RWTH Aachen

Department of Computer Science
*Technical Report*

# Lazy Abstraction for Size-Change Termination

Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2010-14

RWTH Aachen · Department of Computer Science · August 2010

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

# Lazy Abstraction for Size-Change Termination*

Michael Codish[1], Carsten Fuhs[2], Jürgen Giesl[2], and Peter Schneider-Kamp[3]

[1] Department of Computer Science, Ben-Gurion University, Israel
[2] LuFG Informatik 2, RWTH Aachen University, Germany
[3] IMADA, University of Southern Denmark, Odense, Denmark

**Abstract.** Size-change termination is a widely used means of proving termination where source programs are first abstracted to size-change graphs which are then analyzed to determine if they satisfy the size-change termination property. Here, the choice of the abstraction is crucial to the success of the method, and it is an open problem how to choose an abstraction such that no critical loss of precision occurs. This paper shows how to couple the search for a suitable abstraction and the test for size-change termination via an encoding to a single SAT instance. In this way, the problem of choosing the right abstraction is solved en passant by a SAT solver. We show that for the setting of term rewriting, the integration of this approach into the dependency pair framework works smoothly and gives rise to a new class of *size-change* reduction pairs. We implemented size-change reduction pairs in the termination prover AProVE and evaluated their usefulness in extensive experiments.

## 1 Introduction

Proving termination is a fundamental problem in verification. The challenge of termination analysis is to design a program abstraction that captures the properties needed to prove termination as often as possible, while providing a decidable sufficient criterion for termination. The size-change termination method (SCT) [20] is one such technique where programs are abstracted to *size-change graphs* which describe how the sizes of program data are affected by the transitions made in a computation. Size is measured by a well-founded base order. A set of size-change graphs has the *SCT property* iff for every path through any infinite concatenation of these graphs, a value would descend infinitely often w.r.t. the base order. This contradicts the well-foundedness of the base order, which implies termination of the original program. Lee et al. prove in [20] that the problem to determine if a set of size-change graphs has the SCT property is PSPACE-complete. The size-change termination method has been successfully applied in a variety of different application areas [2, 6, 18, 24, 27, 28].

Another approach emerges from the term rewriting community where termination proofs are performed by identifying suitable orders on terms and showing that every transition in a computation leads to a reduction w.r.t. the order. This

* Supported by the G.I.F. grant 966-116.6, the DFG grant GI 274/5-3, and the Danish Natural Science Research Council.

approach provides a decidable sufficient termination criterion for a given class of orders and can be considered as a program abstraction because terms are viewed modulo the order. Tools based on these techniques have been successfully applied to prove termination automatically for a wide range of different programming languages (e.g., Prolog [22, 26], Haskell [16], and Java Bytecode [23]).

A major bottleneck when applying SCT is due to the fact that it is a 2-phase process: First, a suitable program abstraction must be found, and then, the resulting size-change graphs are checked for termination. It is an open problem how to choose an abstraction such that no critical loss of precision occurs. Thus, our aim is to couple the search for a suitable abstraction with the test for size-change termination. To this end we model the search for an abstraction as the search for an order on terms (like in term rewriting). Then we can encode both the abstraction and the test for the SCT property into a single SAT instance.

Using a SAT-based search for orders to prove termination is well established by now. For instance [7, 8, 12, 25, 29] describe encodings for RPO, polynomial orders, or KBO. However, there is one major obstacle when using SAT for SCT. SCT is PSPACE-complete and hence (unless NP = PSPACE), there is no polynomial-size encoding of SCT to SAT. Thus, we focus on a subset of SCT which is in NP and can therefore be effectively encoded to SAT. This subset, called SCNP, is introduced by Ben-Amram and Codish in [5] where experimental evidence indicates that the restriction to this subset of SCT hardly makes any difference in practice. We illustrate our approach in the context of term rewrite systems (TRSs). The basic idea is to give a SAT encoding for the following question:

> For a given TRS (and a class of base orders such as RPO), is there a base order such that the resulting size-change graphs have the SCT property?

In [28], Thiemann and Giesl also apply the SCT method to TRSs and show how to couple it with the *dependency pair* (DP) method [1]. However, they take the 2-phase approach, first (manually) choosing a base order, and then checking if the induced size-change graphs satisfy the SCT property. Otherwise, one might try a different order. The implementation of [28] in the tool AProVE [14] only uses the (weak) embedding order in combination with argument filters [1] as base order. It performs a naive search which enumerates all argument filters. The new approach in this paper leads to a significantly more powerful implementation.

Using SCNP instead of SCT has an additional benefit. SCNP can be directly simulated by a new class of orders which can be used for *reduction pairs* in the DP framework. Thus, the techniques (or "processors") of the DP framework do not have to be modified at all for the combination with SCNP. This makes the integration of the size-change method with DPs much smoother than in [28] and it also allows to use this integration directly in arbitrary (future) extensions of the DP framework. The orders simulating SCNP are novel in the rewriting area.

The paper is structured as follows: Sect. 2 and 3 briefly present the DP framework and the SCT method for DPs. Sect. 4 adapts the SCNP approach to term rewriting. Sect. 5 shows how to encode the search for a base order which satisfies the SCNP property into a single SAT problem. Sect. 6 presents our experimental evaluation in the AProVE tool [14]. We conclude in Sect. 7.

## 2 Term Rewrite Systems and Dependency Pairs

We assume familiarity with term rewriting [3] and briefly introduce the main ideas of the DP method. The basic idea is (i) to describe all (finitely many) paths in the program from one function call to the next by special rewrite rules, called *dependency pairs*. Then (ii) one has to prove that these paths cannot follow each other infinitely often in a computation.

To represent a path from a function call of $f$ with arguments $s_1, \ldots, s_n$ to a function call of $g$ with arguments $t_1, \ldots, t_m$, we extend the signature by two new *tuple symbols* $F$ and $G$. Then a function call is represented by a *tuple term*, i.e., by a term rooted by a tuple symbol, but where no tuple symbols occur below the root. The DP for this path is the rule $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$.

The DP framework operates on *DP problems* $(\mathcal{P}, \mathcal{R})$, which are pairs of two TRSs. Here, for all $s \to t \in \mathcal{P}$, both $s$ and $t$ are tuple terms, whereas for all $l \to r \in \mathcal{R}$, both $l$ and $r$ are *base terms*, i.e., they do not contain tuple symbols. In the *initial* DP problem $(\mathcal{P}, \mathcal{R})$, $\mathcal{P}$ contains all DPs and $\mathcal{R}$ contains all rules of the TRS. Then, to show that this problem does not allow infinite chains of function calls, there is a large number of *processors* for analyzing and simplifying such DP problems. We refer to [1, 13, 15, 17] for further details on the DP framework.

The most common processor for simplifying DP problems is the reduction pair processor. In a *reduction pair* $(\succsim, \succ)$, $\succsim$ is a stable monotonic[4] quasi-order comparing either two tuple terms or two base terms. Moreover, $\succ$ is a stable well-founded order on terms, where $\succsim$ and $\succ$ are compatible (i.e., $\succ \circ \succsim \subseteq \succ$ and $\succsim \circ \succ \subseteq \succ$). Given such a reduction pair and a DP problem $(\mathcal{P}, \mathcal{R})$, if we can show a weak decrease (i.e., a decrease w.r.t. $\succsim$) for all rules from $\mathcal{R}$ and all DPs from $\mathcal{P}$, we can delete all those DPs from $\mathcal{P}$ that are strictly decreasing (i.e., that decrease w.r.t. $\succ$). In other words, we are asking the following question:

> For a given DP problem $(\mathcal{P}, \mathcal{R})$, is there a reduction pair that orients all rules of $\mathcal{R}$ and $\mathcal{P}$ weakly and at least one of the rules of $\mathcal{P}$ strictly?

If we can delete all DPs by repeatedly applying this processor, then the initial DP problem does not allow infinite chains of function calls. Consequently, there is no infinite reduction with the original TRS $\mathcal{R}$, i.e., $\mathcal{R}$ is terminating.

## 3 Size-Change Termination and Dependency Pairs

Size-change termination [20] is a program abstraction where termination is decidable. As mentioned in the introduction, an abstract program is a finite set of size-change graphs which describe, in terms of size-change, the possible transitions between consecutive function calls in the original program.

Size-change termination and the DP framework have some similarities: (i) size-change graphs provide a representation of the paths from one function call to the next, and (ii) in a second stage we show that these graphs do not allow infinite descent. So these steps correspond to steps (i) and (ii) in the DP method.

The main difference between SCT and the DP method is the stage when the

[4] If monotonicity of $\succsim$ is not required, we speak of a *non-monotonic* reduction pair.

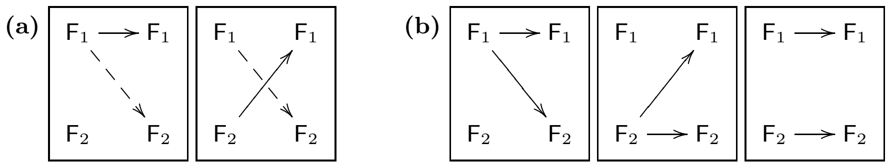

**Fig. 1.** Size-change graphs from Ex. 2

undecidable termination problem is abstracted to a decidable one. For SCT, we use a base order to obtain the finite representation of the paths by size-change graphs. For DPs, no such abstraction is performed and, indeed, the termination of a DP problem is undecidable. Here, the abstraction step is only the second stage where typically a decidable class of base orders is used.

