Uncurrying for Termination*

Nao Hirokawa[1], Aart Middeldorp[2], and Harald Zankl[2]

[1] School of Information Science
Japan Advanced Institute of Science and Technology, Japan
hirokawa@jaist.ac.jp
[2] Institute of Computer Science
University of Innsbruck, Austria
{aart.middeldorp,harald.zankl}@uibk.ac.at

Abstract. First-order applicative term rewrite systems provide a natural framework for modeling higher-order aspects. In this paper we present a transformation from untyped applicative term rewrite systems to functional term rewrite systems that preserves and reflects termination. Our transformation is less restrictive than other approaches. In particular, head variables in right-hand sides of rewrite rules can be handled. To further increase the applicability of our transformation, we present a version for dependency pairs.

1 Introduction

In this paper we are concerned with proving termination of first-order applicative term rewrite systems. These systems provide a natural framework for modeling higher-order aspects found in functional programming languages. The signature of an applicative term rewrite system consists of constants and a single binary function symbol called application and denoted by the infix and left-associative symbol $\circ$. Proving termination of applicative term rewrite systems is challenging because the rewrite rules lack sufficient structure. As a consequence, simplification orders are not effective as $\circ$ is the only function symbol of non-zero arity. Moreover, the dependency pair method is of little help as $\circ$ is the only defined non-constant symbol.

The main contribution of this paper is a new transformation that recovers the structure in applicative rewrite rules. Our transformation can deal with partial applications as well as head variables in right-hand sides of rewrite rules. The key ingredient is the addition of sufficiently many uncurrying rules to the transformed system. These rules are also crucial for a smooth transition into the dependency pair framework. Unlike the transformation of applicative dependency pair problems presented in [10, 17], our uncurrying processor preserves minimality (cf. Section 6), which means that it can be used at any node in a modular (non-)termination proof attempt.

* This research is supported by FWF (Austrian Science Fund) project P18763, Grant-in-Aid for Young Scientists 20800022 of the Ministry of Education, Culture, Sports, Science and Technology of Japan, and STARC.

The remainder of this paper is organised as follows. After recalling existing results in Section 2, we present a new uncurrying transformation and prove that it is sound and complete for termination in Section 3. Despite its simplicity, the transformation has some subtleties which are illustrated by several examples. Two extensions to the dependency pair framework are presented in Section 4. Our results are empirically evaluated in Section 5 and we conclude with a discussion of related work in Section 6. Parts of Section 3 were first announced in a note by the first two authors that was presented at the 3rd International Workshop on Higher-Order Rewriting (Seattle, 2006).

2 Preliminaries

We assume familiarity with term rewriting [5] in general and termination [20] in particular.

Definition 1. *An* applicative *term rewrite system (ATRS for short) is a TRS over a signature that consists of constants and a single binary function symbol called application denoted by the infix and left-associative symbol* $\circ$*. In examples we often use juxtaposition instead of* $\circ$*.*

Every ordinary TRS can be transformed into an applicative rewrite system by currying.

Definition 2. *Let* $\mathcal{F}$ *be a signature. The currying system* $\mathcal{C}(\mathcal{F})$ *consists of the rewrite rules* $f_{i+1}(x_1,\dots,x_i,y) \to f_i(x_1,\dots,x_i) \circ y$ *for every* n*-ary function symbol* $f \in \mathcal{F}$ *and every* $0 \leqslant i < n$*. Here* $f_n = f$ *and, for every* $0 \leqslant i < n$*,* f_i *is a fresh function symbol of arity* i*.*

The currying system $\mathcal{C}(\mathcal{F})$ is confluent and terminating. Hence every term t has a unique normal form $t{\downarrow_{\mathcal{C}(\mathcal{F})}}$.

Definition 3. *Let* $\mathcal{R}$ *be a TRS over the signature* $\mathcal{F}$*. The* curried *system* $\mathcal{R}{\downarrow_{\mathcal{C}(\mathcal{F})}}$ *is the ATRS consisting of the rules* $l{\downarrow_{\mathcal{C}(\mathcal{F})}} \to r{\downarrow_{\mathcal{C}(\mathcal{F})}}$ *for every* $l \to r \in \mathcal{R}$*. The signature of* $\mathcal{R}{\downarrow_{\mathcal{C}(\mathcal{F})}}$ *contains the application symbol* $\circ$ *and a constant* f_0 *for every function symbol* $f \in \mathcal{F}$*.*

In the following we write $\mathcal{R}{\downarrow_{\mathcal{C}}}$ for $\mathcal{R}{\downarrow_{\mathcal{C}(\mathcal{F})}}$ whenever $\mathcal{F}$ can be inferred from the context or is irrelevant. Moreover, we write f for f_0.

Example 4. The TRS $\mathcal{R} = \{\mathsf{0}{+}y \to y, \mathsf{s}(x){+}y \to \mathsf{s}(x{+}y)\}$ is transformed into the ATRS $\mathcal{R}{\downarrow_{\mathcal{C}}} = \{+\ \mathsf{0}\ y \to y, +\ (\mathsf{s}\ x)\ y \to \mathsf{s}\ (+\ x\ y)\}$. Every rewrite sequence in $\mathcal{R}$ can be transformed into a sequence in $\mathcal{R}{\downarrow_{\mathcal{C}}}$, but the reverse does not hold. For instance, with respect to the above example, the rewrite step $+\ (\mathsf{s}\ (+\ \mathsf{0}))\ \mathsf{0} \to \mathsf{s}\ (+\ (+\ \mathsf{0})\ \mathsf{0})$ in $\mathcal{R}{\downarrow_{\mathcal{C}}}$ does not correspond to a rewrite step in $\mathcal{R}$. Nevertheless, termination of $\mathcal{R}$ implies termination of $\mathcal{R}{\downarrow_{\mathcal{C}}}$.

Theorem 5 (Kennaway *et al.* [15])**.** *A TRS* $\mathcal{R}$ *is terminating if and only if* $\mathcal{R}{\downarrow_{\mathcal{C}}}$ *is terminating.* □

As an immediate consequence we get the following transformation method for proving termination of ATRSs.

Corollary 6. *An ATRS $\mathcal{R}$ is terminating if and only if there exists a terminating TRS $\mathcal{S}$ such that $\mathcal{S}\downarrow_{\mathcal{C}} = \mathcal{R}$ (modulo renaming).* □

In [10] this method is called *transformation $\mathcal{A}$*. As can be seen from the following example, the method does not handle partially applied terms and, more seriously, head variables. Hence the method is of limited applicability as it cannot cope with the higher-order aspects modeled by ATRSs.

Example 7. Consider the ATRS $\mathcal{R}$ (from [2])

$$\begin{array}{lll} 1\colon & \mathsf{id}\ x \to x & 4\colon \quad \mathsf{map}\ f\ \mathsf{nil} \to \mathsf{nil} \\ 2\colon & \mathsf{add}\ 0 \to \mathsf{id} & 5\colon \mathsf{map}\ f\ (:\ x\ y) \to :\ (f\ x)\ (\mathsf{map}\ f\ y) \\ 3\colon & \mathsf{add}\ (\mathsf{s}\ x)\ y \to \mathsf{s}\ (\mathsf{add}\ x\ y) & \end{array}$$

Rules 1 and 4 are readily translated into functional form: $\mathsf{id}_1(x) \to x$ and $\mathsf{map}_2(f, \mathsf{nil}) \to \mathsf{nil}$. However, we cannot find functional forms for rules 2 and 3 because the 'arity' of add is 1 in rule 2 and 2 in rule 3. Because of the presence of the head variable f in the subterm $f\ x$, there is no functional term t such that $t\downarrow_{\mathcal{C}} = :\ (f\ x)\ (\mathsf{map}\ f\ y)$. Hence also rule 5 cannot be transformed.

3 Uncurrying

In this section we present an uncurrying transformation that can deal with ATRSs like in Example 7. Throughout this section we assume that $\mathcal{R}$ is an ATRS over a signature $\mathcal{F}$.

