# Termination of Integer Term Rewriting[*]

C. Fuhs, J. Giesl, M. Plücker
LuFG Informatik 2
RWTH Aachen University
Aachen, Germany

P. Schneider-Kamp
Dept. of Mathematics & CS
University of Southern Denmark
Odense, Denmark

S. Falke
CS Department
University of New Mexico
Albuquerque, NM, USA

**Abstract**

When using rewrite techniques for termination analysis of programs, a main problem are pre-defined data types like integers. We extend term rewriting by built-in integers and adapt the dependency pair framework to prove termination of *integer term rewriting* automatically.

## 1 Introduction

Rewrite techniques and tools have been successfully applied to prove termination automatically for different programming languages. The advantage of rewrite techniques is that they are very powerful for algorithms on user-defined data structures, since they can generate well-founded orders comparing arbitrary forms of terms. But in contrast to techniques for termination of imperative programs, rewrite techniques do not support data structures like integers which are pre-defined in most programming languages.

To solve this problem, we extend TRSs by built-in integers and adapt the popular dependency pair (DP) framework for termination of TRSs to integers in Sect. 2. In Sect. 3, we improve the main processor of the adapted DP framework by considering *conditions* and explain how to generate suitable orders for termination proofs of integer TRSs (ITRSs). Sect. 4 evaluates our implementation in AProVE [6].

## 2 Integer Dependency Pair Framework

We represent each integer by a pre-defined constant of the same name. So the signature is split into two disjoint subsets $\mathscr{F}$ and $\mathscr{F}_{int}$. $\mathscr{F}_{int}$ contains $\mathbb{Z} = \{0,1,-1,\ldots\}$, $\mathbb{B} = \{\mathsf{true},\mathsf{false}\}$, and pre-defined operations $ArithOp = \{+,-,*,/,\%\}$, $RelOp = \{>,\geqslant,<,\leqslant,==,!=\}$, and $BoolOp = \{\wedge,\Rightarrow\}$. An *ITRS* $\mathscr{R}$ is a (finite) TRS over $\mathscr{F} \uplus \mathscr{F}_{int}$ where for all rules $\ell \to r$, we have $\ell \in \mathscr{T}(\mathscr{F} \cup \mathbb{Z} \cup \mathbb{B}, \mathscr{V})$ and $\ell \notin \mathbb{Z} \cup \mathbb{B}$. The rewrite relation $\hookrightarrow_{\mathscr{R}}$ of an ITRS $\mathscr{R}$ is defined as $\overset{\mathsf{i}}{\to}_{\mathscr{R} \cup PD}$, where $PD$ is an infinite set of rules to evaluate the pre-defined operations. For example, $PD$ contains the rules $2*21 \to 42$, $42 \geqslant 23 \to \mathsf{true}$, and $\mathsf{true} \wedge \mathsf{false} \to \mathsf{false}$. So pre-defined operations can only be evaluated if both their arguments are integers resp. Booleans. For example, consider the ITRSs $\mathscr{R}_1 = \{(1),(2),(3)\}$ where $\mathsf{sum}(x,y)$ computes $\sum_{i=y}^{x} i$.

$$\mathsf{sum}(x,y) \to \mathsf{sif}(x \geqslant y, x, y) \;\; (1) \qquad \mathsf{sif}(\mathsf{true},x,y) \to y + \mathsf{sum}(x,y+1) \;\; (2) \qquad \mathsf{sif}(\mathsf{false},x,y) \to 0 \;\; (3)$$

The term $\mathsf{sum}(1,1)$ can be rewritten as follows: $\underline{\mathsf{sum}(1,1)} \hookrightarrow_{\mathscr{R}_1} \mathsf{sif}(\underline{1 \geqslant 1},1,1) \hookrightarrow_{\mathscr{R}_1} \underline{\mathsf{sif}(\mathsf{true},1,1)} \hookrightarrow_{\mathscr{R}_1} 1 + \mathsf{sum}(1,\underline{1+1}) \hookrightarrow_{\mathscr{R}_1} 1 + \underline{\mathsf{sum}(1,2)} \hookrightarrow_{\mathscr{R}_1} 1 + \mathsf{sif}(\underline{1 \geqslant 2},1,2) \hookrightarrow_{\mathscr{R}_1} 1 + \underline{\mathsf{sif}(\mathsf{false},1,2)} \hookrightarrow_{\mathscr{R}_1} \underline{1+0} \hookrightarrow_{\mathscr{R}_1} 1$.

