Harnessing First Order Termination Provers Using Higher Order Dependency Pairs *

Carsten Fuhs[1] and Cynthia Kop[2]

[1] RWTH Aachen University, LuFG Informatik 2, 52056 Aachen, Germany
fuhs@informatik.rwth-aachen.de
[2] Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
kop@few.vu.nl

Abstract. Many functional programs and higher order term rewrite systems contain, besides higher order rules, also a significant first order part. We discuss how an automatic termination prover can split a rewrite system into a first order and a higher order part. The results are applicable to all common styles of higher order rewriting with simple types, although some dependency pair approach is needed to use them.

Key words: higher order rewriting, termination, dependency pairs, modularity

1 Introduction

Termination of term rewrite systems has been an area of active research for several decades. In recent years the field of *automatically* proving termination has flourished, and several strong provers have been developed to participate in the annual *International Termination Competition*; there is a wide range of automated methods available for (and used in) these tools: the dependency pair framework [3,17,14], polynomial and matrix orderings [9,8], recursive path orderings [7], semantic labelling [36], and many more techniques.

In higher order termination, however, fewer results have been obtained so far. Recursive and monotonic semantic path orderings have been generalised to a higher order setting [21,5,6], but other automatable term orderings have not (yet?) been extended to this setting.

In the last three years there has been a lot of work on higher order dependency pair approaches and several strong results have been obtained, such as the ability to use argument filterings and to restrict to non-collapsing dependency pairs [28,32,26]. But after simplifying the ordering requirements on terms with this approach, we still have little but a higher order RPO to compare them.

However, in many (realistic) term rewrite systems, only a small number of the rules use functional variables or λ-abstraction. The majority of the rules usually consists entirely of first order symbols. It would therefore be convenient to analyse termination of at least those rules directly with first order techniques.

* This research is supported by the G.I.F. grant 966-116.6 and by the Netherlands Organisation for Scientific Research (NWO-EW) under grant 612.000.629 (HOT).

The progress in dependency pair approaches opens possibilities: we can split the dependency pairs into those which could be considered first order and the higher order remainder, and analyse termination of these parts separately. While the first order dependency pairs classically still need to be regarded together with *all* rules from the underlying system (some of which are higher order), it may be possible to remove these higher order rules, or replace them by first order ones.

In this paper we discuss how to reduce the termination of an orthogonal or finitely branching higher order term rewrite system to the termination of a first order (sub-)system and a (smaller) dependency pair problem. The technique is comparable to a *usable rules* [14,18] approach, but focusses on first order rules. We aim to be as general as possible by not choosing a definition of dependency pairs and assuming as little as possible about the formalism. Consequently, the results presented in this paper can be used for all the common styles of higher order rewriting, and with both dynamic and static dependency pairs [31,32].

We have implemented the method in the higher order termination prover WANDA [24], using the tool AProVE [12] to analyse termination of the first order part of a higher order rewrite system. As far as we know, this is the first time a tool for termination of higher order rewriting is combined with a first order termination tool. Experimental results (see Section 5) demonstrate that this combination significantly improves the strength of the prover.

Higher Order Rewriting "Higher order rewriting", rewriting with some form of functional variables, comes in several forms: typed and untyped, with and without λ-abstraction. To understand the relevance of this work, it should be noted that these styles are *fundamentally different*.

Without giving complete definitions, consider the system with two function symbols: $\mathtt{app} : \mathtt{o} \Rightarrow \mathtt{o} \Rightarrow \mathtt{o}$ (which takes two arguments of type $\mathtt{o}$ and returns an object of type $\mathtt{o}$) and $\mathtt{lam} : (\mathtt{o} \Rightarrow \mathtt{o}) \Rightarrow \mathtt{o}$ (which takes a functional argument of type $\mathtt{o} \Rightarrow \mathtt{o}$ and returns an object of type $\mathtt{o}$), and a single rule $\mathtt{app}\ (\mathtt{lam}\ F)\ x \to F\ x$. In simply-typed applicative systems, terms are built from typed constants and a binary application operator. The given system terminates, because the size of a term decreases with every reduction step. In higher order systems with λ-abstraction, β-reduction may increase the size of a term. Here, defining $\omega = \mathtt{lam}\ (\lambda x.\mathtt{app}\ x\ x)$, there is a loop $\mathtt{app}\ \omega\ \omega \to (\lambda x.\mathtt{app}\ x\ x)\ \omega \to_\beta \mathtt{app}\ \omega\ \omega$.

Since terms in a formalism with λ-abstraction may include *anonymous functions* (such as $\lambda x.\mathtt{app}\ x\ x$) whose presence may give rise to non-termination, these formalisms cannot easily be simulated with applicative systems. In addition, in an applicative system it is impossible to express rules like this derivation rule:

$$\mathtt{D}\ (\lambda x.\mathtt{sin}(Z(x))) \to \lambda x.(\mathtt{D}\ (\lambda y.Z(y))\ x) \times \mathtt{cos}(Z(x))$$

As we will see below, applicative systems can be transformed into standard first order TRSs via some kind of uncurrying; thus, this work is primarily relevant for formalisms which do have λ-abstraction.

Related work Other work on using first order techniques in higher order rewriting is often focussed on applicative systems, where only terms without binders

like λ are considered. By [22], currying untyped TRSs does not affect termination and certain other properties. In [19] an uncurrying transformation from untyped applicative systems to standard first order TRSs is used, which preserves and reflects termination. The result of [13] is similar, but works on a more restricted set of problems, and presents termination techniques that operate directly on applicative systems. In [1,2] simply typed applicative systems are discussed; here, leading variables are eliminated by instantiating them with "template" terms of the right type. Having this, they can be transformed into many-sorted TRSs.

However, these results do not apply to systems with binders and β-reduction, nor does it seem likely that they can easily be extended to such a formalism.

In [11] termination is studied for Haskell programs, a (higher order) polymorphic functional language, via a translation to first order term rewriting. The approach relies on symbolic partial evaluation of a start term, which is made feasible essentially by Haskell's deterministic evaluation strategy. In a general term rewrite setting, however, there is no fixed strategy, which renders the construction from [11] infeasible. Moreover, we are interested in termination of *all* terms, while the construction in [11] considers only terms of a given form.

In this paper, we consider typed higher order rewriting which may have binders; we show how *part* of a higher order termination problem can be dealt with as a first order problem (leaving the truly higher order part to higher order techniques). An early work in this context, [33], considers termination of the combination of typed λ-calculus with first order TRSs. A first modularity result with higher order rules is given in [20], where the authors show that a terminating first order system combined with a number of higher order rules is terminating if the higher order rules satisfy certain restrictions, and the first order part is non-duplicating. The restriction on the first order rules is not present in the current work, nor do we pose limitations on the higher order part.

Another relevant work is [32], which studies static dependency pairs for a subset of the HRS formalism and defines a usable rules approach. The usable rules for a set of first order dependency pairs are all first order. However, this approach does not give an equivalence result like our Theorem 9. In addition, we do not choose a definition of dependency pairs or a formalism.

2 Preliminaries

As stated in the introduction, we aim for generality. Rather than focussing on a formalism, we will discuss the basic definitions used in common styles of higher order rewriting with simple types. Consequently, these definitions are incomplete, but our results can be used for instance with AFSs [21], HRSs [29] and CRSs [23].

