Maximal Termination*

Carsten Fuhs[1], Jürgen Giesl[1], Aart Middeldorp[2], Peter Schneider-Kamp[1], René Thiemann[2], and Harald Zankl[2]

[1] LuFG Informatik 2, RWTH Aachen University, Germany
[2] Institute of Computer Science, University of Innsbruck, Austria

Abstract. We present a new approach for termination proofs that uses polynomial interpretations (with possibly negative coefficients) together with the "maximum" function. To obtain a powerful automatic method, we solve two main challenges: (1) We show how to adapt the latest developments in the dependency pair framework to our setting. (2) We show how to automate the search for such interpretations by integrating "max" into recent SAT-based methods for polynomial interpretations. Experimental results support our approach.

1 Introduction

The use of *polynomial interpretations* [13] is standard in automated termination analysis of term rewrite systems (TRSs). This is especially true for termination proofs in the popular *dependency pair (DP) framework* [1,4,6,9] that is implemented in most automated termination tools for TRSs.

A *polynomial interpretation* $\mathcal{P}ol$ maps every n-ary function symbol f to a polynomial $f_{\mathcal{P}ol}$ over n variables $x_1, \ldots, x_n$. The mapping is extended to terms by defining $[x]_{\mathcal{P}ol} = x$ for variables x and $[f(t_1, ..., t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}([t_1]_{\mathcal{P}ol}, ..., [t_n]_{\mathcal{P}ol})$. If $\mathcal{P}ol$ is clear from the context, we also write $[t]$ instead of $[t]_{\mathcal{P}ol}$. Traditionally, one uses polynomials with *natural* coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$. Then $[t] \in \mathbb{N}$ for every ground term t. For example, consider the interpretation $\mathcal{P}ol$ with $\mathsf{0}_{\mathcal{P}ol} = 0$, $\mathsf{s}_{\mathcal{P}ol} = x_1 + 1$, and $\mathsf{minus}_{\mathcal{P}ol} = x_1$. Then $[\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y))]_{\mathcal{P}ol} = x + 1$.

An interpretation $\mathcal{P}ol$ induces an order $\succ_{\mathcal{P}ol}$ and quasi-order $\succsim_{\mathcal{P}ol}$ where $s \succ_{\mathcal{P}ol} t$ ($s \succsim_{\mathcal{P}ol} t$) iff $[s] > [t]$ ($[s] \geqslant [t]$) holds for all instantiations of variables with natural numbers. So with $\mathcal{P}ol$ above we have $\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succ_{\mathcal{P}ol} \mathsf{minus}(x, y)$. Recently, two extensions to *integer* polynomials were proposed:

(a) [7] used polynomial interpretations with integer coefficients where ground terms could also be mapped to arbitrary integers. However, this approach only works for analyzing *innermost* instead of *full* termination.
(b) [10] proposed interpretations of the form $\max(p, 0)$ where p is a polynomial with integer coefficients. Thus, ground terms are still mapped to numbers from $\mathbb{N}$. So one could define $\mathsf{minus}_{\mathcal{P}ol} = \max(x_1 - x_2, 0)$ which would result in $\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \approx_{\mathcal{P}ol} \mathsf{minus}(x, y)$. Here $\approx_{\mathcal{P}ol}$ denotes the equivalence relation associated with $\succsim_{\mathcal{P}ol}$, where for any quasi-order $\succsim$ we have $\approx \, = \, \succsim \cap \precsim$.

* *Proc. RTA '08*, LNCS, 2008. Supported by the DFG (Deutsche Forschungsgemeinschaft) grant GI 274/5-2 and the FWF (Austrian Science Fund) project P18763.

The drawback is that the approach of [10] was not easy to automate and that it could only be combined with a weak version of the DP technique.

In this paper, we present a new approach which improves upon (a) and (b):

- It uses integer polynomials together with the function "max", where ground terms are only mapped to natural numbers, as in [10]. But in contrast to [10], we permit arbitrary combinations of polynomials and "max", e.g., "$p + \max(q, \max(r, s))$" where p, q, r, s are integer polynomials. And in contrast to [7], integer polynomials may be used for interpreting *any* function symbol.
- It uses the newest and most powerful version of the DP technique as in [7].
- In contrast to [7], it can also prove *full* instead of *innermost* termination.
- In contrast to [10], we show how to search for arbitrary polynomial interpretations with "max" automatically in an efficient way using SAT solving.

After recapitulating the DP framework in Sect. 2, Sect. 3 extends it to handle *non-monotonic* quasi-orders like integer polynomial orders with "max". Sect. 4 shows how to search for such interpretations automatically using SAT solving. Sect. 5 discusses our implementation in the provers AProVE [5] and $\mathsf{T_TT_2}$ [17].

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 $\mathcal{R}$. We denote the set of all dependency pairs of $\mathcal{R}$ by $DP(\mathcal{R})$.

Example 1. Consider the TRS SUBST from [8] and [18, Ex. 6.5.42]:

$$
\begin{array}{lll}
\lambda(x) \circ y \to \lambda(x \circ (1 \star (y \circ \uparrow))) & \mathsf{id} \circ x \to x & 1 \circ (x \star y) \to x \\
(x \star y) \circ z \to (x \circ z) \star (y \circ z) & 1 \circ \mathsf{id} \to 1 & \uparrow \circ (x \star y) \to y \\
(x \circ y) \circ z \to x \circ (y \circ z) & \uparrow \circ \mathsf{id} \to \uparrow &
\end{array}
$$

The dependency pairs are

$$
\begin{array}{lll}
\lambda(x) \circ^\sharp y \to x \circ^\sharp (1 \star (y \circ \uparrow)) \quad (1) & & (x \star y) \circ^\sharp z \to y \circ^\sharp z \\
\lambda(x) \circ^\sharp y \to y \circ^\sharp \uparrow \quad (2) & & (x \circ y) \circ^\sharp z \to x \circ^\sharp (y \circ z) \\
(x \star y) \circ^\sharp z \to x \circ^\sharp z & & (x \circ y) \circ^\sharp z \to y \circ^\sharp z
\end{array}
$$

The main result of the DP framework states that a TRS $\mathcal{R}$ is terminating iff there is no infinite *minimal* $DP(\mathcal{R})$*-chain*. For any set of dependency pairs $\mathcal{P}$, a *minimal* $\mathcal{P}$*-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 $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ and where all $t_i\sigma$ are terminating w.r.t. $\mathcal{R}$.

The DP framework has several techniques (so-called *DP processors*) to prove absence of infinite chains. Thm. 2 recapitulates one of the most important processors, the so-called *reduction pair processor*. It uses *reduction pairs* $(\succsim, \succ)$ to compare terms. Here, $\succsim$ is a stable monotonic quasi-order and $\succ$ is a stable well-founded order, where $\succsim$ and $\succ$ are compatible (i.e., $\succ \circ \succsim \subseteq \succ$ or $\succsim \circ \succ \subseteq \succ$).

If $\mathcal{P}$ is the current set of dependency pairs,[3] then 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 from $\mathcal{P}$. Afterwards, the reduction pair processor can be applied again to the remaining set of DPs (possibly using a different reduction pair). This process is repeated until all DPs have been removed.