The SCT method can be used with any base order. It only requires the information which arguments of a function call become (strictly or weakly) smaller w.r.t. the base order. To prove termination, the base order has to be well-founded. For the adaptation to TRSs we will use a reduction pair $(\succsim, \succ)$ for this purpose and introduce the notion of a size-change graph directly for dependency pairs.

If the TRS has a DP $F(s_1, \dots, s_n) \to G(t_1, \dots, t_m)$, then the corresponding size-change graph has nodes $\{F_1, \dots, F_n, G_1, \dots, G_m\}$ representing the argument positions of $F$ and $G$. The labeled edges in the size-change graph indicate whether there is a strict or weak decrease between these arguments.

**Definition 1 (size-change graphs).** *Let $(\succsim, \succ)$ be a (possibly non-monotonic) reduction pair on base terms, and let $F(s_1, \dots, s_n) \to G(t_1, \dots, t_m)$ be a DP. The* size-change graph *resulting from this DP and from $(\succsim, \succ)$ is the graph $(V_s, V_t, E)$ with source vertices $V_s = \{F_1, \dots, F_n\}$, target vertices $V_t = \{G_1, \dots, G_m\}$, and labeled edges $E = \{(F_i, G_j, \succ) \mid s_i \succ t_j\} \cup \{(F_i, G_j, \succsim) \mid s_i \succsim t_j\}$.*

Size-change graphs are depicted as in Fig. 1. Each graph consists of source vertices, on the left, target vertices, on the right, and edges drawn as full and dashed arrows to indicate strict and weak decrease (i.e., corresponding to "$\succ$" and "$\succsim$", respectively). We introduce the main ideas underlying SCT by example.

*Example 2.* Consider the TRS $\{(1), (2)\}$. It has the DPs (3) and (4).

$$\mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(x, \mathsf{s}(x)) \quad (1) \qquad\qquad \mathsf{F}(\mathsf{s}(x), y) \to \mathsf{F}(x, \mathsf{s}(x)) \quad (3)$$
$$\mathsf{f}(x, \mathsf{s}(y)) \to \mathsf{f}(y, x) \quad (2) \qquad\qquad \mathsf{F}(x, \mathsf{s}(y)) \to \mathsf{F}(y, x) \quad (4)$$

We use a reduction pair based on the embedding order where $\mathsf{s}(x) \succ x$, $\mathsf{s}(y) \succ y$, $\mathsf{s}(x) \succsim \mathsf{s}(x)$, $\mathsf{s}(y) \succsim \mathsf{s}(y)$. Then we get the size-change graphs in Fig. 1**(a)**. Between consecutive function calls, the first argument decreases in size or becomes smaller than the original second argument. In both cases, the second argument weakly decreases compared to the original first argument. By repeated composition of the size-change graphs, we obtain the three "idempotent" graphs in Fig. 1**(b)**. All of them exhibit *in situ decrease* (at $\mathsf{F}_1$ in the first graph, at $\mathsf{F}_2$ in the second graph, and at both $\mathsf{F}_1$ and $\mathsf{F}_2$ in the third graph). This means that the original size-change graphs from Fig. 1**(a)** *satisfy the SCT property.*

Earlier work [28] shows how to combine SCT with term rewriting. Let $\mathcal{R}$ be a TRS and $(\succsim, \succ)$ a reduction pair such that if $s \succ t$ (or $s \succsim t$), then $t$ contains no defined symbols of $\mathcal{R}$, i.e., no root symbols of left-hand sides of rules from $\mathcal{R}$.[5] Let $\mathcal{G}$ be the set of size-change graphs resulting from all DPs with $(\succsim, \succ)$. In [28] the authors prove that if $\mathcal{G}$ satisfies SCT then $\mathcal{R}$ is innermost terminating.

*Example 3.* If one restricts the reduction pair in Ex. 2 to just terms without defined symbols, then one still obtains the same size-change graphs. Since these graphs satisfy the SCT property, one can conclude that the TRS is indeed innermost terminating. Note that to show termination without SCT, an order like RPO would fail (since the first rule requires a lexicographic comparison and the second requires a multiset comparison). While termination could be proved by polynomial orders, as in [28] one could unite these rules with the rules for the Ackermann function. Then SCT with the embedding order would still work, whereas a direct application of RPO or polynomial orders fails.

So our example illustrates a major strength of SCT. A proof of termination is obtained by using just a *simple* base order and by considering each idempotent graph in the closure under composition afterwards. In contrast, without SCT, one would need more complex termination arguments.

In [28] the authors show that when constructing the size-change graphs from the DPs, one can even use *arbitrary* reduction pairs as base orders, provided that all rules of the TRS are weakly decreasing. In other words, this previous work essentially addresses the following question for any DP problem $(\mathcal{P}, \mathcal{R})$:

> For a given base order where $\mathcal{R}$ is weakly decreasing, do all idempotent size-change graphs, under composition closure, exhibit in situ decrease?

Note that in [28], the base order is always given and the only way to search for a base order automatically would be a hopeless generate-and-test approach.

## 4 Approximating SCT in NP

In [5] the authors identify a subset of SCT, called SCNP, that is powerful enough for practical use and is in NP. For SCNP just as for SCT, programs are abstracted to sets of size-change graphs. But instead of checking SCT by the closure under composition, one identifies a suitable ranking function to certify the termination of programs described by the set of graphs. Ranking functions map "program states" to elements of a well-founded domain and one has to show that they (strictly) decrease on all program transitions described by the size-change graphs.

In the rewriting context, program states are terms. Here, instead of a ranking function one can use an arbitrary stable well-founded order $\sqsupset$. Let $(V_s, V_t, E)$ be a size-change graph with source vertices $V_s = \{F_1, \ldots, F_n\}$, target vertices $V_t = \{G_1, \ldots, G_m\}$, and let $(\succsim, \succ)$ be the reduction pair on base terms which was used for the construction of the graph. Now the goal is to extend the order $\succ$ to a well-founded order $\sqsupset$ which can also compare tuple terms and which *satisfies*

[5] Strictly speaking, this is not a reduction pair, since it is only stable under substitutions which do not introduce defined symbols.

the size-change graph (i.e., $F(s_1, \ldots, s_n) \sqsupset G(t_1, \ldots, t_m)$). Similarly, we say that $\sqsupset$ satisfies a *set* of size-change graphs iff it satisfies all the graphs in the set.

If the size-change graphs describe the transitions of a program, then the existence of a corresponding ranking function obviously implies termination of the program. As in [28], to ensure that the size-change graphs really describe the transitions of the TRS correctly, one has to impose suitable restrictions on the reduction pair (e.g., by demanding that all rules of the TRS are weakly decreasing w.r.t. $\succsim$). Then one can indeed conclude termination of the TRS.

In [21], a class of ranking functions is identified which can simulate SCT. So if a set of size-change graphs has the SCT property, then there is a ranking function of that class satisfying these size-change graphs. However, this class is typically exponential in size [21]. To obtain a subset of SCT in NP, [5] considers a restricted class of ranking functions. A set of size-change graphs has the *SCNP* property iff it is satisfied by a ranking function from this restricted class.

Our goal is to adapt this class of ranking functions to term rewriting. The main motivation is to facilitate the *simultaneous* search for a ranking function on the size-change graphs and for the base order which is used to derive the size-change graphs from a TRS. It means that we are searching both for a program abstraction to size-change graphs, and also for the ranking function which proves that these graphs have the SCNP (and hence also the SCT) property.

This is different from [5], where the concrete structure of the program has already been abstracted away to size-change graphs that must be given as inputs. It is also different from the earlier adaption of SCT to term rewriting in [28], where the base order was fixed. As shown by the experiments with [28] in Sect. 6, fixing the base order for the size-change graphs leads to severe limitations in power.

The following example illustrates the SCNP property and presents a ranking function (resp. a well-founded order $\sqsupset$) satisfying a set of size-change graphs.

*Example 4.* Consider the TRS from Ex. 2 and its size-change graphs in Fig. 1**(a)**. Here, the base order is the reduction pair $(\succsim, \succ)$ resulting from the embedding order. We now extend $\succ$ to an order $\sqsupset$ which can also compare tuple terms and which satisfies the size-change graphs in this example. To compare tuple terms $\mathsf{F}(s_1, s_2)$ and $\mathsf{F}(t_1, t_2)$, we first map them to the multisets $\{\, \langle s_1, 1\rangle, \langle s_2, 0\rangle \,\}$ and $\{\, \langle t_1, 1\rangle, \langle t_2, 0\rangle \,\}$ of *tagged* terms (where a tagged term is a pair of a term and a number). Now a multiset $S$ of tagged terms is greater than a multiset $T$ iff for every $\langle t, m\rangle \in T$ there is an $\langle s, n\rangle \in S$ where $s \succ t$ or both $s \succsim t$ and $n > m$.

For the first graph, we have $s_1 \succ t_1$ and $s_1 \succsim t_2$ and hence the multiset $\{\, \langle s_1, 1\rangle, \langle s_2, 0\rangle \,\}$ is greater than $\{\, \langle t_1, 1\rangle, \langle t_2, 0\rangle \,\}$. For the second graph, $s_1 \succsim t_2$ and $s_2 \succ t_1$ also implies that the multiset $\{\, \langle s_1, 1\rangle,\ \langle s_2, 0\rangle \,\}$ is greater than $\{\, \langle t_1, 1\rangle, \langle t_2, 0\rangle \,\}$. Thus, if we define our well-founded order $\sqsupset$ in this way, then it indeed satisfies both size-change graphs of the example. Since this order $\sqsupset$ belongs to the class of ranking functions defined in [5], this shows that the size-change graphs in Fig. 1**(a)** have the SCNP property.

