some DP $u \to v$.[7] Here, $u \to v$ should again be variable renamed. As our DP problem contains two DPs (7) and (9), we get the following two constraints (by considering all possibilities $u \to v \in \{(7), (9)\}$). If both constraints are satisfied, then we can include (7) in $\mathcal{P}_{bound}$.

$$\mathsf{COND}(\mathsf{gt}(x, y), x, y) = \mathsf{MINUS}(x', y') \quad \Rightarrow \quad \mathsf{MINUS}(x, y) \succsim \mathsf{c} \tag{15}$$
$$\mathsf{COND}(\mathsf{gt}(x, y), x, y) = \mathsf{COND}(\mathsf{true}, x', y') \quad \Rightarrow \quad \mathsf{MINUS}(x, y) \succsim \mathsf{c} \tag{16}$$

Def. 10 introduces the syntax and semantics of such conditional constraints.

Definition 10 (Conditional Constraint). *For given relations $\succsim$ and $\succ$, the set $\mathcal{C}$ of conditional constraints is the smallest set with*

[6] While such interpretations always result in well-founded orders, they are difficult to generate automatically. In contrast, the search for integer polynomial interpretations is as for ordinary polynomial interpretations, e.g., by using SAT solving as in [7].

[7] Moreover, $\mathsf{COND}(\mathsf{gt}(x, y), x, y)\sigma$ must be innermost terminating, $\mathsf{COND}(\mathsf{gt}(x, y), x, y)\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} u\sigma$, and $u\sigma$ must be in normal form, since we consider *minimal innermost* chains.

- $\{TRUE,\ s \succsim t,\ s \succ t,\ s = t\} \subseteq \mathcal{C}$ *for all terms* s *and* t
- *if* $\{\varphi_1, \varphi_2\} \subseteq \mathcal{C}$, *then* $\varphi_1 \Rightarrow \varphi_2 \in \mathcal{C}$ *and* $\varphi_1 \wedge \varphi_2 \in \mathcal{C}$
- *if* $\varphi \in \mathcal{C}$ *and* $y \in \mathcal{V}$, *then* $\forall y\ \ \varphi \in \mathcal{C}$

Now we define which normal $\mathcal{F}$-substitutions[8] σ *satisfy a constraint* $\varphi \in \mathcal{C}$, *denoted "*$\sigma \models \varphi$*":*

- $\sigma \models TRUE$ *for all normal* $\mathcal{F}$*-substitutions* σ
- $\sigma \models s \succsim t$ *iff* $s\sigma \succsim t\sigma$ *and* $\sigma \models s \succ t$ *iff* $s\sigma \succ t\sigma$
- $\sigma \models s = t$ *iff* $s\sigma$ *is innermost terminating,* $s\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma$, $t\sigma$ *is a normal form*
- $\sigma \models \varphi_1 \Rightarrow \varphi_2$ *iff* $\sigma \not\models \varphi_1$ *or* $\sigma \models \varphi_2$
- $\sigma \models \varphi_1 \wedge \varphi_2$ *iff* $\sigma \models \varphi_1$ *and* $\sigma \models \varphi_2$
- $\sigma \models \forall y\ \varphi$ *iff* $\sigma' \models \varphi$ *for all normal* $\mathcal{F}$*-substitutions* σ' *where* $\sigma'(x) = \sigma(x)$ *for all* $x \neq y$

A constraint φ *is* valid *("*$\models \varphi$*") iff* $\sigma \models \varphi$ *holds for* all *normal* $\mathcal{F}$*-substitutions* σ.

Now we refine the reduction pair processor by taking conditions into account. To this end, we modify the definition of $\mathcal{P}_{bound}$ and introduce $\mathcal{P}_{\succ}$ and $\mathcal{P}_{\succsim}$.

Theorem 11 (Conditional General Reduction Pair Processor). *Let* $(\succsim, \succ)$ *be a general reduction pair. Let* c *be a fresh constant and let*

$$\begin{aligned} \mathcal{P}_{\succ} &= \{\ s \to t \in \mathcal{P} \ \ |\ \ \models \textstyle\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ s \succ t)\ \} \\ \mathcal{P}_{\succsim} &= \{\ s \to t \in \mathcal{P} \ \ |\ \ \models \textstyle\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ s \succsim t)\ \} \\ \mathcal{P}_{bound} &= \{\ s \to t \in \mathcal{P} \ \ |\ \ \models \textstyle\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ s \succsim \mathsf{c})\ \} \end{aligned}$$

where u' *results from* u *by renaming its variables into fresh variables. Then the following DP processor Proc is sound. Here,* $\mathcal{U}(\mathcal{P})$ *is defined as in Def. 5.*