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. To ensure that it suffices to regard only the *usable* rules instead of *all* rules in the reduction pair processor, one has to demand that $\succsim$ is $\mathcal{C}_\varepsilon$*-compatible*, i.e., that $\mathsf{c}(x,y) \succsim x$ and $\mathsf{c}(x,y) \succsim y$ holds for a fresh function symbol c [6,10]. This requirement is satisfied by virtually all quasi-orders used in practice.[4]

Theorem 2 ([6,10]). *Let $(\succsim, \succ)$ be a reduction pair where $\succsim$ is $\mathcal{C}_\varepsilon$-compatible. Then the following DP processor Proc is sound (i.e., if there is no infinite minimal Proc($\mathcal{P}$)-chain, then there is also no infinite minimal $\mathcal{P}$-chain):*

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

For any function symbol f, let $Rls(f) = \{\ell \to r \in \mathcal{R} \mid \mathrm{root}(\ell) = f\}$. For any term t, the usable rules *$\mathcal{U}(t)$ are the smallest set such that*

$$\mathcal{U}(f(t_1,\ldots,t_n)) = Rls(f) \;\cup\; \bigcup\nolimits_{\ell \to r \in Rls(f)} \mathcal{U}(r) \;\cup\; \bigcup\nolimits_{i=1}^{n} \mathcal{U}(t_i)$$

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

Example 3. For the TRS of Ex. 1, we use the reduction pair $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ with

$$\lambda_{\mathcal{P}ol} = x_1 + 1 \qquad \star_{\mathcal{P}ol} = \max(x_1, x_2)$$
$$\circ_{\mathcal{P}ol} = \circ^\sharp_{\mathcal{P}ol} = x_1 + x_2 \qquad \mathsf{1}_{\mathcal{P}ol} = \mathsf{id}_{\mathcal{P}ol} = \uparrow_{\mathcal{P}ol} = 0$$

Then all (usable) rules and dependency pairs are weakly decreasing (w.r.t. $\succsim_{\mathcal{P}ol}$). Furthermore, the DPs (1) and (2) are strictly decreasing (w.r.t. $\succ_{\mathcal{P}ol}$) and can be removed by Thm. 2. Afterwards, we use the following interpretation where the remaining DPs are strictly decreasing and the rules are still weakly decreasing:

$$\circ^\sharp_{\mathcal{P}ol} = x_1 \qquad \star_{\mathcal{P}ol} = \max(x_1, x_2) + 1$$
$$\circ_{\mathcal{P}ol} = x_1 + x_2 + 1 \qquad \lambda_{\mathcal{P}ol} = \mathsf{1}_{\mathcal{P}ol} = \mathsf{id}_{\mathcal{P}ol} = \uparrow_{\mathcal{P}ol} = 0$$

Termination of SUBST cannot be proved with Thm. 2 using reduction pairs based on linear polynomial interpretations, cf. [3]. Thus, this example shows the

[3] For readability, we consider sets of DPs instead of *DP problems* [4]. This suffices to present our new results, since the DP processors of this paper only modify the DPs.

[4] An exception are *equivalences* like $\approx$, which are usually not $\mathcal{C}_\varepsilon$-compatible [10].

usefulness of polynomial interpretations with "max". Up to now, only restricted forms of such interpretations were available in termination tools. For example, already in 2004, T$_\mathsf{T}$T used interpretations like $\max(x_1 - x_2, 0)$, but no tool offered arbitrary interpretations with polynomials and "max" like $\max(x_1, x_2) + 1$.

While SUBST's original termination proof was very complicated [8], easier proofs were developed later, using the techniques of *distribution elimination* or *semantic labeling* [18]. Indeed, the only tool that could prove termination of SUBST automatically up to now (TPA [12]) used semantic labeling.[5] In contrast, Ex. 3 shows that there is an even simpler proof without semantic labeling.

3 Termination With Integer Polynomials and "max"

Our aim is to use polynomial interpretations with *integer* polynomials, together with the function "max". More precisely, we want to use interpretations that map n-ary function symbols to arbitrary functions from $\mathbb{N}^n \to \mathbb{N}$. But Ex. 4 demonstrates that such interpretations may not be used in Thm. 2, since then $\succsim_{\mathcal{P}ol}$ is not monotonic, and thus, $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is not a reduction pair.

Example 4. Consider this non-terminating TRS (inspired by [7, Ex. 4]):

$$\mathsf{f}(\mathsf{s}(x), x) \to \mathsf{f}(\mathsf{s}(x), \mathsf{round}(x))$$
$$\mathsf{round}(\mathsf{0}) \to \mathsf{0} \qquad\qquad \mathsf{round}(\mathsf{s}(\mathsf{0})) \to \mathsf{s}(\mathsf{0})$$
$$\mathsf{round}(\mathsf{0}) \to \mathsf{s}(\mathsf{0}) \qquad\qquad \mathsf{round}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{s}(\mathsf{round}(x)))$$

Here, $\mathsf{round}(x)$ evaluates to x if x is odd and to x or $\mathsf{s}(x)$ otherwise. We use the interpretation $\mathcal{P}ol$ with $\mathsf{F}_{\mathcal{P}ol} = x_1 + \max(x_1 - x_2, 0)$, $\mathsf{ROUND}_{\mathcal{P}ol} = x_1$, $\mathsf{0}_{\mathcal{P}ol} = 0$, and $\mathsf{s}_{\mathcal{P}ol} = \mathsf{round}_{\mathcal{P}ol} = x_1 + 1$, where F and ROUND are the tuple symbols for f and round, respectively. Then all DPs are strictly decreasing and the usable round-rules are weakly decreasing. So if we were allowed to use $\mathcal{P}ol$ in Thm. 2, then we could remove all DPs and falsely prove termination.

Ex. 4 shows the reason for unsoundness when dropping the requirement of monotonicity of $\succsim$. Thm. 2 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. For instance in Ex. 4, we have $\mathsf{round}(\mathsf{0}) \succsim_{\mathcal{P}ol} \mathsf{0}$, but $\mathsf{F}(\mathsf{s}(\mathsf{0}), \underline{\mathsf{round}(\mathsf{0})}) \not\succsim_{\mathcal{P}ol} \mathsf{F}(\mathsf{s}(\mathsf{0}), \underline{\mathsf{0}})$.

In [10], this problem was solved by requiring $\ell \approx r$ instead of $\ell \succsim r$. Then such rules are not just weakly decreasing but *equivalent* w.r.t. $\succsim$. This requirement is not satisfied in Ex. 4 as $\mathsf{round}(\mathsf{0}) \not\approx_{\mathcal{P}ol} \mathsf{0}$. In general, this equivalence even has to be required for *all* rules $\ell \to r$ (not just the usable ones), since the step from *all* rules to the *usable* rules in the proof of Thm. 2 also relies on the monotonicity of $\succsim$. Thus, up to now one had to apply the following reduction pair processor when using non-monotonic reduction pairs. The soundness of this processor immediately results from [4, Thm. 28] and [10, Thm. 23 and Cor. 31],

[5] For the semantic labeling, TPA uses only a (small) fixed set of functions, including certain fixed polynomials and the function "max". So in contrast to our automation in Sect. 4, TPA does not search for arbitrary combinations of polynomials and "max".

cf. [3].[6] Here, a *non-monotonic* reduction pair $(\succsim, \succ)$ consists of a stable quasi-order $\succsim$ and a compatible stable well-founded order $\succ$. But we do not require monotonicity of $\succsim$ (and $\succsim$ does not have to be $\mathcal{C}_\varepsilon$-compatible either). However, the equivalence relation $\approx$ associated with $\succsim$ must be monotonic.[7]

Theorem 5. *Let $(\succsim, \succ)$ be a non-monotonic reduction pair. Then Proc is sound:*

$$Proc(\mathcal{P}) = \begin{cases} \mathcal{P} \setminus \succ & \text{if } \mathcal{P} \subseteq \succ \cup \succsim \text{ and (a) or (b) holds:} \\ & \text{(a) } \mathcal{P} \cup \mathcal{U}(\mathcal{P}) \text{ is non-duplicating and } \mathcal{U}(\mathcal{P}) \subseteq \approx \\ & \text{(b) } \mathcal{R} \subseteq \approx \\ \mathcal{P} & \text{otherwise} \end{cases}$$

However, demanding $\ell \approx r$ for the usable rules as in Thm. 5(a) is a very strong requirement which makes the termination proof fail in many examples, cf. Ex. 11 and 12. Therefore, as already suggested in [7], one should take into account on which positions the quasi-order $\succsim$ is monotonically *increasing* resp. *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 is at a *decreasing* position, one requires $r \succsim \ell$. Finally, if f is at a position which is neither increasing nor decreasing, one requires $\ell \approx r$.

To modify our definition of usable rules accordingly, we need a *monotonicity specification* which specifies which arguments of a symbol have to be increasing ("$\Uparrow$") or decreasing ("$\Downarrow$"). Afterwards, we search for a (non-monotonic) reduction pair that is *compatible* with the monotonicity specification.