In term rewriting, size-change graphs correspond to DPs and the arcs of the size-change graphs are built by only comparing the arguments of the DPs (which are *base terms*). The ranking function then corresponds to a well-founded order

on tuple terms. We now reformulate the class of ranking functions of [5] in the term rewriting context by defining *SCNP reduction pairs*. The advantage of this reformulation is that it allows us to integrate the SCNP approach directly into the DP framework and that it allows a SAT encoding of both the search for suitable base orders and of the test for the SCNP property. In [5], the class of ranking functions for SCNP is defined incrementally. We follow this, but adapt the definitions of [5] to the term rewriting setting and prove that the resulting orders always constitute reduction pairs. More precisely, we proceed as follows:
**step one:** $(\succsim, \succ)$ is an arbitrary reduction pair on base terms that we start with (e.g., based on RPO and argument filters or on polynomial orders). The main observation that can be drawn from the SCNP approach is that it is helpful to compare base terms and tuple terms in a different way. Thus, our goal is to extend $(\succsim, \succ)$ appropriately to a reduction pair $(\sqsupseteq, \sqsupset)$ that can also compare tuple terms. By defining $(\sqsupseteq, \sqsupset)$ in the same way as the ranking functions of [5], it can simulate the SCNP approach.
**step two:** $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a reduction pair on *tagged* base terms, i.e., on pairs $\langle t, n\rangle$, where $t$ is a base term and $n \in \mathbb{N}$. Essentially, $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a lexicographic combination of the reduction pair $(\succsim, \succ)$ with the usual order on $\mathbb{N}$.
**step three:** $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ extends $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ to compare *multisets* of tagged base terms. The *status* $\mu$ determines how $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is extended to multisets.
**step four:** $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ is a full reduction pair (i.e., it is the reduction pair $(\sqsupseteq, \sqsupset)$ we were looking for). The *level mapping* $\ell$ determines which arguments of a tuple term are selected and tagged, resulting in a multiset of tagged base terms. On tuple terms, $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ behaves according to $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ on the multisets as determined by $\ell$, and on base terms, it behaves like $(\succsim, \succ)$.

Thus, we start with extending a reduction pair $(\succsim, \succ)$ on base terms to a reduction pair $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ on tagged base terms. We compare tagged terms lexicographically by $(\succsim, \succ)$ and by the standard orders $\geq$ and $>$ on numbers.

**Definition 5 (comparing tagged terms).** *Let $(\succsim, \succ)$ be a reduction pair on terms. We define the corresponding reduction pair $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ on tagged terms:*

- $\langle t_1, n_1\rangle \succsim^{\mathbb{N}} \langle t_2, n_2\rangle \quad\Leftrightarrow\quad t_1 \succ t_2 \;\vee\; (t_1 \succsim t_2 \wedge n_1 \geq n_2)$.
- $\langle t_1, n_1\rangle \succ^{\mathbb{N}} \langle t_2, n_2\rangle \quad\Leftrightarrow\quad t_1 \succ t_2 \;\vee\; (t_1 \succsim t_2 \wedge n_1 > n_2)$.

The motivation for tagged terms is that we will use different tags (i.e., numbers) for the different argument positions of a function symbol. For instance, when comparing the terms $s = \mathsf{F}(\mathsf{s}(x), y)$ and $t = \mathsf{F}(x, \mathsf{s}(x))$ as in Ex. 4, one can assign the tags 1 and 0 to the first and second argument position of $\mathsf{F}$, respectively. Then, if $(\succsim, \succ)$ is the reduction pair based on the embedding order, we have $\langle \mathsf{s}(x), 1\rangle \succ^{\mathbb{N}} \langle x, 1\rangle$ and $\langle \mathsf{s}(x), 1\rangle \succ^{\mathbb{N}} \langle \mathsf{s}(x), 0\rangle$. In other words, the first argument of $s$ is greater than both the first and the second argument of $t$.

The following lemma states that if $(\succsim, \succ)$ is a reduction pair on terms, then $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a reduction pair on tagged terms (where we do not require monotonicity, since monotonicity is not defined for tagged terms). This lemma will be needed for our main theorem (Thm. 12) which proves that the reduction pair defined to simulate SCNP is really a reduction pair.

**Lemma 6 (reduction pairs on tagged terms).** *Let $(\succsim, \succ)$ be a reduction pair. Then $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a non-monotonic reduction pair on tagged terms.*[6]

The next step is to introduce a "reduction pair" $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ on multisets of tagged base terms, where $\mu$ is a status which determines how $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is extended to multisets. Of course, there are many possibilities for such an extension. In Def. 7, we present the four extensions which correspond to the ranking functions defining SCNP in [5]. The main difference to the definitions in [5] is that we do not restrict ourselves to *total* base orders. Hence, the notions of maximum and minimum of a multiset of terms are not defined in the same way as in [5].

**Definition 7 (multiset extensions of reduction pairs).** *Let $(\succsim, \succ)$ be a reduction pair on (tagged) terms. We define an extended reduction pair $(\succsim^{\mu}, \succ^{\mu})$ on multisets of (tagged) terms, for $\mu \in \{max, min, ms, dms\}$. Let $S$ and $T$ be multisets of (tagged) terms.*

1. (max order) $S \succsim^{max} T$ *holds iff* $\forall t \in T.\ \exists s \in S.\ s \succsim t$.
   $S \succ^{max} T$ *holds iff* $S \neq \varnothing$ *and* $\forall t \in T.\ \exists s \in S.\ s \succ t$.
2. (min order) $S \succsim^{min} T$ *holds iff* $\forall s \in S.\ \exists t \in T.\ s \succsim t$.
   $S \succ^{min} T$ *holds iff* $T \neq \varnothing$ *and* $\forall s \in S.\ \exists t \in T.\ s \succ t$.
3. (multiset order [9]) $S \succ^{ms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \dots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \dots, t_k \}$, $S_{\text{strict}} \succ^{max} T_{\text{strict}}$*, and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.
   $S \succsim^{ms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \dots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \dots, t_k \}$, *either* $S_{\text{strict}} \succ^{max} T_{\text{strict}}$ *or* $S_{\text{strict}} = T_{\text{strict}} = \varnothing$*, and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.
4. (dual multiset order [4]) $S \succ^{dms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \dots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \dots, t_k \}$, $S_{\text{strict}} \succ^{min} T_{\text{strict}}$*, and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.
   $S \succsim^{dms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \dots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \dots, t_k \}$, *either* $S_{\text{strict}} \succ^{min} T_{\text{strict}}$ *or* $S_{\text{strict}} = T_{\text{strict}} = \varnothing$*, and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.

Here $\succ^{ms}$ is to the standard multiset extension of an order $\succ$ as used, e.g., for the classical definition of RPO. However, our use of tagged terms as elements of the multiset introduces a lexicographic aspect that is missing in RPO.

*Example 8.* Consider again the TRS from Ex. 2 with the reduction pair based on the embedding order. We have $\{\mathsf{s}(x), y\} \succsim^{max} \{x, \mathsf{s}(x)\}$, since for both terms in $\{x, \mathsf{s}(x)\}$ there is an element in $\{\mathsf{s}(x), y\}$ which is weakly greater (w.r.t. $\succsim$). Similarly, $\{x, \mathsf{s}(y)\} \succsim^{max} \{y, x\}$. However, $\{x, \mathsf{s}(y)\} \not\succ^{max} \{y, x\}$, since not every element from $\{y, x\}$ has a strictly greater one in $\{x, \mathsf{s}(y)\}$. We also have $\{x, \mathsf{s}(y)\} \succsim^{min} \{y, x\}$, but $\{\mathsf{s}(x), y\} \not\succsim^{min} \{x, \mathsf{s}(x)\}$, since for $y$ in $\{\mathsf{s}(x), y\}$, there is no term in $\{x, \mathsf{s}(x)\}$ which is weakly smaller.

We have $\{\mathsf{s}(x), y\} \not\succ^{ms} \{x, \mathsf{s}(x)\}$, since even if we take $\{\mathsf{s}(x)\} \succ^{max} \{x\}$, we still do not have $y \succsim \mathsf{s}(x)$. Moreover, also $\{\mathsf{s}(x), y\} \not\succsim^{ms} \{x, \mathsf{s}(x)\}$. Otherwise, for every element of $\{x, \mathsf{s}(x)\}$ there would have to be a *different* weakly greater element in $\{\mathsf{s}(x), y\}$. In contrast, we have $\{x, \mathsf{s}(y)\} \succ^{ms} \{y, x\}$. The element $\mathsf{s}(y)$ is replaced by the strictly smaller element $y$ and for the remaining element $x$ on the right-hand side there is a weakly greater one on the left-hand side. Similarly, we also have $\{\mathsf{s}(x), y\} \not\succsim^{dms} \{x, \mathsf{s}(x)\}$ and$\{x, \mathsf{s}(y)\} \succ^{dms} \{y, x\}$.

[6] All proofs can be found in Appendix A.

So there is no $\mu$ such that the multiset of arguments strictly decreases in some DP and weakly decreases in the other DP. We can only achieve a weak decrease for all DPs. To obtain a strict decrease in such cases, one can add tags.