Types Given a set of *base types* $\mathcal{B}$, *types* are built according to the grammar:

$$\mathcal{T} = \mathcal{B} \mid \mathcal{T} \Rightarrow \mathcal{T}$$

The $\Rightarrow$ associates to the right; a type of the form $\sigma \Rightarrow \tau$ is called *functional*. Every type can be written in the form $\sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \iota$ with $n \geq 0$ and $\iota \in \mathcal{B}$.

Terms A *term* is an expression s over a set $\mathcal{F}$ of typed function symbols and a set $\mathcal{V}$ of typed variables, for which we can derive $s : \sigma$ for some type σ using the following recursive rules (which also define the set $FV(s)$ of *free variables* of s):

(`var`) $x : \tau$	if $x : \tau \in \mathcal{V}$	$FV(x) = \{x\}$
(`fun`) $f : \tau$	if $f : \tau \in \mathcal{F}$	$FV(f) = \emptyset$
(`abs`) $\lambda x.s : \sigma \Rightarrow \tau$	if $x : \sigma \in \mathcal{V}$ and $s : \tau$	$FV(\lambda x.s) = FV(s) \setminus \{x\}$
(`app`) $s \cdot t : \tau$	if $s : \sigma \Rightarrow \tau$ and $t : \sigma$	$FV(s \cdot t) = FV(s) \cup FV(t)$

The $\cdot$ operator associates to the left and is usually omitted; a term $s\ t\ r$ is short for $(s \cdot t) \cdot r$. We consider term equality modulo renaming of bound variables (α-conversion), so $\lambda x.s = \lambda y.s[x := y]$ if y does not occur in s.
Note that this is a general definition of terms; there are several higher order formalisms which do not allow, for instance, a term $(\lambda x.s) \cdot t$, or $f\ s : \sigma \Rightarrow \tau$.

Define $head(s)$, the *head symbol* of s, as the first part of an application: $head(s) = s$ if s is a variable, constant or abstraction, and $head(u\ v) = head(u)$.

Meta-terms Some formalisms, like Klop's CRSs [23] or Blanqui's definition of IDTSs [4], use special *meta-terms* to construct rules. A meta-term is a typed expression generated with clauses (`var`), (`fun`), (`abs`), (`app`) and additionally:

(`meta`) $Z(s_1, \ldots, s_n) : \tau$ if $s_1 : \sigma_1, \ldots, s_n : \sigma_n$ and $Z : [\sigma_1, \ldots, \sigma_n] \Rightarrow \tau \in \mathcal{MV}$

where $\mathcal{MV}$ is a fresh set of *meta-variables*, each equipped with a vector of input types ($\sigma_1, \ldots, \sigma_n$, where n may be 0) and an output type (τ); the s_i are meta-terms. Evidently, all terms are also meta-terms. Meta-terms can be used to match a term which may contain some bound variables, for instance in a rule like:

$$\texttt{map}\ (\lambda x.F(x))\ (\texttt{cons}\ h\ t) \to \texttt{cons}\ F(h)\ (\texttt{map}\ (\lambda x.F(x))\ t)$$

Note that not all higher order formalisms use meta-variables; for instance Jouannaud's and Okada's AFSs [21] use variables for matching instead, at the price of some (easy) expressivity. Nipkow's HRSs [29] also use variables, but here terms are equivalence classes modulo β/η, which is not always a practical modelling. In the examples in this paper, we will use meta-variables to define rules.

Contexts and subterms A *context* is a term containing one occurrence of a special symbol $\Box_\sigma : \sigma$. Contexts are usually denoted as $C[]$, and $C[]$ with $\Box_\sigma$ replaced by some t of type σ is denoted $C[t]$. If $s = C[t]$, then t is a *subterm* of s, denoted $s \trianglerighteq t$. If C is non-empty, then t is a strict subterm of s, denoted $s \triangleright t$.

Substitutions A *substitution* is a type-preserving function mapping variables and meta-variables to terms; substitutions on a finite domain are usually denoted $[x_1 := s_1, \ldots, x_n := s_n]$. A substitution γ may be applied on (meta-)terms by placewise replacing variables and meta-variables by their image in γ; depending on the rewriting formalism the result might be β-normalised. Formally:

$$\begin{array}{rll}
x\gamma = x & & \text{if } x \in \mathcal{V},\ x \notin \mathrm{dom}(\gamma) \\
x\gamma = \gamma(x) & & \text{if } x \in \mathcal{V},\ x \in \mathrm{dom}(\gamma) \\
(f\ s_1 \cdots s_n)\gamma = f\ (s_1\gamma) \cdots (s_n\gamma) & & (f \in \mathcal{F},\ n \geq 0) \\
((\lambda x.q)\ s_1 \cdots s_n)\gamma = (\lambda x.q\gamma)\ (s_1\gamma) \cdots (s_n\gamma) & & (n \geq 0, **) \\
Z(s_1, \ldots, s_n)\gamma = q[x_1 := s_1\gamma, \ldots, x_n := s_n\gamma] & & \text{if } \gamma(Z) = \lambda x_1 \ldots x_n.q
\end{array}$$

(**): When substituting an abstraction $\lambda x.q$, the variable x may not occur in either domain or range of γ. Using α-conversion, this is always defined. We assume that $\gamma(Z)$ has the form $\lambda x_1 \dots x_n.q$ whenever $Z : [\sigma_1, \dots, \sigma_n] \Rightarrow \tau \in \mathcal{MV}$.

This definition is incomplete. The cases where formalisms differ, in particular $(x\ s_1 \cdots s_n)\gamma$ with $n \geq 1$, are omitted. However, the given cases are the only ones we will need.

Rules A *rewrite rule* is a pair $l \to r$ of (meta-)terms such that l and r have the same type and $head(l)$ is a function symbol or abstraction. Let $\mathcal{R}$ be a (possibly infinite) set of rewrite rules. The *rewrite relation* $\to_{\mathcal{R}}$ generated by $\mathcal{R}$ is given by: $s \to_{\mathcal{R}} t$ if $s = C[l\gamma],\ t = C[r\gamma]$ for some $l \to r \in \mathcal{R}$, substitution γ and context C; write $s \to_{\mathcal{R},top} t$ if C is empty and $s \to_{\mathcal{R},in} t$ otherwise.
Depending on the formalism, this rewrite relation may only be defined on terms of a given form, for instance β/η-normal form; however, base-type variables and terms $f\ s_1 \cdots s_n$ of base type always have such a form if the s_i do.

A set of rules $\mathcal{R}$ is *finitely branching* if, for any term s, there are only finitely many different t with $s \to_{\mathcal{R}} t$. This is commonly the case when $\mathcal{R}$ is finite. A set of rules is *terminating* if there is no infinite reduction $s_0 \to_{\mathcal{R}} s_1 \to_{\mathcal{R}} \dots$

Example 1. An example system we will use throughout this paper is the system $\mathcal{R}_{\texttt{list}}$, a module for list manipulation, with the following function symbols:

$$\begin{array}{lll} \texttt{nil} : \texttt{list} & \texttt{append} : \texttt{list} \Rightarrow \texttt{list} \Rightarrow \texttt{list} & \texttt{reverse} : \texttt{list} \Rightarrow \texttt{list} \\ \texttt{cons} : \texttt{nat} \Rightarrow \texttt{list} & \texttt{map} : (\texttt{nat} \Rightarrow \texttt{nat}) \Rightarrow \texttt{list} \Rightarrow \texttt{list} & \texttt{shuffle} : \texttt{list} \Rightarrow \texttt{list} \\ & & \texttt{mirror} : \texttt{list} \Rightarrow \texttt{list} \end{array}$$