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

To ease readability, in Thm. 11 we only consider the conditions resulting from *two* DPs $s \to t$ and $u \to v$ which follow each other in minimal innermost chains. To consider also conditions resulting from $n+1$ adjacent DPs, one would have to modify $\mathcal{P}_{\succ}$ as follows (of course, $\mathcal{P}_{\succsim}$ and $\mathcal{P}_{bound}$ have to be modified analogously).

$$\mathcal{P}_{\succ} = \{ s \to t \in \mathcal{P} \mid\ \models \textstyle\bigwedge_{u_1 \to v_1, \ldots, u_n \to v_n \in \mathcal{P}} (t = u_1' \wedge v_1' = u_2' \wedge \ldots \wedge v_{n-1}' = u_n' \ \Rightarrow\ s \succ t)\ \}$$

Here, the variables in u_i' and v_i' must be renamed in order to be disjoint to the variables in u_j' and v_j' for $j \neq i$. Moreover, instead of regarding DPs which *follow* $s \to t$ in chains, one could also regards DPs which *precede* $s \to t$. Then instead of (or in addition to) the premise "$t = u'$", one would have the premise "$v' = s$".

The question remains how to check whether conditional constraints are valid, since this requires reasoning about reductions resp. reachability. We now introduce a calculus of seven rules to simplify conditional constraints. For example, the constraint (15) is trivially valid, since its condition is unsatisfiable.

[8] A *normal $\mathcal{F}$-substitution* σ instantiates all variables by normal forms that do not contain tuple symbols (i.e., for any $x \in \mathcal{V}$, all function symbols in $\sigma(x)$ are from $\mathcal{F}$).

The reason is that there is no substitution σ with $\sigma \models \mathsf{COND}(\mathsf{gt}(x,y),x,y) = \mathsf{MINUS}(x',y')$, since COND is no defined function symbol (i.e., it is a *constructor*) and therefore, COND-terms can only be reduced to COND-terms.

This leads to the first inference rule. In a conjunction $\varphi_1 \wedge \ldots \wedge \varphi_n$ of conditional constraints φ_i, these rules can be used to replace a conjunct φ_i by a new formula φ_i'. Of course, $\mathit{TRUE} \wedge \varphi$ can always be simplified to φ. Eventually, the goal is to remove all equalities "$p = q$" from the constraints. The soundness of the rules is shown in Thm. 14: if φ_i is replaced by φ_i', then $\models \varphi_i'$ implies $\models \varphi_i$.

I. **Constructor and Different Function Symbol**

$$\frac{f(p_1,...,p_n) = g(q_1,...,q_m) \wedge \varphi \;\Rightarrow\; \psi}{\mathit{TRUE}} \quad \text{if } f \text{ is a constructor and } f \neq g$$

Rule (II) handles conditions like $\mathsf{COND}(\mathsf{gt}(x,y),x,y) = \mathsf{COND}(\mathsf{true},x',y')$ where both terms start with the constructor COND. So (16) is transformed to

$$\mathsf{gt}(x,y) = \mathsf{true} \wedge x = x' \wedge y = y' \quad \Rightarrow \quad \mathsf{MINUS}(x,y) \succsim \mathsf{c} \tag{17}$$

II. **Same Constructors on Both Sides**

$$\frac{f(p_1,...,p_n) = f(q_1,...,q_n) \wedge \varphi \;\Rightarrow\; \psi}{p_1 = q_1 \wedge \ldots \wedge p_n = q_n \wedge \varphi \;\Rightarrow\; \psi} \quad \text{if } f \text{ is a constructor}$$

Rule (III) removes conditions of the form "$x = q$" or "$q = x$" by applying the substitution $[x/q]$ to the constraint.[9] So (17) is transformed to

$$\mathsf{gt}(x,y) = \mathsf{true} \quad \Rightarrow \quad \mathsf{MINUS}(x,y) \succsim \mathsf{c} \tag{18}$$

III. **Variable in Equation**

$$\frac{x = q \wedge \varphi \;\Rightarrow\; \psi}{\varphi\,\sigma \;\Rightarrow\; \psi\,\sigma} \quad \text{if } x \in \mathcal{V} \text{ and } \sigma = [x/q] \qquad \frac{q = x \wedge \varphi \;\Rightarrow\; \psi}{\varphi\,\sigma \;\Rightarrow\; \psi\,\sigma} \quad \text{if } x \in \mathcal{V},\ q \text{ has no defined symbols, } \sigma = [x/q]$$

Of course, one can also omit arbitrary conjuncts from the premise of an implication. To ease notation, we regard a conjunction as a set of formulas. So their order is irrelevant and we write $\varphi' \subseteq \varphi$ iff all conjuncts of φ' are also conjuncts of φ. The empty conjunction is TRUE (i.e., $\mathit{TRUE} \Rightarrow \psi$ can be simplified to ψ).