We want to define a reduction pair $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ which is like $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ on tuple terms and like $(\succsim, \succ)$ on base terms. Here, we use a *level mapping* $\ell$ to map tuple terms $F(s_1, \ldots, s_n)$ to multisets of tagged base terms.

**Definition 9 (level mapping).** *For each tuple symbol $F$ of arity $n$, let $\pi(F) \subseteq \{1, \ldots, n\} \times \mathbb{N}$ such that for each $1 \leq j \leq n$ there is at most one $m \in \mathbb{N}$ with $\langle j, m \rangle \in \pi(F)$. Then $\ell(F(s_1, \ldots, s_n)) = \{ \langle s_i, n_i \rangle \mid \langle i, n_i \rangle \in \pi(F) \}$.*

*Example 10.* Consider again the TRS from Ex. 2 with the reduction pair based on the embedding order. Let $\pi$ be a status function with $\pi(\mathsf{F}) = \{ \langle 1, 1 \rangle, \langle 2, 0 \rangle \}$. So $\pi$ selects both arguments of terms rooted with $\mathsf{F}$ for comparison and associates the tag 1 with the first argument and the tag 0 with the second argument. This means that it puts "more weight" on the first than on the second argument. The level mapping $\ell$ defined by $\pi$ transforms the tuple terms from the DPs of our TRS into the following multisets of tagged terms:

$$\begin{aligned} \ell(\mathsf{F}(\mathsf{s}(x), y)) &= \{ \langle \mathsf{s}(x), 1 \rangle, \langle y, 0 \rangle \} & \qquad \ell(\mathsf{F}(x, \mathsf{s}(x))) &= \{ \langle x, 1 \rangle, \langle \mathsf{s}(x), 0 \rangle \} \\ \ell(\mathsf{F}(x, \mathsf{s}(y))) &= \{ \langle x, 1 \rangle, \langle \mathsf{s}(y), 0 \rangle \} & \qquad \ell(\mathsf{F}(y, x)) &= \{ \langle y, 1 \rangle, \langle x, 0 \rangle \} \end{aligned}$$

Now we observe that for the multisets of the tagged terms above, we have

$$\ell(\mathsf{F}(\mathsf{s}(x), y)) \ \succ^{\mathbb{N},max} \ \ell(\mathsf{F}(x, \mathsf{s}(x))) \qquad\qquad \ell(\mathsf{F}(x, \mathsf{s}(y))) \ \succ^{\mathbb{N},max} \ \ell(\mathsf{F}(y, x))$$

So due to the tagging now we can find an order such that both DPs are strictly decreasing. This order corresponds to the ranking function given in Ex. 4.

Finally we define the class of reduction pairs which corresponds to the class of ranking functions considered for SCNP in [5].

**Definition 11 (SCNP reduction pair).** *Let $(\succsim, \succ)$ be a reduction pair on base terms and let $\ell$ be a level mapping. For $\mu \in \{max, min, ms, dms\}$, we define the* SCNP reduction pair $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$*. For base terms $l, r$ we define $l \underset{(\sim)}{\succ}{}^{\mu,\ell} r \Leftrightarrow l \underset{(\sim)}{\succ} r$ and for tuple terms $s$ and $t$ we define $s \underset{(\sim)}{\succ}{}^{\mu,\ell} t \Leftrightarrow \ell(s) \underset{(\sim)}{\succ}{}^{\mathbb{N},\mu} \ell(t)$.*

So we have $s \succ^{max,\ell} t$ for the DPs $s \to t$ in Ex. 2 and the level mapping $\ell$ in Ex. 10. Thm. 12 states that SCNP reduction pairs actually *are* reduction pairs.

**Theorem 12.** *For $\mu \in \{max, min, ms, dms\}$, $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ is a reduction pair.*

We now automate the SCNP criterion of [5]. For a DP problem $(\mathcal{P}, \mathcal{R})$ with the DPs $\mathcal{P}$ and the TRS $\mathcal{R}$, we have to find a suitable base order $(\succsim, \succ)$ to construct the size-change graphs $\mathcal{G}$ corresponding to the DPs in $\mathcal{P}$. So every graph $(V_s, V_t, E)$ from $\mathcal{G}$ with source vertices $V_s = \{F_1, \ldots, F_n\}$ and target vertices $V_t = \{G_1, \ldots, G_m\}$ corresponds to a DP $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$. Moreover, we have an edge $(F_i, G_j, \succ) \in E$ iff $s_i \succ t_j$ and $(F_i, G_j, \succsim) \in E$ iff $s_i \succsim t_j$.

In our example, if we use the reduction pair $(\succsim, \succ)$ based on the embedding order, then $\mathcal{G}$ are the size-change graphs from Fig. 1**(a)**. For instance, the first size-change graph results from the DP (3).

For SCNP, we have to extend $\succ$ to a well-founded order $\sqsupset$ which can also compare tuple terms and which satisfies all size-change graphs in $\mathcal{G}$. For $\sqsupset$, we could take any order $\succ^{\mu,\ell}$ from an SCNP reduction pair $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$. To show that $\sqsupset$ satisfies the size-change graphs from $\mathcal{G}$, one then has to prove $F(s_1,\ldots,s_n) \succ^{\mu,\ell} G(t_1,\ldots,t_m)$ for every DP $F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m)$. Moreover, to ensure that the size-change graphs correctly describe the transitions of the TRS-program $\mathcal{R}$, one also has to require that all rules of the TRS $\mathcal{R}$ are weakly decreasing w.r.t. $\succsim$ (cf. the remarks at the beginning of Sect. 4). Of course, as in [28], this requirement can be weakened (e.g., by only regarding *usable* rules) when proving *innermost* termination.

As in [5], we define $\sqsupset$ as a lexicographic combination of several orders of the form $\succ^{\mu,\ell}$. We define the lexicographic combination of two reduction pairs as $(\succsim_1, \succ_1) \times (\succsim_2, \succ_2) = (\succsim_{1\times 2}, \succ_{1\times 2})$. Here, $s \succsim_{1\times 2} t$ holds iff both $s \succsim_1 t$ and $s \succsim_2 t$. Moreover, $s \succ_{1\times 2} t$ holds iff $s \succ_1 t$ or both $s \succsim_1 t$ and $s \succ_2 t$. It is clear that $(\succsim_{1\times 2}, \succ_{1\times 2})$ is again a reduction pair.

A suitable well-founded order $\sqsupset$ is now constructed automatically as follows. The pair of orders $(\sqsupseteq, \sqsupset)$ is initialized by defining $\sqsupseteq$ to be the relation where only $t \sqsupseteq t$ holds for two tuple or base terms $t$ and where $\sqsupset$ is the empty relation. As long as the set of size-change graphs $\mathcal{G}$ is not empty, a status $\mu$ and a level mapping $\ell$ are synthesized such that $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ orients all DPs weakly and at least one DP strictly. In other words, the corresponding ranking function satisfies one size-change graph and "weakly satisfies" the others. Then the strictly oriented DPs (resp. the strictly satisfied size-change graphs) are removed, and $(\sqsupseteq, \sqsupset) := (\sqsupseteq, \sqsupset) \times (\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ is updated. In this way, the SCNP approach can be simulated by a repeated application of the *reduction pair processor* in the DP framework, using the special class of SCNP reduction pairs.

So in our example, we could first look for a $\mu_1$ and $\ell_1$ where the first DP (3) decreases strictly (w.r.t. $\succ^{\mu_1,\ell_1}$) and the second decreases weakly (w.r.t. $\succsim^{\mu_1,\ell_1}$). Then we would remove the first DP and could now search for a $\mu_2$ and $\ell_2$ such that the remaining second DP (4) decreases strictly (w.r.t. $\succ^{\mu_2,\ell_2}$). The resulting reduction pair would be $(\sqsupseteq, \sqsupset) = (\succsim^{\mu_1,\ell_1}, \succ^{\mu_1,\ell_1}) \times (\succsim^{\mu_2,\ell_2}, \succ^{\mu_2,\ell_2})$.

While in [5], the set of size-change graphs remains fixed throughout the whole termination proof, the DP framework allows to use a lexicographic combination of SCNP reduction pairs which are constructed from *different* reduction pairs $(\succsim, \succ)$ on base terms. In other words, after a base order and a ranking function satisfying one size-change graph and weakly satisfying all others have been found, the satisfied size-change graph (resp. the corresponding DP) is removed, and one can synthesize a possibly different ranking function and also a *possibly different base order* for the remaining DPs (i.e., different abstractions to different size-change graphs can be used in one and the same termination proof).

*Example 13.* We add a third rule to the TRS from Ex. 2: $\mathsf{f}(\mathsf{c}(x), y) \to \mathsf{f}(x, \mathsf{s}(x))$. Now no SCNP reduction pair based only on the embedding order can orient all DPs strictly at the same time anymore, even if one permits combinations with arbitrary argument filters. However, we can first apply an SCNP reduction pair that sets all tags to 0 and uses the embedding order together with an argument filter to collapse the symbol $\mathsf{s}$ to its argument. Then the DP for the newly added rule is oriented strictly and all other DPs are oriented weakly. After removing the new DP, the SCNP reduction pair that we already used for the DPs of Ex. 2 again orients all DPs strictly. Note that the base order for this second SCNP reduction pair is the embedding order without argument filters, i.e., it differs from the base order used in the first SCNP reduction pair.