And moreover ten rules:

$$\begin{aligned} \texttt{append}\ \texttt{nil}\ l &\to l \\ \texttt{append}\ (\texttt{cons}\ h\ t)\ l &\to \texttt{cons}\ h\ (\texttt{append}\ t\ l) \\ \texttt{reverse}\ \texttt{nil} &\to \texttt{nil} \\ \texttt{reverse}\ (\texttt{cons}\ h\ t) &\to \texttt{append}\ (\texttt{reverse}\ t)\ (\texttt{cons}\ h\ \texttt{nil}) \\ \texttt{shuffle}\ \texttt{nil} &\to \texttt{nil} \\ \texttt{shuffle}\ (\texttt{cons}\ h\ t) &\to \texttt{cons}\ h\ (\texttt{shuffle}\ (\texttt{reverse}\ t)) \\ \texttt{mirror}\ \texttt{nil} &\to \texttt{nil} \\ \texttt{mirror}\ (\texttt{cons}\ h\ t) &\to \texttt{append}\ (\texttt{cons}\ h\ (\texttt{mirror}\ t))\ (\texttt{cons}\ h\ \texttt{nil}) \\ \texttt{map}\ (\lambda x.F(x))\ \texttt{nil} &\to \texttt{nil} \\ \texttt{map}\ (\lambda x.F(x))\ (\texttt{cons}\ h\ t) &\to \texttt{cons}\ F(h)\ (\texttt{map}\ (\lambda x.F(x))\ t) \end{aligned}$$

There is only one really higher order function symbol (`map`), as its rules use an abstraction. Intuitively, the first eight rules are first order. Note that `mirror` has a duplicating rule, so the result from [20] cannot be used to prove termination.

Remarks Despite our aim for generality, we do make a number of assumptions:

- the requirement that $head(l) \notin \mathcal{V}$ for left-hand sides l of a rule is not present in Yamada's STTRSs [35] or (certain variations of) Jouannaud's AFSs [21];
- we use applicative rather than functional notation ($f\ s_1 \cdots s_n$ rather than $f(s_1, \dots, s_n)$), where the latter is used in AFSs and Blanqui's IDTSs [4];

– unlike in AFSs, we do not assume the presence of a β-rule;
– unlike in CRSs or ERSs, typing is enforced;
– we assume monomorphic, simple types, while more advanced type classes are regularly used in several of the formalisms.

Only the first of these is essential: all the proofs in this paper pass almost unmodified even if we use functional notation and admit polymorphic types or include a β-rule. We chose this definition because, through simple transformations, we can usually obtain a system as described above without affecting termination: for the removal of head-variables and currying see for instance [25], to ignore typing embed abstractions into some new symbol $T : (\mathtt{term} \Rightarrow \mathtt{term}) \Rightarrow \mathtt{term}$, to add β-reduction create, for every two types σ, τ, a rule $(\lambda x.Z(x))\ y \to Z(y)$ with $Z : [\sigma] \Rightarrow \tau \in \mathcal{MV}$, and for dealing with (ML-style) polymorphism, instantiate all type variables in all closed ways and consider types of the form $\mathtt{list}(\mathtt{nat})$ as base types. These last two transformations lead to an infinite system, but only in so far as infinity was already implicit in the formalism. If the original system was finitely branching or orthogonal, the same holds for the result.

Variables or Meta-variables Due to our aim of giving formalism-independent definitions, matching may be done either with variables or meta-variables. To ease definitions, we will identify meta-variables without arguments with variables. Thus, a meta-variable $Z : [] \Rightarrow \sigma$ is considered as a variable of type σ. In the $\mathcal{R}_{\mathtt{list}}$ example, l, h and t can be seen as variables, while F is a meta-variable.

3 Splitting the system

To give some formal backing to the intuitive notion of a first order rule, we partition the signature $\mathcal{F}$ into two groups: symbols which have some higher order potential (i.e., they have a non-base type, there is a rule where they are not given all arguments allowed by their type, or they match on or rewrite to such symbols) and symbols which do not. The first group, *potentially higher order* symbols, is denoted $\mathtt{PHO}$ and the second one, consisting of *truly first order* symbols, is denoted $\mathtt{TFO}$. Using this partitioning, we obtain the first order rules by uncurrying the rules which only contain symbols in $\mathtt{TFO}$.

Splitting the symbols Let A be the set consisting of those function symbols $f : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \iota$ (with $\iota \in \mathcal{B}$) such that one of the σ_i is functional, or there is a rule $f\ s_1 \cdots s_m \to r$ where $m < n$ or the rule contains any abstraction, meta-variable with arguments or functional (meta-)variable. We define $\mathtt{PHO}$ recursively: $\mathtt{PHO}$ contains all symbols in A and, if there is a rule $f\ \boldsymbol{l} \to r$ where some l_i or r contains a symbol in $\mathtt{PHO}$, then also $f \in \mathtt{PHO}$. Let $\mathtt{TFO} = \mathcal{F} \setminus \mathtt{PHO}$. A term is *truly first order* if it consists only of function symbols in $\mathtt{TFO}$ and base-type variables.

Example 2. In $\mathcal{R}_{\mathtt{list}}$ we have $A = \{\mathtt{map}\}$. Since the symbol $\mathtt{map}$ only occurs in the $\mathtt{map}$ rules, we have $\mathtt{PHO} = \{\mathtt{map}\}$ and hence $\mathtt{TFO} = \{\mathtt{nil}, \mathtt{cons}, \mathtt{append}, \mathtt{reverse}, \mathtt{shuffle}, \mathtt{mirror}\}$. Should we add a symbol $\mathtt{up} : \mathtt{list} \Rightarrow \mathtt{list}$ and a rule $\mathtt{up}\ l \to \mathtt{map}\ (\lambda x.\mathtt{s}\ x)\ l$, then A would still be $\{\mathtt{map}\}$, but $\mathtt{PHO}$ would also include $\{\mathtt{up}\}$.

Splitting the rules We say that a *rule* $f\ l_1 \cdots l_n \to r$ is *truly first order* if $f \in \mathtt{TFO}$ and *potentially higher order* otherwise; write $\mathcal{R}_{\mathtt{TFO}}$ for the set of rules of the first kind and $\mathcal{R}_{\mathtt{PHO}}$ for the second. Note that if a rule is truly first order then both sides are truly first order terms.

Example 3. The truly first order rules $\mathcal{R}_{\mathtt{TFO}}$ of $\mathcal{R}_{\mathtt{list}}$ are those whose left-hand side has a head symbol from $\{\mathtt{append}, \mathtt{reverse}, \mathtt{shuffle}, \mathtt{mirror}\}$. The potentially higher order rules $\mathcal{R}_{\mathtt{PHO}}$ are those with `map` as the head symbol.

We can safely assume that in the truly first order rules $l \to r$, all variables in r also occur in l: if this is not the case, the system is obviously non-terminating.

Splitting infinite chains A term rewrite system (first or higher order) is non-terminating iff there exists a *(minimal) infinite chain* $s_1, t_1, s_2, t_2, \ldots$ where:

- each $s_i \to_{\mathcal{R},top} \cdot \trianglerighteq t_i$
- each $t_i \to^*_{\mathcal{R},in} s_{i+1}$
- the strict subterms of each of the t_i are terminating

This observation is at the heart of any dependency pair approach. Now note that if ever $head(t_i) \in \mathtt{TFO}$ then for all $j > i$ also $head(s_j), head(t_j) \in \mathtt{TFO}$:

Lemma 4. *If* $head(t_i) \in \mathtt{TFO}$ *then also* $head(s_{i+1}), head(t_{i+1}) \in \mathtt{TFO}$.