Definition 8. *The* applicative arity $\mathrm{aa}(f)$ *of a constant $f \in \mathcal{F}$ is defined as the maximum n such that $f \circ t_1 \circ \cdots \circ t_n$ is a subterm in the left- or right-hand side of a rule in $\mathcal{R}$. This notion is extended to terms as follows:*

$$\mathrm{aa}(t) = \begin{cases} \mathrm{aa}(f) & \text{if } t \text{ is a constant } f \\ \mathrm{aa}(t_1) - 1 & \text{if } t = t_1 \circ t_2 \end{cases}$$

Note that $\mathrm{aa}(t)$ *is undefined if the head symbol of t is a variable.*

Definition 9. *The uncurrying system $\mathcal{U}(\mathcal{F})$ consists of the following rewrite rules $f_i(x_1, \ldots, x_i) \circ y \to f_{i+1}(x_1, \ldots, x_i, y)$ for every constant $f \in \mathcal{F}$ and every $0 \leqslant i < \mathrm{aa}(f)$. Here $f_0 = f$ and, for every $i > 0$, f_i is a fresh function symbol of arity i. We say that $\mathcal{R}$ is* left head variable free *if* $\mathrm{aa}(t)$ *is defined for every non-variable subterm t of a left-hand side of a rule in $\mathcal{R}$. This means that no subterm of a left-hand side in $\mathcal{R}$ is of the form $t_1 \circ t_2$ where t_1 is a variable.*

The uncurrying system $\mathcal{U}(\mathcal{F})$, or simply $\mathcal{U}$, is confluent and terminating. Hence every term t has a unique normal form $t\downarrow_{\mathcal{U}}$.

Definition 10. *The* uncurried *system $\mathcal{R}{\downarrow_{\mathcal{U}}}$ is the TRS consisting of the rules $l{\downarrow_{\mathcal{U}}} \to r{\downarrow_{\mathcal{U}}}$ for every $l \to r \in \mathcal{R}$.*

Example 11. The ATRS $\mathcal{R}$ of Example 7 is transformed into $\mathcal{R}{\downarrow_{\mathcal{U}}}$:

$$\begin{aligned} \mathsf{id}_1(x) &\to x & \mathsf{map}_2(f, \mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{add}_1(\mathsf{0}) &\to \mathsf{id} & \mathsf{map}_2(f, :_2(x,y)) &\to :_2(f \circ x, \mathsf{map}_2(f,y)) \\ \mathsf{add}_2(\mathsf{s}_1(x), y) &\to \mathsf{s}_1(\mathsf{add}_2(x,y)) \end{aligned}$$

The TRS $\mathcal{R}{\downarrow_{\mathcal{U}}}$ is an obvious candidate for $\mathcal{S}$ in Corollary 6. However, as can be seen from the following example, the rules of $\mathcal{R}{\downarrow_{\mathcal{U}}}$ are not enough to simulate an arbitrary rewrite sequence in $\mathcal{R}$.

Example 12. The non-terminating ATRS $\mathcal{R} = \{\mathsf{id}\ x \to x, \mathsf{f}\ x \to \mathsf{id}\ \mathsf{f}\ x\}$ is transformed into the terminating TRS $\mathcal{R}{\downarrow_{\mathcal{U}}} = \{\mathsf{id}_1(x) \to x, \mathsf{f}_1(x) \to \mathsf{id}_2(\mathsf{f}, x)\}$. Note that $\mathcal{R}{\downarrow_{\mathcal{U}}}{\downarrow_{\mathcal{C}}} = \{\mathsf{id}_1\ x \to x, \mathsf{f}_1\ x \to \mathsf{id}_2\ \mathsf{f}\ x\}$ is different from $\mathcal{R}$.

In the above example we need rules that connect id_2 and id_1 as well as f_1 and f. The natural idea is now to add $\mathcal{U}(\mathcal{F})$. In the following we write $\mathcal{U}^+(\mathcal{R}, \mathcal{F})$ for $\mathcal{R}{\downarrow_{\mathcal{U}(\mathcal{F})}} \cup \mathcal{U}(\mathcal{F})$. If $\mathcal{F}$ can be inferred from the context or is irrelevant, $\mathcal{U}^+(\mathcal{R}, \mathcal{F})$ is abbreviated to $\mathcal{U}^+(\mathcal{R})$.

Example 13. Consider the ATRS $\mathcal{R}$ in Example 12. We have $\mathrm{aa}(\mathsf{id}) = 2$ and $\mathrm{aa}(\mathsf{f}) = 1$. The TRS $\mathcal{U}^+(\mathcal{R})$ consists of the following rules

$$\begin{aligned} \mathsf{id}_1(x) &\to x & \mathsf{id} \circ x &\to \mathsf{id}_1(x) & \mathsf{f} \circ x &\to \mathsf{f}_1(x) \\ \mathsf{f}_1(x) &\to \mathsf{id}_2(\mathsf{f}, x) & \mathsf{id}_1(x) \circ y &\to \mathsf{id}_2(x,y) \end{aligned}$$

and is easily shown to be terminating.

As the above example shows, we do not yet have a sound transformation. The ATRS $\mathcal{R}$ admits the cycle $\mathsf{f}\ x \to \mathsf{id}\ \mathsf{f}\ x \to \mathsf{f}\ x$. In $\mathcal{U}^+(\mathcal{R})$ we have $\mathsf{f}_1(x) \to \mathsf{id}_2(\mathsf{f}, x)$ but the term $\mathsf{id}_2(\mathsf{f}, x)$ does not rewrite to $\mathsf{f}_1(x)$. It would if the rule $\mathsf{id}\ x\ y \to x\ y$ were present in $\mathcal{R}$. This inspires the following definition.

Definition 14. *Let $\mathcal{R}$ be a left head variable free ATRS. The* η-saturated *ATRS $\mathcal{R}_\eta$ is the smallest extension of $\mathcal{R}$ such that $l \circ x \to r \circ x \in \mathcal{R}_\eta$ whenever $l \to r \in \mathcal{R}_\eta$ and $\mathrm{aa}(l) > 0$. Here x is a variable that does not appear in $l \to r$.*

The rules added during η-saturation do not affect the termination behaviour of $\mathcal{R}$, according to the following lemma whose straightforward proof is omitted. Moreover, $\mathcal{R}_\eta$ is left head variable free if and only if $\mathcal{R}$ is left head variable free.

Lemma 15. *If $\mathcal{R}$ is a left head variable free ATRS then $\to_{\mathcal{R}} = \to_{\mathcal{R}_\eta}$.* □

We can now state the main result of this section.

Theorem 16. *A left head variable free ATRS $\mathcal{R}$ is terminating if $\mathcal{U}^+(\mathcal{R}_\eta)$ is terminating.*

It is important to note that the applicative arities used in the definition of $\mathcal{U}^+(\mathcal{R}_\eta)$ are computed before η-saturation.

Example 17. The non-terminating ATRS $\mathcal{R} = \{\mathsf{f} \to \mathsf{g}\ \mathsf{a}, \mathsf{g} \to \mathsf{f}\}$ is transformed into $\mathcal{U}^+(\mathcal{R}_\eta) = \{\mathsf{f} \to \mathsf{g}_1(\mathsf{a}), \mathsf{g} \to \mathsf{f}, \mathsf{g}_1(x) \to \mathsf{f} \circ x, \mathsf{g} \circ x \to \mathsf{g}_1(x)\}$ because $\mathrm{aa}(\mathsf{f}) = 0$. The resulting TRS is non-terminating. Uncurrying with $\mathrm{aa}(\mathsf{f}) = 1$ produces the terminating TRS $\{\mathsf{f} \to \mathsf{g}_1(\mathsf{a}), \mathsf{g} \to \mathsf{f}, \mathsf{g}_1(x) \to \mathsf{f}_1(x), \mathsf{g} \circ x \to \mathsf{g}_1(x), \mathsf{f} \circ x \to \mathsf{f}_1(x)\}$.

Before presenting the proof of Theorem 16, we revisit Example 7.