By representing the SCNP method via SCNP reduction pairs, we can now benefit from the flexibility of the DP framework. Thus, we can use other termination methods in addition to SCNP. More precisely, as usual in the DP framework, we can apply arbitrary processors one after another in a modular way. This allows us to interleave arbitrary other termination techniques with termination proof steps based on size-change termination, whereas in [28], size-change proofs could only be used as a final step in a termination proof.

## 5 Automation by SAT Encoding

Recently, the search problem for many common base orders has been reduced successfully to SAT problems [7, 8, 12, 25, 29]. In this section, we build on this earlier work and use these encodings as components for a SAT encoding of SCNP reduction pairs. The corresponding decision problem is stated as follows:

> For a DP problem $(\mathcal{P}, \mathcal{R})$ and a given class of base orders, is there a status $\mu$, a level mapping $\ell$, and a concrete base reduction pair $(\succsim, \succ)$ such that the SCNP reduction pair $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ orients all rules of $\mathcal{R}$ and $\mathcal{P}$ weakly and at least one of $\mathcal{P}$ strictly?

We assume a given *base SAT encoding* $[\![.]\!]_{base}$ which maps base term constraints of the form $s \underset{(\sim)}{\succ} t$ to propositional formulas. Every satisfying assignment for the formula $[\![s \underset{(\sim)}{\succ} t]\!]_{base}$ corresponds to a particular order where $s \underset{(\sim)}{\succ} t$ holds.

We also assume a given encoding for partial orders (on tags), cf. [8]. The function $[\![.]\!]_{po}$ maps partial order constraints of the form $n_1 \geq n_2$ or $n_1 > n_2$ where $n_1$ and $n_2$ represent natural numbers (in some fixed number of bits) to corresponding propositional formulas on the bit representations for the numbers.

For brevity, we only show the encoding for SCNP reduction pairs $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ where $\mu = max$. The encodings for the other cases are similar: The encoding for the min comparison is completely analogous. To encode (dual) multiset comparison one can adapt previous approaches to encode multiset orders [25].

First, for each tuple symbol $F$ of arity $n$, we introduce natural number variables denoted $tag_i^F$ for $1 \leq i \leq n$. These encode the tags associated with the argument positions of $F$ by representing them in a fixed number of bits. In our case, it suffices to consider tag values which are less than the sum of the arities of the tuple symbols. In this way, every argument position of every tuple symbol could get a different tag, i.e., this suffices to represent all possible level mappings.

Now consider a size-change graph corresponding to a DP $\delta = s \to t$ with $s = F(s_1, \ldots, s_n)$ and $t = G(t_1, \ldots, t_m)$. The edges of the size-change graph are determined by the base order, which is not fixed. For any $1 \leq i \leq n$ and $1 \leq j \leq m$, we define a propositional formula $weak_{i,j}^{\delta}$ which is *true* iff $\langle s, tag_i^F \rangle \succsim^{\mathbb{N}} \langle t, tag_j^G \rangle$. Similarly, $strict_{i,j}^{\delta}$ is *true* iff $\langle s, tag_i^F \rangle \succ^{\mathbb{N}} \langle t, tag_j^G \rangle$. The definition of

$weak^{\delta}_{i,j}$ and $strict^{\delta}_{i,j}$ corresponds directly to Def. 5. It is based on the encodings $[\![\cdot]\!]_{base}$ and $[\![.]\!]_{po}$ for the base order and for the tags, respectively.

$$\begin{aligned} weak^{\delta}_{i,j} &= [\![s_i \succ t_j]\!]_{base} \ \vee \ ([\![s_i \succsim t_j]\!]_{base} \wedge [\![tag^F_i \geq tag^G_j]\!]_{po}) \\ strict^{\delta}_{i,j} &= [\![s_i \succ t_j]\!]_{base} \ \vee \ ([\![s_i \succsim t_j]\!]_{base} \wedge [\![tag^F_i > tag^G_j]\!]_{po}) \end{aligned}$$

To facilitate the search for level mappings, for each tuple symbol $F$ of arity $n$ we introduce propositional variables $reg^F_i$ for $1 \leq i \leq n$. Here, $reg^F_i$ is *true* iff the $i$-th argument position of $F$ is regarded for comparison. The formulas $[\![s \succsim^{max,\ell} t]\!]$ and $[\![s \succ^{max,\ell} t]\!]$ then encode that the DP $s \to t$ can be oriented weakly or strictly, respectively. By this encoding, one can simultaneously search for a base order that gives rise to the edges in the size-change graph and for a level mapping that satisfies this size-change graph.

$$\begin{aligned} [\![s \succsim^{max,\ell} t]\!] &= \bigwedge_{1 \leq j \leq m} (reg^G_j \to \bigvee_{1 \leq i \leq n} (reg^F_i \wedge weak^{\delta}_{i,j})) \\ [\![s \succ^{max,\ell} t]\!] &= \bigwedge_{1 \leq j \leq m} (reg^G_j \to \bigvee_{1 \leq i \leq n} (reg^F_i \wedge strict^{\delta}_{i,j})) \ \wedge \ \bigvee_{1 \leq i \leq n} reg^F_i \end{aligned}$$

For any DP problem $(\mathcal{P}, \mathcal{R})$ we can now generate a propositional formula which ensures that the corresponding SCNP reduction pair orients all rules from $\mathcal{R}$ and $\mathcal{P}$ weakly and at least one rule from $\mathcal{P}$ strictly:

$$\bigwedge_{l \to r \in \mathcal{R}} [\![l \succsim r]\!]_{base} \quad \wedge \quad \bigwedge_{s \to t \in \mathcal{P}} [\![s \succsim^{max,\ell} t]\!] \quad \wedge \quad \bigvee_{s \to t \in \mathcal{P}} [\![s \succ^{max,\ell} t]\!]$$

Similar to [7, 29], our approach is easily extended to refinements of the DP method where one only regards the *usable* rules of $\mathcal{R}$ and where these usable rules can also depend on the (explicit or implicit) argument filter of the order.

## 6 Implementation and Experiments

We implemented our contributions in the automated termination prover AProVE [14]. To assess their impact, we compared three configurations of AProVE. In the first, we use SCNP reduction pairs in the reduction pair processor of the DP framework. This configuration is parameterized by the choice whether we allow just max comparisons of multisets or all four multiset extensions from Def. 7. Moreover, the configuration is also parameterized by the choice whether we use classical size-change graphs or *extended* size-change graphs as in [28]. In an extended size-change graph, to compare $s = F(s_1, \ldots, s_n)$ with $t = G(t_1, \ldots, t_m)$, the source and target vertices $\{s_1, \ldots, s_n\}$ and $\{t_1, \ldots, t_m\}$ are extended by additional vertices $s$ and $t$, respectively. Now an edge from $s$ to $t_j$ indicates that the whole term $s$ is greater (or equal) to $t_j$, etc. So these additional vertices also allow us to compare the whole terms $s$ and $t$. By adding these vertices, size-change termination incorporates the standard comparison of terms as well.

In the second configuration, we use the base orders directly in the reduction pair processor (i.e., here we disregard SCNP reduction pairs). In the third configuration, we use the implementation of the SCT method as described in [28]. For

| order | | SCNP fast | SCNP max | SCNP all | reduction pairs | SCT [28] |
|---|---|---|---|---|---|---|
| EMB | proved | 346 | 346 | **347** | 325 | 341 |
| | runtime | **2882.6** | 3306.4 | 3628.5 | 2891.3 | 10065.4 |
| LPO | proved | 500 | **530** | 527 | 505 | 385 |
| | runtime | **3093.7** | 5985.5 | 7739.2 | 3698.4 | 10015.5 |
| RPO | proved | 501 | **531** | **531** | 527 | 385 |
| | runtime | **3222.2** | 6384.1 | 8118.0 | 4027.5 | 10053.4 |
| POLO | proved | 477 | **514** | **514** | 511 | 378 |
| | runtime | 3153.6 | 5273.6 | 7124.4 | **2941.7** | 9974.0 |

**Table 1.** Comparison of SCNP reduction pairs to SCT and direct reduction pairs.

a fair comparison, we updated that old implementation from the DP *approach* to the modular DP *framework* and used SAT encodings for the base orders. (While this approach only uses the embedding order and argument filters as the base order for the construction of size-change graphs, it uses more complex orders (containing the base order) to weakly orient the rules from the TRS.)

We considered all 1381 examples from the standard TRS category of the *Termination Problem Data Base* (TPDB version 7.0.2) as used in the *International Termination Competition* 2009.[7] The experiments were run on a 2.66 GHz Intel Core 2 Quad and we used a time limit of 60 seconds per example. We applied SAT4J [19] to transform propositional formulas to conjunctive normal form and the SAT solver MiniSAT2 [10] to check the satisfiability of the resulting formulas.