We extend the *DP framework* [1, 5, 7, 8] to ITRSs. The main problem is that proving innermost termination of $\mathscr{R} \cup PD$ *automatically* is not straightforward, as $PD$ is infinite. Therefore, we will not consider the rules $PD$ explicitly, but integrate their handling in the processors of the DP framework. The resulting method should be as powerful as possible for term rewriting on integers, but at the same time it should have the full power of the original DP framework when dealing with other function symbols.

For an ITRS $\mathscr{R}$, the *defined* symbols $\mathscr{D}$ are the root symbols of left-hand sides of rules in $\mathscr{R} \cup PD$, i.e., $\mathscr{D}$ also includes $ArithOp \cup RelOp \cup BoolOp$. However, we ignore these symbols when building DPs.

**Definition 1** (DP). *For all $f \in \mathscr{D} \setminus \mathscr{F}_{int}$, we introduce a fresh* tuple symbol *$F$ with the same arity. If $t = f(t_1,\ldots,t_n)$, let $t^\sharp = F(t_1,\ldots,t_n)$. If $\ell \to r \in \mathscr{R}$ for an ITRS $\mathscr{R}$ and $t$ is a subterm of $r$ with* root$(t) \in \mathscr{D} \setminus \mathscr{F}_{int}$, *then $\ell^\sharp \to t^\sharp$ is a* dependency pair *of $\mathscr{R}$. $DP(\mathscr{R})$ is the set of all DPs.*

For example, $DP(\mathscr{R}_1) = \{\mathsf{SUM}(x,y) \to \mathsf{SIF}(x \geqslant y, x, y) \;\; (4), \;\; \mathsf{SIF}(\mathsf{true},x,y) \to \mathsf{SUM}(x,y+1) \;\; (5)\}$.

[*]Supported by the DFG grant GI 274/5-2 and by the G.I.F. grant 966-116.6.

For any TRS $\mathcal{P}$ and ITRS $\mathcal{R}$, a *$\mathcal{P}$-chain* is a sequence of variable renamed pairs $s_1 \to t_1, s_2 \to t_2, \dots$ from $\mathcal{P}$ such that there is a substitution $\sigma$ (with possibly infinite domain) where $t_i\sigma \hookrightarrow_{\mathcal{R}}^* s_{i+1}\sigma$ and $s_i\sigma$ is in normal form w.r.t. $\hookrightarrow_{\mathcal{R}}$, for all $i$. We get the following corollary from the standard results on DPs.

**Corollary 2.** *An ITRS $\mathcal{R}$ is terminating (w.r.t. $\hookrightarrow_{\mathcal{R}}$) iff there is no infinite $DP(\mathcal{R})$*-chain.

Termination techniques are now called *DP processors* and they operate on sets of DPs (called *DP problems*). A DP processor *Proc* takes a DP problem as input and returns a set of new DP problems which have to be solved instead. *Proc* is *sound* if for all DP problems $\mathcal{P}$ with an infinite $\mathcal{P}$-chain there is also a $\mathcal{P}' \in Proc(\mathcal{P})$ with an infinite $\mathcal{P}'$-chain. Termination proofs 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 termination is proved. Many processors (like the *(estimated) dependency graph processor* [1, 5]) do not rely on the rules of the TRS, but just on the DPs and on the defined symbols. Therefore, they can also be directly applied for ITRSs.

But an adaption is non-trivial for one of the most important processors, the *reduction pair processor*. For a DP problem $\mathcal{P}$, this processor generates constraints which should be satisfied by a suitable order on terms. Here, we consider orders based on *max-polynomial interpretations* [3]. The set of *max-polynomials* is the smallest set containing the integers $\mathbb{Z}$, the variables, and $p+q$, $p*q$, and $\max(p,q)$ for all max-polynomials $p$ and $q$. A *max-polynomial interpretation Pol* maps every $n$-ary function symbol $f$ to a max-polynomial $f_{Pol}$ over $n$ variables $x_1, \dots, x_n$. This mapping is extended to terms as usual.

Consider the interpretation *Pol* where $\mathsf{SUM}_{Pol} = x_1 - x_2$, $\mathsf{SIF}_{Pol} = x_2 - x_3$, $+_{Pol} = x_1 + x_2$, $n_{Pol} = n$ for all $n \in \mathbb{Z}$, and $\geqslant_{Pol} = \mathsf{true}_{Pol} = \mathsf{false}_{Pol} = 0$. For any term $t$ and position $\pi$ in $t$, $t$ is *$\succsim_{Pol}$-dependent* on $\pi$ iff there exist terms $u, v$ where $t[u]_\pi \not\approx_{Pol} t[v]_\pi$. Here, $\approx_{Pol} = \succsim_{Pol} \cap \precsim_{Pol}$. So in our example, $\mathsf{SIF}(b,x,y)$ is $\succsim_{Pol}$-dependent on 2 and 3, but not on 1. A term $t$ is *$\succsim_{Pol}$-increasing* on $\pi$ iff $u \succsim_{Pol} v$ implies $t[u]_\pi \succsim_{Pol} t[v]_\pi$ for all terms $u, v$. So $\mathsf{SIF}(b,x,y)$ is $\succsim_{Pol}$-increasing on 1 and 2, but not on 3.