Proof. Write $t_i = f\ u_1 \cdots u_n$ with $f \in \mathtt{TFO}$. As all rules of the form $f\ l_1 \cdots l_m \to r$ have $n = m$, a $\to_{\mathcal{R},in}$ step on t_i reduces one of the u_j. Therefore s_{i+1} has the same head symbol and its immediate subterms are terminating (as they are reducts from the immediate subterms of t_i). Let $s_{i+1} = l\gamma$ with $l \to r \in \mathcal{R}_{\mathtt{TFO}}$ and $r\gamma \trianglerighteq t_{i+1}$. Let p be the smallest subterm of r such that $p\gamma \trianglerighteq t_{i+1}$; since r is a truly first order term p is either a variable (which, as assumed, also occurs in l), or has the form $g\ p_1 \cdots p_k$ with $g \in \mathtt{TFO}$. In the former case, $s_{i+1} \triangleright \gamma(p) \trianglerighteq t_{i+1}$ is terminating because the strict subterms of s_{i+1} are, contradiction. Thus $g\ p_1 \cdots p_k\gamma \trianglerighteq t_{i+1}$ but (by the choice of p) no $p_j\gamma \trianglerighteq t_{i+1}$; we conclude: $head(t_{i+1}) = g \in \mathtt{TFO}$. □

Corollary 5. *If there is an infinite chain, there is one using either only* `TFO` *rules, or only* `PHO` *rules, for the topmost steps. In the first case, all* t_i *have base type as well (since* $f\ s_1 \cdots s_m : \sigma \Rightarrow \tau$ *does not top-reduce if* $f \in \mathtt{TFO}$*).*

4 Simplifying the first order part

Using some dependency pair approach, we could now investigate the two possible forms of chains separately. But does this help us significantly? A priori we cannot use first order results to prove non-existence of (minimal infinite) `TFO` chains, since even in `TFO` chains a step involving higher order symbols might be done in the $\to^*_{\mathcal{R},in}$ reduction. However, note that the rules in $\mathcal{R}_{\mathtt{TFO}}$ do not match on the `PHO` symbols and that, by minimality, any higher order subterm can be assumed to be terminating. Therefore, as we will see, such subterms are mostly harmless.

Splitting with orthogonal rules Orthogonality is a common property in term rewriting with many nice consequences, including confluence. In first order orthogonal systems, termination using an innermost rewriting strategy (which is

often easier to prove) implies general termination [16].

Orthogonality is not defined for all higher order formalisms; however, where defined it implies confluence, and coincides with the first order definition on first order rules (for an overview of such results, see [34, Section 11.6.2]).

We actually use slightly less than orthogonality: we will show that, if $\mathcal{R}$ has unique normal forms and $\mathcal{R}_{\texttt{TFO}}$ is overlay, then the potentially higher order rules can be omitted when studying TFO chains. It is not in general decidable whether a system has unique normal forms, but orthogonality of $\mathcal{R}$, which implies both unique normal forms and $\mathcal{R}_{\texttt{TFO}}$ being overlay, is easy to check automatically.

Roughly, the idea is as follows: by unicity of normal forms and the overlay property, the subterms of all s_i in a minimal infinite TFO chain can be assumed to be normalised. As topmost TFO steps cannot create PHO redexes, higher order subterms anywhere in the chain are normalised, and can be replaced by variables.

We say $\mathcal{R}_{\texttt{TFO}}$ is *overlay* if for all $l \to r,\ u \to v \in \mathcal{R}_{\texttt{TFO}}$, substitutions γ, δ and non-empty contexts C: if $l = C[l']$ with $l'\gamma = u\delta$, then l' is a variable.

In Lemmas 6–8 we will assume that all terminating terms s have a unique normal form, and that $\mathcal{R}_{\texttt{TFO}}$ is overlay. Let $\nu(s)$ denote the normal form $s{\downarrow_{\mathcal{R}}}$ of s and, if $s = f\ s_1 \cdots s_n$, then $\nu'(s) = f\ \nu(s_1) \cdots \nu(s_n)$.

Lemma 6 (TFO steps cannot create PHO redexes). *If all higher order subterms of s are $\mathcal{R}$-normalised – that is, if, when $s \trianglerighteq q$ either $q = f\ q_1 \cdots q_n$ with $f \in \texttt{TFO}$, or q is in $\mathcal{R}$-normal form – then the same holds for the reducts of s.*

Proof. Suppose s has this property and $s \to_{\mathcal{R}} t$; we use induction on the size of s. Since s is not in normal form, $s = f\ s_1 \cdots s_n$ with $f \in \texttt{TFO}$. If $s \to_{\mathcal{R},top} t$, therefore, $s = l\gamma,\ t = r\gamma$ with $l \to r \in \mathcal{R}_{\texttt{TFO}}$; since r contains no higher order symbols, and higher order subterms of any $\gamma(x)$ are normalised, the property holds for $r\gamma$. Otherwise $t = f\ s_1 \cdots s_i' \cdots s_n$ with $s_i \to_{\mathcal{R}} s_i'$; by the induction hypothesis all higher order subterms of s_i' are $\mathcal{R}$-normalised, and by assumption the same holds for the other s_j. □

Lemma 7 (Normalising Chains). *If there exists a minimal infinite chain $s_1 \to_{\mathcal{R}_{\texttt{TFO}},top} \cdot \trianglerighteq t_1 \to^*_{\mathcal{R},in} s_2 \to_{\mathcal{R}_{\texttt{TFO}},top} \cdot \trianglerighteq t_2 \to^*_{\mathcal{R},in} \dots$ there exists also a minimal infinite chain $\nu'(s_1) \to_{\mathcal{R}_{\texttt{TFO}},top} \cdot \trianglerighteq q_1 \to^*_{\mathcal{R}_{\texttt{TFO}},in} \nu'(s_2) \to_{\mathcal{R}_{\texttt{TFO}},top} \cdot \trianglerighteq q_2 \to^*_{\mathcal{R}_{\texttt{TFO}},in} \cdots$*

Proof. For given i, let $l \to r \in \mathcal{R}_{\texttt{TFO}}$, a subterm p of r and a substitution γ be such that $s_i = l\gamma$ and $t_i = p\gamma$; let $\gamma^{\downarrow}$ be the substitution mapping x to $\gamma(x){\downarrow_{\mathcal{R}}}$ for x in the domain of γ and write $l = f\ l_1 \cdots l_n$. Since $\mathcal{R}_{\texttt{TFO}}$ is overlay, $l'\gamma^{\downarrow}$ cannot be an instance of the left-hand side of a rule for any strict subterm l' of l, so each $l_j\gamma{\downarrow_{\mathcal{R}}}$ is exactly $l_j\gamma^{\downarrow}$. Let $q_i = p\gamma^{\downarrow}$; then $\nu'(s_i) = l\gamma^{\downarrow} \to_{\mathcal{R}_{\texttt{TFO}},top} \cdot \trianglerighteq q_i$.