Table 1 compares the power and runtimes of the three configurations depending on the base order. The column "*order*" indicates the base order: embedding order with argument filters (EMB), lexicographic path order with arbitrary permutations and argument filters (LPO), recursive path order with argument filters (RPO), and linear polynomial interpretations with coefficients from $\{0, 1\}$ (POLO). For the first configuration, we used three different settings: full SCNP reduction pairs with extended size-change graphs ("SCNP all"), SCNP reduction pairs restricted to max-comparisons with extended size-change graphs ("SCNP max"), and SCNP reduction pairs restricted to max comparisons and non-extended size-change graphs ("SCNP fast"). The second and third configuration are called "reduction pairs" and "SCT [28]", respectively. For each experiment, we give the number of TRSs which could be proved terminating ("proved") and the analysis time in seconds for running AProVE on all 1381 TRSs ("runtime"). The "best" numbers are always printed in **bold**. For further details on the experiments, we refer to `http://aprove.informatik.rwth-aachen.de/eval/SCNP`. The table allows the following observations:

**(1)** Our SCNP reduction pairs are *much more powerful and significantly faster* than the implementation of [28]. By integrating the search for the base order with SCNP, our new implementation can use a much larger class of base orders and thus, SCNP reduction pairs can prove significantly more examples. The reason for the relatively low speed of [28] is that this approach iterates through argument filters and then generates and analyzes size-change graphs for each of

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

these argument filters. (So the low speed is not due to the repeated composition of size-change graphs in the SCT criterion.)

**(2)** Our new implementation of SCNP reduction pairs is *more powerful* than using the reduction pairs directly. Note that when using extended size-change graphs, *every* reduction pair can be simulated by an SCNP reduction pair.

**(3)** SCNP reduction pairs *add significant power when used for simple orders* like EMB and LPO. The difference is less dramatic for RPO and POLO. Intuitively, the reason is that SCNP allows for multiset comparisons which are lacking in EMB and LPO, while RPO contains multiset comparisons and POLO can often simulate them. Nevertheless, SCNP also adds some power to RPO and POLO, e.g., by extending them by a concept like "maximum". This even holds for more powerful base orders like matrix orders [11]. In Appendix B, we present a TRS where all existing termination tools fail, but where termination can easily be proved automatically by an SCNP reduction pair with a matrix base order.

## 7 Conclusion

We show that the practically relevant part of size-change termination (SCNP) can be formulated as a reduction pair. Thus, SCNP can be applied in the DP framework, which is used in virtually all termination tools for term rewriting.

Moreover, by combining the search for the base order and for the SCNP level mapping into one search problem, we can automatically find the right base order for constructing size-change graphs. Thus, we now generate program abstractions automatically such that termination of the abstracted programs can be shown.

The implementation in AProVE confirms the usefulness of our contribution. Our experiments indicate that the automation of our technique is more powerful than both the direct use of reduction pairs and the SCT adaptation from [28].

## References


1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. J. Avery. Size-change termination and bound analysis. In *Proc. FLOPS '06*, LNCS 3945, pages 192–207, 2006.
3. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.
4. A. M. Ben-Amram and C. S. Lee. Size-change termination in polynomial time. *ACM Transactions on Programming Languages and Systems*, 29(1), 2007.
5. A. M. Ben-Amram and M. Codish. A SAT-based approach to size change termination with global ranking functions. In *Proc. TACAS '08*, LNCS 4963, pages 218–232, 2008.
6. M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
7. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. LPAR '06*, LNAI 4246, pages 30–44, 2006.
8. M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO

termination. *J. Satisfiability, Boolean Modeling and Computation*, 5:193–215, 2008.
9. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
10. N. Eén and N. Sörensson. MiniSAT. `http://minisat.se`.
11. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *J. Automated Reasoning*, 40(2-3):195–220, 2008.
12. C. Fuhs, J. Giesl, A. Middeldorp, R. Thiemann, P. Schneider-Kamp, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT '07*, LNCS 4501, pages 340–354, 2007.
13. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR '04*, LNAI 3542, pages 301–331, 2005.
14. 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, pages 281–286, 2006.
15. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
16. J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 2010. To appear. Preliminary version appeared in *Proc. RTA '06*, LNCS, 4098, pages 297-312, 2006.
17. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
18. N. D. Jones and N. Bohr. Termination analysis of the untyped lambda calculus. In *Proc. RTA '04*, LNCS 3091, pages 1–23, 2004.
19. D. Le Berre and A. Parrain. SAT4J. `http://www.sat4j.org`.
20. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL '01*, pages 81–92, 2001.
21. C. S. Lee. Ranking functions for size-change termination. *ACM Transactions on Programming Languages and Systems*, 31(3):1–42, 2009.
22. M. T. Nguyen, D. De Schreye, J. Giesl, and P. Schneider-Kamp. Polytool: Polynomial interpretations as a basis for termination analysis of logic programs. *Theory and Practice of Logic Programming*, 2010. To appear.
23. C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. RTA '10*, LIPIcs 6, pages 259–276, 2010.
24. A. Podelski and A. Rybalchenko. Transition Invariants. In *Proc. 19th LICS*, pages 32–41. IEEE, 2004.
25. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. FroCoS '07*, LNAI 4720, pages 267–282, 2007.
26. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM Transactions on Computational Logic*, 11(1):1–52, 2009.
27. D. Sereni and N. D. Jones. Termination analysis of higher-order functional programs. In *Proc. APLAS '05*, LNCS 3780, pages 281–297, 2005.
28. R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, 2005.
29. H. Zankl, N. Hirokawa, and A. Middeldorp. KBO orientability. *Journal of Automated Reasoning*, 43(2):173–201, 2009.

## Appendix A.   Proofs

**Lemma 6 (reduction pairs on tagged terms).** *Let* $(\succsim, \succ)$ *be a reduction pair. Then* $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ *is a non-monotonic reduction pair on tagged terms.*

*Proof.* Reflexivity of $\succsim^{\mathbb{N}}$ follows directly from reflexivity of $\succsim$ and of $\geq$. Transitivity of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$ as well as compatibility follow from a simple case analysis on whether we have strict or weak decrease in the term components. Well-foundedness of $\succ^{\mathbb{N}}$ follows from the well-foundedness of lexicographic combinations of well-founded orders.

We now prove stability of $\succsim^{\mathbb{N}}$ by contradiction. Assume that $\succsim^{\mathbb{N}}$ were not stable, i.e., there exist two tagged terms $\langle s, c\rangle$, $\langle t, d\rangle$ and a substitution $\sigma$ with $\langle s, c\rangle \succsim^{\mathbb{N}} \langle t, d\rangle$ and $\langle s\sigma, c\rangle \not\succsim^{\mathbb{N}} \langle t\sigma, d\rangle$. By definition, this implies $\neg(s\sigma \succ t\sigma \vee (s\sigma \succsim t\sigma \wedge c \geq d))$. But by $\langle s, c\rangle \succsim^{\mathbb{N}} \langle t, d\rangle$, we have $s \succ t \vee (s \succsim t \wedge c \geq d)$ which is a contradiction to the stability of $\succ$ and $\succsim$. Stability of $\succ^{\mathbb{N}}$ is completely analogous.

In the proof of Thm. 12, we make use of *multiset covers* [25] as an equivalent representation for the multiset extension $\succsim^{ms}$ and $\succ^{ms}$. This is not only convenient for the proof, but multiset covers can also be used to express the multiset extension in a way which is natural to encode as a propositional formula.

**Definition 14 (multiset cover).** *Let* $S$ *and* $T$ *be multisets with* $|S| = n$ *and* $|T| = m$. *A multiset cover* $\langle \gamma, \varepsilon\rangle$ *is a pair of mappings* $\gamma : \{1, ..., m\} \to \{1, ..., n\}$ *and* $\varepsilon : \{1, ..., n\} \to \{\mathit{false}, \mathit{true}\}$ *such that for each* $1 \leq i \leq n$, *if* $\varepsilon(i) = \mathit{true}$, *then* $\{\, j \mid \gamma(j) = i \,\}$ *is a singleton set.*

For any $t_j \in T$, $\gamma(j) = i$ intuitively means that $t_j$ is *covered* by $s_i$ (i.e., $s_i \succ t_j$ or $s_i \succsim t_j$). Moreover, $\varepsilon(i)$ indicates whether $s_i$ covers elements from $T$ strictly or weakly. Hence, we can now express the $ms$-extension via multiset covers:

- $S \succsim^{ms} T$ holds iff $S = \{s_1 \ldots, s_n\}$, $T = \{t_1, \ldots, t_m\}$, and there exists a multiset cover $\langle \gamma, \varepsilon\rangle$ such that for all $i, j$:

$$\gamma(j) = i \Rightarrow (\texttt{if}\ \ \varepsilon(i)\ \ \texttt{then}\ \ s_i \succsim t_j\ \ \texttt{else}\ \ s_i \succ t_j)$$

- $S \succ^{ms} T$ holds iff $S \succsim^{ms} T$ and $\varepsilon(i) = \mathit{false}$ for some $i$ (i.e., some $s_i$ is replaced by zero or more strictly smaller elements).

**Theorem 12.** *For* $\mu \in \{max, min, ms, dms\}$, $(\succsim^{\mu,\ell}, \succ^{\mu,\ell})$ *is a reduction pair.*

*Proof.* We first illustrate the proof for $\mu = max$ (the proof for $\mu = min$ is analogous). For base terms, $(\succsim^{max,\ell}, \succ^{max,\ell})$ is like $(\succsim, \succ)$ and thus, here it is clearly a reduction pair. Thus, it suffices to consider comparisons between tuple terms.