IV. **Delete Conditions**

$$\frac{\varphi \;\Rightarrow\; \psi}{\varphi' \;\Rightarrow\; \psi} \quad \text{if } \varphi' \subseteq \varphi$$

Rule (IV) is especially useful for omitting conditions $q = x$ where x is a variable which does not occur anywhere else. So one could also transform (17) to

[9] To remove the condition $q = x$, we must ensure that for any normal $\mathcal{F}$-substitution δ, the term $q\delta$ is normal, too. Otherwise, Rule (III) would not be sound, cf. [11].

(18) by Rule (IV). The meaning of (18) is that $\mathsf{MINUS}(x,y)\sigma \succsim \mathsf{c}$ must hold whenever $\mathsf{gt}(x,y)\sigma$ is innermost terminating and $\mathsf{gt}(x,y)\sigma \xrightarrow{\mathsf{i}}^*_{\mathcal{R}} \mathsf{true}$ holds for a normal $\mathcal{F}$-substitution σ. To simplify this constraint further, the next inference rule performs an *induction* on the length of $\mathsf{gt}(x,y)\sigma$'s reduction.[10] Since $\mathsf{gt}(x,y)$ and true do not unify, at least one reduction step is needed, i.e., some rule $\mathsf{gt}(\ell_1,\ell_2) \to r$ must be applied. To detect all possibilities for the first reduction step, we consider all *narrowings* of the term $\mathsf{gt}(x,y)$. We obtain

$$\mathsf{gt}(x,y) \rightsquigarrow_{[x/\mathsf{0},y/v]} \mathsf{false}, \quad \mathsf{gt}(x,y) \rightsquigarrow_{[x/\mathsf{s}(u),y/\mathsf{0}]} \mathsf{true}, \quad \mathsf{gt}(x,y) \rightsquigarrow_{[x/\mathsf{s}(u),y/\mathsf{s}(v)]} \mathsf{gt}(u,v)$$

Thus, we could replace (18) by the following three new constraints where we always apply the respective narrowing substitution to the whole constraint:

$$\begin{aligned} \mathsf{false} = \mathsf{true} \quad &\Rightarrow \quad \mathsf{MINUS}(\mathsf{0},v) \succsim \mathsf{c} && (19)\\ \mathsf{true} = \mathsf{true} \quad &\Rightarrow \quad \mathsf{MINUS}(\mathsf{s}(u),\mathsf{0}) \succsim \mathsf{c} && (20)\\ \mathsf{gt}(u,v) = \mathsf{true} \quad &\Rightarrow \quad \mathsf{MINUS}(\mathsf{s}(u),\mathsf{s}(v)) \succsim \mathsf{c} && (21) \end{aligned}$$

So to transform a constraint $f(x_1,\ldots,x_n) = q \wedge \varphi \;\Rightarrow\; \psi$, we consider all rules $f(\ell_1,\ldots,\ell_n) \to r$. Then the constraint could be replaced by the new constraints

$$r = q\sigma \wedge \varphi\sigma \;\Rightarrow\; \psi\sigma, \qquad \text{where } \sigma = [x_1/\ell_1,\ldots,x_n/\ell_n]. \tag{22}$$

However, we perform a better transformation. Suppose that r contains a recursive call, i.e., a subterm $f(r_1,\ldots,r_n)$, and that the r_i do not contain defined symbols. Obviously, $f(r_1,\ldots,r_n)\sigma$'s reduction is shorter than the reduction of $f(x_1,\ldots,x_n)\sigma$. Thus for $\mu = [x_1/r_1,\ldots,x_n/r_n]$ one can assume

$$\forall y_1,\ldots,y_m \quad f(r_1,\ldots,r_n) = q\mu \wedge \varphi\mu \;\Rightarrow\; \psi\mu \tag{23}$$

as *induction hypothesis* when requiring (22).[11] Here, $y_1,\ldots,y_m$ are all occurring variables except those in r. Of course, we may assume that variables in rewrite rules (i.e., in r) are disjoint from variables in constraints (i.e., in q, φ, and ψ). So instead of (22), it suffices to demand (23) $\Rightarrow$ (22), or equivalently

$$r = q\sigma \wedge \varphi\sigma \wedge (23) \;\Rightarrow\; \psi\sigma. \tag{24}$$

This leads to Rule (V). Here, $x_1,\ldots,x_n$ denote pairwise different variables.