The reduction pair processor requires that all DPs in $\mathcal{P}$ are strictly or weakly decreasing and all *usable rules* $\mathcal{U}_{\mathcal{R}\cup PD}(\mathcal{P})$ are weakly decreasing. Then one can delete all strictly decreasing DPs. The *usable rules* [1, 7] include all rules that can reduce terms in $\succsim_{Pol}$-dependent positions of $\mathcal{P}$'s right-hand sides when instantiating their variables with normal forms. Moreover, as $\succsim_{Pol}$ is not monotonic in general, we require that defined symbols only occur on $\succsim_{Pol}$-increasing positions of right-hand sides.

When using interpretations into the integers, $\succ_{Pol}$ is not well founded. However, for any bound, there is no infinite $\succ_{Pol}$-decreasing sequence that remains greater than the bound. Hence, the reduction pair processor transforms a DP problem into *two* new problems. As before, the first problem results from removing all strictly decreasing DPs. 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 $\mathsf{c}$.

However, there are two problems: (i) *PD* is infinite and thus, there are usually infinitely many usable rules, which is a problem for the automation. (ii) Defined symbols like $+$ often occur on non-$\succsim_{Pol}$-increasing positions (e.g., in the right-hand side of (5) when using *Pol* above). To solve these problems, we now restrict ourselves to so-called *I-interpretations* where $n_{Pol} = n$ for all $n \in \mathbb{Z}$, $+_{Pol} = x_1 + x_2$, $-_{Pol} = x_1 - x_2$, $*_{Pol} = x_1 * x_2$, $\%_{Pol} = |x_1|$, and $/_{Pol} = |x_1| - \min(|x_2| - 1, |x_1|)$. We say that an I-interpretation is *proper* for a term $t$ if all defined symbols except $+$, $-$, and $*$ only occur on $\succsim_{Pol}$-increasing positions of $t$ and if symbols from *RelOp* only occur on $\succsim_{Pol}$-independent positions of $t$.

The concept of *proper* I-interpretations ensures that we can disregard the (infinitely many) usable rules for the symbols from *RelOp* and that the symbols "/" and "%" only have to be estimated "upwards". Moreover, we may allow $+$, $-$, and $*$ on arbitrary positions and we only have to regard the usable rules w.r.t. $\mathcal{R} \cup BO$. Here, *BO* are the (finitely many) rules for the symbols $\wedge$ and $\Rightarrow$ in *BoolOp*.

**Theorem 3** (Reduction Pair Processor for ITRSs)**.** *Let $\mathcal{R}$ be an ITRS, Pol be an I-interpretation, and $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim_{Pol} \mathsf{c}\}$ for a fresh constant* $\mathsf{c}$. *Then the following processor Proc is sound.*

$$Proc(\mathscr{P}) = \begin{cases} \{\, \mathscr{P} \setminus \succ_{Pol},\ \mathscr{P} \setminus \mathscr{P}_{bound} \,\}, & \textit{if } \mathscr{P} \subseteq \succsim_{Pol} \cup \succ_{Pol},\ \mathscr{U}_{\mathscr{R}\cup BO}(\mathscr{P}) \subseteq \succsim_{Pol}, \\ & \textit{and Pol is proper for all right-hand sides of } \mathscr{P} \cup \mathscr{U}_{\mathscr{R}}(\mathscr{P}) \\ \{\, \mathscr{P} \,\}, & \textit{otherwise} \end{cases}$$

To solve the DP problem $\mathscr{P} = \{(4),(5)\}$, we use an I-interpretation *Pol* where $\mathsf{SUM}_{Pol} = x_1 - x_2$ and $\mathsf{SIF}_{Pol} = x_2 - x_3$. We have $\mathscr{U}_{\mathscr{R}\cup BO}(\mathscr{P}) = \varnothing$, as the $+$- and $\geqslant$-rules are not included in $\mathscr{R} \cup BO$. The DP (5) is strictly decreasing, but no DP is bounded, since $\mathsf{SUM}(x,y) \not\succsim_{Pol} \mathsf{c}$ and $\mathsf{SIF}(\mathsf{true},x,y) \not\succsim_{Pol} \mathsf{c}$ for any value of $\mathsf{c}_{Pol}$. Thus, the processor returns the problems $\{(4)\}$ and $\{(4),(5)\}$, i.e., it does not simplify $\mathscr{P}$.