We can write $p = g\ p_1 \cdots p_m,\ s_{i+1} = g\ v_1 \cdots v_m$ and $q_i = g\ u_1 \cdots u_m$, where each $u_j = p_j\gamma^{\downarrow}$; since $p_j\gamma \to^*_{\mathcal{R}} v_j$, we have $u_j{\downarrow_{\mathcal{R}}} = v_j{\downarrow_{\mathcal{R}}}$. Noting that all higher order subterms of q_i are $\mathcal{R}$-normalised, Lemma 6 gives us that $u_j{\downarrow_{\mathcal{R}_{\texttt{TFO}}}} = u_j{\downarrow_{\mathcal{R}}}$. Thus, $q_i \to^*_{\mathcal{R}_{\texttt{TFO}},in} g\ \nu(u_1) \cdots \nu(u_m) = \nu'(t_i) = \nu'(s_{i+1})$ as required. □

Finally, to get rid of (normalised!) higher order subterms, introduce a variable $\bot_\iota$ for all base types ι. For base-type term s, define $\texttt{rep}(s) = f\ \texttt{rep}(s_1) \cdots \texttt{rep}(s_n)$ if $s = f\ s_1 \cdots s_n$ with $f \in \texttt{TFO}$; otherwise $\texttt{rep}(s) = \bot_\iota$. It follows easily that:

Lemma 8 (Replacing higher order terms by variables). *If all higher order subterms of s are $\mathcal{R}$-normalised, and $s \to_{\mathcal{R}_{\mathtt{TFO}}} t$, then $\mathtt{rep}(s) \to_{\mathcal{R}_{\mathtt{TFO}}} \mathtt{rep}(t)$.*

Proof. With induction on p it is evident that, for base-type terms p, always $\mathtt{rep}(p\gamma) = p\gamma^{\mathtt{rep}}$, where $\gamma^{\mathtt{rep}}(x) = \mathtt{rep}(\gamma(x))$. Using induction on the position of the redex in s, this provides the base case ($s \to_{\mathcal{R}_{\mathtt{TFO}},top} t$); the induction case, $s = f\ s_1 \cdots s_n \to_{\mathcal{R}_{\mathtt{TFO}},in} f\ s_1 \cdots s_i' \cdots s_n = t$, holds by induction hypothesis. □

We now have all the preparations to see that if there is an infinite chain with all head symbols in `TFO`, there is one on first order terms and with first order rules.

Theorem 9. *Let $(\mathcal{F}, \mathcal{R})$ be a higher order rewrite system with unique normal forms and let $\mathcal{R}_{\mathtt{TFO}}$ be overlay. Then $\to_{\mathcal{R}}$ is terminating if and only if:*

- *there is no minimal infinite chain using only `PHO` rules in the $\to_{\mathcal{R},top}$ steps, and*
- *$\mathcal{R}_{\mathtt{TFO}}$ is terminating on truly first order terms*

Proof. Suppose $(\mathcal{F}, \mathcal{R})$ is terminating. Then $\mathcal{R}_{\mathtt{TFO}}$ is also terminating (since $\mathcal{R}_{\mathtt{TFO}} \subseteq \mathcal{R}$), and there is no minimal infinite chain at all (since termination of $\to_{\mathcal{R}}$ implies termination of $\to_{\mathcal{R}} \cup \rhd$), let alone using only `PHO` rules.

Suppose both properties hold; by Corollary 5, $\to_{\mathcal{R}}$ is terminating if in addition there is no minimal infinite chain using only `TFO` rules in the $\to_{top}$ steps. Towards a contradiction, suppose that such a chain exists. By Lemma 7 there is a chain which uses only `TFO` rules, $\nu'(s_1) \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq q_1 \to^*_{\mathcal{R}_{\mathtt{TFO}},in} \nu'(s_2) \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq \ldots$; by Lemma 6 (strict subterms of $\nu'(s_1)$ are normalised) higher order subterms are normalised in all terms in the chain. Therefore, by Lemma 8, $\mathtt{rep}(\nu'(s_1)) \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq \mathtt{rep}(q_1) \to^*_{\mathcal{R}_{\mathtt{TFO}},in} \mathtt{rep}(\nu'(s_2)) \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq \ldots$ is an infinite $\mathcal{R}_{\mathtt{TFO}}$ chain on truly first order terms, contradicting termination of $\mathcal{R}_{\mathtt{TFO}}$. □

Thus, given an orthogonal system (or at least, a system where $\mathcal{R}_{\mathtt{TFO}}$ is overlay, and some property guarantees unicity of normal forms), we can split the termination proof into two parts: first, some dependency pair approach, where the dependency pairs for the first order rules can be omitted, and second, proving $\mathcal{R}_{\mathtt{TFO}}$ terminating on truly first order terms.

For the latter part, note that only base-type terms top-reduce, and a base-type, truly first order term corresponds exactly to a purely functional term: we define $\mathtt{uncurry}(f\ s_1 \cdots s_n) = f(\mathtt{uncurry}(s_1), \ldots, \mathtt{uncurry}(s_n))$. The system is terminating if and only if its uncurried version (a many-sorted TRS) is terminating (using [25, Theorem 5], or with a straightforward induction to show that $\mathtt{uncurry}(s) \to_{\mathcal{R}_{\mathtt{TFO}}^{\mathtt{uncurry}}} \mathtt{uncurry}(t)$ if and only if $s \to_{\mathcal{R}_{\mathtt{TFO}}} t$).

Since $\mathcal{R}_{\mathtt{TFO}}$ is a first order overlay TRS with unique normal forms, it is terminating if it is innermost terminating: by [16] this holds for a locally confluent overlay TRS, and by e.g. [34] an innermost terminating (so weakly normalising) TRS with unique normal forms is confluent. Since [10] shows that innermost termination is persistent (a many-sorted TRS is innermost terminating if and only if it is innermost terminating without regarding types), we can send the resulting TRS to any first order termination prover without losing generality, whether or not this prover is type-conscious.

Example 10. $\mathcal{R}_{\texttt{list}}$ is terminating iff the following TRS is terminating:

$$\begin{aligned}
\texttt{append}(\texttt{nil}, l) &\to l\\
\texttt{append}(\texttt{cons}(h,t), l) &\to \texttt{cons}(h, \texttt{append}(t,l))\\
\texttt{reverse}(\texttt{nil}) &\to \texttt{nil}\\
\texttt{reverse}(\texttt{cons}(h,t)) &\to \texttt{append}(\texttt{reverse}(t), \texttt{cons}(h, \texttt{nil}))\\
\texttt{shuffle}(\texttt{nil}) &\to \texttt{nil}\\
\texttt{shuffle}(\texttt{cons}(h,t)) &\to \texttt{cons}(h, \texttt{shuffle}(\texttt{reverse}(t)))\\
\texttt{mirror}(\texttt{nil}) &\to \texttt{nil}\\
\texttt{mirror}(\texttt{cons}(h,t)) &\to \texttt{append}(\texttt{cons}(h, \texttt{mirror}(t)), \texttt{cons}(h, \texttt{nil}))
\end{aligned}$$

and there are no infinite chains using for top steps only the two `map` rules.

Termination of $\mathcal{R}_{\texttt{TFO}}$ cannot be demonstrated with HORPO, even combined with dependency pairs and argument filterings (since the first-order recursive path orderings with these techniques cannot handle it). However, a first order approach using e.g. dependency pairs and a polynomial interpretation to the natural numbers has no trouble with the resulting first order rules.

As for the higher order part, using the static dependency pair approach from [32] there is one dependency pair $\texttt{map}^\sharp\ (\lambda x.F(x))\ (\texttt{cons}\ x\ y) \to \texttt{map}^\sharp\ (\lambda x.F(x))\ y$ with an empty set of usable rules; HORPO easily solves this.

