Higher Order Dependency Pairs for Algebraic Functional Systems

Cynthia Kop[1] **and Femke van Raamsdonk**[1]

1 **Faculty of Sciences, VU, De Boelelaan 1081a, 1081 HV Amsterdam**

Abstract

We extend the termination method using dynamic dependency pairs to higher order rewriting systems with beta as a rewrite step, also called Algebraic Functional Systems (AFSs). We introduce a variation of usable rules, and use monotone algebras to solve the constraints generated by dependency pairs. This approach differs in several respects from those dealing with higher order rewriting modulo beta (e.g. HRSs).

Keywords and phrases higher order rewriting, termination, dynamic dependency pairs

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

An important method to (automatically) prove termination of first order term rewriting is the dependency pair approach by Arts and Giesl [3]. This approach transforms a rewrite system into groups of ordering constraints, such that rewriting is terminating if and only if the groups of constraints are (separately) solvable. Various optimizations of the method have been studied, see for example [7, 6].

This paper contributes to the study of dependency pairs for higher order rewriting. Higher order rewriting comes in different shapes. First, there is rewriting modulo $\alpha\beta\eta$ as in the higher order rewrite systems (HRSs) defined by Nipkow [20]; Klop's CRSs [13] and Khasidashvili's ERSs [12] are in some aspects similar. Various definitions of dependency pairs, often with optimizations, have been given for HRSs [23, 22, 17, 15, 24]. Second, applicative term rewriting systems with functional variables but no abstraction are sometimes considered as a (restricted) form of higher order rewriting. Also in this setting several definitions of dependency pairs exist [16, 18, 19, 1, 2, 8]. The aim of the present paper is to study dependency pairs for a third variant of higher order rewriting: *algebraic functional systems* (AFSs), introduced by Jouannaud and Okada [10]. In AFSs we consider simply typed terms, which are rewritten both using specific rewrite rules and β-reduction, with matching modulo α. While higher order versions of the recursive path ordering are commonly studied in the setting of AFSs [11, 5], there is little work on dependency pairs for this formalism.

We briefly discuss the ideas from studies of dependency pairs for HRSs and for applicative systems in Section 2; we also explain why those approaches do not quite, or not at all apply to the setting with AFSs. We define dependency pairs for AFSs in the so-called *dynamic* style, where functional variables in the right-hand side of a rewrite rule may give rise to dependency pairs. We study the notions of dependency chains, dependency graphs and reduction orders for AFSs with dynamic dependency pairs. To demonstrate that the dynamic approach has adequate strength even without restrictions, we also define a variant of usable rules and apply van de Pol's monotone algebra approach [21] to solve constraints generated by the method. The result is a method to prove termination (a complete method for left-linear systems), which may serve as a basis for further definitions – for example static dependency pairs, or dynamic pairs with restrictions that allow us to drop the subterm property.

submitted to 22nd International Conference on Rewriting Techniques and Applications.
Editor: M. Schmidt-Schauß Editor; pp. 1–29



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Background and Related Work

The extension of dependency pairs to the higher order case is not entirely straightforward and thus many variations exist. This work can roughly be split along two axes. On the one axis, the higher order formalism (we distinguish between applicative rewriting, rewriting modulo β (HRSs), and with β as a separate step (AFSs)), on the other the style of dependency pairs (with the common styles being *dynamic* and *static*). Figure 1 gives an overview.

	Applicative	**HRS**	**AFS**
Dynamic	[16]	[23] [15]	*this paper*
Static	[18] [19]	[4] [22] [17] [24]	[4]
Other	[1] [2] [8]	–	–

Figure 1 References on Higher Order Dependency Pairs

The dynamic and static approach differ in the treatment of leading variables in the right-hand sides of rules (subterms $x \cdot s_1 \cdots s_n$ with $n > 0$ and x a free variable). In the dynamic approach, such subterms lead to a dependency pair; in the static approach they do not. Consequently, first order techniques like argument filterings and usable rules are easier to extend to a static approach, while equivalence results tend to be limited to the dynamic style. Static dependency pairs can only be applied on systems satisfying certain restrictions.

Dependency pairs for applicative term rewriting We first say some words about applicative term rewriting. In applicative systems, terms are built from variables, constants and a binary application operator. Functional variables may be present, as in $x \cdot a$, but there is no abstraction, as in $\lambda x.\, x$. There are various styles of applicative rewriting.

A dynamic approach was defined both for untyped and simply-typed applicative systems in [16], along with a definition of argument filterings. A first static approach appears in [18] and is improved in [19]; the method is restricted to 'plain function passing' systems where, intuitively, leading variables are harmless. Due to the lack of binders, it is also possible to eliminate leading variables by instantiating them, as is done for simply typed systems in [1, 2]; in [8], an uncurrying transformation from untyped applicative systems to normal first order systems is used. These techniques have no parallel in rewriting with binders.

Unfortunately, they are not directly useful in the setting of AFSs, since termination may be lost by adding λ-abstraction and β-reduction. For example, the simply typed applicative system $\mathsf{app} \cdot (\mathsf{abs} \cdot F) \cdot x \to F \cdot x$, with $F : \iota \Rightarrow \iota$ a functional variable, $x : \iota$ a variable, and app, abs constants, is terminating because in every step the size of a term decreases. However, adding λ-abstraction and β-reduction spoils this property: with $\omega = \mathsf{abs} \cdot (\lambda x.\, \mathsf{app} \cdot x \cdot x)$ we have $\mathsf{app} \cdot \omega \cdot \omega = \mathsf{app} \cdot (\mathsf{abs} \cdot (\lambda x.\, \mathsf{app} \cdot x \cdot x)) \cdot \omega \to (\lambda x.\, \mathsf{app} \cdot x \cdot x) \cdot \omega \to \mathsf{app} \cdot \omega \cdot \omega$.

Dynamic Dependency Pairs for HRSs A first, very natural, definition of dependency pairs for HRSs is given in [23]. Here termination is not equivalent to the absence of infinite dependency chains, and a term is required to be greater than its subterms (the *subterm property*), which makes many optimizations impossible. Consequently, most of the focus since has been on the static approach. However, with restrictions on the rules the subterm property may be weakened, as discussed in [15] (extended abstract).

Static Dependency Pairs for HRSs The static approach in [18] is moved to the setting of HRSs in [17], and extended with argument filterings and usable rules in [24]. The static approach omits dependency pairs $f^{\#}(\vec{l}) \rightsquigarrow x(\vec{r})$ with x a variable, which avoids the need of a subterm property. The technique is restricted to *plain function passing* HRSs; for example the (terminating) rule $\mathsf{foo}(\mathsf{bar}(\lambda x.\, F(x))) \to F(\mathsf{a})$ cannot be handled. In addition, bound variables

may become free in a dependency pair. For instance, the rule $\mathsf{I}(\mathsf{s}(n)) \to \mathsf{twice}(\lambda x.\, I(x), n)$ generates a pair $\mathsf{I}^{\#}(\mathsf{s}(n)) \rightsquigarrow \mathsf{I}^{\#}(x)$ which admits an infinite dependency chain.

The definitions for HRSs [23, 17] do not immediately carry over to AFSs, since AFSs may have rules of functional type and β-reduction is a separate rewrite step. A short paper by Blanqui [4] introduces static dependency pairs on a form of rewriting which includes AFSs, but it restricts to base-type rules. The present work considers dynamic dependency pairs and is most related to [23], but is adaptated for the different formalism. Our method conservatively extends the one for first order rewriting and provides a characterization of termination for left-linear AFSs. We have chosen for a dynamic rather than a static approach because, although the static approach is stronger when applicable, the dynamic definitions can be given without restrictions. It would be nice for future work to integrate the two approaches; for the moment they co-exist with each their own advantages and disadvantages.

3 Preliminaries

We consider higher order rewriting as defined by Jouannaud and Okada, also called Algebraic Functional Systems (AFSs). Terms are built from simply typed variables, abstraction and application (as in simply typed λ-calculus), and in addition function symbols which take a fixed number of typed arguments. Terms and matching are modulo α, and β is a rewrite step. We follow roughly the definitions in [25, Chapter 11], as recalled below.

Types and Terms The set of *simple types* (or just *types*) is generated from a given set $\mathcal{B}$ of *base types* and the binary type constructor $\Rightarrow$, which is right-associative. Types are denoted by σ, τ and base types by ι, κ. A type with at least one occurrence of $\Rightarrow$ is called a *functional type*. A *type declaration* is an expression of the form $(\sigma_1 \times \ldots \times \sigma_n) \Rightarrow \tau$; if $n = 0$ this is written as just τ. Type declarations are not types, but are used for typing purposes.

We assume a set $\mathcal{V}$, consisting of infinitely many typed variables for each type, and a set $\mathcal{F}$ disjoint from $\mathcal{V}$, consisting of function symbols each equipped with a type declaration. Variables are denoted by x, y, z and function symbols by f, g, h or using more suggestive notation. To stress the type (declaration) of a symbol a we may write $a : \sigma$. *Terms* over $\mathcal{F}$ are those expressions s for which we can infer $s : \sigma$ for some type σ using the clauses:

(var)	$x : \sigma$	if $x : \sigma \in \mathcal{V}$
(app)	$s \cdot t : \tau$	if $s : \sigma \Rightarrow \tau$ and $t : \sigma$
(abs)	$\lambda x.\, s : \sigma \Rightarrow \tau$	if $x : \sigma \in \mathcal{V}$ and $s : \tau$
(fun)	$f(s_1, \ldots, s_n) : \tau$	if $f : (\sigma_1 \times \ldots \times \sigma_n) \Rightarrow \tau \in \mathcal{F}$ and $s_1 : \sigma_1, \ldots, s_n : \sigma_n$

Note that a function symbol $f : (\sigma_1 \times \ldots \times \sigma_n) \Rightarrow \tau$ takes exactly n arguments, and τ is not necessarily a base type. λ binds occurrences of variables as in the λ-calculus. Terms are considered modulo α-conversion; bound variables are renamed if necessary. The set of variables of s which are not bound is denoted $FV(s)$. Application is left-associative.

A *substitution* $[\vec{x} := \vec{s}]$, with $\vec{x}$ and $\vec{s}$ non-empty finite vectors of equal length, is the homomorphic extension of the type-preserving mapping $\vec{x} \mapsto \vec{s}$ from variables to terms. Substitutions are denoted γ, δ, and the result of applying γ to a term s is denoted $s\gamma$. The *domain* $\mathsf{dom}(\gamma)$ of $\gamma = [\vec{x} := \vec{s}]$ is $\{\vec{x}\}$. Substituting does not capture free variables.

We assume a fresh symbol $\Box_\sigma : \sigma$ for every type σ. A *context* $C[]$ is a term with a single occurrence of some $\Box_\sigma$. The result of replacing $\Box_\sigma$ in $C[]$ by a term s of type σ is denoted $C[s]$. Free variables may be captured; if $C[] = \lambda x.\, \Box_\sigma$ then $C[x] = \lambda x.\, x$. If $s = C[t]$ we say t is a *subterm* of s, notation $s \trianglerighteq t$, or $s \triangleright t$ (strict subterm) if $C[]$ is not the empty context $\Box$.

Rules and Rewriting A *rewrite rule* is a pair of terms $l \to r$ such that l and r are terms of the same type and do not contain a subterm of the form $(\lambda x.\, s) \cdot t$, all free variables of r also occur in l, and l has the form $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$ (with $m \geq n \geq 0$). Given a set of rules $\mathcal{R}$, the *rewrite* or *reduction relation* $\to_{\mathcal{R}}$ on terms is given by the following clauses:

`(rule)`	$C[l\gamma]$	$\to_{\mathcal{R}}$	$C[r\gamma]$	with $l \to r \in \mathcal{R}$, C a context, γ a substitution
`(beta)`	$C[(\lambda x.\, s) \cdot t]$	$\to_{\mathcal{R}}$	$C[s[x := t]]$	

We sometimes use the notation $s \to_\beta t$ for a rewrite step using `(beta)`. An *algebraic functional system* (AFS) is the combination of a set of terms and a rewrite relation on this set, and is usually specified by a set of rules (perhaps with function symbols). A function symbol f is a *defined symbol* of an AFS if there is a rule with left-hand side $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$. A function symbol that is not a defined symbol is a *constructor symbol*. The sets of defined or constructor symbols are denoted by $\mathcal{D}$ or $\mathcal{C}$ respectively. A rewrite rule $l \to r$ is *left-linear* if every free variable occurs at most once in l; an AFS is left-linear if all its rewrite rules are.

▶ **Example 3.1.** Throughout this paper, we will consider as an example the AFS twice. It has four function symbols, $\mathsf{o} : \mathsf{nat}$, $\mathsf{s} : (\mathsf{nat}) \Rightarrow \mathsf{nat}$, $\mathsf{I} : (\mathsf{nat}) \Rightarrow \mathsf{nat}$, $\mathsf{twice} : (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$, and three rewrite rules:

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{I}(\mathsf{o}) & \to & \mathsf{o} & \mathsf{twice}(F) & \to & \lambda y.\, F \cdot (F \cdot y) \\ \mathsf{I}(\mathsf{s}(n)) & \to & \mathsf{s}(\mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot n) & & & \end{array}$$

An example reduction: $\mathsf{I}(\mathsf{s}(\mathsf{o})) \to \mathsf{s}(\mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot \mathsf{o}) \to \mathsf{s}((\lambda y.\, (\lambda x.\, \mathsf{I}(x)) \cdot ((\lambda x.\, \mathsf{I}(x)) \cdot y)) \cdot \mathsf{o}) \to_\beta \mathsf{s}((\lambda x.\, \mathsf{I}(x)) \cdot ((\lambda x.\, \mathsf{I}(x)) \cdot \mathsf{o})) \to_\beta \mathsf{s}((\lambda x.\, \mathsf{I}(x)) \cdot \mathsf{I}(\mathsf{o})) \to \mathsf{s}((\lambda x.\, \mathsf{I}(x)) \cdot \mathsf{o}) \to_\beta \mathsf{s}(\mathsf{I}(\mathsf{o})) \to \mathsf{s}(\mathsf{o})$.