Definition 6. *A* monotonicity specification *is a mapping ν which assigns to every function symbol f and every $i \in \{1, ..., \text{arity}(f)\}$ a subset of $\{\Uparrow, \Downarrow\}$. A reduction pair $(\succsim, \succ)$ is* ν-compatible *iff*

- *if $\Uparrow \in \nu(f, i)$ then $\succsim$ is* monotonically increasing *on f's i-th argument, i.e., $t_i \succsim s_i$ implies $f(t_1, ..., t_i, ..., t_n) \succsim f(t_1, ..., s_i, ..., t_n)$ for all terms $t_1, ..., t_n, s_i$*
- *if $\Downarrow \in \nu(f, i)$ then $\succsim$ is* monotonically decreasing *on f's i-th argument, i.e., $t_i \succsim s_i$ implies $f(t_1, ..., t_i, ..., t_n) \precsim f(t_1, ..., s_i, ..., t_n)$ for all terms $t_1, ..., t_n, s_i$*
- *if $\nu(f, i) = \{\Uparrow, \Downarrow\}$ then*[8] *additionally $\succsim$ must be* independent *on f's i-th argument, i.e., $f(t_1, ..., t_i, ..., t_n) \approx f(t_1, ..., s_i, ..., t_n)$ for all terms $t_1, ..., t_n, s_i$*

We call f ν-dependent *on its i-th argument iff $\nu(f, i) \neq \{\Uparrow, \Downarrow\}$. The concept of* monotonicity *can be extended to positions in a term where $\nu(t, \varepsilon) = \{\Uparrow\}$ and*

[6] An alternative to Thm. 5(a) is presented in [10, Thm. 40] for reduction pairs $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ based on polynomial interpretations. Here, "non-duplication of $\mathcal{P} \cup \mathcal{U}(\mathcal{P})$" is replaced by "$\mathcal{P}ol$-right-linearity of $\mathcal{P} \cup \mathcal{U}(\mathcal{P})$". So for every right-hand side r there must be a linear term r' with $r \approx_{\mathcal{P}ol} r'$ where r' differs from r only in the variables.

[7] Triples like $(\approx, \succsim, \succ)$ were called "reduction triples" in [10]. "Non-monotonic reduction pairs" are also related to the "general reduction pairs" in [7], but there $\succ$ did not have to be well founded. Consequently, the notion of stability was weakened too.

[8] Note that this condition is implied by the first two conditions whenever $\succsim$ is total on ground terms and whenever $s\sigma \succsim t\sigma$ for all ground substitutions σ implies $s \succsim t$.

$$\nu(f(t_1,...,t_n), i\,p) = \begin{cases} \{\Uparrow,\Downarrow\} & \textit{if } \nu(f,i) = \{\Uparrow,\Downarrow\} \textit{ or } \nu(t_i,p) = \{\Uparrow,\Downarrow\} \\ \{\Uparrow\} & \textit{if } \nu(f,i) = \nu(t_i,p) = \{\Uparrow\} \textit{ or } \nu(f,i) = \nu(t_i,p) = \{\Downarrow\} \\ \{\Downarrow\} & \textit{if either } \nu(f,i) = \{\Uparrow\} \textit{ and } \nu(t_i,p) = \{\Downarrow\} \\ & \qquad \textit{or } \nu(f,i) = \{\Downarrow\} \textit{ and } \nu(t_i,p) = \{\Uparrow\} \\ \varnothing & \textit{otherwise} \end{cases}$$

*A position p in a term t is called ν-*dependent *iff $\nu(t,p) \neq \{\Uparrow,\Downarrow\}$.*

Definition 7 (General Usable Rules [7]). *Let ν be a monotonicity specification. For any TRS U, we define $U^{\{\Uparrow,\Downarrow\}} = \varnothing$, $U^{\{\Uparrow\}} = U$, $U^{\{\Downarrow\}} = U^{-1} = \{r \to \ell \mid \ell \to r \in U\}$, and $U^{\varnothing} = U \cup U^{-1}$. For any term t, we define the* general usable rules *$\mathcal{GU}(t)$ as the smallest set such that*[9]

$$\mathcal{GU}(f(t_1,\ldots,t_n)) = Rls(f) \;\cup\; \bigcup\nolimits_{\ell\to r\in Rls(f)} \mathcal{GU}(r) \;\cup\; \bigcup\nolimits_{i=1}^{n} \mathcal{GU}^{\nu(f,i)}(t_i)$$

For a set of DPs $\mathcal{P}$, we define $\mathcal{GU}(\mathcal{P}) = \bigcup_{s\to t\in\mathcal{P}} \mathcal{GU}(t)$. Moreover, we let $\mathcal{U}^{contr}(t)$ be those rules of $\mathcal{R}$ that contributed to $\mathcal{GU}(t)$, i.e., $\mathcal{U}^{contr}(t) = \{\ell\to r \in \mathcal{R} \mid \ell\to r \in \mathcal{GU}(t) \text{ or } r\to\ell\in\mathcal{GU}(t)\}$. Similarly, $\mathcal{U}^{contr}(\mathcal{P}) = \bigcup_{s\to t\in\mathcal{P}} \mathcal{U}^{contr}(t)$.[10]

Example 8. In Ex. 4, as $\mathsf{F}_{\mathcal{P}ol} = x_1 + \max(x_1 - x_2, 0)$, $\succsim_{\mathcal{P}ol}$ is monotonically decreasing on F's second argument. So $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is ν-compatible for the monotonicity specification ν with $\nu(\mathsf{F},2) = \{\Downarrow\}$ and $\nu(\mathsf{F},1) = \nu(\mathsf{ROUND},1) = \nu(\mathsf{s},1) = \nu(\mathsf{round},1) = \{\Uparrow\}$. Due to $\nu(\mathsf{F},2) = \{\Downarrow\}$, the general usable rules are the reversed round-rules. Thus, we cannot falsely prove termination with $\mathcal{P}ol$ anymore, since $\mathcal{P}ol$ does not make the *reversed* round-rules weakly decreasing; for example, we have $\mathsf{0} \prec_{\mathcal{P}ol} \mathsf{round}(\mathsf{0})$.

Our goal is to show that with the modified definition of usable rules above, Thm. 2 can also be used for non-monotonic reduction pairs. However, this is not true in general as shown by the following counterexample, cf. [10, Ex. 32].

Example 9. Consider the following famous TRS of Toyama [16]:

$$\mathsf{f}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x) \qquad\qquad \mathsf{g}(x,y) \to x \qquad\qquad \mathsf{g}(x,y) \to y$$

We use a monotonicity specification ν with $\nu(\mathsf{F},1) = \{\Downarrow\}$, $\nu(\mathsf{F},2) = \{\Uparrow\}$, $\nu(\mathsf{F},3) = \{\Uparrow,\Downarrow\}$ and a ν-compatible reduction pair $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ where $\mathsf{F}_{\mathcal{P}ol} = \max(x_2 - x_1, 0)$, $\mathsf{0}_{\mathcal{P}ol} = 0$, and $\mathsf{1}_{\mathcal{P}ol} = 1$. The only DP is strictly decreasing and there is no (general) usable rule. Hence, one would falsely conclude termination.

To obtain a sound criterion, we therefore impose certain requirements on all rules $\ell \to r \in \mathcal{P} \cup \mathcal{U}^{contr}$. To this end, we need the following notions.

- A rule $\ell \to r$ is *ν-more monotonic* (ν-MM) if variables occur at *more monotonic* positions on the right-hand side than on the left-hand side. More precisely, for every ν-dependent position p of r with $r|_p = x$ there is a position q of ℓ such that $\ell|_q = x$ and $\nu(\ell,q) \subseteq \nu(r,p)$. However, each position of ℓ can only be used once, i.e., for different positions p and p' of r we must choose different positions q and q' of ℓ. To define this notion formally, let $\mathcal{P}os_x^{\nu}(t)$

[9] Note that $\mathcal{GU}(t)$ is no longer a subset of $\mathcal{R}$. We nevertheless refer to $\mathcal{GU}(t)$ as "usable" rules in order to keep the similarity to Thm. 2.