## 3 Conditional Constraints and Generation of I-Interpretations

To solve this problem, we consider *conditions* for inequalities like $s \underset{(\sim)}{\succ} t$ or $s \succsim \mathsf{c}$. So to include (4) in $\mathscr{P}_{bound}$, we do not demand $\mathsf{SUM}(x,y) \succsim \mathsf{c}$ for *all* $x$ and $y$. It suffices to require the inequality only for those instantiations of $x$ and $y$ which can be used in chains. So we require $\mathsf{SUM}(x,y) \succsim \mathsf{c}$ only for instantiations $\sigma$ where (4)'s instantiated right-hand side $\mathsf{SIF}(x \geqslant y,x,y)\sigma$ reduces to an instantiated left-hand side $u\sigma$ for some DP $u \to v$ where $u\sigma$ is in normal form. Here, $u \to v$ should again be variable renamed. As our DP problem contains two DPs (4) and (5), we get the following two *conditional constraints* (by considering all $u \to v \in \{(4),(5)\}$). We include (4) in $\mathscr{P}_{bound}$ if both constraints are satisfied.

$$\mathsf{SIF}(x \geqslant y,x,y) = \mathsf{SUM}(x',y') \;\Rightarrow\; \mathsf{SUM}(x,y) \succsim \mathsf{c} \quad (6) \qquad \mathsf{SIF}(x \geqslant y,x,y) = \mathsf{SIF}(\mathsf{true},x',y') \;\Rightarrow\; \mathsf{SUM}(x,y) \succsim \mathsf{c} \quad (7)$$

To check whether conditional constraints are valid requires reasoning about reachability w.r.t. TRSs with infinitely many rules. To this end, we developed rules to simplify conditional constraints. These rules detect that (6)'s premise is unsatisfiable and hence, (6) is valid. Moreover, they transform (7) into

$$x \succsim y \qquad \Rightarrow \qquad \mathsf{SUM}(x,y) \succsim \mathsf{c} \tag{8}$$

To automate the reduction pair processor, one has to generate an I-interpretation satisfying a given conditional constraint. One starts with an *abstract* I-interpretation. It maps each function symbol to a max-polynomial with *abstract* coefficients. So we could use an abstract I-interpretation *Pol* where $\mathsf{SUM}_{Pol} = a_0 + a_1 x_1 + a_2 x_2$, $\mathsf{SIF}_{Pol} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3$, and $\mathsf{c}_{Pol} = c_0$. Of course, the interpretation for the symbols in $\mathbb{Z} \cup \mathit{ArithOp}$ is fixed as for any I-interpretation (i.e., $+_{Pol} = x_1 + x_2$, etc.).

Then we transform the conditional constraint into an *inequality constraint* by replacing all atomic constraints "$s \succsim t$" by "$[s]_{Pol} \geqslant [t]_{Pol}$" and "$s \succ t$" by "$[s]_{Pol} \geqslant [t]_{Pol} + 1$". So "$\mathsf{SUM}(x,y) \succsim \mathsf{c}$" is transformed into "$a_0 + a_1 x + a_2 y \geqslant c_0$". Here, the abstract coefficients $a_0, a_1, a_2, c_0$ are implicitly existentially quantified and the variables $x, y \in \mathscr{V}$ are universally quantified. So (8) is transformed into

$$\forall x \in \mathbb{Z}, y \in \mathbb{Z} \quad (\, x \geqslant y \quad \Rightarrow \quad a_0 + a_1 x + a_2 y \geqslant c_0 \,) \tag{9}$$

Now we remove universally quantified variables from such constraints. Rule (A) handles conditions "$x \geqslant p$" or "$p \geqslant x$" for a polynomial $p$ without $x$. So (9) is transformed to $\forall y \in \mathbb{Z}, z \in \mathbb{N}$ $a_0 + a_1 (y+z) + a_2 y \geqslant c_0$ (10).

A. **Eliminating Conditions**

$$\frac{\forall x \in \mathbb{Z}, \ldots \quad (x \geqslant p \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p+z] \Rightarrow \psi[x/p+z])} \qquad \frac{\forall x \in \mathbb{Z}, \ldots \quad (p \geqslant x \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p-z] \Rightarrow \psi[x/p-z])}$$

To replace all remaining quantifiers over $\mathbb{Z}$ by quantifiers over $\mathbb{N}$, Rule (B) splits the inequality constraint $\varphi$ into the cases where $y$ is positive resp. negative. Thus, (10) is transformed into the conjunction of (11) and (12).