The symbol I represents the identity function, and therefore no infinite reduction exists. However, this is not trivial to prove; neither orderings like HORPO [11] nor a static dependency pair approach can handle the second rule, due to the subterm $\mathsf{I}(x)$. The static approach gives a requirement $\mathsf{I}^\#(\mathsf{s}(n)) > \mathsf{I}^\#(x)$, where the right-hand side contains a variable which does not occur in the left-hand side. Since $>$ must be closed under substitution, this is impossible to satisfy, as $\mathsf{s}(n)$ might be substituted for x. Applying HORPO leads to a similar problem.

4 Dependency Pairs

An intuition behind the dependency pair approach is to identify those parts of the right-hand sides of rewrite rules which may give rise to an infinite reduction. These are subterms headed by a defined symbol (as in first order term rewriting), and also subterms headed by a free variable, because such a variable can be instantiated by a defined symbol or abstraction. The latter is typical for the *dynamic* approach to higher order dependency pairs.

In this section we will extend the concepts of dependency pairs and dependency chains to AFSs. We show that an AFS is terminating if it does not have an infinite dependency chain, and that absence of dependency chains characterizes termination for left-linear AFSs.

Completed Rules An AFS is *completed* by adding for each rule of the form $l \to \lambda x_1 \ldots x_n.\, r$ with $n > 0$ and r not an abstraction the n new rules $l \cdot x_1 \to \lambda x_2 \ldots x_n.\, r, \ldots, l \cdot x_1 \cdots \cdot x_n \to r$. We do this to avoid creating dependency pairs containing a β-redex. Completing does not affect termination. For example, the system twice is completed by adding $\mathsf{twice}(F) \cdot m \to F \cdot (F \cdot m)$. In the remainder of the paper, we work with completed AFSs.

Candidate terms The definition of a dependency pair uses the notion of candidate terms, intuitively those subterms which might cause non-termination. Subterms that cannot be reduced at the root are omitted, because they are not a minimal starting point of an infinite reduction. Bound variables that become free by taking a subterm are replaced by fresh constants. We denote by $\mathbb{C}$ the set consisting of infinitely many fresh symbols c_x with c_x the same type as x, where x in c_x is not bound and is not subject to α-conversion.

▶ **Definition 4.1.** We say $t[x_1 := c_{x_1}, \dots, x_n := c_{x_n}]$ is a *candidate term* of s if $s \trianglerighteq t$, and $x_1, \dots, x_n$ are the variables which occur bound in s but free in t, and either $t = f(t_1, \dots, t_n) \cdot t_{n+1} \cdots t_m$ with f a defined symbol and $m \geq n \geq 0$, or $t = x \cdot t_1 \cdots t_n$ with x free in s and $n > 0$. We denote the set of candidate terms of s by $Cand(s)$.

In the AFS twice we have $Cand(F \cdot (F \cdot m)) = \{F \cdot (F \cdot m),\ F \cdot m\}$ and $Cand(\mathsf{s}(\mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot n)) = \{\mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot n, \mathsf{twice}(\lambda x.\, \mathsf{I}(x)),\ \mathsf{I}(c_x)\}$. Note that for example $x \cdot y$ is not a candidate term of $g(\lambda x.\, x \cdot y)$ because x occurs only bound.

Dependency Pairs The definition of dependency pair also uses marked function symbols as in the first order case. Let $\mathcal{F}^\# = \mathcal{F} \cup \{f^\# : \sigma \mid f : \sigma \in \mathcal{D}\}$, so $\mathcal{F}$ extended with a marked version for every defined symbol, having the same type declaration. The marked counterpart of a term s, notation $s^\#$, is $f^\#(s_1, \dots, s_n)$ if $s = f(s_1, \dots, s_n)$ with f in $\mathcal{D}$, and just s otherwise. For example, $(\mathsf{twice}(F))^\# = \mathsf{twice}^\#(F)$ and $(\mathsf{twice}(F) \cdot m)^\# = \mathsf{twice}(F) \cdot m$.

▶ **Definition 4.2** (Dependency Pair)**.** The set of *dependency pairs* of a rewrite rule $l \to r$, notation $\mathsf{DP}(l \to r)$, consists of:

- all pairs $l^\# \rightsquigarrow p^\#$ with $p \in Cand(r)$,
- all pairs $l \cdot y_1 \cdots y_k \rightsquigarrow r \cdot y_1 \cdots y_k$ with $1 \leq k \leq n$ if $r : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \iota$, and either $r = x \cdot r_1 \cdots r_m$ with $m \geq 0$ or $r = f(r_1, \dots, r_i) \cdot r_{i+1} \cdots r_m$ with $m \geq i \geq 0$ and $f \in \mathcal{D}$.

We use $\mathsf{DP}(\mathcal{R})$ (or just DP) for the set of all dependency pairs of rewrite rules of an AFS $\mathcal{R}$.

▶ **Example 4.3.** The set of dependency pairs of the AFS twice consists of:

$$\begin{array}{lcllcl}
\mathsf{I}^\#(\mathsf{s}(n)) & \rightsquigarrow & \mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot n & \mathsf{twice}^\#(F) & \rightsquigarrow & F \cdot (F \cdot c_y) \\
\mathsf{I}^\#(\mathsf{s}(n)) & \rightsquigarrow & \mathsf{twice}^\#(\lambda x.\, \mathsf{I}(x)) & \mathsf{twice}^\#(F) & \rightsquigarrow & F \cdot c_y \\
\mathsf{I}^\#(\mathsf{s}(n)) & \rightsquigarrow & \mathsf{I}^\#(c_x) & \mathsf{twice}(F) \cdot m & \rightsquigarrow & F \cdot (F \cdot m) \\
 & & & \mathsf{twice}(F) \cdot m & \rightsquigarrow & F \cdot m
\end{array}$$

The last two dependency pairs originate from the rule added by completion.

To illustrate the second form of dependency pair, consider the system with function symbols $\mathsf{app} : (o) \Rightarrow o \Rightarrow o$ and $\mathsf{abs} : (o \Rightarrow o) \Rightarrow o$, and one rewrite rule: $\mathsf{app}(\mathsf{abs}(x)) \to x$. This system has no dependency pairs of the first form, but does admit a two-step loop: $s := \mathsf{app}(\mathsf{abs}(\lambda x.\, \mathsf{app}(x) \cdot x)) \cdot \mathsf{abs}(\lambda x.\, \mathsf{app}(x) \cdot x) \to (\lambda x.\, \mathsf{app}(x) \cdot x) \cdot \mathsf{abs}(\lambda x.\, \mathsf{app}(x) \cdot x) \to_\beta s$.

Comparing our approach to static dependency pairs as defined in [17], the two main differences are that we avoid bound variables becoming free, and that we include dependency pairs where the right-hand side is headed by a variable. We call such pairs *collapsing*.

Dependency Chains We can now investigate termination by means of *dependency chains*:

▶ **Definition 4.4.** A *dependency chain* is a sequence $[(\rho_i, s_i, t_i) \mid i \in \mathbb{N}]$ such that for all i:

1. $\rho_i \in \mathsf{DP} \cup \{\mathtt{beta}\}$,
2. if $\rho_i = l_i \rightsquigarrow p_i \in \mathsf{DP}$ there exists γ with domain $FV(l_i)$ such that $s_i = l_i\gamma$ and $t_i = p_i\gamma$

3. if $\rho_i = \mathtt{beta}$ then $s_i = (\lambda x.\, u) \cdot v \cdot w_1 \cdots w_k$ and either
   a. $k > 0$ and $t_i = u[x := v] \cdot w_1 \cdots w_k$, or
   b. $k = 0$ and there is w such that $u \trianglerighteq w$ and $x \in FV(w)$ and $w^\#[x := v] = t_i$, but $w \neq x$
4. $t_i \to_{in}^* s_{i+1}$

A step $\to_{in}$ is obtained by rewriting some s_i inside a term of the form $f(s_1, \ldots, s_n) \cdot s_{n+1} \cdots s_m$.

▶ **Theorem 4.5.** *If $\mathcal{R}$ is non-terminating there is an infinite dependency chain over $\mathsf{DP}(\mathcal{R})$.*

Proof Sketch. Say a term s is *minimally non-terminating* (MNT) if s is terminating but all its subterms are not. Let u_{-1} be any MNT term, and subsequently for every $i \in \mathbb{N}$, given an MNT term u_{i-1}, we define $\rho_i \in \mathsf{DP} \cup \{\mathtt{beta}\}$ and terms s_i and t_i. Note (**): *if an MNT term is reduced at any other position than the top, the result is also MNT, or terminating.*

If $u_{i-1} = (\lambda x.\, s) \cdot t$ then $s[x := t]$ is also non-terminating (because eventually a topmost step must be done, and we can see that $s[x := t]$ reduces to the result); let u_i be an MNT subterm of $s[x := t]$ and define $\rho_i, s_i, t_i := \mathtt{beta}, u_{i-1}, u_i^\#$. If $u_{i-1} = (\lambda x.\, s) \cdot t \cdot v_0 \cdots v_k$ then by (**) $u_i := s[x := t] \cdot v_0 \cdots v_k$ is also MNT, so choose $\rho_i, s_i, t_i := \mathtt{beta}, u_{i-1}, u_i$. Otherwise $u_{i-1} = f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$; then $u_{i-1} \to_{in}^*$ some term $l\gamma \cdot w_1 \cdots w_k$, with $r\gamma \cdot \vec{w}$ still non-terminating. If $k > 0$ then by (**) $r\gamma \cdot \vec{w}$ is MNT, so choose $u_i := r\gamma \cdot \vec{w}$ and $\rho_i, s_i, t_i := l \cdot x_1 \cdots x_k \rightsquigarrow r \cdot x_1 \cdots x_k,\ l\gamma \cdot \vec{w}, r\gamma \cdot \vec{w}$. Otherwise let r' be the smallest subterm of r such that $p := r'\delta$ is still non-terminating, where δ replaces the newly free variables x_i by c_{x_i}. Then some analysis shows that p is a candidate of r and $p\gamma$ is also MNT; choose $u_i := p\gamma$ and $\rho_i, s_i, t_i := l^\# \rightsquigarrow p^\#, l^\#\gamma, p^\#\gamma$. This process generates a dependency chain. ◀

The converse of Theorem 4.5 does not hold. Consider the AFS with rules:

$$f(x, y, \mathsf{s}(z)) \to g(h(x, y), \lambda u.\, f(u, x, z)) \quad \text{and} \quad h(x, x) \to f(x, \mathsf{s}(x), \mathsf{s}(\mathsf{s}(x)))$$

This system has the following dependency pairs:

$$\begin{array}{lcllcl} f^\#(x, y, \mathsf{s}(z)) & \rightsquigarrow & h^\#(x, y) & h^\#(x, x) & \rightsquigarrow & f^\#(x, \mathsf{s}(x), \mathsf{s}(\mathsf{s}(x))) \\ f^\#(x, y, \mathsf{s}(z)) & \rightsquigarrow & f^\#(c_u, x, z) & & & \end{array}$$

There is an infinite dependency chain: $f^\#(c_u, \mathsf{s}(c_u), \mathsf{s}(\mathsf{s}(c_u))) \rightsquigarrow f^\#(c_u, c_u, \mathsf{s}(c_u)) \rightsquigarrow h^\#(c_u, c_u) \rightsquigarrow f^\#(c_u, \mathsf{s}(c_u), \mathsf{s}(\mathsf{s}(c_u))) \rightsquigarrow \ldots$ However, the AFS is terminating, intuitively because the bound variable destroys matching possibilities. The crucial point of the example is the combination of bound variables and non-left-linear rules. Theorem 4.6 shows that for left-linear AFSs, the absence of infinite dependency chains actually characterizes termination.

▶ **Theorem 4.6.** *A left-linear AFS $\mathcal{R}$ is terminating if and only if it does not admit an infinite dependency chain.*

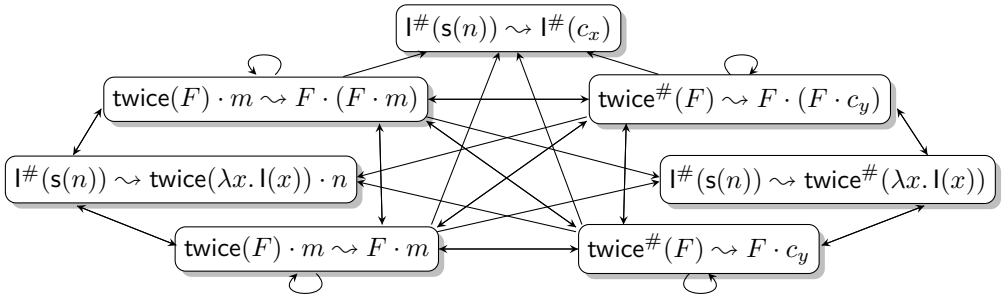
Proof Sketch. In a left-linear system replacing variables by a function symbol that doesn't occur in any rule has no effect on applicability of $\to_{\mathcal{R}}$. Thus a dependency chain effectively produces an infinite reduction $|s_i| \to_{\mathcal{R}} \cdot \rhd |t_i| \to_{\mathcal{R}}^* |s_{i+1}|$ (where $|\cdot|$ replaces any $f^\#$ by its unmarked counterpart), and this implies the existence of an infinite $\to_{\mathcal{R}}$ reduction. ◀

5 The Dependency Graph

As in the first order case, we use a dependency graph to organize the dependency pairs. The definition of a dependency graph is typical for our setting here, namely AFSs with dynamic dependency pairs, but the other notions we use are similar to the first order ones.