[10] More precisely, we use an induction on $\xrightarrow{\mathsf{i}}_{\mathcal{R}} \circ \trianglerighteq$, where $\trianglerighteq$ is the subterm relation. The idea for this inference rule was inspired by our earlier work on termination of simple first-order functional programs [3]. But [3] only considered a very restricted form of functional programs (left-linear, sufficiently complete, non-overlapping constructor systems without defined symbols in arguments of recursive calls), whereas we regard arbitrary TRSs. Moreover, we integrate this idea of performing induction into the whole framework of termination techniques and tools available for TRSs. Finally, in contrast to [3], we do not need an underlying induction theorem prover. Nevertheless, our approach is significantly stronger (e.g., [3] fails on examples like Ex. 12, cf. [11]).

[11] If there are more recursive calls in r, then one can obtain a corresponding induction hypothesis (23) for each recursive call. But similar to Footnote 9, if the r_i contain defined symbols, then one may not assume (23) as induction hypothesis.

V. **Induction (Defined Symbol with Pairwise Different Variables)**

$$\frac{f(x_1,...,x_n) = q \quad \wedge \; \varphi \qquad\qquad \Rightarrow \;\; \psi}{\displaystyle\bigwedge_{f(\ell_1,...,\ell_n)\to r\in\mathcal{R}} \quad (\; r = q\,\sigma \wedge \varphi\,\sigma \wedge \varphi' \;\; \Rightarrow \;\; \psi\,\sigma \;)}$$

if f is a defined symbol and $f(x_1,...,x_n)$ does not unify with q

where $\sigma = [x_1/\ell_1, ..., x_n/\ell_n]$

$$\text{and} \quad \varphi' = \begin{cases} \forall y_1,...,y_m \;\; f(r_1,\ldots,r_n) = q\mu \wedge \varphi\,\mu \Rightarrow \psi\,\mu, \quad \text{if} \\ \qquad\qquad \bullet\; r \text{ contains the subterm } f(r_1,...,r_n), \\ \qquad\qquad \bullet\; \text{there is no defined symbol in any } r_i, \\ \qquad\qquad \bullet\; \mu = [x_1/r_1,...,x_n/r_n], \text{ and} \\ \qquad\qquad \bullet\; y_1,...,y_m \text{ are all occurring variables except } \mathcal{V}(r) \\ \mathit{TRUE}, \quad \text{otherwise} \end{cases}$$

In our example, the above rule transforms the original constraint (18) into the three new constraints (19), (20), and (25). Here, (25) is obtained from the narrowing step $\mathsf{gt}(x,y) \rightsquigarrow_{[x/\mathsf{s}(u),y/\mathsf{s}(v)]} \mathsf{gt}(u,v)$, i.e., we have $\sigma = [x/\mathsf{s}(u), y/\mathsf{s}(v)]$, $r_1 = u$, $r_2 = v$, and $\mu = [x/u, y/v]$. There are no variables $y_1,\ldots,y_m$.

$$\begin{aligned} &\mathsf{gt}(u,v) = \mathsf{true} \\ \wedge\; &(\mathsf{gt}(u,v) = \mathsf{true} \;\Rightarrow\; \mathsf{MINUS}(u,v) \succsim \mathsf{c}) \quad \Rightarrow \quad \mathsf{MINUS}(\mathsf{s}(u),\mathsf{s}(v)) \succsim \mathsf{c} \end{aligned} \tag{25}$$

To simplify (25) further, now we can "apply" the induction hypothesis, since its condition $\mathsf{gt}(u,v) = \mathsf{true}$ is guaranteed to hold. So we can transform (25) to

$$\mathsf{gt}(u,v) = \mathsf{true} \;\wedge\; \mathsf{MINUS}(u,v) \succsim \mathsf{c} \;\Rightarrow\; \mathsf{MINUS}(\mathsf{s}(u),\mathsf{s}(v)) \succsim \mathsf{c}. \tag{26}$$

In general, to simplify conditions one may of course also instantiate universally quantified variables.[12] This leads to the following rule.

VI. **Simplify Condition**

$$\frac{\varphi \;\wedge\; (\forall y_1,\ldots,y_m \;\; \varphi' \Rightarrow \psi' \;) \;\;\Rightarrow\;\; \psi}{\varphi \;\wedge \qquad\qquad\qquad\qquad \psi'\,\sigma \;\;\Rightarrow\;\; \psi}$$

if $DOM(\sigma) \subseteq \{y_1,\ldots,y_m\}$, there is no defined symbol and no tuple symbol in any $\sigma(y_i)$, and $\varphi'\,\sigma \subseteq \varphi$

To simplify the remaining constraints (19), (20), and (26), note that (19) can be eliminated by Rule (I) since it has an unsatisfiable condition $\mathsf{false} = \mathsf{true}$. Moreover, Rule (II) can delete the trivial condition $\mathsf{true} = \mathsf{true}$ of the constraint (20). For (26), with Rule (IV) one can of course always omit conditions like $\mathsf{gt}(u,v) = \mathsf{true}$ from conditional constraints. In this way, all conditions with equalities $p = q$ are removed in the end.