Splitting the rules in a finitely branching system The requirements for Theorem 9 are essential; consider for example the following system, where the higher-order part lacks the "unique normal forms" property:

$$\begin{array}{ll}
\texttt{f}\ x\ \texttt{b} \to \texttt{g}\ x\ x & \texttt{h}\ (\lambda x.F(x)) \to F(\texttt{a})\\
\texttt{g}\ x\ \texttt{a} \to \texttt{f}\ x\ x & \texttt{h}\ (\lambda x.F(x)) \to F(\texttt{b})
\end{array}$$

Although $\mathcal{R}_{\texttt{TFO}}$ (which consists of the two rules on the left) is terminating and orthogonal, there is an infinite chain with all top-steps in $\mathcal{R}_{\texttt{TFO}}$:

$$\begin{aligned}
\texttt{f}\ (\texttt{h}\ (\lambda x.x))\ \texttt{b} &\to \texttt{g}\ (\texttt{h}\ (\lambda x.x))\ (\texttt{h}\ (\lambda x.x)) \to \texttt{g}\ (\texttt{h}\ (\lambda x.x))\ \texttt{a}\\
&\to \texttt{f}\ (\texttt{h}\ (\lambda x.x))\ (\texttt{h}\ (\lambda x.x)) \to \texttt{f}\ (\texttt{h}\ (\lambda x.x))\ \texttt{b}
\end{aligned}$$

This happens because the first order part is duplicating, and $\texttt{h}\ (\lambda x.x)\ \texttt{a}$ reduces both to $\texttt{a}$ and to $\texttt{b}$ (the F in the corresponding rules is a meta-variable, so a β-step is implied). Note that the role of the higher order part could be taken over by a pair of first order rules, $\texttt{c}(x,y) \to x, \texttt{c}(x,y) \to y$: $\mathcal{R}_{\texttt{TFO}}$ is not $\mathcal{C}_\varepsilon$*-terminating.* Following a technique originally due to Gramlich [15], and occurring in definitions for *usable rules* for full termination [14,18,32], we will see that absence of minimal infinite chains for $\mathcal{R}_{\texttt{TFO}}$ holds if $\mathcal{R}_{\texttt{TFO}}$ (seen as a first order TRS) is $\mathcal{C}_\varepsilon$-terminating.

Roughly, the idea is thus: in a finitely branching system, any term s which is not headed by a symbol in `TFO` can be replaced by the list $s' := \texttt{c}\ t_1\ (\texttt{c}\ t_2 \dots\ (\texttt{c}\ t_n \bot))$ of its immediate reducts; by the two `c`-rules, s' still reduces to all reducts of s. Doing this replacement everywhere in a term does not affect the applicability of first order rules. Thus, in a term $f\ s_1 \cdots s_n$ where all s_i are terminating, the transformation can be repeated until only first order symbols, `c` and $\bot$ remain.

In the following definitions and Lemma 11, let $\mathcal{R}$ be finitely branching.

For all base types ι, let $\bot_\iota$ be a variable of type ι and let $\mathtt{c}_\iota : \iota \Rightarrow \iota \Rightarrow \iota$ be a new function symbol. Let $\mathcal{R}^C_{\mathtt{TFO}} := \mathcal{R}_{\mathtt{TFO}} \cup \{\mathtt{c}_\iota\ x\ y \to x,\ \mathtt{c}_\iota\ x\ y \to y \mid \iota \in \mathcal{B}\}$. Now, for terminating base-type terms s we define $\psi(s)$ and $A(s)$ with a shared induction on $\to_{\mathcal{R}} \cup \rhd$, as follows:

- $\psi(f\ s_1 \cdots s_n) = f\ \psi(s_1) \cdots \psi(s_n)$ if $f \in \mathtt{TFO}$; $\psi(s) = A(s)$ for other s
- $A(s) = D_\iota(\{t \mid s \to_{\mathcal{R}} t\})$ (if $s : \iota$), where D_ι is a function on finite sets of terminating terms, defined by: $D_\iota(X) = \bot_\iota$ if $X = \emptyset$, and $\mathtt{c}_\iota\ \psi(t)\ D_\iota(X \setminus \{t\})$ if X is nonempty and t is its smallest element (ordered lexicographically).

Note that $\{t \mid s \to_{\mathcal{R}} t\}$ is finite by the assumption that $\mathcal{R}$ is finitely branching.

Lemma 11. *If $s \to_{\mathcal{R}} t$ with s a terminating base-type term, then $\psi(s) \to^*_{\mathcal{R}^C_{\mathtt{TFO}}} \psi(t)$.*

Proof. First note that:

1. for truly first order terms q and substitutions γ whose domain includes $FV(q)$, if $q\gamma$ is terminating then $\psi(q\gamma) = q\gamma^\psi$, where $\gamma^\psi(x) = \psi(\gamma(x))$ for x in the domain of γ. This follows immediately with induction on q.
2. $D_\iota(X) \to^*_{\mathcal{R}^C_{\mathtt{TFO}}} \psi(q)$ for any $q \in X$, by a straightforward induction on the size of X. Therefore $A(s) \to^*_{\mathcal{R}^C_{\mathtt{TFO}}} \psi(t)$ if $s \to_{\mathcal{R}} t$.

We prove Lemma 11 by induction on the size of s. If $head(s) \notin \mathtt{TFO}$, then by (2), $\psi(s) = A(s) \to^*_{\mathcal{R}^C_{\mathtt{TFO}}} \psi(t)$. Otherwise, let $s = f\ s_1 \cdots s_n$ with $f \in \mathtt{TFO}$; all s_i have base type, so if a step is done in one of the s_i, we can apply the induction hypothesis. If $s \to_{\mathcal{R},top} t$ then $s = l\gamma$, $t = r\gamma$ for some $l \to r \in \mathcal{R}_{\mathtt{TFO}}$ and substitution γ. Using (1): $\psi(s) = \psi(l\gamma) = l\gamma^\psi \to_{\mathcal{R}^C_{\mathtt{TFO}}} r\gamma^\psi = \psi(r\gamma) = \psi(t)$. □

Theorem 12. *A finitely branching higher order term rewrite system $(\mathcal{F}, \mathcal{R})$ is terminating if:*

- *there is no infinite chain using only* PHO *rules in the $\to_{\mathcal{R},top}$ steps and*
- *$\mathcal{R}^C_{\mathtt{TFO}}$ is terminating on truly first order terms*

Proof. By Corollary 5, it suffices if termination of $\mathcal{R}^C_{\mathtt{TFO}}$ implies that there is no minimal infinite chain using only TFO rules in the $\to_{\mathcal{R},top}$ steps. So suppose there is such a chain $s_1 \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq t_1 \to^*_{\mathcal{R},in} s_2 \ldots$ We must see that $\mathcal{R}^C_{\mathtt{TFO}}$ is non-terminating or, equivalently, that there is an infinite $\to_{\mathcal{R}^C_{\mathtt{TFO}}} \cdot \trianglerighteq$ reduction, on truly first order terms. For a term $u = f\ u_1 \cdots u_n$ with all u_j terminating, let $\psi'(u) = f\ \psi(u_1) \cdots \psi(u_n)$. Then each $\psi'(t_i) \to^*_{\mathcal{R}^C_{\mathtt{TFO}}} \psi'(s_{i+1})$ by Lemma 11, and $\psi'(s_i) = \psi'(l_i\gamma_i) = l_i\gamma_i^\psi \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq p_i\gamma_i^\psi = \psi'(t_i)$ by Observation (1) from its proof. Thus, $\psi'(s_1) \to_{\mathcal{R}_{\mathtt{TFO}},top} \cdot \trianglerighteq \psi'(t_1) \to^*_{\mathcal{R}^C_{\mathtt{TFO}},in} \ldots$ gives the required infinite reduction. □

Note that, unlike Theorem 9, Theorem 12 is not an equivalence. Even if $\mathcal{R}$ is terminating, $\mathcal{R}^C_{\mathtt{TFO}}$ may not be. Consequently, if proving termination of $\mathcal{R}_{\mathtt{TFO}}$ fails, a (sufficiently advanced) higher order approach might still succeed.