The *dependency graph* of an AFS $\mathcal{R}$ has the dependency pairs of $\mathcal{R}$ as nodes, and an edge from node $l \rightsquigarrow p$ to node $l' \rightsquigarrow p'$ if there is a finite dependency chain $[(l \rightsquigarrow p, s_1, t_1), (\mathtt{beta}, s_2, t_2), \ldots, (\mathtt{beta}, s_{k-1}, t_{k-1}), (l' \rightsquigarrow p', s_k, t_k)]$ with all but the first and the last elements `beta`.

▶ **Example 5.1.** The dependency graph of the AFS twice:



A *cycle* is a non-empty set $\mathcal{C}$ of dependency pairs such that between every two pairs $\rho, \pi \in \mathcal{C}$ there is a non-empty path in the graph using only nodes in $\mathcal{C}$. A cycle that is not contained in any other cycle is called a *strongly connected component* (SCC). To prove termination we must show that cycles in a dependency graph are in some sense well-behaved (see Theorem 6.2). Due to clause 3b in Definition 4.4, there is an edge from any node of the form $l \rightsquigarrow x \cdot r_1 \ldots \cdot r_n$ with x a variable to all other nodes. Hence a rule with a functional variable in its right-hand side gives rise to many cycles. Here, exactly, lies the appeal of the static approach, which eliminates the need for such pairs. However, this barrier is not impossible to overcome, and as discussed, the dynamic approach can deal with systems where the static approach fails.

A set $D \subseteq \mathsf{DP}$ is *looping* if there is an infinite dependency chain using only dependency pairs from D and `beta`. By termination of simply typed β-reduction, $\emptyset$ is not looping.

Because the dependency graph cannot be computed in general, one uses *approximations* of the dependency graph, which have the same nodes but possibly more edges. A brute method to find an approximation of the dependency graph is to have an edge between $l \rightsquigarrow p$ and $l' \rightsquigarrow p'$ as soon as the head of p is a variable, or if p and l' both have the form $f(s_1, \ldots, s_n) \cdot s_{n+1} \cdots s_m$ for some function symbol f and some $m \geq n \geq 0$. It is interesting to study more sophisticated methods to find approximations, but this is left for future work.

In the remainder of this paper, we will assume that dependency graphs (and hence also their approximations) have only finitely many nodes. This is the case if the AFS under consideration has finitely many rewrite rules. However, note that also for infinite AFSs (arising for example by instantiation of polymorphic rewrite rules) we can work with finite dependency graphs, if (infinite) sets of dependency pairs are represented by a single node.

▶ **Lemma 5.2.** *Let G be an approximation of the dependency graph of an AFS $\mathcal{R}$. Suppose that every cycle in G is non-looping. Then $\mathcal{R}$ is terminating.*

Proof Sketch. Given an infinite dependency chain, there must be a dependency pair ρ_i which occurs infinitely often (by the finiteness assumption). Then $\{\rho_j \mid j > i\}$ is a cycle. ◀

▶ **Example 5.3.** The dependency graph (approximation) of twice from Example 5.1 admits many cycles, such as $\{\mathsf{twice}(F) \cdot n \rightsquigarrow F \cdot (F \cdot n)\}$ or the following cycle $\mathcal{C}_{\mathsf{twice}}$:

$$\left\{ \begin{array}{rcl@{\qquad}rcl} \mathsf{I}^\#(\mathsf{s}(n)) & \rightsquigarrow & \mathsf{twice}(\lambda x.\mathsf{I}(x)) \cdot n & \mathsf{twice}^\#(F) & \rightsquigarrow & F \cdot (F \cdot c_y) \\ \mathsf{I}^\#(\mathsf{s}(n)) & \rightsquigarrow & \mathsf{twice}^\#(\lambda x.\mathsf{I}(x)) & \mathsf{twice}^\#(F) & \rightsquigarrow & F \cdot c_y \\ \mathsf{twice}(F) \cdot m & \rightsquigarrow & F \cdot (F \cdot m) & \mathsf{twice}(F) \cdot m & \rightsquigarrow & F \cdot m \end{array} \right\}$$

$\mathcal{C}_{\mathsf{twice}}$ is an SCC and includes all cycles. Therefore twice is terminating if $\mathcal{C}_{\mathsf{twice}}$ is non-looping.

6 Reduction Orders

The challenge, then, is to prove the absence of looping cycles. We use the following definition:

▶ **Definition 6.1.** A *reduction triple* consists of a well-founded ordering $>$, a quasi-ordering $\geq$ and a sub-relation $\geq_1$ of $\geq$, such that:

1. $>$ and $\geq$ are *compatible*: either $> \cdot \geq \subseteq >$ or $\geq \cdot > \subseteq >$;
2. $>$, $\geq$ and $\geq_1$ are all *stable* (that is, closed under substitution);
3. $\geq_1$ is *monotonic*: (that is, if $s \geq_1 t$ with s, t sharing a type, then $C[s] \geq_1 C[t]$);
4. $\geq_1$ contains `beta` (that is, always $(\lambda x.\, s) \cdot t \geq_1 s[x := t]$).

A *reduction pair* is a pair $(>, \geq)$ such that $(>, \geq, \geq)$ is a reduction triple; this corresponds with the original (first order) notion of reduction pair. The reduction triple is a generalisation of this notion, where $\geq$ itself is not required to be monotonic; we will need a non-monotonic $\geq$ in Section 6.1 to compare terms with different types. To deal with subterm reduction in dependency chains, an additional definition is needed. We say $\geq$ has the *limited subterm property* if: for all x, s, t, u such that $s \trianglerighteq u$ and u is neither an abstraction nor a single variable, there is a substitution γ such that $(\lambda x.\, s) \cdot t \geq (u^{\#})\gamma[x := t]$. Intuitively, the substitution γ is used to replace free variables in u that are bound in s by fresh constants c_x. However, we will also use a more liberal replacement of those variables, hence the general γ.

The following theorem shows how reduction triples can be used with dependency pairs.

▶ **Theorem 6.2.** *A set $D = D_1 \uplus D_2$ of dependency pairs is non-looping if D_2 is non-looping, and there is a reduction triple $(>, \geq, \geq_1)$ such that*

- *$l > p$ for all $l \rightsquigarrow p \in D_1$,*
- *$l \geq p$ for all $l \rightsquigarrow p \in D_2$,*
- *$l \geq_1 r$ for all $l \to r \in \mathcal{R}$,*
- *either D is non-collapsing or $\geq$ satisfies the limited subterm property.*

Proof Sketch. If D is looping it has an infinite chain which (as D_2 is non-looping) contains infinitely many pairs in D_1. If D is non-collapsing we can find such a chain without `beta` steps, and have $s_i \geq t_i \geq s_{i+1}$ for all i, and if $\rho_i \in D_1$ even $s_i > t_i$, contradicting well-foundedness of $>$. If D is collapsing then let $[(\rho_i, s_i, t_i) | i \in \mathbb{N} | i \geq j]$ be an infinite dependency chain over D; if $\rho_j \in D_1$ then $s_j > t_j \geq s_{j+1}$, if $\rho_j \in D_2$ then $s_j \geq t_j \geq s_{j+1}$ and if $\rho_j =$ `beta` then there is some substitution δ such that $s_j \geq t_j\delta \geq s_{j+1}\delta$. Since $[(\rho_i, s_i\delta, t_i\delta) | i \in \mathbb{N} | i \geq j+1]$ is also a dependency chain we can continue this reasoning recursively, obtaining a decreasing $\geq$ sequence with infinitely many $>$ steps, contradicting well-foundedness. ◀

Theorem 6.2 can be used to prove that every cycle in the dependency graph approximation of an AFS is non-looping; termination follows with Lemma 5.2. See also Section 9 for an algorithm. For left-linear AFSs, we even have a characterization of termination.

▶ **Theorem 6.3.** *A left-linear AFS with dependency graph approximation G is terminating if and only if for every cycle in G the requirements of Theorem 6.2 are satisfied.*

▶ **Example 6.4.** Termination of twice is proved if there is a reduction triple $(>, \geq, \geq_1)$ with the limited subterm property, such that $l \geq_1 r$ for all rules, and $l > p$ for every dependency pair in $\mathcal{C}_{\mathsf{twice}}$ from Example 5.3 (choosing $D_2 = \emptyset$, which is non-looping).

6.1 Type Changing

The situation so far is not completely satisfactory, because both $>$ and $\geq$ may have to compare terms of different types. Consider for example the dependency pair $\mathsf{twice}^\#(F) \rightsquigarrow F \cdot c_y$ from twice where the two sides have a different type. Moreover, the comparison in the definition of limited subterm property may concern terms of different types. This is problematic because term orderings do not usually compare terms of arbitrary different types; neither any version of the higher order path ordering [11, 5] nor monotone algebras [21] are equipped for this.

A solution is to manipulate the ordering requirements. Let $(\succ, \succeq)$ be a reduction pair (so a pair such that $(\succ, \succeq, \succeq)$ is a reduction triple). Define $>$, $\geq$, and $\geq_1$ as follows:

- $s > t$ if there are fresh variables $x_1, \ldots, x_n$ and terms $u_1, \ldots, u_m$ such that $s \cdot x_1 \cdots x_n \succ t \cdot u_1 \cdots u_m$ and both sides have some base type;
- $s \geq t$ if there are fresh variables $x_1, \ldots, x_n$ and terms $u_1, \ldots, u_m$ such that $s \cdot x_1 \cdots x_n \; R \; t \cdot u_1 \cdots u_m$ and both sides have some base type, where R is $\succeq \cup \succ \cdot \succeq \cup \succeq \cdot \succ$;
- $s \geq_1 t$ if $s \succeq t$ and s, t have the same type.

▶ **Lemma 6.5.** *$(>, \geq, \geq_1)$ as generated from a reduction pair $(\succ, \succeq)$ is a reduction triple.*

Proof. This is easy, noting: (1) if $s \geq_1 t$ then by monotonicity $s\vec{x} \succeq t\vec{x}$, (2) if $s > t$ then for any $\vec{u}$ there are $\vec{v}$ such that $s \cdot \vec{u} \succ t \cdot \vec{v}$ (by stability of $\succ$), (3) similar for $\geq$. ◀

The relations $>$ and $\geq$ are not necessarily computable, but we will not need to work with them directly. To prove some set of dependency pairs D non-looping, we can choose for every pair $l \rightsquigarrow p \in D$ a corresponding base-type pair $\overline{l} \rightsquigarrow \overline{p}$, and prove either $\overline{l} \succ \overline{p}$ or $\overline{l} \succeq \overline{p}$. For example, we could assign $\overline{l} := l \cdot x_1 \cdots x_n$ and $\overline{p} := p \cdot c_{y_1} \cdots c_{y_m}$, where the c_{y_i} are chosen arbitrarily. This is the choice we will use in examples in this paper. Other choices for $\overline{p}$, for instance made in such a way as to duplicate existing requirements, are also possible.

To make sure that $\geq$ satisfies the limited subterm property, we consider a base-type version of subterm reduction, which is strongly related to β-reduction.

▶ **Definition 6.6.** $\rhd^!$ is the relation on base-type terms (and $\trianglerighteq^!$ its reflexive closure) generated by the following clauses:

- $(\lambda x.\, s) \cdot t_0 \cdots t_n \rhd^! u$ if $s[x := t_0] \cdot t_1 \cdots t_n \trianglerighteq^! u$
- $f(s_1, \ldots, s_m) \cdot t_1 \cdots t_n \trianglerighteq^! u$ if $s_i \cdot \vec{c} \trianglerighteq^! u$
- $s \cdot t_1 \cdots t_n \trianglerighteq^! u$ if $t_i \cdot \vec{c} \trianglerighteq^! u$ (s may have any form)

Here, $s \cdot \vec{c}$ is a term s applied to constants c_y of the right type. Note that if $s \rhd t$ and s has base type, there are terms $u_1, \ldots, u_n$ and substitution γ such that $s \rhd^! t\gamma \cdot u_1 \cdots u_n$. Consequently, $\geq$ satisfies the limited subterm property if $\succ \cup \succeq$ contains $\rhd^!$ and $f(\vec{x}) \succeq f^\#(\vec{x})$ for all $f \in \mathcal{D}$ (the *marking property*). We can derive the following theorem.

▶ **Theorem 6.7.** *A set of dependency pairs $D = D_1 \uplus D_2$ is non-looping if D_2 is non-looping and there is a reduction pair $(\succ, \succeq)$ such that:*

1. *$\overline{l} \succ \overline{p}$ for all $l \rightsquigarrow p \in D_1$;*
2. *$\overline{l} \succeq \overline{p}$ for all $l \rightsquigarrow p \in D_2$;*
3. *$l \succeq r$ for all $l \to r \in \mathcal{R}$;*
4. *if D is collapsing, then $\succ \cup \succeq$ contains $\rhd^!$, and $f(\vec{x}) \succeq f^\#(\vec{x})$ for all $f \in \mathcal{D}$.*

▶ **Example 6.8.** To prove that $\mathcal{C}_{\mathsf{twice}}$ is non-looping it suffices to find a reduction pair $(\succ, \succeq)$ such that $l \succeq r$ for all rules, $\succeq$ satisfies the subterm and marking properties, and furthermore:

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{I}^\#(\mathsf{s}(n)) & \succ & \mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot n & \mathsf{twice}^\#(F) \cdot x & \succ & F \cdot (F \cdot c_y) \\ \mathsf{I}^\#(\mathsf{s}(n)) & \succ & \mathsf{twice}^\#(\lambda x.\, \mathsf{I}(x)) \cdot c_z & \mathsf{twice}^\#(F) \cdot x & \succ & F \cdot c_y \\ \mathsf{twice}(F) \cdot m & \succ & F \cdot (F \cdot m) & \mathsf{twice}(F) \cdot m & \succ & F \cdot m \end{array}$$

This completes the basis of dynamic dependency pairs for AFSs. But is this approach any easier than proving $l > r$ for all rewrite rules? Unless the dependency graph has no cycles we still have to prove $l \geq r$ for all rules and with an ordering like HORPO [11] this is barely an improvement. In Section 7 we will therefore discuss a variation of usable rules, which allows us to drop a number of ordering requirements. In Section 8 we will define a variation of the monotone algebra approach that is especially suited to dependency pairs.