Example 18. Consider again the ATRS $\mathcal{R}$ of Example 7. Proving termination of the transformed TRS $\mathcal{U}^+(\mathcal{R}_\eta)$

$$
\begin{array}{rcr}
\mathsf{id}_1(x) \to x & \mathsf{:} \circ x \to \mathsf{:}_1(x) & \mathsf{id} \circ x \to \mathsf{id}_1(x) \\
\mathsf{add}_1(0) \to \mathsf{id} & \mathsf{:}_1(x) \circ y \to \mathsf{:}_2(x, y) & \mathsf{add} \circ x \to \mathsf{add}_1(x) \\
\mathsf{add}_2(0, y) \to \mathsf{id}_1(y) & & \mathsf{add}_1(x) \circ y \to \mathsf{add}_2(x, y) \\
\mathsf{add}_2(\mathsf{s}_1(x), y) \to \mathsf{s}_1(\mathsf{add}_2(x, y)) & & \mathsf{s} \circ x \to \mathsf{s}_1(x) \\
\mathsf{map}_2(f, \mathsf{nil}) \to \mathsf{nil} & & \mathsf{map} \circ x \to \mathsf{map}_1(x) \\
\mathsf{map}_2(f, \mathsf{:}_2(x, y)) \to \mathsf{:}_2(f \circ x, \mathsf{map}_2(f, y)) & & \mathsf{map}_1(x) \circ y \to \mathsf{map}_2(x, y)
\end{array}
$$

is straightforward with the dependency pair method (recursive SCC algorithm with three applications of the subterm criterion).

The following two lemmata state factorisation properties which are used in the proof of Theorem 16. The easy induction proofs are omitted.

Lemma 19. *Let s and t be terms. If $\mathrm{aa}(s) > 0$ then $s{\downarrow_\mathcal{U}} \circ t{\downarrow_\mathcal{U}} \to_\mathcal{U}^* (s \circ t){\downarrow_\mathcal{U}}$. If $\mathrm{aa}(s) \leqslant 0$ or if $\mathrm{aa}(s)$ is undefined then $s{\downarrow_\mathcal{U}} \circ t{\downarrow_\mathcal{U}} = (s \circ t){\downarrow_\mathcal{U}}$.* □

For a substitution σ, we write $\sigma{\downarrow_\mathcal{U}}$ for the substitution $\{x \mapsto \sigma(x){\downarrow_\mathcal{U}} \mid x \in \mathcal{V}\}$.

Lemma 20. *Let σ be a substitution. For every term t, $t{\downarrow_\mathcal{U}}\sigma{\downarrow_\mathcal{U}} \to_\mathcal{U}^* (t\sigma){\downarrow_\mathcal{U}}$. If t is head variable free then $t{\downarrow_\mathcal{U}}\sigma{\downarrow_\mathcal{U}} = (t\sigma){\downarrow_\mathcal{U}}$.* □

Proof (of Theorem 16). We show that $s{\downarrow_\mathcal{U}} \to^+_{\mathcal{U}^+(\mathcal{R}_\eta)} t{\downarrow_\mathcal{U}}$ whenever $s \to_{\mathcal{R}_\eta} t$. This entails that any infinite $\mathcal{R}_\eta$ derivation is transformed into an infinite $\mathcal{U}^+(\mathcal{R}_\eta)$ derivation. The theorem follows from this observation and Lemma 15. Let $s = C[l\sigma]$ and $t = C[r\sigma]$ with $l \to r \in \mathcal{R}_\eta$. We use induction on the size of the context C.

- If $C = \Box$ then $s{\downarrow_\mathcal{U}} = (l\sigma){\downarrow_\mathcal{U}} = l{\downarrow_\mathcal{U}}\sigma{\downarrow_\mathcal{U}}$ and $r{\downarrow_\mathcal{U}}\sigma{\downarrow_\mathcal{U}} \to_\mathcal{U}^* (r\sigma){\downarrow_\mathcal{U}} = t{\downarrow_\mathcal{U}}$ by Lemma 20. Hence $s{\downarrow_\mathcal{U}} \to^+_{\mathcal{U}^+(\mathcal{R}_\eta)} t{\downarrow_\mathcal{U}}$.
- Suppose $C = \Box \circ s_1 \circ \cdots \circ s_n$ and $n > 0$. Since $\mathcal{R}_\eta$ is left head variable free, $\mathrm{aa}(l)$ is defined. If $\mathrm{aa}(l) = 0$ then

$$
\begin{aligned}
s{\downarrow_\mathcal{U}} &= (l\sigma \circ s_1 \circ \cdots \circ s_n){\downarrow_\mathcal{U}} = l\sigma{\downarrow_\mathcal{U}} \circ s_1{\downarrow_\mathcal{U}} \circ \cdots \circ s_n{\downarrow_\mathcal{U}} \\
&= l{\downarrow_\mathcal{U}}\sigma{\downarrow_\mathcal{U}} \circ s_1{\downarrow_\mathcal{U}} \circ \cdots \circ s_n{\downarrow_\mathcal{U}}
\end{aligned}
$$

and

$$\begin{aligned} r\downarrow_{\mathcal{U}}\sigma\downarrow_{\mathcal{U}} \circ s_1\downarrow_{\mathcal{U}} \circ \cdots \circ s_n\downarrow_{\mathcal{U}} &\to^*_{\mathcal{U}} (r\sigma)\downarrow_{\mathcal{U}} \circ s_1\downarrow_{\mathcal{U}} \circ \cdots \circ s_n\downarrow_{\mathcal{U}} \\ &\to^*_{\mathcal{U}} (r\sigma \circ s_1 \circ \cdots \circ s_n)\downarrow_{\mathcal{U}} = t\downarrow_{\mathcal{U}} \end{aligned}$$

by applications of Lemmata 19 and 20. Hence $s\downarrow_{\mathcal{U}} \to^+_{\mathcal{U}^+(\mathcal{R}_\eta)} t\downarrow_{\mathcal{U}}$. If $\mathrm{aa}(l) > 0$ then $l \circ x \to r \circ x \in \mathcal{R}_\eta$ for some fresh variable x. We have $s = C'[(l \circ x)\tau]$ and $t = C'[(r \circ x)\tau]$ for the context $C' = \Box \circ s_2 \circ \cdots \circ s_n$ and the substitution $\tau = \sigma \cup \{x \mapsto s_1\}$. Since C' is smaller than C, we can apply the induction hypothesis which yields the desired result.

– In the remaining case $C = s_1 \circ C'$. The induction hypothesis yields

$$C'[l\sigma]\downarrow_{\mathcal{U}} \to^+_{\mathcal{U}^+(\mathcal{R}_\eta)} C'[r\sigma]\downarrow_{\mathcal{U}}$$

If $\mathrm{aa}(s_1) \leqslant 0$ or if $\mathrm{aa}(s_1)$ is undefined then $s\downarrow_{\mathcal{U}} = s_1\downarrow_{\mathcal{U}} \circ C'[l\sigma]\downarrow_{\mathcal{U}}$ and $t\downarrow_{\mathcal{U}} = s_1\downarrow_{\mathcal{U}} \circ C'[r\sigma]\downarrow_{\mathcal{U}}$ by Lemma 19. If $\mathrm{aa}(s_1) > 0$ then $s_1\downarrow_{\mathcal{U}} = f_i(u_1, \ldots, u_i)$ for the head symbol f of s_1 and some terms $u_1, \ldots, u_i$. So

$$s\downarrow_{\mathcal{U}} = f_{i+1}(u_1, \ldots, u_i, C'[l\sigma]\downarrow_{\mathcal{U}})$$

and

$$t\downarrow_{\mathcal{U}} = f_{i+1}(u_1, \ldots, u_i, C'[r\sigma]\downarrow_{\mathcal{U}})$$

Hence in both cases we obtain $s\downarrow_{\mathcal{U}} \to^+_{\mathcal{U}^+(\mathcal{R}_\eta)} t\downarrow_{\mathcal{U}}$. □

The next example shows that the left head variable freeness condition cannot be weakened to the well-definedness of $\mathrm{aa}(l)$ for every left-hand side l.

Example 21. Consider the non-terminating ATRS $\mathcal{R} = \{\mathsf{f}\ (x\ \mathsf{a}) \to \mathsf{f}\ (\mathsf{g}\ \mathsf{b}), \mathsf{g}\ \mathsf{b} \to \mathsf{h}\ \mathsf{a}\}$. The transformed TRS $\mathcal{U}^+(\mathcal{R}_\eta)$ consists of the rules

$$\begin{array}{lll} \mathsf{f}_1(x \circ \mathsf{a}) \to \mathsf{f}_1(\mathsf{g}_1(\mathsf{b})) & \mathsf{f} \circ x \to \mathsf{f}_1(x) & \mathsf{h} \circ x \to \mathsf{h}_1(x) \\ \mathsf{g}_1(\mathsf{b}) \to \mathsf{h}_1(\mathsf{a}) & \mathsf{g} \circ x \to \mathsf{g}_1(x) & \end{array}$$

and is terminating because its rules are oriented from left to right by the lexicographic path order with precedence $\circ \succ \mathsf{g}_1 \succ \mathsf{f}_1 \succ \mathsf{h}_1 \succ \mathsf{a} \succ \mathsf{b}$. Note that $\mathrm{aa}(\mathsf{f}\ (x\ \mathsf{a})) = 0$.