So to finish the termination proof of Ex. 1, we can include the DP (7) in $\mathcal{P}_{bound}$ if the constraints $\mathsf{MINUS}(\mathsf{s}(u),\mathsf{0}) \succsim \mathsf{c}$ and $\mathsf{MINUS}(u,v) \succsim \mathsf{c} \Rightarrow \mathsf{MINUS}(\mathsf{s}(u),\mathsf{s}(v)) \succsim \mathsf{c}$ are satisfied. Of course, these constraints obviously hold for $\mathcal{P}ol_2$ if we choose $\mathsf{c}_{\mathcal{P}ol_2} = 1$. Then the DP (9) is strictly decreasing and (7) is bounded from below and thus, the reduction pair processor transforms the remaining DP problem $\{(7),(9)\}$ into $\{(7)\}$ and $\{(9)\}$. Now the resulting DP

[12] As in Footnote 9, one may only instantiate them by terms without defined symbols.

problems are easy to solve and thus, innermost termination of Ex. 1 is proved.

The rules (I) - (VI) are not always sufficient to exploit the conditions of a constraint. We demonstrate this with the following example.

Example 12. We regard a TRS $\mathcal{R}$ containing the gt*-rules (4) - (6) together with*

$$\begin{array}{rlrl} \mathsf{plus}(n,\mathsf{0}) \to n & & \mathsf{f}(\mathsf{true},x,y,z) \to \mathsf{f}(\mathsf{gt}(x,\mathsf{plus}(y,z)),x,\mathsf{s}(y),z) \\ \mathsf{plus}(n,\mathsf{s}(m)) \to \mathsf{s}(\mathsf{plus}(n,m)) & & \mathsf{f}(\mathsf{true},x,y,z) \to \mathsf{f}(\mathsf{gt}(x,\mathsf{plus}(y,z)),x,y,\mathsf{s}(z)) \end{array}$$

The termination of gt and of plus is easy to show. So the initial DP problem can easily be transformed into $\{(27),(28)\}$ with

$$\mathsf{F}(\mathsf{true},x,y,z) \to \mathsf{F}(\mathsf{gt}(x,\mathsf{plus}(y,z)),x,\mathsf{s}(y),z) \tag{27}$$
$$\mathsf{F}(\mathsf{true},x,y,z) \to \mathsf{F}(\mathsf{gt}(x,\mathsf{plus}(y,z)),x,y,\mathsf{s}(z)) \tag{28}$$

To include (27) in $\mathcal{P}_{bound}$, we have to impose the following constraint:

$$\mathsf{F}(\mathsf{gt}(x,\mathsf{plus}(y,z)),x,\mathsf{s}(y),z) = \mathsf{F}(\mathsf{true},x',y',z') \;\;\Rightarrow\;\; \mathsf{F}(\mathsf{true},x,y,z) \succsim \mathsf{c} \tag{29}$$

With the rules (II) and (IV), it can be transformed into

$$\mathsf{gt}(x,\mathsf{plus}(y,z)) = \mathsf{true} \;\;\Rightarrow\;\; \mathsf{F}(\mathsf{true},x,y,z) \succsim \mathsf{c} \tag{30}$$

Now we want to use induction. However, Rule (V) is only applicable for conditions $f(x_1,\ldots,x_n) = q$ where $x_1,\ldots,x_n$ are *pairwise different variables*. To obtain such conditions, we use the following rule. Here, x denotes a fresh variable.

VII. **Defined Symbol without Pairwise Different Variables**	
$\dfrac{f(p_1,\ldots,p_i,\ldots,p_n) = q \wedge \varphi \;\Rightarrow\; \psi}{p_i = x \wedge f(p_1,\ldots,\;x\;,\ldots,p_n) = q \wedge \varphi \;\Rightarrow\; \psi}$	if f is a defined symbol and ($p_i \notin \mathcal{V}$ or $p_i = p_j$ for a $j \neq i$)

So the constraint (30) is transformed into

$$\mathsf{plus}(y,z) = w \wedge \mathsf{gt}(x,w) = \mathsf{true} \;\Rightarrow\; \mathsf{F}(\mathsf{true},x,y,z) \succsim \mathsf{c}$$