7 Formative Rules

In the first order setting, the result corresponding with Theorem 6.2 is optimized: it is sufficient to consider for a cycle only its *usable rules* instead of all rules. The definition of usable rules cannot easily be extended to our setting, because we admit collapsing dependency pairs. Therefore we take a different approach with the same goal of restricting attention to rules which are in some way relevant to a set of dependency pairs. Where usable rules are defined from the right-hand sides of dependency pairs, our *formative rules* are based on the left-hand sides. We will use the notion of *simple terms*:

▶ **Definition 7.1.** A term s is *simple* if:

- it is linear,
- it has no subterm of the form $x \cdot s_1 \cdots s_n$ with $n > 0$ and x a free variable,
- there is no occurrence of a free variable below an abstraction.

Many examples of AFSs, such as rules from functional programming, have a simple left-hand side. The intuition behind formative rules is that, for rewrite rules with a simple left-hand side, only the formative rules can contribute to the creation of its pattern.

▶ **Definition 7.2.** For β-normal terms s, let $Symb(s)$ be recursively defined as follows:

$$\begin{array}{rcl} Symb(\lambda x.\, s : \sigma) & = & \{\langle ABS, \sigma\rangle\} \cup Symb(s) \\ Symb(f(s_1, \ldots, s_n) \cdot s_{n+1} \cdots s_m : \sigma) & = & \{\langle f, \sigma\rangle\} \cup Symb(s_1) \cup \ldots \cup Symb(s_m) \\ Symb(x \cdot s_1 \cdots s_n : \sigma) & = & \{\langle VAR, \sigma\rangle\} \cup Symb(s_1) \cup \ldots \cup Symb(s_n) \ (n > 0) \\ Symb(x) & = & \emptyset \end{array}$$

The formative symbols and rules of any term are defined by a (possibly) infinite process:

- the starting point: $FS_0(s) = Symb(s)$
- for all $n \geq 0$, the set $FR_n(s)$ consists of rules $l \cdot x_1 \cdots x_k \to r \cdot x_1 \cdots x_k$ if $l \to r \in \mathcal{R}$ and
  - $k = 0,\ r = \lambda x.\, r' : \sigma$ and $\langle ABS, \sigma\rangle \in FS_n(s)$, or
  - $r = f(\vec{u}) \cdot \vec{v} : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_k \Rightarrow \tau$ and $\langle f, \tau\rangle \in FS_n(s)$, or
  - $r = x \cdot \vec{v} : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_k \Rightarrow \tau$ and $\langle f, \tau\rangle \in FS_n(s)$ for some $f \in \mathcal{F} \cup \{ABS, VAR\}$; $|\vec{v}| \geq 0$
- $FS_{n+1}(s) = FS_n(s) \cup \bigcup_{l \to r \in FR_n(s)} Symb(l)$

Now $FR(s)$ is defined as the union of all $FR_n(s)$ (this is a finite union for finite AFSs) in the case that both s is simple and all rules in this union have a simple left-hand side. Otherwise, $FR(s) = \mathcal{R}$. The set of formative rules of a dependency pair, $FR(f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m \rightsquigarrow p)$, is defined as $\bigcup_{1 \leq i \leq m} FR(l_i)$. For a set D of dependency pairs, $FR(D) = \bigcup_{l \rightsquigarrow p \in D} FR(l \rightsquigarrow p)$.

Note that $FR_n(s)$ and $FS_{n+1}(s)$ can easily be calculated (automatically) from $FS_n(s)$; to compute $FR(s)$ a tool would simply repeat this process until either a rule with a non-simple left-hand-side is included (in which case $FR(s) = \mathcal{R}$), or until no new symbols are added.

▶ **Example 7.3.** Recall the rules for the (completed) system twice:

$$\begin{array}{llllllll} (A) & \mathsf{I}(\mathsf{o}) & \to & \mathsf{o} & (C) & \mathsf{twice}(F) & \to & \lambda y.\, F \cdot (F \cdot y) \\ (B) & \mathsf{I}(\mathsf{s}(n)) & \to & \mathsf{s}(\mathsf{twice}(\lambda x.\, \mathsf{I}(x)) \cdot n) & (D) & \mathsf{twice}(F) \cdot m & \to & F \cdot (F \cdot m) \end{array}$$

In this context, let $l = \mathsf{s}(n)$. Then

$$\begin{array}{lllllll} FS_0(l) & = & \{\langle \mathsf{s}, \mathsf{nat}\rangle\} & \quad & FS_1(l) & = & \{\langle \mathsf{s}, \mathsf{nat}\rangle, \langle \mathsf{twice}, \mathsf{nat}\rangle, \langle \mathsf{I}, \mathsf{nat}\rangle\} \\ FR_0(l) & = & \{(B),\ (D)\} & & FR_1(l) & = & \{(B),\ (D)\} = FR_0(l) \end{array}$$

We have $FR(\mathsf{I}^{\#}(\mathsf{s}(n)) \rightsquigarrow p) = FR(\mathsf{s}(n)) = \{(B),\ (D)\}$ for any p. Note that for a dependency pair with left-hand side $\mathsf{twice}(F) \cdot n$ or $\mathsf{twice}^{\#}(F)$ the set of formative rules is empty (since $Symb(F) = Symb(n) = \emptyset$). Therefore, the formative rules of the SCC $\mathcal{C}_{\mathsf{twice}}$ are (B) and (D).

Using formative rules Formative rules are constructed in such a way that to reduce to a term of the form $l\gamma$ we only need its formative rules:

▶ **Lemma 7.4.** *If s is terminating and $s \to^*_{\mathcal{R}} l\gamma$, then there exists a substitution δ on the same domain as γ such that each $\delta(x) \to^*_{\mathcal{R}} \gamma(x)$ and $s \to^*_{FR(l)} l\delta$.*

Proof Sketch. We assume l is simple and not a variable (otherwise this is trivial). Transform the reduction $s \to^*_{\mathcal{R}} l\gamma$ into a reduction without any headmost steps with a rule $l' \to \lambda x.\, r'$ (this is possible because the rules have been completed). Then perform induction on s first, using $\to_{\mathcal{R}} \cup \rhd$, the length of the reduction second. If s is headed by a beta-redex we can start with a β-step because $l\gamma$ is not (and complete with IH1), if s reduces to $l\gamma$ without any headmost steps we use the $\rhd$ part of IH1 (variable capture is not an issue because γ can be assumed to have empty domain if l is an abstraction) and if $s \to^*_{\mathcal{R}} l'\gamma' \cdot t_1 \cdots t_n \to_{\mathcal{R}} r'\gamma' \cdot t_1 \cdots t_n \to^*_{\mathcal{R}'} l\gamma$ with either r' headed by a variable or the latter part not using any headmost steps, then $l'' := l' \cdot x_1 \cdots x_n \to r' \cdot x_1 \cdots x_n =: r''$ is a formative rule of l and can be assumed simple, so we use the second induction hypothesis to get $s \to^*_{FR(l'')} l''\delta' \to_{\mathcal{R}} r''\delta'$ and the first induction hypothesis to have $r''\delta' \to^*_{FR(l)} l\delta$; this suffices because $FR(l'') \subseteq FR(l)$. ◀

With this we can strengthen the definition of dependency chains, and adapt Theorem 4.5:

▶ **Lemma 7.5.** *If $\mathcal{R}$ is non-terminating, there is an infinite dependency chain over $\mathsf{DP}(\mathcal{R})$ such that for all i: $t_i \to^*_{in} s_{i+1}$ using only rules from $FR(l_{i+1})$.*

Thus, we can restrict attention to dependency chains using only formative rules, and adapt the definition of *looping* and the results of Sections 5 and 6 accordingly. We obtain:

▶ **Theorem 7.6** (Complete Result)**.** *A set of dependency pairs $D = D_1 \uplus D_2$ is non-looping if D_2 is non-looping and there is a reduction triple such that:*

1. *$l > p$ for $l \rightsquigarrow p \in D_1$,*
2. *$l \geq p$ for $l \rightsquigarrow p \in D_2$,*
3. *$l \geq_1 r$ for $l \to r \in FR(D)$,*
4. *If D is collapsing, then $\geq$ additionally satisfies the limited subterm property.*

Also $\emptyset$ is non-looping. An AFS with rules $\mathcal{R}$ and dependency graph approximation G is terminating if all cycles in G are non-looping, which holds if all SCCs are non-looping.

In requirement (3) in Theorem 6.7 we can also restrict attention to the formative rules of D instead of considering all rules. It remains to find a suitable reduction triple or pair.

8 Monotone Algebras

A semantical method to prove termination of rewriting is to interpret terms in a well-founded algebra, and show that whenever $s \to t$ their interpretations decrease: $[\![s]\!] > [\![t]\!]$. For TRSs, such an algebra is called a termination model if $[\![l]\!] > [\![r]\!]$ for all rules $l \to r$ and some additional properties guarantee that this implies $[\![C[l\gamma]]\!] > [\![C[r\gamma]]\!]$ for all contexts C and substitutions γ. A TRS is terminating if and only if it has a termination model [9, 26]. Van de Pol [21] generalizes this approach to HRSs, with higher order rewriting modulo $\alpha\beta\eta$, and shows that a HRS is terminating if it has a termination model; the converse does not hold.

Here we consider interpretations of AFS terms in a monotone algebra, and use the orderings to solve dependency pair constraints. Since $>$ does not have to be monotonic when using dependency pairs, the theory of [21] can be significantly simplified. We interpret all base types with the same algebra to avoid problems with comparing differently-typed terms.

▶ **Definition 8.1** (Weakly Monotonic Functionals)**.** Let $\mathcal{A}$ be an algebra with a well-founded partial order $>$ and minimum element 0. We assume there is a binary operator $\vee$ on $\mathcal{A}$ such that $x \vee y \geq x, y$ for all $x, y \in \mathcal{A}$ and $x \vee 0 = x$. Terms will be interpreted by elements of, and weakly monotonic functionals over, $\mathcal{A}$. Intuitively, a functional f is weakly monotonic if $f(x) \geq f(y)$ whenever $x \geq y$; however, f only needs to be defined on weakly monotonic input. We inductively define the weakly monotonic functionals for all types, and relations $\sqsupset_{wm}$ and $\sqsupseteq_{wm}$ on these functionals:

- the interpretation for base types: $\mathcal{WM}_\iota = \mathcal{A}$ for all $\iota \in \mathcal{B}$,
- the orderings on $\mathcal{WM}_\iota$ (with $\iota \in \mathcal{B}$): $\sqsupset_{wm}$ equals $>$, and $\sqsupseteq_{wm}$ is its reflexive closure,
- the interpretation for functional types: $\mathcal{WM}_{\sigma\Rightarrow\tau}$ consists of the functions mapping elements of $\mathcal{WM}_\sigma$ to elements of $\mathcal{WM}_\tau$, such that $\sqsupseteq_{wm}$ is preserved (that is, if $x \sqsupseteq_{wm} y$ in $\mathcal{WM}_\sigma$ then $f(x) \sqsupseteq_{wm} f(y)$ in $\mathcal{WM}_\tau$),
- the orderings on $\mathcal{WM}_{\sigma\Rightarrow\tau}$: we have $f \sqsupset_{wm} g$ iff $f(x) \sqsupset_{wm} g(x)$ for all $x \in \mathcal{WM}_\sigma$, and $f \sqsupseteq_{wm} g$ iff $f(x) \sqsupseteq_{wm} g(x)$ for all $x \in \mathcal{WM}_\sigma$.

$\sqsupset_{wm}$ and $\sqsupseteq_{wm}$ are an order and quasi-order respectively, and strongly compatible. If either $x \sqsupset_{wm} y$ or $x = y$ then $x \sqsupseteq_{wm} y$, but the converse implication does not hold.

Constant functions are weakly monotonic functionals: for $n \in \mathcal{A}$ and $\sigma = \tau_1 \Rightarrow \ldots \Rightarrow \tau_k \Rightarrow \iota$ (note that ι always refers to a base type), let $n_\sigma = \boldsymbol{\lambda} x_1 \ldots x_k.n$ (the function in $\mathcal{WM}_\sigma$ taking k arguments and returning n). The function $\boldsymbol{\lambda} f.f(\vec{0})$ is also in $\mathcal{WM}_{\sigma\Rightarrow\iota}$, where $f(\vec{0})$ is short for $f(0_{\tau_1}, \ldots, 0_{\tau_k})$. A weakly monotonic functional not defined in [21], but which will be needed to deal with term application, is *max*:

$$\begin{array}{rcll} \max_\iota(x,y) & = & x \vee y & (\text{for } x, y \in \mathcal{A}) \\ \max_{\sigma\Rightarrow\tau}(f,y) & = & \boldsymbol{\lambda} x.\, \max_\tau(f(x), y) & (\text{for } f \in \mathcal{WM}_{\sigma\Rightarrow\tau}, y \in \mathcal{A}) \end{array}$$

Using induction on the type of the first argument, it is easy to see that $\max_\sigma \in \mathcal{WM}_{\sigma\Rightarrow\iota\Rightarrow\sigma}$.