The uncurrying transformation is not always useful.

Example 22. Consider the one-rule TRS $\mathcal{R} = \{\mathsf{C}\ x\ y\ z\ u \to x\ z\ (x\ y\ z\ u)\}$ from [7]. The termination of $\mathcal{R}$ is proved by the lexicographic path order with empty precedence. The transformed TRS $\mathcal{U}^+(\mathcal{R}_\eta)$ consists of

$$\begin{array}{rl} \mathsf{C}_4(x, y, z, u) \to x \circ z \circ (x \circ y \circ z \circ u) & \\ \mathsf{C} \circ x \to \mathsf{C}_1(x) & \mathsf{C}_2(x, y) \circ z \to \mathsf{C}_3(x, y, z) \\ \mathsf{C}_1(x) \circ y \to \mathsf{C}_2(x, y) & \mathsf{C}_3(x, y, z) \circ u \to \mathsf{C}_4(x, y, z, u) \end{array}$$

None of the tools that participated in the termination competitions between 2005 and 2007 is able to prove the termination of this TRS.

We show that the converse of Theorem 16 also holds. Hence the uncurrying transformation is not only sound but also complete for termination. (This does not contradict the preceding example.)

Definition 23. *For a term t over the signature of the TRS $\mathcal{U}^+(\mathcal{R})$, we denote by $t{\downarrow_{\mathcal{C}'}}$ the result of identifying different function symbols in $t{\downarrow_{\mathcal{C}}}$ that originate from the same function symbol in $\mathcal{F}$. The notation ${\downarrow_{\mathcal{C}'}}$ is extended to TRSs and substitutions in the obvious way.*

Example 24. For the ATRS $\mathcal{R}$ of Example 12 we have $\mathcal{R}{\downarrow_{\mathcal{U}}}{\downarrow_{\mathcal{C}'}} = \mathcal{R}$.

Lemma 25. *For every t, C, and σ, $C[t\sigma]{\downarrow_{\mathcal{C}'}} = C{\downarrow_{\mathcal{C}'}}[t{\downarrow_{\mathcal{C}'}}\sigma{\downarrow_{\mathcal{C}'}}]$.*

Proof. Straightforward induction on C and t. □

Lemma 26. *Let $\mathcal{R}$ be a left head variable free ATRS. If s and t are terms over the signature of $\mathcal{U}^+(\mathcal{R})$ then $s \to_{\mathcal{R}_\eta{\downarrow_{\mathcal{U}}}} t$ if and only if $s{\downarrow_{\mathcal{C}'}} \to_{\mathcal{R}_\eta} t{\downarrow_{\mathcal{C}'}}$.*

Proof. This follows from Lemma 25 and the fact that $\mathcal{R}_\eta{\downarrow_{\mathcal{U}}}{\downarrow_{\mathcal{C}'}} = \mathcal{R}_\eta$. □

Lemma 27. *Let $\mathcal{R}$ be a left head variable free ATRS. If s and t are terms over the signature of $\mathcal{U}^+(\mathcal{R})$ and $s \to_{\mathcal{U}} t$ then $s{\downarrow_{\mathcal{C}'}} = t{\downarrow_{\mathcal{C}'}}$.*

Proof. This follows from Lemma 25 in connection with the observation that all rules in $\mathcal{U}{\downarrow_{\mathcal{C}'}}$ have equal left- and right-hand sides. □

Theorem 28. *If a left head variable free ATRS $\mathcal{R}$ is terminating then $\mathcal{U}^+(\mathcal{R}_\eta)$ is terminating.*

Proof. Assume that $\mathcal{U}^+(\mathcal{R}_\eta)$ is non-terminating. Since $\mathcal{U}$ is terminating, any infinite rewrite sequence has the form $s_1 \to_{\mathcal{R}_\eta{\downarrow_{\mathcal{U}}}} t_1 \to_{\mathcal{U}}^* s_2 \to_{\mathcal{R}_\eta{\downarrow_{\mathcal{U}}}} t_2 \to_{\mathcal{U}}^* \cdots$. Applications of Lemmata 26 and 27 transform this sequence into $s_1{\downarrow_{\mathcal{C}'}} \to_{\mathcal{R}_\eta} t_1{\downarrow_{\mathcal{C}'}} = s_2{\downarrow_{\mathcal{C}'}} \to_{\mathcal{R}_\eta} t_2{\downarrow_{\mathcal{C}'}} = \cdots$. It follows that $\mathcal{R}_\eta$ is non-terminating. Since $\to_{\mathcal{R}} = \to_{\mathcal{R}_\eta}$ by Lemma 15, we conclude that $\mathcal{R}$ is non-terminating. □

We conclude this section by describing a trivial mirroring technique for TRSs. This technique can be used to eliminate some of the left head variables in an ATRS.

Definition 29. *Let t be a term. The term t^M is defined as follows: $t^M = t$ if t is a variable and $t^M = f(t_n^M, \ldots, t_1^M)$ if $t = f(t_1, \ldots, t_n)$. Moreover, if $\mathcal{R}$ is a TRS then $\mathcal{R}^M = \{l^M \to r^M \mid l \to r \in \mathcal{R}\}$.*

We obviously have $s \to_{\mathcal{R}} t$ if and only if $s^M \to_{\mathcal{R}^M} t^M$. This gives the following result.

Theorem 30. *A TRS $\mathcal{R}$ is terminating if and only if $\mathcal{R}^M$ is terminating.* □

Example 31. Consider the one-rule ATRS $\mathcal{R} = \{x\ (\mathsf{a}\ \mathsf{a}\ \mathsf{a}) \to \mathsf{a}\ (\mathsf{a}\ \mathsf{a})\ x\}$. While $\mathcal{R}$ has a head variable in its left-hand side, the mirrored version $\mathcal{R}^M = \{\mathsf{a}\ (\mathsf{a}\ \mathsf{a})\ x \to x\ (\mathsf{a}\ \mathsf{a}\ \mathsf{a})\}$ is left head variable free. The transformed TRS $\mathcal{U}^+((\mathcal{R}^M)_\eta)$

$$\mathsf{a}_2(\mathsf{a}_1(\mathsf{a}), x) \to x \circ \mathsf{a}_2(\mathsf{a}, \mathsf{a}) \qquad \mathsf{a} \circ x \to \mathsf{a}_1(x) \qquad \mathsf{a}_1(x) \circ y \to \mathsf{a}_2(x, y)$$

is easily proved terminating with dependency pairs and a linear polynomial interpretation.

4 Uncurrying with Dependency Pairs

In this section we incorporate the uncurrying transformation into the dependency pair framework [4,9,11,13,17]. Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. The signature $\mathcal{F}$ is extended with *dependency pair symbols* $f^\sharp$ for every symbol $f \in \{\mathrm{root}(l) \mid l \to r \in \mathcal{R}\}$, where $f^\sharp$ has the same arity as f, resulting in the signature $\mathcal{F}^\sharp$. If $l \to r \in \mathcal{R}$ and t is a subterm of r with a defined root symbol that is not a proper subterm of l then the rule $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. Here $l^\sharp$ and $t^\sharp$ are the result of replacing the root symbols in l and t by the corresponding dependency pair symbols. The set of dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$. A *DP problem* is a pair of TRSs $(\mathcal{P}, \mathcal{R})$ such that the root symbols of the rules in $\mathcal{P}$ do neither occur in $\mathcal{R}$ nor in proper subterms of the left- and right-hand sides of rules in $\mathcal{P}$. The problem is said to be *finite* if there is no infinite sequence $s_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_1 \to^*_{\mathcal{R}} s_2 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to^*_{\mathcal{R}} \cdots$ such that all terms $t_1, t_2, \ldots$ are terminating with respect to $\mathcal{R}$. Such an infinite sequence is said to be *minimal*. The main result underlying the dependency pair approach states that termination of a TRS $\mathcal{R}$ is equivalent to finiteness of the DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$.