As before, we can uncurry the resulting system to obtain a many-sorted TRS. This time, however, dropping types may result in losing termination.

Example 13. Consider the system with six function symbols,

$$\begin{array}{llll} 0 : \mathtt{nat} & \mathtt{avg} : \mathtt{nat} \Rightarrow \mathtt{nat} \Rightarrow \mathtt{nat} & \mathtt{fun} : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat} \\ \mathtt{s} : \mathtt{nat} \Rightarrow \mathtt{nat} & \mathtt{check} : \mathtt{nat} \Rightarrow \mathtt{nat} & \mathtt{apply} : \mathtt{nat} \Rightarrow \mathtt{nat} \Rightarrow \mathtt{nat} \end{array}$$

and the following rules:

$$\begin{array}{rl} \mathtt{avg}\ 0\ 0 \to 0 & \mathtt{avg}\ x\ (\mathtt{s}\ (\mathtt{s}\ (\mathtt{s}\ y))) \to \mathtt{s}\ (\mathtt{avg}\ (\mathtt{s}\ x)\ y) \\ \mathtt{avg}\ 0\ (\mathtt{s}\ 0) \to 0 & \\ \mathtt{avg}\ 0\ (\mathtt{s}\ 0) \to \mathtt{s}\ 0 & \mathtt{apply}\ (\mathtt{fun}\ (\lambda x.F(x)))\ y \to F(\mathtt{check}\ y) \\ \mathtt{avg}\ 0\ (\mathtt{s}\ (\mathtt{s}\ 0)) \to \mathtt{s}\ 0 & \mathtt{check}\ 0 \to 0 \\ \mathtt{avg}\ (\mathtt{s}\ x)\ y \to \mathtt{avg}\ x\ (\mathtt{s}\ y) & \mathtt{check}\ (\mathtt{s}\ x) \to \mathtt{s}\ (\mathtt{check}\ x) \end{array}$$

The `fun` symbol signifies an encoding of a function in the natural numbers, and `apply` decodes it. To avoid losing termination, the `apply` function employs a check that the function is applied only on a constructor ground term.

This system does not satisfy the requirements from [20], nor can the static framework from [32] be applied. However, we *can* use the dynamic approach from [26]. Thus, by Theorem 12 (not Theorem 9, because the first order part does not have unique normal forms), it suffices to show termination of the TRS:

$$\begin{array}{lll} \mathtt{avg}(0,0) \to 0 & \mathtt{avg}(\mathtt{s}(x),y) \to \mathtt{avg}(x,\mathtt{s}(y)) & \mathtt{c}(x,y) \to x \\ \mathtt{avg}(0,\mathtt{s}(0)) \to 0 & \mathtt{avg}(x,\mathtt{s}(\mathtt{s}(\mathtt{s}(y)))) \to \mathtt{s}(\mathtt{avg}(\mathtt{s}(x),y)) & \mathtt{c}(x,y) \to y \\ \mathtt{avg}(0,\mathtt{s}(0)) \to \mathtt{s}(0) & \mathtt{check}(0) \to 0 & \\ \mathtt{avg}(0,\mathtt{s}(\mathtt{s}(0))) \to \mathtt{s}(0) & \mathtt{check}(\mathtt{s}(x)) \to \mathtt{s}(\mathtt{check}(x)) & \end{array}$$

And additionally find a higher order reduction pair which satisfies $l \geq r$ for all rules, and moreover $\mathtt{apply}^\sharp\ (\mathtt{fun}\ (\lambda x.F(x)))\ y > F(\mathtt{check}(y))$.

For the first part, all rules are strictly oriented with a polynomial interpretation of $f_0 = 1,\ f_{\mathtt{s}}(x) = x + 1,\ f_{\mathtt{avg}}(x,y) = 3x + 2y,\ f_{\mathtt{check}}(x) = 2x,\ f_{\mathtt{c}}(x,y) = x{+}y{+}1$. For the latter part, consider an argument filtering $\pi(\mathtt{check}\ x) = \mathtt{check}_\pi$, $\pi(\mathtt{s}\ x) = x,\ \pi(\mathtt{avg}\ x\ y) = \mathtt{avg}_\pi$. It suffices to find a reduction pair such that:

$$\begin{array}{lll} \mathtt{apply}^\sharp\ (\mathtt{fun}\ (\lambda x.F(x)))\ y > F(\mathtt{check}_\pi) & \mathtt{check}_\pi \geq 0 & \mathtt{avg}_\pi \geq \mathtt{avg}_\pi \\ \mathtt{apply}\ (\mathtt{fun}\ (\lambda x.F(x)))\ y \geq F(\mathtt{check}_\pi) & \mathtt{check}_\pi \geq \mathtt{check}_\pi & \mathtt{avg}_\pi \geq 0 \end{array}$$

Which is satisfied with HORPO, using a precedence $\mathtt{fun} >_\mathcal{F} \mathtt{check}_\pi, \mathtt{avg}_\pi >_\mathcal{F} 0$.

Discussion The restriction to finitely branching systems cannot be dropped, as might be demonstrated with a higher order adaptation of [30, Example 4.6]. However, in practice it is no great problem: a system given by a finite set of rules, even polymorphic rules, is finitely branching in common higher order formalisms.

5 Experiments

We have implemented the contributions of this paper in the higher order termination tool WANDA [24], using a combination of dynamic and static dependency

pairs. WANDA is a participant in the higher order category of the annual International Termination Competition.[3] Here, termination tools compete for power on benchmarks from several categories, with examples from the *Termination Problem Database (TPDB)*. This database is a collection of termination problems from research papers and applications that has been accumulated over the years.[4]

In the competition of 2010, WANDA could prove termination of 7 out of the 12 randomly chosen examples from the TPDB in the category *Higher-Order Rewriting - Union Beta*, coming a close second to THOR, which could handle the same examples plus `Mixed_HO_10/prefixsum.xml` (in the mean time WANDA can also deal with this example). This shows that WANDA is among the state-of-the-art higher order termination provers.

We have coupled WANDA with the first order termination tool AProVE [12] as a black-box to analyse termination of the first order TRSs generated by WANDA. To assess our contributions empirically, we have conducted experiments on an Intel Xeon CPU 5140 with four cores clocked at 2.33 GHz, investigating full termination of in total 152 higher order rewrite systems. As in the termination competition, the proof attempt is aborted after a timeout of 60 seconds.

The Higher Order category in the current TPDB (v8.0) is not very rich in examples (there are only 40 benchmarks). Therefore, we additionally consider higher order termination (union beta) for the 110 (originally untyped) applicative TRSs of the TPDB which could automatically be assigned a simple type.[5] We assume λ-abstraction is allowed in term formation, even though the rules do not use it. Of course, this solves a different problem than the one originally intended; thus these results should not be compared to first order tools analysing the same examples as untyped applicative systems. Additionally, we tested the systems from Examples 1 and 13. We did not include examples from the Haskell category, because WANDA's type system cannot yet deal with the polymorphism present.