Term Interpretation. Using an interpretation $\mathcal{J}$ of function symbols, van de Pol associates to each closed term a weakly monotonic functional. Although the definition in [21] considers terms modulo $\alpha\beta\eta$, this is not a significant blockade because we can handle application as a function symbol. The following is our own adaptation of the translation in [21]:

▶ **Definition 8.2.** For all function symbols $f : (\sigma_1 \times \ldots \times \sigma_n) \Rightarrow \tau$ let $\mathcal{J}_f \in \mathcal{WM}_\sigma$, where σ is $\sigma_1 \Rightarrow \ldots \sigma_n \Rightarrow \tau$. A valuation is a function α with a finite domain of variables, such that $\alpha(x) \in \mathcal{WM}_\sigma$ for $x : \sigma$ in its domain. For any AFS-term s and valuation α whose domain contains all $x \in FV(s)$, let $[\![s]\!]_{\mathcal{J},\alpha}$ be the weakly monotonic functional defined as follows:

$$\begin{array}{lcl}
[\![x]\!]_{\mathcal{J},\alpha} & = & \alpha(x) \text{ if } x \in \mathcal{V} \\
[\![f(s_1,\ldots,s_n)]\!]_{\mathcal{J},\alpha} & = & \mathcal{J}_f([\![s_1]\!],\ldots,[\![s_n]\!]) \\
[\![\lambda x.\, s]\!]_{\mathcal{J},\alpha} & = & \boldsymbol{\lambda} n.[\![s]\!]_{\mathcal{J},\alpha\cup\{x\mapsto n\}} \text{ if } x \notin \mathsf{dom}(\alpha) \\
[\![s \cdot t]\!]_{\mathcal{J},\alpha} & = & \max([\![s]\!]_{\mathcal{J},\alpha}([\![t]\!]_{\mathcal{J},\alpha}), [\![t]\!]_{\mathcal{J},\alpha}(\vec{0}))
\end{array}$$

▶ **Example 8.3.** In our running example, consider an interpretation into the natural numbers Say $\mathcal{J}_{\mathsf{I}} = \boldsymbol{\lambda} n.n$ and $\mathcal{J}_{\mathsf{s}} = \boldsymbol{\lambda} n.n+1$. Then $[\![\mathsf{I}(\mathsf{s}(x))]\!]_{\mathcal{J},\alpha} = \alpha(x)+1$.

Reduction Pair Since this definition uses weak rather than strict monotonicity it cannot be used directly like in first order rewriting: $[\![l]\!] \sqsupset_{wm} [\![r]\!]$ does not in general imply $[\![C[l\gamma]]\!] \sqsupset_{wm} [\![C[r\gamma]]\!]$. This issue (which van de Pol works around by defininining an additional relation) disappears in the context of dependency pairs. Using Theorem 6.7 we obtain a number of requirements $[\![l]\!] \sqsupset_{wm} [\![r]\!]$ or $[\![l]\!] \sqsupseteq_{wm} [\![r]\!]$, and additionally, for collapsing D, the subterm and marking properties must be satisfied. The latter is a simple restriction, the former holds if the value of a function is always greater than or equal to the value of its arguments.

▶ **Theorem 8.4.** *Let $\mathcal{J}$ be a symbol interpretation such that:*

- *$\mathcal{J}_f \sqsupseteq_{wm} \mathcal{J}_{f^\#}$ for all $f \in \mathcal{D}$*
- *$\mathcal{J}$ maps each c_x to the appropriate 0_σ*
- *for all $f : (\sigma_1 \times \ldots \times \sigma_n) \Rightarrow \tau_1 \Rightarrow \ldots \Rightarrow \tau_m \Rightarrow \iota \in \mathcal{F}$, all $1 \leq i \leq n$ and all $n \in \mathcal{WM}_{\sigma_i}$: $\mathcal{J}_f(0_{\sigma_1},\ldots,n,\ldots,0_{\sigma_n},0_{\tau_1},\ldots,0_{\tau_m}) \sqsupseteq_{wm} n(\vec{0})$.*

Define $s \succ t$ if $[\![s]\!]_{\mathcal{J},\alpha} \sqsupset_{wm} [\![t]\!]_{\mathcal{J},\alpha}$ for all valuations α and $s \succeq t$ if $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![t]\!]_{\mathcal{J},\alpha}$ for all valuations α. Then $(\succ,\succeq)$ is a reduction pair which satisfies the subterm and marking properties from Theorem 6.7.

Proof. Compatibility is evident, weak monotonicity holds by a simple case distinction and stability by the substitution Lemma [21, Theorem 3.2.1]. By the interpretation of application also $\to_\beta$ is contained in $\succeq$, and subterm reduction is included by an inductive argument which uses the last two requirements. The marking property is given by the first requirement. ◀

It is not immediately obvious how to use monotone algebras automatically; a lot will depend on the chosen interpretation for the function symbols. Common first order methods, like polynomial or matrix interpretations, are not likely to be succesful in the presence of functional variables. However, it is very likely that higher order parallels exist, such as an interpretation with primitive recursive functions. While a proper study of such methods is beyond the scope of this paper, the example below might give some initial ideas.

▶ **Example 8.5.** Suppose we have to satisfy a requirement $\mathsf{map}(F,\mathsf{cons}(x,y)) \succeq \mathsf{cons}(F \cdot x, \mathsf{map}(F,y))$, where $F : \mathsf{nat} \Rightarrow \mathsf{nat}$. We consider an interpretation in the natural numbers (with standard $>$, 0, and $\vee$ giving the highest of two numbers) using primitive recursive functions. Let $G(f,m,n)$ be the recursive function defined by: $G(f,m,0) = \max(f(m),m)$ and $G(f,m,n+1) = f(n+1, 2G(f,m,n))$. This function is weakly monotonic in each of f, m and n, and moreover $G(f,m,n+k) \geq G(f,m,n)+G(f,m,k)$ for all $n,k > 0$. Also $G(f,m,n) \geq m$, and $G(f,m,n) \geq f(0)$ if f is weakly monotonic. Choose $\mathcal{J}_{\mathsf{cons}} = \boldsymbol{\lambda} nm.n+m+1$ and $\mathcal{J}_{\mathsf{map}} = \boldsymbol{\lambda} fn.G(f,n,n+1)$, and let $\alpha = \{F \mapsto f, x \mapsto n, y \mapsto m\}$ be a valuation. Then:

$$\begin{array}{lcl}
[\![\mathsf{map}(F,\mathsf{cons}(x,y))]\!]_{\mathcal{J},\alpha} & = & G(f,n+m+1,n+m+2) \\
& \sqsupseteq_{wm} & G(f,n+m+1,n+1) + G(f,n+m+1,m+1) \\
& = & f(n+1) + 2G(f,n+m+1,n) + G(f,n+m+1,m+1) \\
& \sqsupseteq_{wm} & f(n) + 2(n+m+1) + G(f,m,m+1) \\
& \sqsupset_{wm} & \max(f(n),n) + G(f,m,m+1) + 1 \\
& = & [\![\mathsf{cons}(F \cdot x, \mathsf{map}(F,y))]\!]_{\mathcal{J},\alpha}
\end{array}$$

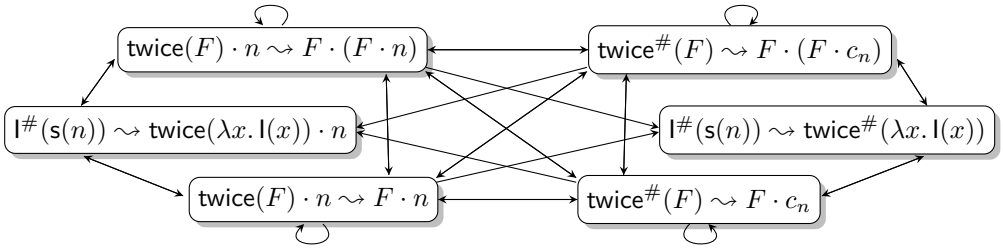
9 Conclusion

A Termination Algorithm The combination of Theorem 7.6 and Section 6.1 provides an algorithm to prove termination of an AFS. First calculate the system's dependency pairs and take an approximation of the (finite) dependency graph. Then:

1. remove all nodes from G which are not on a cycle;
2. if G is empty return `terminating`; otherwise find an SCC $\mathcal{C}$;
3. determine a partition in $\mathcal{C} = \mathcal{C}_1 \uplus \mathcal{C}_2$ and find a reduction pair $(\succ, \succeq)$ such that $\overline{l} \succ \overline{p}$ for $l \rightsquigarrow p \in \mathcal{C}_1$, $\overline{l} \succeq \overline{p}$ for $l \rightsquigarrow p \in \mathcal{C}_2$, $l \succeq r$ for $l \to r \in FR(\mathcal{C})$ and either $\mathcal{C}$ is non-collapsing, or $\succ \cup \succeq$ contains $\rhd^!$ and $f(\vec{x}) \succeq f^\#(\vec{x})$ for all $f \in \mathcal{D}$; if this step fails, return `fail`;
4. remove all pairs in $\mathcal{C}_1$ from the graph, since any cycle $\mathcal{C}'$ which includes such a pair is a subcycle of $\mathcal{C}$ and thus also proved non-looping by $(\succ, \succeq)$; continue with (1).

The algorithm iterates over a graph approximation, simplifying SCCs until none remain; note that this moves in the direction of the dependency pair framework as defined in [6].

▶ **Example 9.1.** Consider our running example twice, whose dependency graph was shown in Example 5.1. As instructed in step (1) of the algorithm, we remove nodes not on a cycle.



In step (2) we choose the SCC of all pairs in the graph; its formative rules are calculated in Example 7.3. For step (3) let $\mathcal{C}_1 := \{\mathsf{I}^\#(\mathsf{s}(n)) \rightsquigarrow \mathsf{twice}(\lambda x.\mathsf{I}(x)) \cdot n, \mathsf{I}^\#(\mathsf{s}(n)) \rightsquigarrow \mathsf{twice}^\#(\lambda x.\mathsf{I}(x))\}$ and $\mathcal{C}_2$ the set containing the other pairs. We have the following proof obligations:

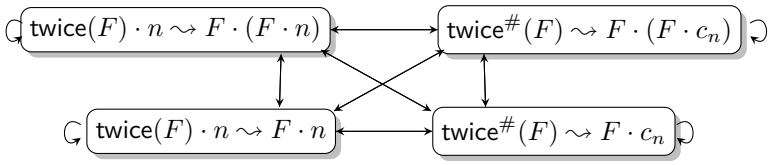
A.	$\mathsf{I}^\#(\mathsf{s}(n))$	$\succ$	$\mathsf{twice}(\lambda x.\mathsf{I}(x)) \cdot n$	*E.*	$\mathsf{twice}^\#(F) \cdot x$	$\succeq$	$F \cdot (F \cdot c_y)$
B.	$\mathsf{I}^\#(\mathsf{s}(n))$	$\succ$	$\mathsf{twice}^\#(\lambda x.\mathsf{I}(x)) \cdot c_z$	*F.*	$\mathsf{twice}^\#(F) \cdot x$	$\succeq$	$F \cdot c_y$
C.	$\mathsf{twice}(F) \cdot m$	$\succeq$	$F \cdot (F \cdot m)$	*G.*	$\mathsf{I}(\mathsf{s}(n))$	$\succeq$	$\mathsf{s}(\mathsf{twice}(\lambda x.\mathsf{I}(x)) \cdot n)$
D.	$\mathsf{twice}(F) \cdot m$	$\succeq$	$F \cdot m$	*H.*	$\mathsf{twice}(F) \cdot m$	$\succeq$	$F \cdot (F \cdot m)$

Requirement (H) is a duplicate of (C). Using an interpretation in functionals over the natural numbers where each $\mathcal{J}_{c_x} = 0$, and assuming $\mathcal{J}_{\mathsf{twice}} = \mathcal{J}_{\mathsf{twice}^\#}$, (B) is implied by (A), and (E) by (C), and (F) by (D). The remaining requirements are satisfied with $\mathcal{J}_{\mathsf{I}^\#} = \mathcal{J}_{\mathsf{I}} = \boldsymbol{\lambda} n.n$ and $\mathcal{J}_{\mathsf{s}} = \boldsymbol{\lambda} n.n + 1$ and $\mathcal{J}_{\mathsf{twice}^\#} = \mathcal{J}_{\mathsf{twice}} = \boldsymbol{\lambda} f.\boldsymbol{\lambda} n.f(f(n))$:

A.	$n + 1$	$>$	$\max((\boldsymbol{\lambda} n.n)((\boldsymbol{\lambda} n.n)n), n) = \max(n, n) = n$
C.	$\max(F(F(n)), n)$	$\geq$	$\max(F(\max(F(n), n)), \max(F(n), n))$
D.	$\max(F(F(n)), n)$	$\geq$	$\max(F(n), n)$
G.	$n + 1$	$\geq$	$\max(n, n) + 1 = n + 1$

The calculations for (A) and (G) are obvious. With some reasoning (distinguishing the cases $n > F(n)$, and $F(n) \geq n$ and noting that $F(n) \geq n$ implies $F(F(n)) \geq F(n)$ by weak monotonicity), (C) and (D) also hold.