[10] $\mathcal{U}^{contr}$ are the "usable rules w.r.t. an argument filtering" from [6].

be the set of all ν-dependent positions p of t with $t|_p = x$. Then a rule $\ell \to r$ is *ν-MM* if for each variable x there is an injective mapping α from $\mathcal{P}os^\nu_x(r)$ to $\mathcal{P}os^\nu_x(\ell)$ such that $\nu(\ell, \alpha(p)) \subseteq \nu(r, p)$ for all $p \in \mathcal{P}os^\nu_x(r)$.
So for the right-hand side of the DP in Ex. 9, we have $\mathcal{P}os^\nu_x(\mathsf{F}(x,x,x)) = \{1,2\}$. Hence, x would have to occur on at least two different ν-dependent positions q and q' in the left-hand side $\mathsf{F}(0,1,x)$. Moreover, we would need $\nu(\mathsf{F}(0,1,x),q) \subseteq \nu(\mathsf{F}(x,x,x),1) = \{\Downarrow\}$ and $\nu(\mathsf{F}(0,1,x),q') \subseteq \nu(\mathsf{F}(x,x,x),2) = \{\Uparrow\}$. However, this DP is not ν-MM as $\mathcal{P}os^\nu_x(\mathsf{F}(0,1,x)) = \varnothing$.
- $\ell \to r$ is *weakly ν-MM* if for each x with $\mathcal{P}os^\nu_x(\ell) \neq \varnothing$, there is an injective mapping α from $\mathcal{P}os^\nu_x(r)$ to $\mathcal{P}os^\nu_x(\ell)$ such that $\nu(\ell,\alpha(p)) \subseteq \nu(r,p)$ for all $p \in \mathcal{P}os^\nu_x(r)$. So in contrast to ν-MM, now we also permit variables that occur at dependent positions of r, but not at any dependent position of ℓ. Therefore, the DP of Ex. 9 is weakly ν-MM.
- $\ell \to r$ is *ν-right-linear* (ν-RL) if all variables occur at most once at a ν-dependent position in r. Formally, $\ell \to r$ is *ν-RL* iff for all $x \in \mathcal{V}(r)$: $|\mathcal{P}os^\nu_x(r)| \leqslant 1$. So the DP in Ex. 9 is not ν-RL since x occurs twice at ν-dependent positions in the right-hand side.

A TRS is *(weakly) ν-MM* resp. *ν-RL* iff all its rules satisfy that condition.

We now extend the processor from Thm. 2 to non-monotonic reduction pairs. Thm. 10 shows that to remove all strictly decreasing DPs, it is still sufficient if the (general) usable rules are weakly decreasing, provided that $\mathcal{P} \cup \mathcal{U}^{contr}(\mathcal{P})$ satisfies ν-MM. Alternatively, one can also require weak ν-MM and ν-RL.

As shown in [7], if one only wants to prove *innermost* termination, then Thm. 10 can be used even without the conditions (weak) ν-MM and ν-RL. However, we now extend this result to *full* termination. Of course, if $\mathcal{P} \cup \mathcal{U}^{contr}(\mathcal{P})$ is not (weakly) ν-MM resp. ν-RL and one wants to prove full termination with a non-monotonic reduction pair, then one has to use Thm. 5 instead.

Theorem 10. *Let ν be a monotonicity specification and let $(\succsim, \succ)$ be a ν-compatible non-monotonic reduction pair. Then Proc is sound:*[11]

$$Proc(\mathcal{P}) = \begin{cases} \mathcal{P} \setminus \succ & \text{if } \mathcal{P} \subseteq \succ \cup \succsim,\ \mathcal{GU}(\mathcal{P}) \subseteq \succsim, \text{ and one of (a) or (b) holds:} \\ & \quad \text{(a) } \mathcal{P} \cup \mathcal{U}^{contr}(\mathcal{P}) \text{ is } \nu\text{-MM} \\ & \quad \text{(b) } \mathcal{P} \cup \mathcal{U}^{contr}(\mathcal{P}) \text{ is weakly } \nu\text{-MM and } \nu\text{-RL} \\ \mathcal{P} & \text{otherwise} \end{cases}$$

Example 11. To modify Ex. 4 into a terminating TRS, we replace the f-rule by

$$\mathsf{f}(\mathsf{s}(x), x) \ \to\ \mathsf{f}(\mathsf{s}(x), \mathsf{round}(\mathsf{s}(x)))$$

similar to [7, Ex. 9]. We use the monotonicity specification from Ex. 8. The interpretation $\mathcal{P}ol$ from Ex. 4 is modified by defining $\mathsf{round}_{\mathcal{P}ol} = x_1$. Then $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is ν-compatible, all DPs are strictly decreasing, and the (general) usable rules (i.e., the *reversed* round-rules) are weakly decreasing. Moreover, all rules in $\mathcal{P} \cup \mathcal{U}^{contr}(\mathcal{P})$ are ν-MM. Thus, by Thm. 10(a) we can transform the initial DP problem $\mathcal{P} = DP(\mathcal{R})$ into $\mathcal{P} \setminus \succ = \varnothing$ and prove termination.

In contrast, this was not possible by the method of [10] which requires $\ell \approx r$

[11] The proof can be found in [3].

for all usable rules. There is no (possibly non-monotonic) reduction pair that satisfies $\mathsf{round}(0) \approx 0 \approx \mathsf{s}(0)$ and $\mathsf{F}(\mathsf{s}(x), x) \succ \mathsf{F}(\mathsf{s}(x), \mathsf{round}(\mathsf{s}(x)))$. The method of [7] can only prove innermost termination of this example. However, this TRS does not belong to a known class of TRSs where innermost termination implies termination. So in fact, up to now all tools failed on this example.

Example 12. The following example illustrates Thm. 10(b):

$$\begin{aligned}
\mathsf{p}(0) &\to 0 & \mathsf{minus}(x, 0) &\to x\\
\mathsf{p}(\mathsf{s}(x)) &\to x & \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{minus}(x, y)\\
\mathsf{div}(0, \mathsf{s}(y)) &\to 0 & \mathsf{minus}(x, \mathsf{s}(y)) &\to \mathsf{p}(\mathsf{minus}(x, y))\\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y))) &&\\
\mathsf{log}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(y))) &\to 0 &&\\
\mathsf{log}(\mathsf{s}(\mathsf{s}(x)), \mathsf{s}(\mathsf{s}(y))) &\to \mathsf{s}(\mathsf{log}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y)))) &&
\end{aligned}$$

We use a monotonicity specification ν with $\nu(\mathsf{s}, 1) = \nu(\mathsf{p}, 1) = \nu(\mathsf{minus}, 1) = \nu(\mathsf{MINUS}, 1) = \nu(\mathsf{div}, 1) = \nu(\mathsf{DIV}, 1) = \nu(\mathsf{LOG}, 1) = \{\Uparrow\}$, $\nu(\mathsf{minus}, 2) = \{\Downarrow\}$, $\nu(\mathsf{P}, 1) = \nu(\mathsf{MINUS}, 2) = \nu(\mathsf{div}, 2) = \nu(\mathsf{DIV}, 2) = \nu(\mathsf{LOG}, 2) = \{\Uparrow, \Downarrow\}$, and the interpretation $\mathsf{p}_{\mathcal{P}ol} = \max(x_1 - 1, 0)$, $\mathsf{minus}_{\mathcal{P}ol} = \max(x_1 - x_2, 0)$, $0_{\mathcal{P}ol} = \mathsf{P}_{\mathcal{P}ol} = 0$, $\mathsf{s}_{\mathcal{P}ol} = \mathsf{MINUS}_{\mathcal{P}ol} = \mathsf{div}_{\mathcal{P}ol} = \mathsf{LOG}_{\mathcal{P}ol} = x_1 + 1$, $\mathsf{DIV}_{\mathcal{P}ol} = x_1 + 2$. Now $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is ν-compatible, all DPs except $\mathsf{MINUS}(x, \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)$ are strictly decreasing, and the remaining DP and the usable p-, minus-, and div-rules are weakly decreasing. In addition, all DPs and usable rules are weakly ν-MM and ν-RL. Hence, by Thm. 10(b) we can remove all DPs except $\mathsf{MINUS}(x, \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)$. Afterwards, we use $\mathsf{MINUS}_{\mathcal{P}ol'} = x_2$ and $\mathsf{s}_{\mathcal{P}ol'} = x_1 + 1$ to delete this remaining DP. (Now there are no usable rules.) Hence, termination is proved.

Note that here, Thm. 10(a) does not apply as the DP $\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y))$ is not ν-MM: the first occurrence of y in the right-hand side is at a non-increasing position, whereas the only occurrence of y in the left-hand side is at a ν-independent, and thus increasing position.