In order to prove a DP problem finite, a number of *DP processors* have been developed. DP processors are functions that take a DP problem as input and return a set of DP problems as output. In order to be employed to prove termination they need to be *sound*, that is, if all DP problems in a set returned by a DP processor are finite then the initial DP problem is finite. In addition, to ensure that a DP processor can be used to prove non-termination it must be *complete* which means that if one of the DP problems returned by the DP processor is not finite then the original DP problem is not finite.

In this section we present two DP processors that uncurry applicative DP problems, which are DP problems over applicative signatures containing two application symbols: $\circ$ and $\circ^\sharp$.

4.1 Uncurrying Processor

Definition 32. *Let $(\mathcal{P}, \mathcal{R})$ be an applicative DP problem. The DP processor $\mathcal{U}_1$ is defined as*

$$(\mathcal{P}, \mathcal{R}) \mapsto \begin{cases} \{(\mathcal{P}{\downarrow_{\mathcal{U}(\mathcal{F})}}, \mathcal{U}^+(\mathcal{R}_\eta, \mathcal{F}))\} & \text{if } \mathcal{P} \cup \mathcal{R} \text{ is left head variable free} \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

where $\mathcal{F}$ consists of all function symbols of $\mathcal{P} \cup \mathcal{R}$ minus the root symbols of $\mathcal{P}$.

Theorem 33. *The DP processor $\mathcal{U}_1$ is sound and complete.*

Proof. Let $\mathcal{F}$ be the set of function symbols of $\mathcal{P} \cup \mathcal{R}$ minus the root symbols of $\mathcal{P}$. We first show soundness. Let $(\mathcal{P}, \mathcal{R})$ be an applicative DP problem with the property that $\mathcal{P} \cup \mathcal{R}$ is left head variable free. Suppose the DP problem $(\mathcal{P}{\downarrow_{\mathcal{U}}}, \mathcal{U}^+(\mathcal{R}_\eta))$ is finite. We have to show that $(\mathcal{P}, \mathcal{R})$ is finite. Suppose to the

contrary that $(\mathcal{P},\mathcal{R})$ is not finite. So there exists a minimal rewrite sequence

$$s_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_1 \to^*_{\mathcal{R}} s_2 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to^*_{\mathcal{R}} \cdots \tag{1}$$

By Lemmata 15 and 20 together with the claim in the proof of Theorem 16, this sequence can be transformed into $s_1{\downarrow_{\mathcal{U}}} \xrightarrow{\epsilon}_{\mathcal{P}\downarrow_{\mathcal{U}}} u_1 \to^*_{\mathcal{U}} t_1{\downarrow_{\mathcal{U}}} \to^*_{\mathcal{U}^+(\mathcal{R}_\eta)} s_2{\downarrow_{\mathcal{U}}} \xrightarrow{\epsilon}_{\mathcal{P}\downarrow_{\mathcal{U}}} u_2 \to^*_{\mathcal{U}} t_2{\downarrow_{\mathcal{U}}} \to^*_{\mathcal{U}^+(\mathcal{R}_\eta)} \cdots$. It remains to show that all terms u_1, u_2, ... are terminating with respect to $\mathcal{U}^+(\mathcal{R}_\eta)$. Fix i. We have $u_i{\downarrow_{\mathcal{C}'}} = t_i{\downarrow_{\mathcal{U}}}{\downarrow_{\mathcal{C}'}} = t_i$. Due to the minimality of (1), t_i is terminating with respect to $\mathcal{R}$ and, according to Lemma 15, also with respect to $\mathcal{R}_\eta$. Hence, due to the proof of Theorem 28, u_i is terminating with respect to $\mathcal{U}^+(\mathcal{R}_\eta)$.

Next we show completeness of the DP processor $\mathcal{U}_1$. So let $(\mathcal{P},\mathcal{R})$ be an applicative DP problem with the property that $\mathcal{P}\cup\mathcal{R}$ is left head variable free and suppose that the DP problem $(\mathcal{P}{\downarrow_{\mathcal{U}}},\mathcal{U}^+(\mathcal{R}_\eta))$ is not finite. So there exists a minimal rewrite sequence $s_1 \xrightarrow{\epsilon}_{\mathcal{P}\downarrow_{\mathcal{U}}} t_1 \to^*_{\mathcal{U}^+(\mathcal{R}_\eta)} s_2 \xrightarrow{\epsilon}_{\mathcal{P}\downarrow_{\mathcal{U}}} t_2 \to^*_{\mathcal{U}^+(\mathcal{R}_\eta)} \cdots$. Using Lemmata 26 and 27 this sequence can be transformed into $s_1{\downarrow_{\mathcal{C}'}} \xrightarrow{\epsilon}_{\mathcal{P}} t_1{\downarrow_{\mathcal{C}'}} \to^*_{\mathcal{R}_\eta} s_2{\downarrow_{\mathcal{C}'}} \xrightarrow{\epsilon}_{\mathcal{P}} t_2{\downarrow_{\mathcal{C}'}} \to^*_{\mathcal{R}_\eta} \cdots$. In order to conclude that the DP problem $(\mathcal{P},\mathcal{R})$ is not finite, it remains to show that the terms $t_1{\downarrow_{\mathcal{C}'}}$, $t_2{\downarrow_{\mathcal{C}'}}$, ... are terminating with respect to $\mathcal{R}_\eta$. This follows from the assumption that the terms t_1, t_2, ... are terminating with respect to $\mathcal{U}^+(\mathcal{R}_\eta)$ in connection with Lemma 26. □

The following example from [17] shows that the $\mathcal{A}$ transformation of [10] is not sound because it does not preserve minimality.[3]

Example 34. Consider the applicative DP problem $(\mathcal{P},\mathcal{R})$ with $\mathcal{P}$ consisting of the rewrite rule $(\mathsf{g}\ x)\ (\mathsf{h}\ y)^\sharp\ z \to z\ z^\sharp\ z$ and $\mathcal{R}$ consisting of the rules

$$\begin{aligned} \mathsf{c}\ x\ y &\to x & \mathsf{c}\ (\mathsf{g}\ x)\ y &\to \mathsf{c}\ (\mathsf{g}\ x)\ y \\ \mathsf{c}\ x\ y &\to y & \mathsf{c}\ x\ (\mathsf{g}\ y) &\to \mathsf{c}\ x\ (\mathsf{g}\ y) \end{aligned}$$

The DP problem $(\mathcal{P},\mathcal{R})$ is not finite because of the following minimal rewrite sequence:

$$\begin{aligned} (\mathsf{g}\ x)\ (\mathsf{h}\ x)^\sharp\ (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x) &\xrightarrow{\epsilon}_{\mathcal{P}} (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x)\ (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x)^\sharp\ (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x) \\ &\to_{\mathcal{R}} (\mathsf{g}\ x)\ (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x)^\sharp\ (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x) \\ &\to_{\mathcal{R}} (\mathsf{g}\ x)\ (\mathsf{h}\ x)^\sharp\ (\mathsf{c}\ \mathsf{g}\ \mathsf{h}\ x) \end{aligned}$$

Applying the DP processor $\mathcal{U}_1$ produces $(\mathcal{P}{\downarrow_{\mathcal{U}}},\mathcal{U}^+(\mathcal{R}_\eta))$ with $\mathcal{P}{\downarrow_{\mathcal{U}}}$ consisting of the rewrite rule $\mathsf{g}_1(x)\circ\mathsf{h}_1(y)\circ^\sharp z \to z\circ z\circ^\sharp z$ and $\mathcal{U}^+(\mathcal{R}_\eta)$ consisting of the rules

$$\begin{aligned} \mathsf{c}_2(x,y) &\to x & \mathsf{c}_2(\mathsf{g}_1(x),y) &\to \mathsf{c}_2(\mathsf{g}_1(x),y) & \mathsf{g}\circ x &\to \mathsf{g}_1(x) \\ \mathsf{c}_2(x,y) &\to y & \mathsf{c}_2(x,\mathsf{g}_1(y)) &\to \mathsf{c}_2(x,\mathsf{g}_1(y)) & \mathsf{h}\circ x &\to \mathsf{h}_1(x) \\ \mathsf{c}\circ x &\to \mathsf{c}_1(x) & \mathsf{c}_1(x)\circ y &\to \mathsf{c}_2(x,y) \end{aligned}$$

[3] Since minimality is not part of the definition of finite DP problems in [10], this does not contradict the results in [10].