Thus we move on to step (4) and remove the two nodes in $\mathcal{C}_1$ from the graph:



All nodes are still interconnected, so we continue with the SCC of all pairs. Interestingly, $FR(\mathcal{C}) = \emptyset$. Therefore it suffices to find a reduction pair with the usual properties and:

$$\begin{array}{lcllcl} \mathsf{twice}(F) \cdot n & \succ & F \cdot (F \cdot n) & \mathsf{twice}^{\#}(F) \cdot n & \succ & F \cdot (F \cdot c_y) \\ \mathsf{twice}(F) \cdot n & \succ & F \cdot n & \mathsf{twice}^{\#}(F) \cdot n & \succ & F \cdot c_y \end{array}$$

This is satisfied with an algebra interpretation with $\mathcal{J}_{\mathsf{twice}^{\#}} = \mathcal{J}_{\mathsf{twice}} = \boldsymbol{\lambda} fn.\max(f(f(n)), n)+1$. Thus we remove the final four nodes from the graph, and conclude that twice is terminating.

Summary and Future Work We have defined a first basic dependency pair method for AFSs, with a variation of usable rules which takes into account the possible presence of collapsing dependency pairs. We have explained that besides orderings such as HORPO also monotone algebras can be used to solve the ordering constraints.

We intend to further study dependency pairs for AFSs with restrictions. For example, if function symbols have a base output type we can drop requirements, yielding an easier method. If we restrict to rules without abstractions in the left-hand sides, we may weaken the subterm property to obtain a stronger method, and define for instance argument filterings (in the extended abstract [15] a first step in this direction is given for HRSs).

A preliminary version of the dependency pair method with argument filterings is implemented in the tool WANDA v1.0 [14]. We intend to improve the implementation by taking into account also the dependency graph, strongly connected components and formative rules.

This work aims to contribute to the larger goal of understanding dependency pairs for higher order rewriting, and creating tools to automatically prove termination in this setting.

Acknowledgments. We are very grateful for the constructive comments of the referees that helped to improve the paper. We also gratefully acknowledge remarks from Jan Willem Klop.

References

1 T. Aoto and Y. Yamada. Dependency pairs for simply typed term rewriting. In J. Giesl, editor, *Proceedings of RTA 2005*, volume 3467 of *LNCS*, pages 120–134, Nara, Japan, April 2005. Springer.

2 T. Aoto and Y. Yamada. Argument filterings and usable rules for simply typed dependency pairs. In S. Ghilardi and R. Sebastiani, editors, *Proceedings of FroCoS 2009*, volume 5749 of *LNAI*, pages 117–132, Trento, Italy, September 2009. Springer.

3 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

4 F. Blanqui. Higher-order dependency pairs. In *Proceedings of WST 2006*, Seattle, USA, August 2006.

5 F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: The end of a quest. In *Lecture Notes in Computer Science (CSL '08)*, volume 5213 of *LNCS*, pages 1–14, Bertinoro, Italy, July 2008. Springer.

6 J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proceedings of LPAR 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331. Springer, 2005.

7 N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 205(4):474–511, 2007.

8 Nao Hirokawa, Aart Middeldorp, and Harald Zankl. Uncurrying for termination. In *Proceedings of the 15th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning*, volume 5330 of *Lecture Notes in Artificial Intelligence*, pages 667–681, Doha, 2008. Springer-Verlag.

9 G. Huet and D.C. Oppen. Equations and rewrite rules: a survey. In R.V Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, London, 1980.

10 J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS'91)*, pages 350–361, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

11 J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th annual IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 402–411, Trento, Italy, July 1999.

12 Z. Khasidashvili. Expression Reduction Systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220, Tblisi, Georgia, 1990.

13 J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, The Netherlands, 1980. PhD Thesis.

14 C. Kop. Wanda. `http://www.few.vu.nl/~kop/code.html`.

15 C. Kop and F. van Raamsdonk. Higher-order dependency pairs with argument filterings. In *Proceedings of WST 2010*, Edinburgh, UK, July 2010. `http://www.few.vu.nl/~kop/wst10.pdf`.

16 K. Kusakari. On proving termination of term rewriting systems with higher-order variables. *IPSJ Transactions on Programming*, 42(SIG 7 PRO11):35–45, 2001.

17 K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, 92(10):2007–2015, 2009.

18 K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting. *AAECC*, 18(5):407–431, 2007.

19 K. Kusakari and M. Sakai. Static dependency pair method for simply-typed term rewriting and related techniques. *IEICE Transactions*, 2(92-D):235–247, 2009.

20 T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, Amsterdam, The Netherlands, July 1991.

21 J.C. van de Pol. *Termination of Higher-order Rerwite Systems*. PhD thesis, University of Utrecht, 1996.

22 M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E88-D(3):583–593, 2005.

23 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.

24 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. To appear.

25 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

26 H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.

A Appendix: complete proofs

A.1 Some Lemmas about Variable Replacement

To start this appendix, we consider subterm reduction and replacing variables with symbols c_x. First of all we note again that α-substitution does not affect the symbols c_x. Thus, the candidate terms of any given term are *not* preserved under α-substitution. However, this is not a problem, because we only require a single representative for any candidate.

In this section we will discuss some Lemmas which will primarily be used for Theorem 4.6.

▶ **Definition 1.1.** A substitution χ is *a variable replacement* if $\chi(x)$ is in $\mathbb{C}$ for all $x \in \mathsf{dom}(\chi)$.

▶ **Definition 1.2.** Let $\overline{\rhd}$ be the relation such that $s \overline{\rhd} t$ if there are a term u and variable replacement χ such that $s \rhd u$ and $t = u\chi$, and $\mathsf{dom}(\chi) \subseteq FV(u) \setminus FV(s)$. Let $\overline{\trianglerighteq}$ be its transitive closure.

▶ **Lemma 1.3.** *Let s be a term, l a linear term not containing any symbols c_x, γ a substitution whose domain contains only variables in $FV(l)$ and χ a variable replacement. Suppose $s\chi = l\gamma$. Then there is a substitution δ on the same domain as γ such that $s = l\delta$ and each $\gamma(x) = \chi(\delta(x))$ for all x.*

Proof. Induction over the form of l. If l is a variable in the domain of γ, then $\gamma(l) = s\chi$; we can choose $\delta(l) = s$. This suffices because $\mathsf{dom}(\gamma) = \{l\}$. If l is a variable not in the domain of γ, then γ is the empty substitution, and $s\chi = s = l$ (since $l = l\gamma$ is not a constructor c_x). We choose δ the empty substitution as well. If $l = \lambda x.\, l'$ then s cannot be a variable (since c_y is not an abstraction), so $s = \lambda x.\, s'$. We can assume x is fresh, so the induction hypothesis provides δ such that $\gamma = \chi \circ \delta$ (and therefore x does not occur in either domain or range of δ) and $s' = l'\delta$; then also $s = l\delta$.

If $l = l_1 \cdot l_2$ then let $\gamma_i = \gamma_{\restriction FV(l_i)}$. By linearity of l each γ_i has disjunct domain, and they contain only variables occurring in $FV(l_i)$. We have $s\chi = (l_1\gamma) \cdot (l_2\gamma)$, so s can only have the form $s_1 \cdot s_2$, with $s_i\chi = l_i\gamma$ (note, after all, that if s were a variable, then $s\chi$ would either be a variable or a constructor symbol, not an application). The induction hypothesis supplies δ_1, δ_2 on the same domains as γ_1, γ_2 such that $s_i = l_i\delta_i$ and $\gamma_i(x) = \chi(\delta_i(x))$ for x in the respective domains. Due to disjunct domains we can safely define $\delta := \delta_1 \cup \delta_2$. Finally, if $l = f(l_1, \ldots, l_n)$ we again use left-linearity to split γ into $\gamma_1, \ldots, \gamma_n$ and inductively find $\delta_1, \ldots, \delta_n$; choosing $\delta := \delta_1 \cup \ldots \cup \delta_n$ suffices. ◀

▶ **Lemma 1.4.** *If $\mathcal{R}$ is left-linear, s, t terms and χ a variable replacement, and $s\chi \to_{\mathcal{R}} t$, then there exists a term t' such that $s \to_{\mathcal{R}} t'$ and $t = t'\chi$.*

Proof. By induction on the derivation of $s\chi \to_{\mathcal{R}} t$. If $s\chi \to_{\mathcal{R}} t$ by a topmost step, so $s\chi = l\gamma$ and $t = r\gamma$, then according to Lemma 1.3 we can find δ, t' such that $s = l\delta$ and $\chi(\delta(x)) = \gamma(x)$ for $x \in \mathsf{dom}(\gamma)$. We can safely assume that $\mathsf{dom}(\delta) = FV(l)$, and therefore $s = l\delta\chi$ and $t = r\delta\chi$; defining $t' := r\delta$ we have satisfied the requirements.

Next, consider the inductive cases; since a term c_x does not have any strict subterms, $s\chi$ has the same form (application, abstraction or function application) as s. Each of the cases is immediate with the induction hypothesis (and demonstrated hereafter). If $s = s_1 \cdot s_2$ and $t = t_1 \cdot (s_2\chi)$ with $s_1\chi \to_{\mathcal{R}} t_1$, then the induction hypothesis provides t_1' such that $s_1 \to_{\mathcal{R}} t_1'$ and $t_1'\chi = t_1$. Let $t' := t_1' \cdot s_2$. Then $s \to_{\mathcal{R}} t'$ and $t'\chi = t_1'\chi \cdot s_2\chi = (t_1 \cdot s_2)\chi = t$. The case when $s = s_1 \cdot s_2$ and $t = s_1\chi \cdot t_2$ with $s_2\chi \to_{\mathcal{R}} t_2$ is exactly the same. If $s = f(s_1, \ldots, s_n)$ and $t = f(s_1\chi, \ldots, t_i, \ldots, s_n\chi)$ and $s_i\chi \to_{\mathcal{R}} t_i$, the induction hypothesis provides t_i' such that

$s_i \to_{\mathcal{R}} t_i'$ and $t_i'\chi = t_i$. We are done defining $t' := f(s_1, \ldots, t_i', \ldots, t_n)$. Finally, if $s = \lambda x.\, s_0$ and $t = \lambda x.\, t_0$ then the induction hypothesis provides t_0' and we can define $t' = \lambda x.\, t_0'$. ◀

▶ **Lemma 1.5.** *If $s\chi \trianglerighteq t$, then there exists t' such that $s \trianglerighteq t'$ and $t'\chi = t$ (for χ a variable replacement).*

Proof. With induction over the size of s. First suppose $s\chi = t$; then we are done choosing $t' := s$. Otherwise, $s\chi = C[t]$ for some non-empty context C; s cannot be a variable since neither variables nor symbols c_x have subterms. Thus, s is a functional term, an application, or an abstraction. If $s = f(s_1, \ldots, s_i, \ldots, s_n)$ and $s_i\chi = D[t]$, then $s_i\chi \trianglerighteq t$, so the induction hypothesis supplies t' with $s \rhd D[t] \trianglerighteq t'$ and $t'\chi = t$. If $s = s_1 \cdot s_2$ then we complete similarly. If $s = \lambda x.\, s'$ and $s'\chi = D[t]$, then still the induction hypothesis supplies suitable t'. ◀

▶ **Lemma 1.6.** *Assume $\mathcal{R}$ is left-linear. If there is an infinite $\to_{\mathcal{R}} \cdot \overline{\trianglerighteq}$ reduction starting in s, then there is an infinite $\to_{\mathcal{R}}$ reduction starting in s.*

Proof. Reasoning from contradiction, let s be a term from which no infinite $\to_{\mathcal{R}}$ reduction originates; we must prove there is no infinite $\to_{\mathcal{R}} \cdot \overline{\trianglerighteq}$ reduction starting in s. We can perform induction on s by $\to_{\mathcal{R}}$, so any $\to_{\mathcal{R}}$-reduct t of s can be assumed not to start an infinite $\to_{\mathcal{R}} \cdot \overline{\rhd}$ reduction. Towards a contradiction, assume there is an infinite sequence $s \to_{\mathcal{R}} t\,\overline{\trianglerighteq}\,u \to_{\mathcal{R}} v\,\overline{\trianglerighteq}\,w \ldots$. Expanding the definition of $\overline{\trianglerighteq}$ we find $C[], u', \chi$ such that $t = C[u']$ and $u = u'\chi$, with $\mathsf{dom}(\chi) \subseteq FV(u') \setminus FV(t)$. By Lemma 1.4 there is v' such that $u' \to_{\mathcal{R}} v'$ and $v = v'\chi$. Since $v\,\overline{\trianglerighteq}\,w$ we can write $v = \trianglerighteq w'$ for some w' with $w = w'\epsilon$, with ϵ a variable replacement such that $\mathsf{dom}(\epsilon) \subseteq FV(w') \setminus FV(v)$. But according to Lemma 1.5 there is a term w'' such that $v' = D[w'']$ and $w' = w''\chi$. Define δ as the restriction of $\chi \cup \epsilon$ to $FV(w'')$. Then $w = w'\epsilon = w''\delta$. Looking back to where we started, we have $s \to_{\mathcal{R}} t = C[u'] \to_{\mathcal{R}} C[v'] = C[D[w'']]$. Since $\mathsf{dom}(\delta)$ contains only variables in $\mathsf{dom}(\chi) \cup \mathsf{dom}(\epsilon)$, it contains no variables in $FV(C[v'])$, which after all is a subset of $FV(s)$. Thus, $C[D[w'']]\,\overline{\trianglerighteq}\,w$. Consequence, t *does* start an infinite $\to_{\mathcal{R}} \cdot \overline{\trianglerighteq}$ reduction, contradicting our assumption! ◀

▶ **Lemma 1.7.** *If $r \rhd p'$ and $p'\chi = p$, for χ a variable replacement substitution whose domain contains only variables in $FV(p') \setminus FV(r)$, then $r\gamma\,\overline{\rhd}\,p\gamma$ for any substitution γ.*

Proof. Write $r = C[p']$ and assume the variables bound in r are fresh (so do not occur in either domain or range of γ). By definition of substitution $r\gamma = C\gamma[p'\gamma]$, so $r\gamma\,\overline{\rhd}\,p'\gamma\chi$. Since the range of χ does not contain any variables at all, and because the range of γ does not contain variables in $\mathsf{dom}(\gamma)$, this equals $p'\delta\gamma = p\gamma$ as required. ◀

A.2 Proofs for Section 4

Proof of Theorem 4.5. We must see that if $\mathcal{R}$ is non-terminating, then there is an infinite dependency chain over $\mathsf{DP}(\mathcal{R})$. Given any non-terminating term, let u_{-1} be a minimal subterm that is still non-terminating (u_{-1} is *MNT*). Now, for any $i \in \mathbb{N}$, let u_i be a given MNT term, and consider an infinite reduction starting in u_i. It cannot be that u_i is an abstraction, since abstractions can only be reduced by reducing their immediate subterm, contradicting minimality of u_i. For the same reason u_i cannot have the form $x \cdot v_1 \cdots v_n$ with x a variable, or the form $f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$ with f a constructor symbol. What remains are the forms $u_i = (\lambda x.\, s) \cdot t \cdot v_1 \cdots v_n$ or $u_i = f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$ with $f \in \mathcal{D}$.