B. **Split**

$$\frac{\forall y \in \mathbb{Z} \quad \varphi}{\forall y \in \mathbb{N} \quad \varphi \qquad \wedge \qquad \forall y \in \mathbb{N} \quad \varphi[y/-y]}$$

$\forall y \in \mathbb{N}, z \in \mathbb{N} \quad a_0 + a_1\,(y+z) + a_2 y \geqslant c_0$ (11) $\qquad \forall y \in \mathbb{N}, z \in \mathbb{N} \quad a_0 + a_1\,(-y+z) - a_2 y \geqslant c_0$ (12)

Note that (11) can be reformulated as "$\forall y \in \mathbb{N}, z \in \mathbb{N} \quad (a_1 + a_2)\,y + a_1 z + (a_0 - c_0) \geqslant 0$". So we now have to ensure non-negativeness of "polynomials" over variables like $y$ and $z$ ranging over $\mathbb{N}$, where the "coefficients" are polynomials like "$a_1 + a_2$" over the abstract variables. To this end, it suffices to require that these "coefficients" are $\geqslant 0$ [9]. In other words, now one can eliminate all universally quantified variables like $y, z$ and transform (11) into the *Diophantine constraint* "$a_1 + a_2 \geqslant 0 \wedge a_1 \geqslant 0 \wedge a_0 - c_0 \geqslant 0$".

To search for abstract coefficients that satisfy the resulting Diophantine constraints, one fixes upper and lower bounds for their values. Then one can translate such Diophantine constraints into a SAT problem which can be handled by SAT solvers efficiently [2]. The constraints resulting from the initial inequality constraint (9) are for example satisfied by $a_0 = 0$, $a_1 = 1$, $a_2 = -1$, and $c_0 = 0$. With these values, the abstract interpretation $a_0 + a_1 x_1 + a_2 x_2$ for SUM is turned into the concrete interpretation $x_1 - x_2$. With the resulting concrete I-interpretation *Pol*, we would have $\mathscr{P}_\succ = \{(5)\}$ and $\mathscr{P}_{bound} = \{(4)\}$. The reduction pair processor of Thm. 3 would therefore transform the initial DP problem $\mathscr{P} = \{(4), (5)\}$ into the two problems $\mathscr{P} \setminus \mathscr{P}_\succ = \{(4)\}$ and $\mathscr{P} \setminus \mathscr{P}_{bound} = \{(5)\}$. Both are easy to solve.

## 4 Experiments and Conclusion

We adapted the DP framework to ITRSs. To evaluate our approach, we implemented it in AProVE [6] and tested it on a data base of 117 ITRSs containing also numerous examples from papers on termination of imperative programs. With a timeout of 1 minute per example, the new version of AProVE proves termination of 104 examples (88.9 %). We also tested the previous version of AProVE (AProVE08) and the tool $\mathsf{T_TT_2}$ [10] that do not support built-in integers. Here, we converted integers into terms constructed with 0, s, pos, and neg (e.g., $-1$ is represented as "neg(s(0))") and we added rules for pre-defined operations on integers in this representation. Although AProVE08 won the last *Termination Competition* 2008 for term rewriting and $\mathsf{T_TT_2}$ was second, AProVE08 resp. $\mathsf{T_TT_2}$ only proved termination of 24 (20.5 %) resp. 6 examples (5.1 %). This clearly shows the benefits of built-in integers in term rewriting. For details on our experiments and to run the new version of AProVE, we refer to `http://aprove.informatik.rwth-aachen.de/eval/Integer/`. A longer version of this paper appeared in [4].

## References

[1] T. Arts, J. Giesl. Termination of term rewriting using dependency pairs. *Th. Comp. Sc.*, 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. In *Proc. RTA'08*, LNCS 5117, pp. 110-125, 2008.

[4] C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In *Proc. RTA'09*, LNCS 5595, pp. 32-47, 2009.

[5] 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.

[6] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR'06*, LNAI 4130, pp. 281-286, 2006.

[7] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155-203, 2006.

[8] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *I & C*, 199(1,2):172-199, 2005.

[9] H. Hong and D. Jakuš. Testing positiveness of polynomials. *J. Aut. Reasoning*, 21(1):23-38, 1998.

[10] M. Korp, C. Sternagel, H. Zankl, A. Middeldorp. Tyrolean Termination Tool 2. *Proc. RTA'09*, LNCS, 2009.