1. $\underline{\succsim^{max,\ell} \text{ and } \succ^{max,\ell} \text{ are compatible}}$
   We only show $\succ^{max,\ell} \circ \succsim^{max,\ell} \subseteq \succ^{max,\ell}$. The proof for $\succsim^{max,\ell} \circ \succ^{max,\ell} \subseteq \succ^{max,\ell}$ is analogous. Let $s = F(\overline{s})$, $t = G(\overline{t})$, and $u = H(\overline{u})$ where $s \succ^{max,\ell} t \wedge t \succsim^{max,\ell} u$. Clearly, $s \succ^{max,\ell} t$ means

   $$\ell(F(\overline{s})) \neq \emptyset \wedge \forall \langle t_j, d_j \rangle \in \ell(G(\overline{t})).\ \exists \langle s_i, c_i \rangle \in \ell(F(\overline{s})).\ \langle s_i, c_i \rangle \succ^{\mathbb{N}} \langle t_j, d_j \rangle$$

   Furthermore, $t \succsim^{max,\ell} u$ means

   $$\forall \langle u_j, e_j \rangle \in \ell(H(\overline{u})).\ \exists \langle t_i, d_i \rangle \in \ell(G(\overline{t})).\ \langle t_i, d_i \rangle \succsim^{\mathbb{N}} \langle u_j, e_j \rangle$$

   Since $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$ are compatible by Lemma 6, we obtain

   $$\ell(F(\overline{s})) \neq \emptyset \wedge \forall \langle u_j, e_j \rangle \in \ell(H(\overline{u})).\ \exists \langle s_i, c_i \rangle \in \ell(F(\overline{s})).\ \langle s_i, c_i \rangle \succ^{\mathbb{N}} \langle u_j, e_j \rangle$$

   and thus $s \succ^{max,\ell} t$.

2. $\underline{\succsim^{max,\ell} \text{ is reflexive and both } \succsim^{max,\ell} \text{ and } \succ^{max,\ell} \text{ are transitive and stable}}$
   The claim $F(\overline{s}) \succsim^{max,\ell} F(\overline{s})$ is equivalent to $\ell(F(\overline{s})) \succsim^{\mathbb{N},max} \ell(F(\overline{s}))$, which means

   $$\forall \langle s_j, c_j \rangle \in \ell(F(\overline{s})).\ \exists \langle s_i, c_i \rangle \in \ell(F(\overline{s})).\ \langle s_i, c_i \rangle \succsim^{\mathbb{N}} \langle s_j, c_j \rangle$$

   This holds by choosing $\langle s_i, c_i \rangle = \langle s_j, c_j \rangle$ and by reflexivity of $\succsim^{\mathbb{N}}$ (Lemma 6). Transitivity and stability of $\succsim^{max,\ell}$ and $\succ^{max,\ell}$ follow in an analogous way from the respective properties of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$.

3. $\underline{\succsim^{max,\ell} \text{ is monotonic}}$
   Since $\succsim^{max,\ell}$ is obviously monotonic on base terms, we only consider the case where $s, t$ are base terms with $s \succsim^{max,\ell} t$ (i.e., $s \succsim t$) and $F$ is a tuple symbol. One has to show $s' \succsim^{max,\ell} t'$ for the tuple terms $s' = F(u_1, \ldots, u_{i-1}, s, u_{i+1}, \ldots, u_n)$ and $t' = F(u_1, \ldots, u_{i-1}, t, u_{i+1}, \ldots, u_n)$. Clearly, $s' \succsim^{max,\ell} t'$ holds iff

   $$\forall \langle v_j, d_j \rangle \in \ell(t').\ \exists \langle v_l, d_l \rangle \in \ell(s').\ \langle v_l, d_l \rangle \succsim^{\mathbb{N}} \langle v_j, d_j \rangle$$

   For $\langle v_j, d_j \rangle = \langle u_p, e_p \rangle$ with $p \in \{1, \ldots, i-1, i+1, n\}$, we choose $\langle v_l, d_l \rangle = \langle v_j, d_j \rangle$ (since $\succsim^{\mathbb{N}}$ is reflexive by Lemma 6). For $\langle v_j, d_j \rangle = \langle t, c \rangle$, we choose $\langle v_l, d_l \rangle = \langle s, c \rangle$. By $s \succsim t$ and $c \geq c$, we also have $\langle s, c \rangle \succsim^{\mathbb{N}} \langle t, c \rangle$.

4. $\underline{\succ^{max,\ell} \text{ is well founded}}$
   Analogous to the proof that the standard multiset extension of an order $\succ$ is well founded iff $\succ$ itself is well founded.

We now illustrate the proof for $\mu = ms$ (the proof for $\mu = dms$ is analogous).

1. $\underline{\succsim^{ms,\ell} \text{ and } \succ^{ms,\ell} \text{ are compatible, and both } \succsim^{ms,\ell} \text{ and } \succ^{ms,\ell} \text{ are transitive}}$
   We only show $\succ^{ms,\ell} \circ \succsim^{ms,\ell} \subseteq \succ^{ms,\ell}$. The proofs for $\succsim^{ms,\ell} \circ \succ^{ms,\ell} \subseteq \succ^{ms,\ell}$, $\succ^{ms,\ell} \circ \succ^{ms,\ell} \subseteq \succ^{ms,\ell}$, and $\succsim^{ms,\ell} \circ \succsim^{ms,\ell} \subseteq \succsim^{ms,\ell}$ are analogous. Let $s = F(\overline{s})$, $t = G(\overline{t})$, and $u = H(\overline{u})$ where $s \succ^{ms,\ell} t \wedge t \succsim^{ms,\ell} u$.

Clearly, $s \succ^{ms,\ell} t$ means that there exists a multiset cover $\langle \gamma_1, \varepsilon_1 \rangle$ such that

$$\begin{array}{l} \forall \langle i,c\rangle \in \pi(F).\ \forall \langle j,d\rangle \in \pi(G). \\ \quad \gamma_1(j) = i \Rightarrow (\texttt{if}\ \varepsilon_1(i)\ \texttt{then}\ \langle s_i,c\rangle \succsim^{\mathbb{N}} \langle t_j,d\rangle\ \texttt{else}\ \langle s_i,c\rangle \succ^{\mathbb{N}} \langle t_j,d\rangle) \\ \text{and } \exists \langle i,c\rangle \in \pi(F).\neg\varepsilon(i) \end{array}$$

Moreover, $t \succsim^{ms,\ell} u$ means that there exists a multiset cover $\langle \gamma_2, \varepsilon_2 \rangle$ such that:

$$\begin{array}{l} \forall \langle j,d\rangle \in \pi(G).\ \forall \langle k,e\rangle \in \pi(H). \\ \quad \gamma_2(k) = j \Rightarrow (\texttt{if}\ \varepsilon_2(j)\ \texttt{then}\ \langle t_j,d\rangle \succsim^{\mathbb{N}} \langle u_k,e\rangle\ \texttt{else}\ \langle t_j,d\rangle \succ^{\mathbb{N}} \langle u_k,e\rangle) \end{array}$$

We now construct a multiset cover $\langle \gamma_3, \varepsilon_3 \rangle$ to show that $s \succ^{ms,\ell} u$ holds as well. Let $\gamma_3(k) = \gamma_1(\gamma_2(k))$ and let

$$\varepsilon_3(i) = \begin{cases} \mathit{true}, & \text{if } \varepsilon_1(i) \wedge i = \gamma_1(j) \text{ for some } j \text{ with } \varepsilon_2(j) \\ \mathit{false}, & \text{otherwise} \end{cases}$$

To see that $\langle \gamma_3, \varepsilon_3 \rangle$ actually is a multiset cover showing $s \succ^{ms,\ell} u$, we need to prove:

(a) For each $1 \le i \le n$, if $\varepsilon_3(i)$ then $\{\ k \mid \gamma_3(k) = i\ \}$ is a singleton set.
This holds by construction.

(b) $\forall \langle i,c\rangle \in \pi(F).\ \forall \langle k,e\rangle \in \pi(H).$
$\gamma_3(k) = i \Rightarrow (\texttt{if}\ \varepsilon_3(i)\ \texttt{then}\ \langle s_i,c\rangle \succsim^{\mathbb{N}} \langle u_k,e\rangle\ \texttt{else}\ \langle s_i,c\rangle \succ^{\mathbb{N}} \langle u_k,e\rangle)$
Let $\gamma_3(k) = i$. Then $\exists j.\ j = \gamma_2(k) \wedge i = \gamma_1(j)$. Hence, we have:

$$\langle s_i,c\rangle \mathrel{{}_{(\sim)}^{\succ}}{}^{\mathbb{N}} \langle t_j,d\rangle \mathrel{{}_{(\sim)}^{\succ}}{}^{\mathbb{N}} \langle u_k,e\rangle$$

where $x \mathrel{{}_{(\sim)}^{\succ}}{}^{\mathbb{N}} y$ iff $x \succ^{\mathbb{N}} y$ or $x \succsim^{\mathbb{N}} y$.
By transitivity of $\succsim^{\mathbb{N}}$, then $\varepsilon_1(i) \wedge \varepsilon_2(j)$ implies $\langle s_i,c\rangle \succsim^{\mathbb{N}} \langle u_k,e\rangle$. Likewise, by transitivity of $\succ^{\mathbb{N}}$ and compatibility of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$, $\neg(\varepsilon_1(i) \wedge \varepsilon_2(j))$ implies $\langle s_i,c\rangle \succ^{\mathbb{N}} \langle u_k,e\rangle$.
Since $\langle \gamma_1, \varepsilon_1 \rangle$ and $\langle \gamma_2, \varepsilon_2 \rangle$ are multiset covers, $\varepsilon_1(i) \wedge \varepsilon_2(j)$ holds iff $\varepsilon_3(i)$ holds, which shows the proposition.