Example 13. To continue, we can now perform induction on gt *which yields*

$$\mathsf{plus}(y,z) = v \wedge \mathsf{false} = \mathsf{true} \Rightarrow \mathsf{F}(\mathsf{true},\mathsf{0},y,z) \succsim \mathsf{c} \tag{31}$$
$$\mathsf{plus}(y,z) = \mathsf{0} \wedge \mathsf{true} = \mathsf{true} \Rightarrow \mathsf{F}(\mathsf{true},\mathsf{s}(u),y,z) \succsim \mathsf{c} \tag{32}$$
$$\mathsf{plus}(y,z) = \mathsf{s}(v) \wedge \mathsf{gt}(u,v) = \mathsf{true} \wedge (34) \Rightarrow \mathsf{F}(\mathsf{true},\mathsf{s}(u),y,z) \succsim \mathsf{c} \tag{33}$$

Here, (34) is the induction hypothesis:

$$\forall y,z \;\; \mathsf{plus}(y,z) = v \wedge \mathsf{gt}(u,v) = \mathsf{true} \;\Rightarrow\; \mathsf{F}(\mathsf{true},u,y,z) \succsim \mathsf{c} \tag{34}$$

With Rule (I) we delete constraint (31) and Rule (II) simplifies constraint (32) to "$\mathsf{plus}(y,z) = \mathsf{0} \;\Rightarrow\; \mathsf{F}(\mathsf{true},\mathsf{s}(u),y,z) \succsim \mathsf{c}$*". Similar to our previous example, by induction via* plus *and by removing the constraint with the unsatisfiable condition* $\mathsf{s}(\mathsf{plus}(n,m)) = \mathsf{0}$*, we finally transform it to*

$$\mathsf{F}(\mathsf{true},\mathsf{s}(u),\mathsf{0},\mathsf{0}) \succsim \mathsf{c} \tag{35}$$

The other constraint (33) is simplified further by induction via plus *as well:*

$$n = \mathsf{s}(v) \wedge \mathsf{gt}(u, v) = \mathsf{true} \wedge (34) \Rightarrow \mathsf{F}(\mathsf{true}, \mathsf{s}(u), n, \mathsf{0}) \succsim \mathsf{c} \tag{36}$$

$$\mathsf{s}(\mathsf{plus}(n, m)) = \mathsf{s}(v) \wedge \mathsf{gt}(u, v) = \mathsf{true} \wedge (34) \wedge \varphi' \Rightarrow \mathsf{F}(\mathsf{true}, \mathsf{s}(u), n, \mathsf{s}(m)) \succsim \mathsf{c} \tag{37}$$

where φ' is the new induction hypothesis. We apply Rules (III) and (IV) on (36) to obtain "$\mathsf{gt}(u, v) = \mathsf{true} \Rightarrow \mathsf{F}(\mathsf{true}, \mathsf{s}(u), \mathsf{s}(v), \mathsf{0}) \succsim \mathsf{c}$". By another induction on gt *and by applying Rules (I), (II), (IV), and (VI) we get the final constraints*

$$\mathsf{F}(\mathsf{true}, \mathsf{s}(\mathsf{s}(i)), \mathsf{s}(\mathsf{0}), \mathsf{0}) \succsim \mathsf{c} \tag{38}$$

$$\mathsf{F}(\mathsf{true}, \mathsf{s}(i), \mathsf{s}(j), \mathsf{0}) \succsim \mathsf{c} \Rightarrow \mathsf{F}(\mathsf{true}, \mathsf{s}(\mathsf{s}(i)), \mathsf{s}(\mathsf{s}(j)), \mathsf{0}) \succsim \mathsf{c} \tag{39}$$

In the only remaining constraint (37) we delete φ' with Rule (IV) and by removing the outermost s *in the first condition with Rule (II), we get*

$$\mathsf{plus}(n, m) = v \wedge \mathsf{gt}(u, v) = \mathsf{true} \wedge (34) \Rightarrow \mathsf{F}(\mathsf{true}, \mathsf{s}(u), n, \mathsf{s}(m)) \succsim \mathsf{c}$$

Now we can simplify the condition by applying the induction hypothesis (34). In (34), the variables y and z were universally quantified. We instantiate y with n and z with m. With Rule (VI) we replace (34) by the new condition $\mathsf{F}(\mathsf{true}, u, n, m) \succsim \mathsf{c}$. By deleting the first two remaining conditions we finally get

$$\mathsf{F}(\mathsf{true}, u, n, m) \succsim \mathsf{c} \Rightarrow \mathsf{F}(\mathsf{true}, \mathsf{s}(u), n, \mathsf{s}(m)) \succsim \mathsf{c} \tag{40}$$