In the first case, note that by minimality of u_i eventually a headmost step must be taken; therefore, any infinite reduction starting in u_i has the form $u_i \to_{\mathcal{R}}^* (\lambda x.\, s') \cdot t' \cdot v_1' \cdots v_n' \to_\beta$

$s'[x := t'] \cdot v'_1 \cdots v'_n \to_{\mathcal{R}} \ldots$. Since also $s[x := t] \cdot v_1 \cdots v_n \to^*_{\mathcal{R}} s'[x := t'] \cdot v'_1 \cdots v'_n$ the immediate beta-reduct of u_i is also non-terminating. If $n > 0$, this reduct is minimal: any strict subterm of $s[x := t] \cdot v_1 \cdots v_n$ is either a subterm of some v_j (and therefore also of u_i) or a subterm of $s[x := t] \cdot v_1 \cdots v_j$ with $j < n$ (and therefore a subterm of a reduct of $(\lambda x.\, s) \cdot t \cdot v_1 \cdots v_j$, which is a subterm of u_i)). Choose $u_{i+1} := s[x := t] \cdot v_1 \cdots v_n$ and let $\rho_{i+1}, s_{i+1}, t_{i+1} := \mathtt{beta}, u_i, u_{i+1}$. Note that $s_i^\# = s_i$ and $t_i^\# = t_i$. On the other hand, if $n = 0$, let w be a minimal subterm of s_i such that $w[x := t]$ is still non-terminating. Since both w and t are terminating (by minimality of u_i), $w \neq x$, but $FV(w)$ does contain x. If $w = \lambda y.\, w'$ then also $w'[x := t]$ is non-terminating, contradicting minimality of w. Therefore w can only be an application $w_1 \cdot w_2$ or a functional term $f(w_1, \ldots, w_n)$ and we have $(w[x := t])^\# = w^\#[x := t]$. Noting that all $w_i[x := t]$ are terminating by minimality of w, we see that $w[x := t]$ is MNT. Thus, we choose $u_{i+1} := w[x := t]$ and $\rho_{i+1}, s_{i+1}, t_{i+1} := \mathtt{beta}, u_i, u_{i+1}^\#$.

In the second case, $u_i = f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$, only finitely many steps in the v_j can be taken before a head step appears in any infinite reduction, say $u_i \to^*_{in} f(v'_1, \ldots, v'_n) \cdot v'_{n+1} \cdots v'_m = l\gamma \cdot v'_{j+1} \cdots v'_m$ with $n \leq j \leq m$ and $r\gamma \cdot v'_{j+1} \cdots v'_m$ still non-terminating ($l \to r \in \mathcal{R}$). Since the rules were completed, we can always find such a rule that either $m = j$ or r is not an abstraction: if $m > j$ then we know that $r\gamma \cdot v'_{j+1} \cdots v'_m$ is still MNT, so if $r = \lambda x.\, r'$ eventually a headmost $\to_\beta$ step must be taken. Like above, we then see that $r'\gamma[x := v'_{j+1}] \cdot v'_{j+2} \cdots v'_m$ is also non-terminating, so we might instead have used the rule $l \cdot x \to r'$. If r' is still an abstraction and also $m > j + 1$ we can repeat this change, until we eventually (as $m - j$ decreases in every step) find a rule $l \to r$, substitution γ and j such that $u_i \to^*_{in} l\gamma \cdot v'_{j+1} \cdots v'_m \to_{\mathcal{R}} r\gamma \cdot v'_{j+1} \cdots v'_m$ and either r is not an abstraction or $m = j$.

First suppose $m > j$, so as observed above $r\gamma \cdot v'_{j+1} \cdots v'_m$ is still MNT. Then, after a finite number of internal steps it must be possible to reach a head step; therefore $r\gamma$ cannot be headed by a variable or a functional term $g(\vec{w})$ with g a constructor; either $r\gamma$ is headed by an abstraction, or by a functional term $f(\vec{w})$ with $f \in \mathcal{D}$. Since we know that r itself is not an abstraction, this can only be the case if r is (headed by) a variable, or (headed by) a functional term $f(\vec{w})$ with $f \in \mathcal{D}$. Therefore $l \cdot x_{j+1} \cdots x_m \rightsquigarrow r \cdot x_{j+1} \cdots x_m$ is a dependency pair. Define $\rho_{i+1}, s_{i+1}, t_{i+1} := l \cdot \vec{x} \rightsquigarrow r \cdot \vec{x},\ l\gamma \cdot v'_{j+1} \cdots v'_m,\ r\gamma \cdot v'_{j+1} \cdots v'_m$.
Finally, suppose $m = j$, so $u_i \to^*_{in} l\gamma$ and $r\gamma$ is non-terminating. As in the definition of candidate terms, write r with all bound variables different symbols and let r' be a minimal subterm of r such that $p\gamma$ is still non-terminating, where $p = r'\delta$ and δ is a substitution $[x_1 := c_{x_1}, \ldots, x_n := c_{x_n}]$ with $\{x_1, \ldots, x_n\} = FV(r') \setminus FV(r)$ – since r itself is non-terminating such an r' exists. Then r' cannot be a variable free in r, as $\gamma(r')$ would be a strict subterm of l (contradicting minimality of u_i). Also r' cannot be an abstraction $\lambda x.\, r''$, as then $r''\delta\gamma$ would also be non-terminating, and (since $\to_{\mathcal{R}}$ is preserved under substitution) so would $r''\delta[x := c_x]\gamma$ be. By minimality of r' we also see that r' cannot have the form $f(r_1, \ldots, r_n) \cdot r_{n+1} \cdots r_m$ with $f \in \mathcal{C}$ (as then some $r_i\delta\gamma$ would have to be non-terminating), nor can it have the form $x \cdot r_1 \cdots r_n$ with x bound since there is no rule to reduce $c_x \cdot r_1 \cdots r_j$. We see that r' can only be $x \cdot r_0 \cdots r_m$ with x a variable free in r or $f(r_1, \ldots, r_n) \cdot r_{n+1} \cdots r_m$ with f a defined symbol, and therefore $p \in Cand(r)$. In either case, $p\gamma$ is MNT as well. Choose $u_{i+1} := p\gamma$ and define: $\rho_{i+1}, s_{i+1}, t_{i+1} := l^\# \rightsquigarrow p^\#,\ l^\#\gamma,\ p^\#\gamma$. Note that, since l and p are not single variables, $p^\#\gamma$ and $l^\#\gamma$ are exactly $(l\gamma)^\#$ and $(p\gamma)^\#$.

Due to the construction of $[\rho_i, s_i, t_i | i \in \mathbb{N}]$ it is easy to see that this chain forms an infinite dependency chain. We merely need to observe that in every step $t_i = u_i^\#$, and that by the definition of $\to^*_{\mathcal{R},in}$ it immediately follows that $u_i \to^*_{\mathcal{R},in} l\gamma$ implies $u_i^\# \to^*_{\mathcal{R},in} l\gamma^\#$. ◀

Proof of Theorem 4.6. By Theorem 4.5 non-termination of $\mathcal{R}$ implies the existence of an infinite dependency chain, whether $\mathcal{R}$ is left-linear or not. For the other direction, let an

infinite dependency chain $[\rho_i, s_i, t_i | i \in \mathbb{N}]$ be given. Note that for a candidate term p of r we have $r\gamma \mathrel{\overline{\trianglerighteq}} p\gamma$ for any substitution γ by Lemma 1.7, regardless of the choice for the c_x in the definition of candidate term. Also the normal subterm reduction relation $\trianglelefteq$ is included in $\overline{\trianglerighteq}$. Therefore, whether $\rho_i \in \mathsf{DP}$ or $\rho_i = \mathtt{beta}$, $|s_i| \to_{\mathcal{R}} \cdot \overline{\trianglerighteq} |t_i|$ for all $i \in \mathbb{N}$, where $|u|$ is u with function symbols $f^\#$ replaced by just f. When $t_i \to_{\mathcal{R}}^* s_{i+1}$ it is clear that also $|t_i| \to_{\mathcal{R}}^* |s_{i+1}|$. Since every $\to_{\mathcal{R}}$ step is also a $\to_{\mathcal{R}} \cdot \overline{\trianglerighteq}$ step, this procedure generates an infinite $\to_{\mathcal{R}} \cdot \overline{\trianglerighteq}$ reduction, which by Lemma 1.6 implies non-termination of $\to_{\mathcal{R}}$. ◀

A.3 Proofs of Section 5

The following Lemma was stated as fact in the text.

▶ **Lemma 1.8.** *$\emptyset$ is not dangerous.*

Proof of Lemma 1.8. Any infinite dependency chain with all $\rho_i = \mathtt{beta}$ has $t_i = s_{i+1}$ for all i: if s_{i+1} is headed by an abstraction, $t_i \to_{in}^+$ cannot hold. Also, either $s_i \to_\beta t_i$ or $s_i \to_\beta \cdot \rhd |t_i| = t_i$ (since $t_i = s_{i+1}$ is an application, it is equal to its marked version). Therefore we can find contexts $C_1, C_2, \ldots$ such that $s_0 \to_\beta C_1[s_1] \to_\beta C_1[C_2[s_2]]$. But such a chain can not exist, as typed β-reduction is terminating. ◀

The text mentions that we will assume the dependency graph is finite, but that this is not necessary as it may be finitely approximated. In this appendix, we will give the proofs for a potentially infinite graph, but with a finite approximation.

▶ **Definition 1.9** (Revised Dependency Graph Approximation)**.** An approximation of a (possibly infinite) dependency graph is a graph with a finite number of nodes, each representing a set of dependency pairs. There is an edge from node A to node B if the original graph contains an edge from any element of A to any element of B (but more edges than this may be present).

An infinite system often corresponds with a finite polymorphic system, in which case the group splitting is quite natural. Evidently any dependency graph has a finite approximation, for example the graph with a single node representing all nodes in the original graph, and an edge to itself.

Proof of Lemma 5.2. In the infinite definition, a cycle $\mathcal{C}$ in G corresponds with the set $\mathcal{C}' = \{\rho | \rho \in \mathsf{DP}(\mathcal{R})|$ some node in $\mathcal{C}$ represents $\rho\}$. The assumption in the lemma gives that any such set is non-looping.

By Theorem 4.6 $\mathcal{R}$ is terminating if there is no infinite reduction chain. So suppose (towards a contradiction) there is one, say $[(\rho_i, s_i, t_i) | \rho_i \in \mathsf{DP}(\mathcal{R})]$. Since the graph approximation G is finite, there will be some node A such that infinitely many ρ_i are in A; say $\rho_{i_1}, \rho_{i_2}, \ldots$ are all in A. Let $\mathcal{C}$ be the set of nodes corresponding with some ρ_j such that $j > i_1$. Then $\mathcal{C}$ is a cycle in the graph:

- If $\mathcal{C} = \{A\}$, so all ρ_j with $j > i$ are either $\mathtt{beta}$ or in $\mathcal{C}$, then the complete graph has an edge between ρ_{i_1} and ρ_{i_2}, so G has an edge from A to itself. Thus, $\mathcal{C}$ is a cycle.
- If $\mathcal{C}$ contains some $B \neq A$, then there is a path in G from A to B and a path from B to A. This because for any $\rho_j \in B$ we can find k such that $i_k < j < i_{k+1}$ (since $i_1 < i_2 < \ldots$ is infinite and j is not in this sequence). But then there is a path in the dependency graph from ρ_{i_k} to ρ_j and a path from ρ_j to $\rho_{i_{k+1}}$.
  Consequently, there is a path from A to A (via B) and for any two other nodes B, C there is a path from B to C (via A); $\mathcal{C}$ is a cycle.