The technique of [10] cannot handle the DP $\mathsf{LOG}(\ldots) \to \mathsf{LOG}(\mathsf{div}(\ldots), \ldots)$, because it would have to find an interpretation which makes the div-rules equivalent. In contrast, Thm. 10 only requires a weak decrease for the div-rules. Indeed, all existing termination tools failed on this example.

4 Automation

The most efficient implementations to search for polynomial interpretations are based on SAT solving [2]. However, [2] only handled the search for polynomial interpretations with natural coefficients as well as interpretations of the form $\max(p - n, 0)$ where p is a polynomial with natural coefficients and $n \in \mathbb{N}$. So we permitted interpretations like $\max(x_1 - 1, 0)$, but not interpretations like $\max(x_1 - x_2, 0)$ (as needed in Ex. 11 and 12) or $\max(x_1, x_2)$ (as needed in Ex. 1).

We want to use SAT solvers to search for *arbitrary* interpretations using polynomials and "max". Compared to existing related approaches, there are two challenges: the additional use of "max" in polynomial interpretations (Sect. 4.1) and the handling of non-monotonic quasi-orders and general usable rules (Sect. 4.2).

4.1 Automating Polynomial Interpretations with "max"

We start with encoding the "classical" reduction pair processor of Thm. 2 as a SAT problem. This is simpler than encoding Thm. 10, because in Thm. 2 we use a monotonic reduction pair $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ and thus, the applicability conditions and the usable rules $\mathcal{U}$ do not depend on a monotonicity specification. But in contrast to our earlier encoding from [2], now $\mathcal{P}ol$ can be an interpretation that combines polynomials and "max" arbitrarily.[12]

Definition 13 (max-polynomial). *Let $\mathcal{V}$ be the set of variables. The set of* max-polynomials $\mathbb{P}_M$ *over a set of numbers M is the smallest set such that*

- *$M \subseteq \mathbb{P}_M$ and $\mathcal{V} \subseteq \mathbb{P}_M$*
- *if $p, q \in \mathbb{P}_M$, then $p + q \in \mathbb{P}_M$, $p - q \in \mathbb{P}_M$, $p * q \in \mathbb{P}_M$, and $\max(p, q) \in \mathbb{P}_M$*

At the moment, we only consider interpretations $\mathcal{P}ol$ that map every function symbol to a max-polynomial over $\mathbb{N}$ that does not contain any subtraction "$-$". Obviously, then $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is a $\mathcal{C}_\varepsilon$-compatible (monotonic) reduction pair.

To find such interpretations automatically, one starts with an *abstract* polynomial interpretation. It maps each function symbol to a max-polynomial over a set $\mathcal{A}$ of *abstract* coefficients. In other words, one has to determine the degree and the shape of the max-polynomial, but the actual coefficients are left open. For example, for the TRS of Ex. 1 we could use an abstract polynomial interpretation $\mathcal{P}ol$ where $\star_{\mathcal{P}ol} = \max(a_1\, x_1 + a_2\, x_2,\ a_1'\, x_1 + a_2'\, x_2)$, $\uparrow_{\mathcal{P}ol} = b$, $\circ_{\mathcal{P}ol} = x_1 + x_2$, etc.[13] Here, a_1, a_2, a_1', a_2', b are abstract coefficients.

Now to apply the reduction pair processor of Thm. 2, we have to find an instantiation of the abstract coefficients satisfying the following condition. Then all dependency pairs that are strictly decreasing (i.e., $[s] \geqslant [t] + 1$) can be removed.

$$\bigwedge_{s \to t\, \in\, \mathcal{P}} [s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol} \;\wedge\; \bigvee_{s \to t\, \in\, \mathcal{P}} [s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol} + 1 \;\wedge\; \bigwedge_{\ell \to r\, \in\, \mathcal{U}(\mathcal{P})} [\ell]_{\mathcal{P}ol} \geqslant [r]_{\mathcal{P}ol} \quad (3)$$

Here, all rules in $\mathcal{P} \cup \mathcal{U}(\mathcal{P})$ are variable-renamed to have pairwise different variables. The polynomials $[s]_{\mathcal{P}ol}$, $[t]_{\mathcal{P}ol}$, etc. are again max-polynomials over $\mathcal{A}$. So with the interpretation $\mathcal{P}ol$ above, to make the last rule of Ex. 1 weakly decreasing (i.e., $\uparrow \circ (x \star y) \succsim_{\mathcal{P}ol} y$) we obtain the inequality $[\uparrow \circ (x \star y)]_{\mathcal{P}ol} \geqslant [y]_{\mathcal{P}ol}$:

$$b + \max(a_1\, x + a_2\, y,\ a_1'\, x + a_2'\, y) \geqslant y \quad (4)$$

We have to find an instantiation of the abstract coefficients $a_1, a_2, \ldots$ such that (4) holds for *all* instantiations of the variables x and y. In other words, the variables from $\mathcal{V}$ occurring in such inequalities are universally quantified.

Several techniques have been proposed to transform such inequalities further in order to remove such universally quantified variables [11]. However, the existing techniques only operate on inequalities without "max". Therefore, we now present new inference rules to eliminate "max" from such inequalities.

Our inference rules operate on *conditional* constraints of the form

[12] Of course, in an analogous way, one can also integrate the "minimum" function and indeed, we did this in our implementations.

[13] Here we already fixed $\circ$'s interpretation to simplify the presentation. Our implementations use heuristics to determine when to use an interpretation with "max".

$$p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \;\Rightarrow\; p \geqslant q \tag{5}$$

Here, $n \geqslant 0$ and $p_1, ..., p_n, q_1, ..., q_n$ are polynomials with abstract coefficients without "max". In contrast, p, q are max-polynomials with abstract coefficients.

The first inference rule eliminates an inner occurrence of "max" from the inequality $p \geqslant q$. If p or q have a sub-expression $\max(p', q')$ where p' and q' do not contain "max", then we can replace this sub-expression by p' or q' when adding the appropriate condition $p' \geqslant q'$ or $q' \geqslant p' + 1$, respectively.

I. **Eliminating "max"**

$$\frac{p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \quad\quad \Rightarrow \;\ldots \max(p', q') \ldots}{\begin{array}{lll} p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \wedge p' \geqslant q' & \Rightarrow \;\ldots \quad p' \quad \ldots & \wedge \\ p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \wedge q' \geqslant p' + 1 & \Rightarrow \;\ldots \quad q' \quad \ldots & \end{array}}$$

if p' and q' do not contain "max"

Obviously, by repeated application of inference rule (I), all occurrences of "max" can be removed. In our example, the constraint (4) is transformed into the following new constraint that does not contain "max" anymore.

$$a_1\, x + a_2\, y \geqslant a_1'\, x + a_2'\, y \;\Rightarrow\; b + a_1\, x + a_2\, y \geqslant y \quad\quad \wedge \tag{6}$$

$$a_1'\, x + a_2'\, y \geqslant a_1\, x + a_2\, y + 1 \;\Rightarrow\; b + a_1'\, x + a_2'\, y \geqslant y \tag{7}$$

Since the existing methods for eliminating universally quantified variables only work for *unconditional* inequalities, the next inference rule eliminates the conditions $p_i \geqslant q_i$ from a constraint of the form (5).[14] To this end, we introduce two new abstract polynomials $\overline{p}$ and $\overline{q}$ (that do not contain "max"). The polynomial $\overline{q}$ over the variables $x_1, ..., x_n$ is used to "measure" the polynomials $p_1, ..., p_n$ resp. $q_1, ..., q_n$ in the premise of (5) and the unary polynomial $\overline{p}$ measures the polynomials p and q in the conclusion of (5). We write $\overline{q}[p_1, ..., p_n]$ to denote the result of instantiating the variables $x_1, ..., x_n$ in $\overline{q}$ by $p_1, ..., p_n$, etc.