So to summarize, the constraint (29) can be transformed into (35), (38), (39), and (40). These constraints are satisfied by the interpretation $\mathcal{P}ol$ where $\mathsf{F}_{\mathcal{P}ol} = x_2 - x_3 - x_4$, $\mathsf{s}_{\mathcal{P}ol} = x_1 + 1$, $\mathsf{0}_{\mathcal{P}ol} = 0$, and $\mathsf{c}_{\mathcal{P}ol} = 1$. Therefore, we can include the DP (27) in $\mathcal{P}_{bound}$. For a similar reason, the other DP (28) is also bounded. Moreover, both DPs are strictly decreasing and there are no usable rules since F *is not $\succsim_{\mathcal{P}ol}$-dependent on 1. Hence, the reduction pair processor can remove both DPs and innermost termination of Ex. 12 is proved.*

We define $\varphi \vdash \varphi'$ iff φ' results from φ by repeatedly applying the above inference rules to the conjuncts of φ. Thm. 14 states that these rules are sound.

Theorem 14 (Soundness). *If $\varphi \vdash \varphi'$, then $\models \varphi'$ implies $\models \varphi$.*

With Thm. 14 we can now refine the reduction pair processor from Thm. 11.

Corollary 15 (Conditional General Reduction Pair Processor with Inference). *Let $(\succsim, \succ)$ be a general reduction pair and let* c *be a fresh constant. For every $s \to t \in \mathcal{P}$ and every inequality $\psi \in \{ s \succ t,\ s \succsim t,\ s \succsim \mathsf{c} \}$, let φ_ψ be a constraint with $\bigwedge_{u \to v \in \mathcal{P}} (t = u' \;\Rightarrow\; \psi) \;\vdash\; \varphi_\psi$. Here, u' results from u by renaming its variables into fresh variables. Then the processor Proc from Thm. 11 is still sound if we define $\mathcal{P}_\succ = \{ s \to t \in \mathcal{P} \mid\ \models \varphi_{s \succ t} \}$, $\mathcal{P}_\succsim = \{ s \to t \in \mathcal{P} \mid\ \models \varphi_{s \succsim t} \}$, and $\mathcal{P}_{bound} = \{ s \to t \in \mathcal{P} \mid\ \models \varphi_{s \succsim \mathsf{c}} \}$.*

For automation, one of course needs a strategy for the application of the rules (I) - (VII). Essentially, we propose to apply the rules with the priority (I), (II), (IV)′, (VI), (III), (VII), (V), cf. [11]. Here, (IV)′ is a restriction of (IV) which only deletes conditions $q = x$ where x is a variable which does not occur anywhere else. Moreover, to ensure termination of the inference rules, one has to impose a limit on the number of inductions with Rule (V). In the end, we use Rule (IV) to remove all remaining conditions containing "=" or "⇒". Moreover, if there are several conditions of the form $s \,_{(}\!\succsim\!_{)}\, t$, we remove all but one of them.

Thus, the constraints φ_ψ in Cor. 15 are conjunctions where the conjuncts have the form "$t_1 \,_{(}\!\succsim\!_{)}\, t_2$" or "$s_1 \,_{(}\!\succsim\!_{)}\, s_2 \Rightarrow t_1 \,_{(}\!\succsim\!_{)}\, t_2$". However, most existing methods and tools for the generation of orders and of polynomial interpretations can only handle *unconditional* inequalities [4, 7]. To transform such conditional constraints into unconditional ones, note that any constraint "$s \succsim \mathsf{c} \Rightarrow t \succsim \mathsf{c}$" can be replaced by "$t \succsim s$". More generally, if one uses polynomial orders, then any constraint "$s_1 \,_{(}\!\succsim\!_{)}\, s_2 \Rightarrow t_1 \,_{(}\!\succsim\!_{)}\, t_2$" can be replaced by "$[t_1] - [t_2] \geq [s_1] - [s_2]$". So in Ex. 13, instead of (39) and (40), we would require $[\mathsf{F}(\mathsf{true}, \mathsf{s}(\mathsf{s}(i)), \mathsf{s}(\mathsf{s}(j)), \mathsf{0})] \geq [\mathsf{F}(\mathsf{true}, \mathsf{s}(i), \mathsf{s}(j), \mathsf{0})]$ and $[\mathsf{F}(\mathsf{true}, \mathsf{s}(u), n, \mathsf{s}(m))] \geq [\mathsf{F}(\mathsf{true}, u, n, m)]$.

In practice, it is not recommendable to fix the reduction pair $(\succsim, \succ)$ in advance and to check the validity of the constraints of the reduction pair processor afterwards. Instead, one should leave the reduction pair open and first simplify the constraints of the reduction pair processor using the above inference rules. Afterwards, one uses the existing techniques to generate a reduction pair (e.g., based on integer polynomial interpretations) satisfying the resulting constraints.