This DP problem is not finite:

$$\begin{aligned}\mathsf{g}_1(x) \circ \mathsf{h}_1(x) \circ^\sharp (\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x) &\xrightarrow{\epsilon}_{\mathcal{P}\downarrow_\mathcal{U}} (\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x) \circ (\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x) \circ^\sharp (\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x)\\ &\to^*_{\mathcal{U}^+(\mathcal{R}_\eta)} (\mathsf{g} \circ x) \circ (\mathsf{h} \circ x) \circ^\sharp (\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x)\\ &\to^*_{\mathcal{U}^+(\mathcal{R}_\eta)} \mathsf{g}_1(x) \circ \mathsf{h}_1(x) \circ^\sharp (\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x)\end{aligned}$$

Note that $\mathsf{c}_2(\mathsf{g},\mathsf{h}) \circ x$ is terminating with respect to $\mathcal{U}^+(\mathcal{R}_\eta)$.

The uncurrying rules are essential in this example, even though in the original DP problem all occurrences of each constant have the same number of arguments. Indeed, the $\mathcal{A}$ transformation leaves out the uncurrying rules, resulting in a DP problem that admits infinite rewrite sequences but no minimal ones since one has to instantiate the variable z in $\mathsf{g}_1(x) \circ \mathsf{h}_1(y) \circ^\sharp z \to z \circ z \circ^\sharp z$ by a term that contains a subterm of the form $\mathsf{c}_2(\mathsf{g}_1(s), t)$ or $\mathsf{c}_2(s, \mathsf{g}_1(t))$ and the rules $\mathsf{c}_2(\mathsf{g}_1(x), y) \to \mathsf{c}_2(\mathsf{g}_1(x), y)$ and $\mathsf{c}_2(x, \mathsf{g}_1(y)) \to \mathsf{c}_2(x, \mathsf{g}_1(y))$ ensure that these terms are non-terminating.

4.2 Freezing

A drawback of $\mathcal{U}_1$ is that dependency pair symbols are excluded from the uncurrying process. Typically, all pairs in $\mathcal{P}$ have the same root symbol $\circ^\sharp$. The next example shows that uncurrying root symbols of $\mathcal{P}$ can be beneficial.

Example 35. After processing the ATRS consisting of the rule $\mathsf{a}\ x\ \mathsf{a} \to \mathsf{a}\ (\mathsf{a}\ \mathsf{a})\ x$ with the recursive SCC algorithm and $\mathcal{U}_1$, the rule $\mathsf{a}_1(x) \circ^\sharp \mathsf{a} \to \mathsf{a}_1(\mathsf{a}_1(\mathsf{a})) \circ^\sharp x$ must be oriented. This cannot be done with a linear polynomial interpretation. If we transform the rule into $\mathsf{a}^\sharp_2(x, \mathsf{a}) \to \mathsf{a}^\sharp_2(\mathsf{a}_1(\mathsf{a}), x)$ this becomes trivial.

To this end we introduce a simple variant of *freezing* [19].

Definition 36. *A* simple freeze *is a partial mapping $\divideontimes$ that assigns to a function symbol of arity $n > 0$ an argument position $i \in \{1, \dots, n\}$. Every simple freeze $\divideontimes$ induces the following partial mapping on non-variable terms $t = f(t_1, \dots, t_n)$, also denoted by $\divideontimes$:*

- *if $\divideontimes(f)$ is undefined or $n = 0$ then $\divideontimes(t) = t$,*
- *if $\divideontimes(f) = i$ and $t_i = g(u_1, \dots, u_m)$ then*

$$\divideontimes(t) = \divideontimes^f_g(t_1, \dots, t_{i-1}, u_1, \dots, u_m, t_{i+1}, \dots, t_n)$$

  where $\divideontimes^f_g$ is a fresh $m + n - 1$-ary function symbol,
- *if $\divideontimes(f) = i$ and t_i is a variable then $\divideontimes(t)$ is undefined.*

We denote $\{\divideontimes(l) \to \divideontimes(r) \mid l \to r \in \mathcal{R}\}$ by $\divideontimes(\mathcal{R})$.

Now uncurrying for dependency pair symbols is formulated with the simple freeze $\divideontimes(\circ^\sharp) = 1$, transforming $f_n(t_1, \dots, t_n) \circ^\sharp t_{n+1}$ to $\divideontimes^{\circ^\sharp}_{f_n}(t_1, \dots, t_n, t_{n+1})$. Writing $f^\sharp_{n+1}$ for $\divideontimes^{\circ^\sharp}_{f_n}$, we obtain the uncurried term $f^\sharp_{n+1}(t_1, \dots, t_n, t_{n+1})$. In Example 35 we have $\divideontimes(\{\mathsf{a}_1(x) \circ^\sharp \mathsf{a} \to \mathsf{a}_1(\mathsf{a}_1(\mathsf{a})) \circ^\sharp x\}) = \{\mathsf{a}^\sharp_2(x, \mathsf{a}) \to \mathsf{a}^\sharp_2(\mathsf{a}_1(\mathsf{a}), x)\}$.

Definition 37. *A term t is* strongly root stable *with respect to a TRS $\mathcal{R}$ if $t\sigma \to_{\mathcal{R}}^* \cdot \xrightarrow{\epsilon}_{\mathcal{R}} u$ does not hold for any substitution σ and term u. Let $\circledast$ be a simple freeze. A DP problem $(\mathcal{P}, \mathcal{R})$ is* $\circledast$-stable *if $\circledast(\mathcal{P})$ is well-defined and t_i is strongly root stable for $\mathcal{R}$ whenever $s \to f(t_1, \dots, t_n) \in \mathcal{P}$ and $\circledast(f) = i$.*

Definition 38. *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and $\circledast$ a simple freeze. The DP processor $\circledast$ is defined as*

$$(\mathcal{P}, \mathcal{R}) \mapsto \begin{cases} \{(\circledast(\mathcal{P}), \mathcal{R})\} & \text{if } (\mathcal{P}, \mathcal{R}) \text{ is } \circledast\text{-stable} \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

Furthermore, the DP processor $\mathcal{U}_2$ is defined as the composition $\circledast \circ \mathcal{U}_1$, where $\circledast(\circ^\sharp) = 1$.

Theorem 39. *The DP processor $\circledast$ is sound and complete.*

Proof. We show that every minimal rewrite sequence $s_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* s_2 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* \cdots$ can be transformed into the minimal sequence $\circledast(s_1) \xrightarrow{\epsilon}_{\circledast(\mathcal{P})} \circledast(t_1) \to_{\mathcal{R}}^* \circledast(s_2) \xrightarrow{\epsilon}_{\circledast(\mathcal{P})} \circledast(t_2) \to_{\mathcal{R}}^* \cdots$ and vice versa. This follows from the following three observations.

- $s_i \xrightarrow{\epsilon}_{\mathcal{P}} t_i$ if and only if $\circledast(s_i) \xrightarrow{\epsilon}_{\circledast(\mathcal{P})} \circledast(t_i)$

  We have $s_i \xrightarrow{\epsilon}_{\mathcal{P}} t_i$ if and only if $s_i = l\sigma$ and $t_i = r\sigma$ with $l \to r \in \mathcal{P}$. Since $\circledast(\mathcal{P})$ is well-defined, the latter is equivalent to $\circledast(s_i) = \circledast(l\sigma) = \circledast(l)\sigma \xrightarrow{\epsilon}_{\circledast(\mathcal{P})} \circledast(r)\sigma = \circledast(r\sigma) = \circledast(t_i)$.
- $t_i \to_{\mathcal{R}}^* s_{i+1}$ if and only if $\circledast(t_i) \to_{\mathcal{R}}^* \circledast(s_{i+1})$