II. **Eliminating Conditions**

$$\frac{p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \;\Rightarrow\; p \geqslant q}{\overline{p}[p] - \overline{p}[q] \geqslant \overline{q}[p_1, \ldots, p_n] - \overline{q}[q_1, \ldots, q_n]}$$

if $\overline{q}$ and $\overline{p}$ do not contain "max", $\overline{p}$ is strictly monotonic, and $\overline{q}$ is weakly monotonic

Here, the monotonicity conditions mean that $x > y \Rightarrow \overline{p}[x] > \overline{p}[y]$ must hold and similarly that $x_1 \geqslant y_1 \wedge \ldots \wedge x_n \geqslant y_n \Rightarrow \overline{q}[x_1, \ldots, x_n] \geqslant \overline{q}[y_1, \ldots, y_n]$.

To see why Rule (II) is sound, let $\overline{p}[p] - \overline{p}[q] \geqslant \overline{q}[p_1, \ldots, p_n] - \overline{q}[q_1, \ldots, q_n]$ hold and assume that there is an instantiation σ of all variables in the polynomials with numbers that refutes $p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \;\Rightarrow\; p \geqslant q$. Now $p_1\sigma \geqslant q_1\sigma \wedge \ldots \wedge p_n\sigma \geqslant q_n\sigma$ implies $\overline{q}[p_1, \ldots, p_n]\sigma \geqslant \overline{q}[q_1, \ldots, q_n]\sigma$ by weak monotonicity of $\overline{q}$. Hence, $\overline{p}[p]\sigma - \overline{p}[q]\sigma \geqslant 0$. Since the instantiation σ is a counterexample to our original constraint, we have $p\sigma \not\geqslant q\sigma$ and thus $p\sigma < q\sigma$. But then strict monotonicity of $\overline{p}$ would imply $\overline{p}[p]\sigma - \overline{p}[q]\sigma < 0$ which gives a contradiction.

[14] Such conditional polynomial constraints also occur in other applications, e.g., in the termination analysis of logic programs. Indeed, we used a rule similar to inference rule (II) in the tool Polytool for termination analysis of logic programs [15]. However, Polytool only applies classical polynomial interpretations without "max".

If we choose[15] the abstract polynomials $\overline{p} = c\,x_1$ and $\overline{q} = d\,x_1$ for (6) and $\overline{p} = c'\,x_1$ and $\overline{q} = d'\,x_1$ for (7), then (6) and (7) are transformed into the following unconditional inequalities. (Note that we also have to add the inequalities $c \geqslant 1$ and $c' \geqslant 1$ to ensure that $\overline{p}$ is strictly monotonic.)

$$c \cdot (b + a_1\,x + a_2\,y) - c \cdot y \geqslant d \cdot (a_1\,x + a_2\,y) - d \cdot (a'_1\,x + a'_2\,y) \qquad \wedge \qquad (8)$$

$$c' \cdot (b + a'_1\,x + a'_2\,y) - c' \cdot y \geqslant d' \cdot (a'_1\,x + a'_2\,y) - d' \cdot (a_1\,x + a_2\,y + 1) \qquad (9)$$

Of course, such inequalities can be transformed into inequalities with 0 on their right-hand side. For example, (8) is transformed to

$$(c\,a_1 - d\,a_1 + d\,a'_1)\;x \;+\; (c\,a_2 - c - d\,a_2 + d\,a'_2)\;y \;+\; c\,b \;\geqslant\; 0 \qquad (10)$$

Thus, we now have to ensure non-negativeness of "polynomials" over variables like x, y, where the "coefficients" are polynomials over the abstract variables like $c\,a_1 - d\,a_1 + d\,a'_1$. To this end, it suffices to require that all these "coefficients" are $\geqslant 0$ [11]. In other words, now one can eliminate all universally quantified variables like x, y and (10) is transformed into the *Diophantine constraint*

$$c\,a_1 - d\,a_1 + d\,a'_1 \geqslant 0 \qquad \wedge \qquad c\,a_2 - c - d\,a_2 + d\,a'_2 \geqslant 0 \qquad \wedge \qquad c\,b \geqslant 0$$

III. **Eliminating Universally Quantified Variables**	
$\dfrac{p_0 + p_1\,x_1^{e_{11}} \ldots x_n^{e_{n1}} + \cdots + p_k\,x_1^{e_{1k}} \ldots x_n^{e_{nk}} \geqslant 0}{p_0 \geqslant 0 \;\wedge\; p_1 \geqslant 0 \;\wedge \ldots \wedge\; p_k \geqslant 0}$	if the p_i neither contain "max" nor any variable from $\mathcal{V}$

To search for suitable values for the abstract coefficients that satisfy the resulting Diophantine constraints, one fixes an upper bound for these values. Then we showed in [2] how to translate such Diophantine constraints into a satisfiability problem for propositional logic which can be handled by SAT solvers efficiently. In our example, the constraints resulting from the initial inequality (4) are for example satisfied by $a_1 = 1$, $a_2 = 0$, $a'_1 = 0$, $a'_2 = 1$, $b = 0$, $c = 1$, $d = 1$, $c' = 1$, $d' = 0$. With these values, the abstract interpretation $\max(a_1\,x_1 + a_2\,x_2,\; a'_1\,x_1 + a'_2\,x_2)$ for $\star$ is turned into the concrete interpretation $\max(x_1, x_2)$.

4.2 Automating Thm. 10

Now we show how to automate the improved reduction pair processor of Thm. 10. As before, our aim is to translate the resulting constraints into Diophantine constraints and further into propositional satisfiability problems.

Again, we start with an *abstract* polynomial interpretation $\mathcal{P}ol$. But since the values for the abstract coefficients can now be from $\mathbb{Z}$, we add the constraint

$$[f] \geqslant 0 \qquad \text{for all function symbols } f \qquad (11)$$

to ensure the well-foundedness of the resulting order. In the TRS of Ex. 12, we could start with an abstract interpretation where $\mathsf{minus}_{\mathcal{P}ol} = \max(m_1 x_1 + m_2 x_2, m_0)$. Here, m_0, m_1, m_2 are abstract coefficients which can later be instan-

[15] A good heuristic is to choose $\overline{q} = b_1 x_1 + \ldots + b_n x_n$ where all b_i are from $\{0, 1\}$ and $\overline{p} = a \cdot x_1$ where $1 \leqslant a \leqslant \max(\Sigma_{i=1}^{n} b_i,\ 1)$.

tiated by integers. Thus, we obtain the constraint $\max(m_1x_1 + m_2x_2, m_0) \geqslant 0$.

The challenge when automating Thm. 10 is that the general usable rules $\mathcal{GU}$ and the conditions (weakly) ν-MM and ν-RL depend on the (yet unknown) monotonicity specification ν, which itself enforces constraints on the quasi-order $\succsim_{\mathcal{P}ol}$ that one searches for. Nevertheless, if one uses max-polynomial interpretations, then the search for reduction pairs can still be mechanized efficiently. More precisely, we show how to encode all conditions of Thm. 10 as a formula which is independent of ν. In other words, this formula only contains Diophantine and Boolean variables. The latter are used to encode ν. The formula has the form

$$\mathit{Orient} \wedge \mathit{Usable} \wedge \big(\mathit{More} \ \vee \ (\mathit{Wmore} \wedge \mathit{Rlinear})\big) \wedge \mathit{Compat} \wedge \mathit{Depend} \tag{12}$$

where *Orient* requires that the DPs and general usable rules are weakly decreasing and at least one DP is strictly decreasing. Here, we use Boolean variables that state which rules are usable and *Usable* ensures that these variables have the correct values. *More*, *Wmore*, and *Rlinear* correspond to ν-MM, weak ν-MM, and ν-RL, respectively. *Compat* requires that $\succsim_{\mathcal{P}ol}$ is ν-compatible. Finally, the formula *Depend* computes the sets $\nu(t,p)$ from the monotonicity specification ν.