More precisely, we start the following procedure $\mathsf{REDUCTION\_PAIR}(\mathcal{P})$ with $\mathcal{P} = DP(\mathcal{R})$. If $\mathsf{REDUCTION\_PAIR}(DP(\mathcal{R}))$ returns "*Yes*", then innermost termination is proved. Of course, this procedure can be refined by also applying other DP processors than just the reduction pair processor to $\mathcal{P}$.

Procedure $\mathsf{REDUCTION\_PAIR}(\mathcal{P})$

1. If $\mathcal{P} = \varnothing$ then stop and return "*Yes*".
2. Choose non-empty subsets $\mathcal{P}_\succ \subseteq \mathcal{P}$ and $\mathcal{P}_{bound} \subseteq \mathcal{P}$. Let $\mathcal{P}_\succsim = \mathcal{P} \setminus \mathcal{P}_\succ$.
3. Generate the following constraint φ (where $\succsim$ and $\succ$ are *not yet fixed*):

$$\textstyle\bigwedge_{s\to t\in\mathcal{P}_\succ,\, u\to v\in\mathcal{P}} (t = u' \;\Rightarrow\; s \succ t) \wedge \bigwedge_{s\to t\in\mathcal{P}_{bound},\, u\to v\in\mathcal{P}} (t = u' \;\Rightarrow\; s \succsim \mathsf{c}) \wedge$$
$$\textstyle\bigwedge_{s\to t\in\mathcal{P}_\succsim,\, u\to v\in\mathcal{P}} (t = u' \;\Rightarrow\; s \succsim t) \wedge \bigwedge_{\ell\to r\in\mathcal{U}(\mathcal{P})} (\ell \succsim r)$$

4. Use Rules (I) - (VII) to transform φ to a constraint φ' without "=".
5. Generate an integer polynomial interpretation satisfying φ', cf. e.g. [7].
6. If $\mathsf{REDUCTION\_PAIR}(\mathcal{P}_\succsim)$ = "*Yes*" and $\mathsf{REDUCTION\_PAIR}(\mathcal{P} \setminus \mathcal{P}_{bound})$ = "*Yes*", then return "*Yes*". Otherwise, return "*Maybe*".

5 Conclusion

We have extended the reduction pair processor of the DP method in order to handle TRSs that terminate because of bounded increase. To be able to measure the *increase* of arguments, we permitted the use of general reduction pairs (e.g., based on integer polynomial interpretations). Moreover, to exploit the *bounds* given by conditions, we developed a calculus based on induction which simplifies the constraints needed for the reduction pair processor.

We implemented the new reduction pair processor of Cor. 15 in our termination prover AProVE [9]. To demonstrate the power of our method, [11] contains a collection of typical TRSs with bounded increase. These include examples with non-boolean (possibly nested) functions in the bound, examples with combinations of bounds, examples containing increasing or decreasing defined symbols, examples with bounds on lists, examples with different increases in different arguments, increasing TRSs that go beyond the shape of functional programs, etc. Although AProVE was the most powerful tool for termination analysis of TRSs in the *International Competition of Termination Tools*, up to now AProVE (as well as all other tools participating in the competition) failed on all TRSs from our collection. In contrast, with the results from this paper, the new version of AProVE can prove innermost termination for all of them. Thus, these results represent a substantial advance in automated termination proving. To experiment with our implementation, the new version of AProVE can be accessed via the web at `http://aprove.informatik.rwth-aachen.de/eval/Increasing/`.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge, 1998.
3. J. Brauburger and J. Giesl. Termination analysis by inductive evaluation. In *Proc. 15th CADE*, LNAI 1421, pages 254–269, 1998.
4. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *J. Aut. Reason.*, 34(4):325–363, 2005.
5. B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In *Proc. CAV '06*, LNCS 4144, pages 415–418, 2006.
6. M.-L. Fernández. Relaxing monotonicity for innermost termination. *Information Processing Letters*, 93(3):117–123, 2005.
7. 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, 2007. To appear.
8. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR'04*, LNAI 3452, pages 301–331, 2005.
9. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. *Proc. IJCAR '06*, LNAI 4130, p. 281–286, 2006.
10. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
11. J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. Technical Report AIB-2007-03, RWTH Aachen, 2007. Available from `http://aib.informatik.rwth-aachen.de`.
12. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
13. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
14. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
15. P. Manolios and D. Vroon. Termination analysis with calling context graphs. In *Proc. CAV '06*, LNCS 4144, pages 401–414, 2006.
16. C. Marché and H. Zantema. The termination competition. In *Proc. RTA '07*, 2007. To appear.