  Since t_i and s_{i+1} have the same root symbol we can write $t_i = f(u_1, \dots, u_n)$ and $s_{i+1} = f(u'_1, \dots, u'_n)$. If $\circledast(f)$ is undefined or $n = 0$ then $\circledast(s_i) = s_i \to_{\mathcal{R}}^* t_i = \circledast(t_i)$. Suppose $\circledast(f) = k$. Since t_i is an instance of a right-hand side of a pair in $\mathcal{P}$ and $\circledast(\mathcal{P})$ is well-defined, u_k cannot be a variable. Write $u_k = g(v_1, \dots, v_m)$. According to $\circledast$-stability, u_k is root stable and thus $u'_k = g(v'_1, \dots, v'_m)$. Hence

  $$t_i = f(u_1, \dots, u_{k-1}, g(v_1, \dots, v_m), u_{k+1}, \dots, u_n)$$
  $$s_{i+1} = f(u'_1, \dots, u'_{k-1}, g(v'_1, \dots, v'_m), u'_{k+1}, \dots, u'_n)$$

  and

  $$\circledast(t_i) = \circledast^f_g(u_1, \dots, u_{k-1}, v_1, \dots, v_m, u_{k+1}, \dots, u_n)$$
  $$\circledast(s_{i+1}) = \circledast^f_g(u'_1, \dots, u'_{k-1}, v'_1, \dots, v'_m, u'_{k+1}, \dots, u'_n)$$

  Consequently, $t_i \to_{\mathcal{R}}^* s_{i+1}$ if and only if $u_j \to_{\mathcal{R}}^* u'_j$ for $1 \leqslant j \leqslant n$ with $j \neq k$ and $v_j \to_{\mathcal{R}}^* v'_j$ for $1 \leqslant j \leqslant m$ if and only if $\circledast(t_i) \to_{\mathcal{R}}^* \circledast(s_{i+1})$.
- t_i terminates wrt $\mathcal{R}$ if and only if $\circledast(t_i)$ terminates wrt $\mathcal{R}$

  This follows immediately from the observation above that all reductions in t_i take place in the arguments u_j or v_j. □

Corollary 40. *The DP processor $\mathcal{U}_2$ is sound and complete.* □

The next example shows that ❄-stability is essential for soundness.

Example 41. Consider the non-terminating ATRS $\mathcal{R}$ consisting of the two rules $\mathsf{f}\,\mathsf{a} \to \mathsf{g}\,\mathsf{a}$ and $\mathsf{g} \to \mathsf{f}$, which induces the infinite DP problem $(\mathcal{P}, \mathcal{R})$ with $\mathcal{P}$ consisting of the rules $\mathsf{f}^\sharp\,\mathsf{a} \to \mathsf{g}^\sharp\,\mathsf{a}$ and $\mathsf{f}^\sharp\,\mathsf{a} \to \mathsf{g}^\sharp$. Since $\mathcal{P}{\downarrow_\mathcal{U}} = \mathcal{P}$ and $\mathcal{U}_1$ is sound, the DP problem $(\mathcal{P}, \mathcal{U}^+(\mathcal{R}_\eta))$ is also infinite. The set $❄(\mathcal{P}{\downarrow_\mathcal{U}})$ consists of $\mathsf{f}_1^\sharp(\mathsf{a}) \to \mathsf{g}_1^\sharp(\mathsf{a})$ and $\mathsf{f}_1^\sharp(\mathsf{a}) \to \mathsf{g}^\sharp$. Clearly, the DP problem $(❄(\mathcal{P}), \mathcal{U}^+(\mathcal{R}_\eta))$ is finite. Note that $(\mathcal{P}, \mathcal{U}^+(\mathcal{R}_\eta))$ is not ❄-stable as $\mathsf{g} \xrightarrow{\epsilon}_{\mathcal{U}^+(\mathcal{R}_\eta)} \mathsf{f}$.

Since ❄-stability is undecidable in general, for automation we need to approximate strong root stability. We present a simple criterion which is based on the term approximation TCAP from [10], where it was used to give a better approximation of dependency graphs.

Definition 42 ([10]). *Let $\mathcal{R}$ be a TRS and t a term. The term $\mathsf{TCAP}_\mathcal{R}(t)$ is inductively defined as follows. If t is a variable, $\mathsf{TCAP}_\mathcal{R}(t)$ is a fresh variable. If $t = f(t_1, \ldots, t_n)$ then we let $u = f(\mathsf{TCAP}_\mathcal{R}(t_1), \ldots, \mathsf{TCAP}_\mathcal{R}(t_n))$ and define $\mathsf{TCAP}_\mathcal{R}(t)$ to be u if u does not unify with the left-hand side of a rule in $\mathcal{R}$, and a fresh variable otherwise.*

Lemma 43. *A term t is strongly root stable for a TRS $\mathcal{R}$ if $\mathsf{TCAP}_\mathcal{R}(t) \notin \mathcal{V}$.*

Proof. The only possibility for $\mathsf{TCAP}_\mathcal{R}(t) \notin \mathcal{V}$ is when $t = f(t_1, \ldots, t_n)$ and $u = f(\mathsf{TCAP}_\mathcal{R}(t_1), \ldots, \mathsf{TCAP}_\mathcal{R}(t_n))$ does not unify with a left-hand side of a rule in $\mathcal{R}$. Assume to the contrary that t is not strongly root stable. Then there are a substitution σ and a left-hand side l of a rule in $\mathcal{R}$ such that $t\sigma \xrightarrow{>\epsilon}{}^*_\mathcal{R} l\tau$. Write $l = f(l_1, \ldots, l_n)$. We have $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$ with $t_i\sigma \to^*_\mathcal{R} l_i\tau$ for $1 \leqslant i \leqslant n$. Hence $\mathsf{TCAP}_\mathcal{R}(t_i)\delta_i = l_i\tau$ for some substitution δ_i ([10, proof of Theorem 13]). Since the terms $\mathsf{TCAP}_\mathcal{R}(t_1), \ldots, \mathsf{TCAP}_\mathcal{R}(t_n)$ are linear and do not share variables, it follows that u unifies with l, contradicting the assumption. □

Example 44. Consider the DP problem $(\mathcal{P}{\downarrow_\mathcal{U}}, \mathcal{U}^+(\mathcal{R}_\eta))$ of Example 35 with $\mathcal{P}{\downarrow_\mathcal{U}} = \{\mathsf{a}_1(x) \circ^\sharp \mathsf{a} \to \mathsf{a}_1(\mathsf{a}_1(\mathsf{a})) \circ^\sharp x\}$ and $\mathcal{U}^+(\mathcal{R}_\eta) = \{\mathsf{a} \circ x \to \mathsf{a}_1(x), \mathsf{a}_1(x) \circ y \to \mathsf{a}_2(x, y), \mathsf{a}_2(x, \mathsf{a}) \to \mathsf{a}_2(\mathsf{a}_1(\mathsf{a}), x)\}$. Since $\mathsf{TCAP}_{\mathcal{U}^+(\mathcal{R}_\eta)}(\mathsf{a}_1(\mathsf{a}_1(\mathsf{a}))) = \mathsf{a}_1(\mathsf{a}_1(\mathsf{a}))$ is not a variable, $\mathsf{a}_1(\mathsf{a}_1(\mathsf{a}))$ is strongly root stable. Hence $(\mathcal{P}{\downarrow_\mathcal{U}}, \mathcal{U}^+(\mathcal{R}_\eta))$ is ❄-stable.

5 Experiments

The results of this paper are implemented in the termination prover T$_\text{T}$T2.[4] For experimentation the 195 ATRSs from the termination problem data base (TPDB)[5] have been employed. All tests have been performed on a single core of a server equipped with eight dual-core AMD Opteron® processors 885 running at a clock rate of 2.6GHz and 64GB of main memory. Comprehensive details of the experiments[6] give evidence that the proposed transformations can be

[4] `http://colo6-c703.uibk.ac.at/ttt2/`

[5] `http://www.lri.fr/~marche/tpdb/`

[6] `http://colo6-c703.uibk.ac.at/ttt2/uncurry/`

Table 1. Experimental results.

	direct			as processor			
	6	16	16+30	none	$\mathcal{A}$	$\mathcal{U}_1$	$\mathcal{U}_2$
subterm criterion	1	47	48	41	–	41	58
matrix (dimension 1)	4	90	101	66	71	95	101
matrix (dimension 2)	7	108	131	108	115	136	138

implemented very efficiently, e.g., for the most advanced strategy all 195 systems are analyzed within about 15 seconds. We considered two popular termination methods, namely the subterm criterion [13] and matrix interpretations [8] of dimensions one and two and with coefficients ranging over $\{0, 1\}$. Both methods are integrated within the dependency pair framework using dependency graph reasoning and usable rules as proposed in [10–12].