We start with defining *Depend*. To represent a monotonicity specification ν, for every function symbol f of arity n and every $1 \leqslant i \leqslant n$ we introduce two Boolean variables $\Uparrow_{f,i}$ and $\Downarrow_{f,i}$ which encode the set $\nu(f,i)$. So $\Uparrow_{f,i}$ is *true* iff $\Uparrow \in \nu(f,i)$ and likewise for $\Downarrow_{f,i}$. *Depend* is the conjunction of the following formulas for every term t in $\mathcal{P} \cup \mathcal{U}(\mathcal{P})$ and every position p of t. They introduce two Boolean variables $\Uparrow_{t,p}$ and $\Downarrow_{t,p}$ to encode the sets $\nu(t,p)$ according to Def. 6.

$$\begin{array}{ll}
\Uparrow_{t,\varepsilon} & \Leftrightarrow \mathit{true} \\
\Uparrow_{f(t_1,\dots,t_n),i\,p} & \Leftrightarrow \big(\Uparrow_{f,i} \wedge \Uparrow_{t_i,p}\big) \quad \vee \quad \big(\Downarrow_{f,i} \wedge \Downarrow_{t_i,p}\big) \quad \vee \\
& \big(\Uparrow_{f,i} \wedge \Downarrow_{f,i}\big) \quad \vee \quad \big(\Uparrow_{t_i,p} \wedge \Downarrow_{t_i,p}\big) \\
\Downarrow_{t,\varepsilon} & \Leftrightarrow \mathit{false} \\
\Downarrow_{f(t_1,\dots,t_n),i\,p} & \Leftrightarrow \big(\Uparrow_{f,i} \wedge \Downarrow_{t_i,p}\big) \quad \vee \quad \big(\Downarrow_{f,i} \wedge \Uparrow_{t_i,p}\big) \quad \vee \\
& \big(\Uparrow_{f,i} \wedge \Downarrow_{f,i}\big) \quad \vee \quad \big(\Uparrow_{t_i,p} \wedge \Downarrow_{t_i,p}\big)
\end{array}$$

Next we define *Usable*. We use two Boolean variables us_f and $\overline{\mathsf{us}}_f$ for every defined symbol f. Here, us_f (resp. $\overline{\mathsf{us}}_f$) is *true* if the f-rules (resp. reversed f-rules) are usable according to Def. 7. So whenever an f occurs at a non-decreasing position of a right-hand side of $\mathcal{P}$ then the f-rules are usable. Similarly, if f occurs at a non-increasing position, then the reversed f-rules are usable. Moreover, if (possibly reversed) f-rules are already usable then this may yield new usable rules due to right-hand sides of f-rules. Here, one has to keep the direction of the rules for non-decreasing positions and reverse the direction for non-increasing positions. This gives rise to the following formula *Usable*.

$$\bigwedge_{s \to t \in \mathcal{P},\, t|_p = f(\dots),\, f \text{ defined}} (\neg\Downarrow_{t,p} \Rightarrow \mathsf{us}_f) \quad \wedge \quad (\neg\Uparrow_{t,p} \Rightarrow \overline{\mathsf{us}}_f) \qquad \wedge$$

$$\bigwedge_{\ell \to r \in Rls(f),\, r|_p = g(\dots),\, g \text{ defined}} \big(\mathsf{us}_f \Rightarrow (\neg\Downarrow_{r,p} \Rightarrow \mathsf{us}_g) \ \wedge \ (\neg\Uparrow_{r,p} \Rightarrow \overline{\mathsf{us}}_g)\big) \ \wedge \ \big(\overline{\mathsf{us}}_f \Rightarrow (\neg\Downarrow_{r,p} \Rightarrow \overline{\mathsf{us}}_g) \ \wedge \ (\neg\Uparrow_{r,p} \Rightarrow \mathsf{us}_g)\big)$$

With the Boolean variables us_f and $\overline{\mathsf{us}}_f$ we can easily formalize that the rules in $\mathcal{P} \cup \mathcal{GU}(\mathcal{P})$ are weakly decreasing and that at least one pair is strictly decreasing. We obtain the following constraint *Orient* which is analogous to (3).

$$\bigwedge_{s\to t\in\mathcal{P}} [s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol} \quad \wedge \quad \bigvee_{s\to t\in\mathcal{P}} [s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol}+1 \quad \wedge$$

$$\bigwedge_{\ell\to r\in\mathcal{R},\, f=\mathrm{root}(\ell)} \big(\mathsf{us}_f \Rightarrow [\ell]_{\mathcal{P}ol} \geqslant [r]_{\mathcal{P}ol}\big) \quad \wedge \quad \big(\overline{\mathsf{us}}_f \Rightarrow [r]_{\mathcal{P}ol} \geqslant [\ell]_{\mathcal{P}ol}\big)$$

To ensure that $\mathcal{P}\cup\mathcal{U}^{contr}(\mathcal{P})$ is ν-RL, we interpret the Boolean values *true* and *false* as 1 and 0. Then we express ν-RL as a Diophantine constraint which we solve in the same way as the ones obtained from *Orient* later on. For any variable x, any term t, and any set $M \subseteq \{\Uparrow,\Downarrow\}$, let $\#_x^M(t)$ be a polynomial that describes the number of occurrences of x in t at positions p where $\nu(t,p)=M$. Thus, $\#_x^{\varnothing}(t) = \sum_{t|_p=x}(\neg\Uparrow_{t,p}\wedge\neg\Downarrow_{t,p})$ and $\#_x^{\{\Uparrow\}}(t)$, $\#_x^{\{\Downarrow\}}(t)$, $\#_x^{\{\Uparrow,\Downarrow\}}(t)$ are defined accordingly. Moreover, $\#_x(t) = \sum_{t|_p=x}(\neg\Uparrow_{t,p}\vee\neg\Downarrow_{t,p})$ encodes the number of occurrences of x at dependent positions of t. Then the constraint *Rlinear* is:

$$\bigwedge_{s\to t\in\mathcal{P},\, x\in\mathcal{V}(s)} \#_x(t)\leqslant 1 \quad \wedge \bigwedge_{\ell\to r\in\mathcal{R},\, x\in\mathcal{V}(\ell),\, f=\mathrm{root}(\ell)} \big(\mathsf{us}_f \vee \overline{\mathsf{us}}_f \Rightarrow \#_x(r)\leqslant 1\big)$$

More and *Wmore* ensure that $\mathcal{P}\cup\mathcal{U}^{contr}(\mathcal{P})$ is (weakly) ν-MM. For every rule $\ell\to r$ and every variable x at a ν-dependent position p of r, this variable must also occur at a unique less monotonic "partner" position q of ℓ. Thus, we could require $\#_x^{\varnothing}(r) \leqslant \#_x^{\varnothing}(\ell)$, $\#_x^{\{\Uparrow\}}(r) \leqslant \#_x^{\{\Uparrow\}}(\ell)$, and $\#_x^{\{\Downarrow\}}(r) \leqslant \#_x^{\{\Downarrow\}}(\ell)$. However, these requirements would be too strong, because they ignore the possibility that the "partner" position in ℓ may also be *strictly less* monotonic than the one in r. Therefore, for every rule $\ell\to r$ we introduce two new Diophantine variables $pt_x^{\Uparrow}$ and $pt_x^{\Downarrow}$ which stand for the number of those positions $p\in\mathcal{P}os_x^{\nu}(r)$ with $\nu(r,p)=\{\Uparrow\}$ (resp. $\nu(r,p)=\{\Downarrow\}$) where the "partner" position $q\in\mathcal{P}os_x^{\nu}(\ell)$ is non-monotonic (i.e., $\nu(\ell,q)=\varnothing$). Then *Wmore* is the following formula:

$$\bigwedge_{s\to t\in\mathcal{P},\, x\in\mathcal{V}(t)}\big(\#_x(s)\geqslant 1 \Rightarrow mm(s\to t,x)\big) \wedge \bigwedge_{\ell\to r\in\mathcal{R},\, x\in\mathcal{V}(r),\, f=\mathrm{root}(\ell)} \big((\mathsf{us}_f\vee\overline{\mathsf{us}}_f)\wedge\#_x(\ell)\geqslant 1 \Rightarrow mm(\ell\to r,x)\big)$$

where $mm(\ell\to r,x)$ is the following formula to encode ν-MM. Its first part ensures that ℓ contains enough non-monotonic occurrences of x to "cover" all occurrences of x in r that have a non-monotonic "partner" position in ℓ.

$$\#_x^{\varnothing}(r)+pt_x^{\Uparrow}+pt_x^{\Downarrow} \leqslant \#_x^{\varnothing}(\ell) \wedge \#_x^{\{\Uparrow\}}(r)\leqslant pt_x^{\Uparrow}+\#_x^{\{\Uparrow\}}(\ell) \wedge \#_x^{\{\Downarrow\}}(r)\leqslant pt_x^{\Downarrow}+\#_x^{\{\Downarrow\}}(\ell)$$

Now *More* results from *Wmore* by removing the premises "$\#_x(\cdot)\geqslant 1$".