(c) $\exists i.\neg\varepsilon_3(i)$.
This holds due to $\exists i.\neg\varepsilon_1(i)$ and $\neg\varepsilon_1(i) \Rightarrow \neg\varepsilon_3(i)$.

2. <u>$\succsim^{ms,\ell}$ and $\succ^{ms,\ell}$ are stable, and $\succsim^{ms,\ell}$ is reflexive.</u>
This follows from the respective properties of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$.

3. <u>$\succsim^{ms,\ell}$ is monotonic</u>
Since $\succsim^{ms,\ell}$ is obviously monotonic on base terms, we only consider the case where $s, t$ are base terms with $s \succsim^{ms,\ell} t$ (i.e., $s \succsim t$), where $F$ is a tuple symbol, and one has to show $s' \succsim^{ms,\ell} t'$ for the tuple terms $s' = F(u_1, \ldots, u_{i-1}, s, u_{i+1}, \ldots, u_n)$ and $t' = F(u_1, \ldots, u_{i-1}, t, u_{i+1}, \ldots, u_n)$. Here the multiset cover $\langle \gamma, \varepsilon \rangle$ with $\forall i'.\varepsilon(i') \wedge \gamma(i') = i'$ implies $s' \succsim^{ms,\ell} t'$.

4. $\underline{\succ^{ms,\ell} \text{ is well founded}}$
   This can be shown via an adaption of the proof that the standard multiset extension of an order $\succ$ is well founded iff $\succ$ itself is well founded. We additionally need to use transitivity and compatibility of $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$.

## Appendix B. Combining SCNP with Matrix Orders

*Example 15.* Consider the following rewrite system $\mathcal{R}_1$, which is taken from the termination problem data base (`TRS/Zantema_06/03.xml`).

$$\begin{aligned} \mathsf{a}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x)))) &\to \mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{a}(x)))))) \\ \mathsf{a}(\mathsf{c}(x)) &\to \mathsf{c}(\mathsf{a}(x)) \\ \mathsf{c}(\mathsf{b}(x)) &\to \mathsf{b}(\mathsf{c}(x)) \end{aligned}$$

When attempting to prove termination by the standard techniques of the DP framework (such as the dependency graph processor or the reduction pair processor with matrix orders over the naturals), the initial DP problem is eventually transformed into the DP problem $(\mathcal{P}_1, \mathcal{R}_1)$ with $\mathcal{P}_1 = \{\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x)))) \to \mathsf{A}(\mathsf{a}(x))\}$.

We can conclude the termination proof of $\mathcal{R}_1$ by deleting the DP of $\mathcal{P}_1$ using the reduction pair processor. To this end, one can use the following matrix order based on an interpretation $\mathcal{M}$ of dimension 4 and matrix entries from $\{0, 1\}$.

$$\begin{aligned}
\mathsf{A}_{\mathcal{M}}(x_1) &= \begin{pmatrix} 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix} \cdot x_1 \\
\mathsf{a}_{\mathcal{M}}(x_1) &= \begin{pmatrix} 0&1&0&0 \\ 1&0&0&0 \\ 0&1&0&1 \\ 1&0&0&0 \end{pmatrix} \cdot x_1 \\
\mathsf{b}_{\mathcal{M}}(x_1) &= \begin{pmatrix} 0&0&1&1 \\ 0&0&0&0 \\ 1&1&0&0 \\ 0&1&0&0 \end{pmatrix} \cdot x_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \\
\mathsf{c}_{\mathcal{M}}(x_1) &= \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix} \cdot x_1
\end{aligned}$$

This example can easily be modified into an example where an SCNP reduction pair is required. The idea is to exploit that SCNP allows us to perform a "max comparison" on the level of the tuple symbols of the dependency pairs. This is a feature that many base orders like matrix orders over the naturals cannot provide.

To this end, we introduce a unary function symbol $\mathsf{dup}$ and a binary function symbol $\mathsf{collapse}$. Then we add a rule where $\mathsf{dup}$ applied to the left-hand side of the DP in $\mathcal{P}_1$ rewrites to *two copies* of the corresponding right-hand side of the DP, wrapped inside the symbol $\mathsf{collapse}$.

$$\mathsf{dup}(\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))) \to \mathsf{collapse}(\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x)))$$

We also have to make sure that we can still continue the rewrite sequence as before. So we add rules to eliminate any of the copies again.

$$\begin{aligned} \mathsf{collapse}(x, y) &\to \mathsf{dup}(x) \\ \mathsf{collapse}(x, y) &\to \mathsf{dup}(y) \end{aligned}$$

We now unite the rules from $\mathcal{R}_1$ with these three new rules and obtain a TRS $\mathcal{R}_2$ that turns out to be very challenging for current automated termination tools.

$$\begin{aligned} \mathsf{a}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x)))) &\to \mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{a}(x)))))) \\ \mathsf{a}(\mathsf{c}(x)) &\to \mathsf{c}(\mathsf{a}(x)) \\ \mathsf{c}(\mathsf{b}(x)) &\to \mathsf{b}(\mathsf{c}(x)) \\ \mathsf{dup}(\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))) &\to \mathsf{collapse}(\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))) \\ \mathsf{collapse}(x, y) &\to \mathsf{dup}(x) \\ \mathsf{collapse}(x, y) &\to \mathsf{dup}(y) \end{aligned}$$

When attempting the termination proof in the DP framework, after a few proof steps, we get the DP problem $(\mathcal{P}_2, \mathcal{R}_2)$ where $\mathcal{P}_2$ consists of the following dependency pairs.

$$\begin{aligned} \mathsf{DUP}(\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))) &\to \mathsf{COLLAPSE}(\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))) \\ \mathsf{COLLAPSE}(x, y) &\to \mathsf{DUP}(x) \\ \mathsf{COLLAPSE}(x, y) &\to \mathsf{DUP}(y) \end{aligned}$$

Now we can delete the first dependency pair using an SCNP reduction pair with the status $\mu = max$. This reduction pair uses the previous matrix order $\mathcal{M}$ when comparing base terms. For comparing tuple terms, it uses a level mapping $\ell$ with $\ell(\mathsf{DUP}) = \{\langle 1, 0\rangle\}$ and $\ell(\mathsf{COLLAPSE}) = \{\langle 1, 0\rangle, \langle 2, 0\rangle\}$ (i.e., here we do not really need the tags in the level mapping). Thus, the first DP is strictly decreasing because for every term in the multiset $\{\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))\}$ of arguments on the right-hand side there is a strictly larger term in the multiset $\{\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))\}$ of arguments on the left-hand side. After this proof step, applying the dependency graph processor on the resulting DP problem concludes the termination proof for $\mathcal{R}_2$.

Note that without SCNP reduction pairs, termination of this example is not easy to prove. When disabling SCNP reduction pairs, AProVE can no longer accomplish the termination proof and we aborted the proof attempt after 15 minutes. Here, we used the reduction pair processor with reduction pairs from the regarded class of base orders (i.e., matrix orders over the naturals with matrices of dimension 4 and entries from $\{0, 1\}$). Indeed, none of the tools that participated in the termination competition 2009 for standard term rewriting succeeded in finding a termination proof for $\mathcal{R}_2$. This shows that even for very powerful classes of orders like matrix orders, it is not difficult to come up with examples where SCNP reduction pairs enable an automated termination proof whereas none could be found otherwise.

## Aachener Informatik-Berichte

**This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from** `http://aib.informatik.rwth-aachen.de/`**. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email:** `biblio@informatik.rwth-aachen.de`

2005-01 * Fachgruppe Informatik: Jahresbericht 2004
2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture
2005-12 Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15 Uwe Naumann: The Complexity of Derivative Computation
2005-16 Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17 Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18 Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"

2005-19 Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-02 Michael Weber: Parallel Algorithms for Verification of Large Systems

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

2006-13 Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities

2006-14 Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"

2006-15 Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices

2006-16 Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness

2006-17 Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines

2007-01 * Fachgruppe Informatik: Jahresbericht 2006

2007-02 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations

2007-03 Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase

2007-04 Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation

2007-05 Uwe Naumann: On Optimal DAG Reversal

2007-06 Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking

2007-07 Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications

2007-08 Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches

2007-09 Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption

2007-10 Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes

2007-11 Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke

2007-12 Uwe Naumann: An L-Attributed Grammar for Adjoint Code

2007-13 Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs

2007-14 Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes

2007-15 Volker Stolz: Temporal assertions for sequential and concurrent programs

2007-16 Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks

2007-17 René Thiemann: The DP Framework for Proving Termination of Term Rewriting

2007-18 Uwe Naumann: Call Tree Reversal is NP-Complete
2007-19 Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control
2007-20 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems
2007-21 Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains
2007-22 Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets
2008-01 * Fachgruppe Informatik: Jahresbericht 2007
2008-02 Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04 Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05 Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06 Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07 Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08 George B. Mertzios, Stavros D. Nikolopoulos: The $\lambda$-cluster Problem on Parameterized Interval Graphs
2008-09 George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10 George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11 George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13 Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14 Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15 Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves
2008-16 Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17 Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18 Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems

2008-19 Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems

2009-02 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications

2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices

2009-04 Daniel Klünder: Entwurf eingebetteter Software mit abstrakten Zustandsmaschinen und Business Object Notation

2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs

2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time

2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme

2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme

2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata
2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms
2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting
2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs
2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut
2010-11 Martin Zimmermann: Parametric LTL Games
2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut
2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems
2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.