Table 1 differentiates between applying the transformations as a preprocessing step (direct) or within the dependency pair framework (as processor). The direct method of Corollary 6 (Theorem 16, Theorems 16 and 30) applies to 10 (141, 170) systems. If used directly, the numbers in the table refer to the systems that could be proved terminating in case of a successful transformation. Mirroring (when termination of the original system could not be proved) does increase applicability of our (direct) transformation significantly. The right part of Table 1 states the number of successful termination proofs for the processors $\mathcal{A}$ (transformation $\mathcal{A}$ from [10, 17]), $\mathcal{U}_1$ (Definition 32), and $\mathcal{U}_2$ (Definition 38) which shows that the results of this paper really increase termination proving power for ATRSs. Since transformation $\mathcal{A}$ does not preserve minimality (Example 34) one cannot use it together with the subterm criterion. (In [17] it is shown that minimality is preserved when the transformation $\mathcal{A}$ is fused with the reduction pair and usable rules processors.) It is a trivial exercise to extend mirroring to DP problems. Our experiments revealed that (a) mirroring works better for the direct approach (hence we did not incorporate it into the right block of the table) and (b) the uncurrying processors should be applied before other termination processors.

Although Theorem 16 and the processor $\mathcal{U}_2$ are incomparable in power we recommend the usage of the processor. One reason is the increased strength and another one the modularity which allows to prevent pitfalls like Example 22. Last but not least, the processors $\mathcal{U}_1$ and $\mathcal{U}_2$ are not only sound but also complete which makes them suitable for non-termination analysis. Unfortunately T$_{\mathsf{T}}$T$_2$ does only support trivial methods for detecting non-termination of TRSs but we anticipate that these processors ease the job of proving non-termination of ATRSs considerably.

6 Related Work

The $\mathcal{A}$ transformation of Giesl *et al.* [10] works only on *proper* applicative DP problems, which are DP problems with the property that all occurrences of each

constant have the same number of arguments. No uncurrying rules are added to the processed DP problems. This destroys minimality (Example 34), which seriously hampers the applicability of the $\mathcal{A}$ transformation. Thiemann [17, Sections 6.2 and 6.3] addresses the loss of minimality by incorporating reduction pairs, usable rules, and argument filterings into the $\mathcal{A}$ transformation. (These refinements were considered in the column labeled $\mathcal{A}$ in Table 1.) In [17] it is further observed that the $\mathcal{A}$ transformation works better for innermost termination than for termination. A natural question for future work is how $\mathcal{U}_1$ and $\mathcal{U}_2$ behave for innermost termination.

Aoto and Yamada [1, 2] present transformation techniques for proving termination of simply typed ATRSs. After performing η-saturation, head variables are eliminated by instantiating them with 'template' terms of the appropriate type. In a final step, the resulting ATRS is translated into functional form.

Example 45. Consider again the ATRS $\mathcal{R}$ of Example 7. Suppose we adopt the following type declarations: $\mathsf{0} : \mathsf{int}$, $\mathsf{s} : \mathsf{int} \to \mathsf{int}$, $\mathsf{nil} : \mathsf{list}$, $(:) : \mathsf{int} \to \mathsf{list} \to \mathsf{list}$, $\mathsf{id} : \mathsf{int} \to \mathsf{int}$, $\mathsf{add} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$, and $\mathsf{map} : (\mathsf{int} \to \mathsf{int}) \to \mathsf{list} \to \mathsf{list}$. The head variable f in the right-hand side $: (\mathsf{id}\ x)\ (\mathsf{map}\ f\ y)$ has type $\mathsf{int} \to \mathsf{int}$. There are three template terms of this type: s, id, and $\mathsf{add}\ z$. Instantiating f by these three terms in $\mathcal{R}_\eta$ produces the ATRS $\mathcal{R}'$:

$$\begin{array}{rr}
\mathsf{id}\ x \to x & \mathsf{map}\ f\ \mathsf{nil} \to \mathsf{nil} \\
\mathsf{add}\ \mathsf{0} \to \mathsf{id} & \mathsf{map}\ \mathsf{s}\ (:\ x\ y) \to :\ (\mathsf{s}\ x)\ (\mathsf{map}\ \mathsf{s}\ y) \\
\mathsf{add}\ \mathsf{0}\ y \to \mathsf{id}\ y & \mathsf{map}\ \mathsf{id}\ (:\ x\ y) \to :\ (\mathsf{id}\ x)\ (\mathsf{map}\ \mathsf{id}\ y) \\
\mathsf{add}\ (\mathsf{s}\ x)\ y \to \mathsf{s}\ (\mathsf{add}\ x\ y) & \mathsf{map}\ (\mathsf{add}\ z)\ (:\ x\ y) \to :\ (\mathsf{add}\ z\ x)\ (\mathsf{map}\ (\mathsf{add}\ z)\ y)
\end{array}$$

The TRS $\mathcal{R}'{\downarrow_{\mathcal{U}}}$ is terminating because its rules are oriented from left to right by the lexicographic path order. According to the main result of [2], the *simply typed* ATRS $\mathcal{R}$ is terminating, too.

The advantage of the simply typed approach is that the uncurrying rules are not necessary because the application symbol has been eliminated from $\mathcal{R}'{\downarrow_{\mathcal{U}}}$. This typically results in simpler termination proofs. It is worthwhile to investigate whether a version of head variable instantiation can be developed for the untyped case. We would like to stress that with the simply typed approach one obtains termination only for those terms which are simply typed. Our approach, when it works, provides termination for all terms, irrespective of *any* typing discipline. In [3] the dependency pair method is adapted to deal with simply typed ATRSs. Again, head variable instantiation plays a key role.

Applicative term rewriting is not the only model for capturing higher-order aspects. The S-expression rewrite systems of Toyama [18] have a richer structure than applicative systems, which makes proving termination often easier. Recent methods (e.g. [6, 14]) use types to exploit strong computability, leading to powerful termination methods which are directly applicable to higher-order systems. In [16] strong computability is used to analyse the termination of simply typed ATRSs with the dependency pair method.

References

1. Aoto, T., Yamada, T.: Termination of simply typed term rewriting by translation and labelling. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 380–394. Springer (2003)
2. Aoto, T., Yamada, T.: Termination of simply-typed applicative term rewriting systems. In: HOR 2004. Technical Report AIB-2004-03, RWTH Aachen. pp. 61–65 (2004)
3. Aoto, T., Yamada, T.: Dependency pairs for simply typed term rewriting. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 120–134. Springer (2005)
4. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236(1-2), 133–178 (2000)
5. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
6. Blanqui, F., Jouannaud, J.P., Rubio, A.: HORPO with computability closure: A reconstruction. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 138–150. Springer (2007)
7. Dershowitz, N.: 33 Examples of termination. In: French Spring School of Theoretical Computer Science. LNCS, vol. 909, pp. 16–26. Springer (1995)
8. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of rewrite systems. Journal of Automated Reasoning 40(2-3), 195–220 (2008)
9. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 301–331. Springer (2004)
10. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Gramlich, B. (ed.) FroCoS 2005. LNCS (LNAI), vol. 3717, pp. 216–231. Springer (2005)
11. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
12. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. Information and Computation 199(1-2), 172–199 (2005)
13. Hirokawa, N., Middeldorp, A.: Tyrolean termination tool: Techniques and features. Information and Computation 205(4), 474–511 (2007)
14. Jouannaud, J.P., Rubio, A.: Polymorphic higher-order recursive path orderings. Journal of the ACM 54(1) (2007)
15. Kennaway, R., Klop, J.W., Sleep, M.R., de Vries, F.J.: Comparing curried and uncurried rewriting. Journal of Symbolic Computation 21(1), 15–39 (1996)
16. Kusakari, K., Sakai, M.: Enhancing dependency pair method using strong computability in simply-typed term rewriting. Applicable Algebra in Engineering, Communication and Computing 18(5), 407–431 (2007)
17. Thiemann, R.: The DP Framework for Proving Termination of Term Rewriting. PhD thesis, RWTH Aachen (2007). Available as technical report AIB-2007-17
18. Toyama, Y.: Termination of S-expression rewriting systems: Lexicographic path ordering for higher-order terms. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 40–54. Springer (2004)
19. Xi, H.: Towards automated termination proofs through "freezing". In: Nipkow, T. (ed.) RTA 1998. LNCS, vol. 1379, pp. 271–285. Springer (1998)
20. Zantema, H.: Termination. In: Terese (ed.) Term Rewriting Systems 2003. vol. 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press 181–259 (2003)