	WANDAProVE	WANDA without first order back-end
YES	110	100
NO	10	10
MAYBE	25	38
TIMEOUTS	7	4
Avg. runtime	5.17 *s*	2.90 *s*

Fig. 1. Experimental results of WANDA with and without AProVE as first order prover

Our experiments, which are summarised in Figure 1, show that WANDA combined with AProVE can deal with all examples where plain WANDA succeeds, and 10 more. Out of these 10 additional examples, 8 stem from the applicative benchmarks from the TPDB; the other 2 are the examples used in this paper.

On the benchmarks available in the higher order category of TPDB v8.0, the number of termination proofs is unchanged. This is not surprising since each of

[3] See also `http://termination-portal.org/wiki/Termination_Competition`.
[4] For further information we refer to `http://termination-portal.org/wiki/TPDB`.
[5] A variation of these examples has by now been accepted for the next TPDB version.

these benchmarks focusses on the higher order aspect, so improvements on the side of the first order aspect can be expected to have only little impact. Runtime increases moderately from an average of 2.90 seconds to 5.17 seconds, which is still far from the timeout of 60 seconds per example, whereas termination proving power increases by 10%.

For details on our experiments and for access to our example suites, we refer to `http://aprove.informatik.rwth-aachen.de/eval/WANDAProVE/`.

6 Discussion

Overview of the technique Using Theorems 9 and 12 we can use a first order termination prover as a "black box" for a higher order tool, as follows:

1. determine `TFO` and `PHO` as described in Section 3, as well as $\mathcal{R}_{\texttt{TFO}}$;
2. if $\mathcal{R}_{\texttt{TFO}}$ is overlay and $\mathcal{R}$ has unique normal forms, let $\mathcal{R}'$ be the uncurried form of $\mathcal{R}_{\texttt{TFO}}$; if the system does not satisfy these properties (or we cannot determine whether it does) let $\mathcal{R}'$ be the uncurried form of $\mathcal{R}^{C}_{\texttt{TFO}}$;
3. feed $\mathcal{R}'$ into a first order termination prover (ignoring the types, unless a prover for many-sorted TRSs is used);
4. if this returns NO and no $\mathtt{c}_\iota$-rules were added to $\mathcal{R}'$, return NO;
   if it returns YES, continue with a dependency pair approach which omits the dependency pairs headed by symbols in `TFO`;
   otherwise continue with a direct approach for the complete system.

Note that, if the first order prover fails, this algorithm does not abort, but attempts to prove termination of the first order rules along with the rest. It is arguably not very likely that this will be more successful, but a higher order tool may be able to take steps which a type-oblivious first order tool cannot.

Dependency pairs While we have not explicitly used dependency pairs except in the examples, the notion of a minimal infinite chain naturally suggests the use of dependency pairs. Several approaches have been suggested for various forms of higher order rewriting [2,31,32,27]. Theorems 9 and 12 provide a way to remove some (perhaps most!) of the dependency graph components of realistic higher order systems, by delegating these to a first order termination prover.

Contribution The approach outlined in this paper allows (automatic) termination provers to use first order techniques to deal with first order dependency pairs. If we work on finitely branching HRSs with static dependency pairs, Theorem 12 is a direct result of the usable rules approach in [32], but our result holds on *all* common formalisms for higher order rewriting and *any* kind of dependency pair framework. Moreover, for orthogonal systems the result from Theorem 9 is strictly stronger than the theory obtained from this usable rules approach. Our experiments reveal a notable increase of termination proving power by this successful combination of a higher order termination prover with a first order termination prover as a back-end. Therefore, we expect that it will become essential for successful higher order termination provers to either use first order techniques immediately or enlist an external first order termination prover.

Future Work It might be possible to extend the use of first order provers further by identifying groups of dependency pairs where the higher order aspect is not actively used (such as a dependency pair $\mathtt{map}^\sharp(\lambda x.F(x), \mathtt{cons}(h,t)) \to \mathtt{map}^\sharp(\lambda x.F(x), t)$); dropping types, and transforming an abstraction into a single variable, such pairs might also be handled with first order techniques.

Acknowledgements We are very grateful for the constructive remarks of the anonymous referees and Femke van Raamsdonk, which helped improve the paper.

References

1. T. Aoto and Y. Yamada. Dependency pairs for simply typed term rewriting. In *Proc. RTA '05*, volume 3467 of *LNCS*, pages 120–134. Springer, 2005.
2. T. Aoto and Y. Yamada. Argument filterings and usable rules for simply typed dependency pairs. In *Proc. FroCoS '09*, volume 5749 of *LNAI*, pages 117–132. Springer, 2009.
3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
4. F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proc. RTA '00*, volume 1833 of *LNCS*, pages 47–61. Springer, 2000.
5. F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: the end of a quest. In *Proc. CSL '08*, volume 5213 of *LNCS*, pages 1–14. Springer, 2008.
6. C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proc. LPAR '01*, volume 2250 of *LNCS*, pages 531–547. Springer, 2001.
7. M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *Journal of Automated Reasoning*, 2011. To appear.
8. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.
9. 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*, volume 4501 of *LNCS*, pages 340–354. Springer, 2007.
10. C. Fuhs, J. Giesl, M. Parting, P. Schneider-Kamp, and S. Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 2011. To appear.
11. 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*, 33, 2011.
12. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, volume 4130 of *LNAI*, pages 281–286. Springer, 2006.
13. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. FroCoS '05*, volume 3717 of *LNAI*, pages 216–231. Springer, 2005.
14. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

15. B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.
16. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
17. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
18. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
19. N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for termination. In *Proc. LPAR '08*, volume 5330 of *LNAI*, pages 667–681. Springer, 2008.
20. J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 1997.
21. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proc. LICS '99*, pages 402–411, 1999.
22. R. Kennaway, J.W. Klop, M.R. Sleep, and F.-J. de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21(1):15–39, 1996.
23. J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1-2):279–308, 1993.
24. C. Kop. WANDA – a higher order termination tool. `http://www.few.vu.nl/~kop/code.html`.
25. C. Kop. Simplifying algebraic functional systems. In *Proc. CAI '11*, volume 5749 of *LNCS*, pages 201–215. Springer, 2011.
26. C. Kop and F. van Raamsdonk. Higher-order dependency pairs with argument filterings. In *Proc. WST '10*, 2010. `http://www.few.vu.nl/~kop/wst10.pdf`.
27. C. Kop and F. van Raamsdonk. Higher order dependency pairs for algebraic functional systems. In *Proc. RTA '11*, volume 10 of *LIPIcs*, pages 203–218. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
28. K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 18(5):407–431, 2007.
29. T. Nipkow. Higher-order critical pairs. In *Proc. LICS '91*, pages 342–349, 1991.
30. E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136(2):333–360, 1994.
31. M. Sakai, Y. Watanabe, and T. Sakabe. An extension of the dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.
32. S. Suzuki, K. Kusakari, and F. Blanqui. Argument filterings and usable rules in higher-order rewrite systems. *IPSJ Transactions on Programming*, 4(2):1–12, 2011.
33. V. Tannen and G.H. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1):3–28, 1991.
34. Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.
35. T. Yamada. Confluence and termination of simply typed term rewriting systems. In *Proc. RTA '01*, volume 2051 of *LNCS*, pages 338–352. Springer, 2001.
36. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.