Compat ensures that whenever the Boolean variable $\Uparrow_{f,i}$ is *true*, then $f_{\mathcal{P}ol}$ is a max-polynomial that is (weakly) monotonically increasing on its i-th argument (similarly for $\Downarrow_{f,i}$). We express such monotonicity conditions by the *partial derivatives* of $f_{\mathcal{P}ol}$. If $f_{\mathcal{P}ol}$ is differentiable (i.e., $f_{\mathcal{P}ol}$ contains no "max"), then $\succsim_{\mathcal{P}ol}$ is monotonically increasing on f's i-th argument iff $\frac{\partial f_{\mathcal{P}ol}}{\partial x_i}\geqslant 0$ (similarly for monotonic decrease). If $f_{\mathcal{P}ol}$ is a max-polynomial, then it is in general not differentiable, but *piecewise differentiable* and *continuous*. Then

$\succsim_{\mathcal{P}ol}$ is monotonically increasing (resp. decreasing) on f's i-th argument iff $\frac{\partial f_{\mathcal{P}ol}}{\partial x_i}\geqslant 0$ (resp. $\frac{\partial f_{\mathcal{P}ol}}{\partial x_i}\leqslant 0$) holds for all values where $\frac{\partial f_{\mathcal{P}ol}}{\partial x_i}$ is defined.

For instance, $\max(x_1 - 1, 2)$ is not differentiable at $x_1 = 3$. We have $\frac{\partial \max(x_1-1,2)}{\partial x_1} = 0$ for $x_1 < 3$ and $\frac{\partial \max(x_1-1,2)}{\partial x_1} = 1$ for $x_1 > 3$. But as $\frac{\partial \max(x_1-1,2)}{\partial x_1} \geqslant 0$ whenever it is defined, the function $\max(x_1 - 1, 2)$ is indeed monotonically increasing.

Therefore we introduce a new function symbol der_x for partial derivatives. Here, $\mathrm{der}_x(p)$ stands for $\frac{\partial p}{\partial x}$ whenever p is a function depending on x. However, at the moment the expressions $\mathrm{der}_x(p)$ are not "evaluated". Thus, we can also write $\mathrm{der}_x(p)$ if p is not differentiable. Then, *Compat* is the conjunction of the following constraints for all function symbols f and all $1 \leqslant i \leqslant \mathrm{arity}(f)$:

$$\big(\Uparrow_{f,i} \;\Rightarrow\; \mathrm{der}_{x_i}(f_{\mathcal{P}ol}) \geqslant 0\big) \quad\wedge\quad \big(\Downarrow_{f,i} \;\Rightarrow\; 0 \geqslant \mathrm{der}_{x_i}(f_{\mathcal{P}ol})\big)$$

This is indeed sufficient to guarantee that $(\succsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is ν-compatible. In particular, $\Uparrow_{f,i} \wedge \Downarrow_{f,i}$ now implies $\mathrm{der}_{x_i}(f_{\mathcal{P}ol}) = 0$, which ensures that $\succsim_{\mathcal{P}ol}$ is independent on f's i-th argument. Thus, the third condition of Def. 6 is always satisfied for quasi-orders like $\succsim_{\mathcal{P}ol}$, cf. Footnote 8.

So to automate Thm. 10,[16] we start with the constraint (12) instead of (3). In addition, we need the constraints of the form (11). Then we again apply the inference rules (I) - (III) in order to obtain Diophantine constraints.

However, now inequalities also contain "$\mathrm{der}_x(p)$" for max-polynomials p. Here, we apply Rule (I) repeatedly in order to eliminate "max". So by Rule (I), the constraint $\mathrm{der}_{x_1}(\max(m_1 x_1 + m_2 x_2, m_0)) \geqslant 0$ would be transformed into

$$\begin{array}{lll}\big(m_1 x_1 + m_2 x_2 \geqslant m_0 & \Rightarrow & \mathrm{der}_{x_1}(m_1 x_1 + m_2 x_2) \geqslant 0\big) \quad \wedge \\ \big(m_0 \geqslant m_1 x_1 + m_2 x_2 + 1 & \Rightarrow & \mathrm{der}_{x_1}(m_0) \geqslant 0\big)\end{array} \tag{13}$$

To eliminate "der_x" afterwards, we need the following rule for partial derivation:

IV. **Eliminating "der"**

$$\frac{\dots \mathrm{der}_{x_i}(p_0 + p_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} + \dots + p_k\, x_1^{e_{1k}} \dots x_n^{e_{nk}}) \dots}{\dots p_1\, e_{i1}\, x_1^{e_{11}} \dots x_i^{e_{i1}-1} \dots x_n^{e_{n1}} + \dots + p_k\, e_{ik}\, x_1^{e_{1k}} \dots x_i^{e_{ik}-1} \dots x_n^{e_{nk}}}$$ if the p_i neither contain "max" nor any variable from $\mathcal{V}$

So in (13), one could replace $\mathrm{der}_{x_1}(m_1 x_1 + m_2 x_2)$ by m_1 and $\mathrm{der}_{x_1}(m_0)$ by 0.

5 Experiments and Conclusion

We showed how to use integer polynomial interpretations with "max" in termination proofs with DPs and developed a method to encode the resulting search problems into SAT. All our results are implemented in the systems AProVE and $\mathsf{T_TT_2}$. While AProVE and $\mathsf{T_TT_2}$ were already the two most powerful termination provers for TRSs at the *International Competition of Termination Tools* 2007 [14], our contributions increase the power of both tools considerably without affecting their efficiency. More precisely, when using a time limit of 1 minute per example, AProVE and $\mathsf{T_TT_2}$ can now automatically prove termination of 15 ad-

[16] The automation of Thm. 5 works as for Thm. 2. To automate the combination of Thm. 5 and Thm. 10, one first generates the constraints for Thm. 10 and tries to solve them. If one does not find a solution, one checks whether $\mathcal{P} \cup \mathcal{U}(\mathcal{P})$ is non-duplicating. In this case, one uses Thm. 5(a) and otherwise, one uses Thm. 5(b).

ditional examples from the *Termination Problem Data Base* that is used for the competitions. Several of these examples had not been proven terminating by any tool at the competitions before. Moreover, AProVE and $\mathsf{T_TT_2}$ now also succeed on all examples from this paper (i.e., Ex. 1, 11, and 12), whereas all previous tools from the competitions failed (with the exception of TPA that could already solve Ex. 1). Our experiments also show the advantages over the earlier related contributions of [7,10] which were already implemented in AProVE and $\mathsf{T_TT_2}$, respectively. To run the AProVE implementation via a web-interface and for further details, we refer to http://aprove.informatik.rwth-aachen.de/eval/maxpolo.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. 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.
3. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal termination. Technical Report AIB-2008-03, RWTH Aachen, Germany. Available from http://aib.informatik.rwth-aachen.de.
4. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR'04*, LNAI 3452, pp. 301–331, 2005.
5. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. *Proc. IJCAR'06*, LNAI 4130, pp. 281–286, 2006.
6. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
7. J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. *Proc. CADE'07*, LNAI 4603, pp. 443–459, 2007.
8. T. Hardin and A. Laville. Proof of termination of the rewriting system SUBST on CCL. *Theoretical Computer Science*, 46(2,3):305–312, 1986.
9. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
10. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
11. H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
12. A. Koprowski. TPA: Termination proved automatically. In *Proc. RTA'06*, LNCS 4098, pp. 257–266, 2006.
13. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
14. C. Marché and H. Zantema. The termination competition. *Proc. RTA'07*, LNCS 4533, pp. 303–313, 2007.
15. M. Nguyen, D. De Schreye, J. Giesl, P. Schneider-Kamp. Polytool: Polynomial interpretations as a basis for termination analysis of logic programs. KU Leuven, 2008.
16. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
17. $\mathsf{T_TT_2}$. Available from http://colo6-c703.uibk.ac.at/ttt2.
18. H. Zantema. Termination. In *Term Rewriting Systems*, by Terese (ed.), Chapter 6, pp. 181–259, Cambridge University Press, 2003.