Thus there is an infinite dependency chain $\{(\rho_j, s_j, t_j) | j > i_1\}$ using only elements of $\mathcal{C}'$, contradicting the assumption that all cycles are non-looping. ◀

A.4 Proofs of Section 6, part 1

Proof of Theorem 6.2. Let $D = D_1 \uplus D_2$ such that D_2 is non-looping, and a reduction triple $(>, \geq, \geq_1)$ exists such that $l > p$ for all $l \rightsquigarrow p \in D_1$, $l \geq p$ for all $l \rightsquigarrow p \in D_2$, $l \geq_1 r$ for all $l \to r \in \mathcal{R}$ and either D is non-collapsing or $\geq$ satisfies the limited subterm property. We must prove that D is non-looping as well. First some observations:

1. If $s \to_{\mathcal{R}} t$ then $s \geq t$: either $s = C[l\gamma]$ and $t = C[r\gamma]$ for some context C, substitution γ and rule $l \to r$, or $s = C[(\lambda x.\, u) \cdot v]$ and $t = C[u[x := v]]$. In the first case, $l \geq_1 r$ by assumption, $l\gamma \geq_1 r\gamma$ by stability of $\geq_1$ and then $s \geq_1 t$ by monotonicity of $\geq_1$; in the second case $s \geq t$ because $\geq_1$ contains `beta` and is monotonous (note that $\geq_1$ is included in $\geq$, so $s \geq_1 t$ implies $s \geq t$).
2. If D is not collapsing, any infinite dependency chain with all $\rho_i \in D \cup \{\texttt{beta}\}$ has only finitely many `beta` steps: by Lemma 1.8 some ρ_i in such a chain is not `beta`, $\rho_i = l \rightsquigarrow p$. Since D is non-collapsing, p has the form $f(p_1, \ldots, p_n) \cdot p_{n+1} \cdots p_m$, and therefore t_i is headed by a functional term. But then $t_i \to_{in}^* s_{i+1}$ and ρ_{i+1} also cannot be `beta`. Using induction we see that all of the following ρ_j are in D.

Now we show that D is non-looping. Towards a contradiction, assume there is an infinite dependency chain with all $\rho_i \in D \cup \{\texttt{beta}\}$. If D is non-collapsing, we can assume all ρ_i are in D, as by observation 2 every such chain has a `beta`-free tail.

Define δ_0 as the empty substitution. For $i \in \mathbb{N}$, consider ρ_i. If $\rho_i = l \rightsquigarrow p \in D_1$, then $s_i\delta_i = l\gamma\delta_i > p\gamma\delta_i$ for some substitution γ because $>$ is stable. If $\rho_i = l \rightsquigarrow p \in D_2$, then $s_i\delta_i = l\gamma\delta_i \geq p\gamma\delta_i$ because $\geq$ is stable. If $\rho_i = \texttt{beta}$, so D is collapsing, there are two possibilities: either $s_i \to_\beta t_i$ by a head-most reduction, or $s_i = (\lambda x.\, u) \cdot v$, $u \trianglerighteq w$ and $t_i = w^\#[x := v]$, with $x \in FV(w)$ and $w \neq x$. In the first case, $s_i\delta_i \geq t_i\delta_i$ because $\geq_1$ contains `beta` and is stable. In the second case, the limited subterm property provides γ such that $s_i \ (> \cup \geq) \ w^\#[x := t]\gamma$. By monotonicity of $> \cup \geq$ then $s_i\delta_i \ (> \cup \geq) \ t_i\gamma\delta_i$.

Now define $\delta_{i+1} := \gamma\delta_i$ in the last case and $\delta_{i+1} := \delta_i$ otherwise. We have seen that always $s_i\delta_i \ (> \cup \geq) \ t_i\delta_{i+1}$, and if $\rho_i \in D_1$ even $s_i\delta_i > t_i\delta_{i+1}$. Since $\to_{\mathcal{R}}^*$ is included in $\geq$ by observation 1 also $t_{i+1}\delta_{i+1} \geq s_{i+1}\delta_{i+1}$. Thus, an infinite dependency chain over D leads to an infinite $>, \geq$ reduction, and if infinitely many dependency pairs in D_1 occur, this gives an infinite $>$ reduction (by compatibility of $>$ and $\geq$), contradicting well-foundedness. If only finitely many dependency pairs in D_1 occur, eventually all $\rho_i \in D_2 \cup \{\texttt{beta}\}$, and consequently D_2 is dangerous, contradicting the assumption. ◀

Proof of Theorem 6.3. Let $(\mathcal{F}, \mathcal{R})$ be a system with left-linear rules and dependency graph approximation G. If the requirements of Theorem 6.2 are satisfied for all cycles in G, then by Theorem 6.2 all cycles in G are non-looping, so by Lemma 5.2 $\mathcal{R}$ is indeed terminating.

For the other direction, suppose $\mathcal{R}$ is terminating; we will define a reduction triple $(>, \geq, \geq_1)$ with the limited subterm property such that $l > p$ for all $l \rightsquigarrow p \in \mathsf{DP}$ and $l \geq_1 r$ for all rules $l \to r$; this same reduction triple can then be used for all cycles in G (in fact, for all subsets of DP). Let $\sqsupset$ be the restriction of the relation $\to_{\mathcal{R}} \cdot \overline{\trianglerighteq}$ where all variables from the right-hand side occur in the left-hand side (that is, where the $\overline{\trianglerighteq}$ step replaces all newly freed variables by terms c_x - $\overline{\trianglerighteq}$ was defined at the start of this appendix). By Lemma 1.6 termination of $\to_{\mathcal{R}}$ implies well-foundedness of $\sqsupset$. Defining $s > t$ iff $|s| \sqsupset^+ |t|$ and $s \geq_1 t$ iff $|s| \to_{\mathcal{R}}^* |t|$ and $\geq$ as $> \cup \geq_1$ (where $|\cdot|$ just removes markings), we definitely have a

well-founded ordering and a quasi-ordering, and it is easy to see that $>$ and $\geq$ are compatible, that all three relations are stable and that $\geq_1$ is monotonic (even though $>$ is not). Clearly `beta` is contained in $\geq_1$ and if $s \trianglerighteq u$ then $(\lambda x.\, s) \cdot t \to_{\mathcal{R}} s[x := t] \mathrel{\overline{\trianglerighteq}} u[x := t][\vec{y} := \vec{c}]$ (where $\{\vec{y}\} = FV(u) \setminus FV(s)$). It is also clear that $l \geq_1 r$ for all $l \to r \in \mathcal{R}$ and also $l \sqsupset p$ (and therefore $l > p$) for $l \rightsquigarrow p \in \mathsf{DP}$. ◀

A.5 Proofs of Section 6.1

Proof of Lemma 6.5. First note (**): if $s > t$, then for any $u_1, \ldots, u_n$ such that $s \cdot \vec{u}$ is of base type, there are $v_1, \ldots, v_m$ such that $s \cdot \vec{u} \succ t \cdot \vec{v}$: we know $s \cdot \vec{x} \succ t \cdot \vec{w}$ for some $\vec{x}, \vec{w}$ with all x_i occurring neither in s or t, so $s \cdot \vec{u} = s \cdot \vec{x}[\vec{x} := \vec{u}] \succ t \cdot \vec{w}[\vec{x} := \vec{u}]$ by stability, which $= t \cdot (\vec{v}[\vec{x} := \vec{u}])$ because all x_i were fresh (not occurring in s or t). Similarly if $s \geq t$, then for any $u_1, \ldots, u_n$ such that $s \cdot \vec{u}$ has base type, there are $v_1, \ldots, v_m$ such that $s \cdot \vec{u}\ (\succeq \cup \succ \cdot \succeq \cup \succeq \cdot \succ)\ t \cdot \vec{v}$ (both $\succ$ and $\succeq$ are stable).

Now consider $>$. Evidently $>$ is transitive: if $s > t > u$ then by (**) $s \cdot \vec{x} \succ t \cdot \vec{v} \succ u \cdot \vec{w}$, and by transitivity of $\succ$ we conclude $s \cdot \vec{x} \succ u \cdot \vec{w}$. Also $>$ is well-founded, since every sequence $s_1 > s_2 > \ldots$ implies the existence of a sequence $s_1 \cdot \vec{x} \succ s_2 \cdot \vec{t} \succ s_3 \cdot \vec{u} \succ \ldots$; therefore $>$ is also non-reflexive. Finally, $>$ is stable, for if $s > t$ and γ is a substitution, then for fresh $\vec{x}$ there are terms $\vec{u}$ such that $s \cdot \vec{x} \succ t \cdot \vec{u}$; as $\succ$ is closed under substitution also $s \cdot \vec{y} \succ t \cdot \vec{u}[\vec{x} := \vec{y}]$, where $\vec{y}$ are variables which occur neither in s, t or anywhere in the domain or range of γ. Then $s\gamma \cdot \vec{y} = (s \cdot \vec{y})\gamma \succ (t \cdot \vec{u}[\vec{x} := \vec{y}])\gamma = t\gamma \cdot (\vec{u}[\vec{x} := \vec{y}]\gamma)$ as required.

Now considering $\geq_1$, first note that if $s \succeq t$ then by monotonicity of $\succeq$ also $s \cdot \vec{x} \succeq t \cdot \vec{x}$, so $\geq_1$ is indeed a subrelation of $\geq$. Furthermore, $\geq_1$ inherits reflexivity, transitivity, stability and monotonicity from $\succeq$, as well as the property that it contains `beta`.

For compatibility of $>$ and $\geq$, note that either $\succ \cdot \succeq$ is in $\succ$ or $\succeq \cdot \succ$ is in $\succ$. Assume the former. Then $s > t \geq u$ implies that $s \cdot \vec{x} \succ t \cdot \vec{v}\ R\ u \cdot \vec{w}$, where R is one of $\succ$, $\succeq$ or $\succeq \cdot \succ$. In the first case we use transitivity of $\succ$, in the second the assumption, in the third both the assumption and transitivity of $\succ$. If $\succeq \cdot \succ$ is included in $\succ$ the reasoning is symmetric.

Finally, stability of $\geq$ follows from (**) in the same way as with $>$. For transitivity, suppose $s \geq t \geq u$. This implies $s \cdot \vec{x} R_1 t \cdot \vec{v} R_2 u \cdot \vec{w}$, where both R_1 and R_2 are either $\succeq$, $\succ \cdot \succeq$ or $\succeq \cdot \succ$. We must see that $R_1 \cdot R_2$ is included in $\succ \cup \succ \cdot \succeq \cup \succeq \cdot \succ$ for each of the nine combinations. But this can easily be seen to hold by compatibility and transitivity of the two relations involved. ◀

Towards proving some statements in the text, we study $\rhd^!$. The following Lemma is a useful property of the definition of $\rhd^!$.

▶ **Lemma 1.10.** *$\rhd^!$ is stable.*

Proof. With a straightforward induction over the definition of $\rhd^!$; evidently if $s = t$ also $s\gamma = t\gamma$, so we can assume the induction hypothesis holds for $\trianglerighteq^!$ as well. Consider the clause used to derive $s \rhd^! t$. If $s = (\lambda x.\, u) \cdot v_0 \cdots v_n$ and $u[x := v_0] \cdot v_1 \cdots v_n \trianglerighteq^! t$, then by induction $u\gamma[x := v_0\gamma] \cdot v_1\gamma \cdots v_n\gamma = (u[x := v_0] \cdot v_1 \cdots v_n)\gamma \trianglerighteq^! t\gamma$, and therefore $s\gamma = (\lambda x.\, u\gamma) \cdot v_0\gamma \cdots v_n\gamma \rhd^! t\gamma$ by the same clause. If $s = f(u_1, \ldots, u_m) \cdot v_1 \cdots v_n$ and $u_i \cdot \vec{c} \trianglerighteq^! t$, then $s\gamma = f(u_1\gamma, \ldots, u_m\gamma) \cdot v_1\gamma \cdots v_n\gamma$ and by the induction hypothesis $u_i\gamma \cdot \vec{c} = (u_i \cdot \vec{c})\gamma \trianglerighteq^! t$; thus $s\gamma \rhd^! t\gamma$ by the same clause. Finally, if $s = u \cdot v_1 \cdots v_n$ and $v_i \cdot \vec{c} \trianglerighteq^! t$ we equally complete with the induction hypothesis. ◀

The following Lemma was stated as fact in the text.

▶ **Lemma 1.11.** *If $s \rhd t$ and s has base type, there are terms $u_1, \dots, u_n$ and substitution γ such that $s \rhd^! t\gamma \cdot u_1 \cdots u_n$.*

Proof. If $s \rhd t$ then there is a non-empty context C such that $s = C[t]$. We prove the Lemma by induction over the "measure" of C, where the measure $\mu(C)$ is inductively defined as follows:

- $\mu(\Box) = 0$
- $\mu(D[] \cdot v) = \mu(D[])$
- $\mu(\lambda x.\, D[]) = \mu(D[]) + 1$
- $\mu(f(v_1, \dots, D[], \dots, v_m)) = \mu(D[]) + 1$
- $\mu(v \cdot D[]) = \mu(D[]) + 1$

Clearly all contexts have non-negative measure.

If $C[] = (\lambda x.\, D[]) \cdot v \cdot w_1 \cdots w_m$ then $s \rhd^! t\gamma \cdot \vec{u}$ as required if $D[t][x := v] \cdot \vec{w} \trianglerighteq^! t\gamma \cdot \vec{u}$. If $D[] = \Box$ this is satisfied with $\gamma = [x := v]$ and $\vec{u} = \vec{w}$ (note that x is bound in s, so the requirement on γ holds). If $D[]$ is not the empty context, note that $\mu(C[]) = \mu(D[])+1$ whereas $\mu(D[] \cdot \vec{w}) = \mu(D[])$, and moreover this context has base type. Thus we can apply the induction hypothesis and find $\gamma, \vec{u}$ such that $D[t] \cdot \vec{w} \rhd^! t\gamma \cdot \vec{u}$. But then $D[t][x := t] \cdot \vec{w} = (D[t] \cdot \vec{w})[x := v] \rhd^! (t\gamma \cdot \vec{u})[x := v]$ by Lemma 1.10, and this term equals $(t(\gamma \cup [x := v])) \cdot \vec{u}[x := v]$ and thus has the required form.

If $C[] = f(v_1, \dots, D[], \dots, v_m) \cdot w_1 \cdots w_k$, then $\mu(C[]) = \mu(D[] \cdot \vec{c}) + 1$. It suffices to show that $D[t] \cdot \vec{c} \trianglerighteq t\gamma \cdot \vec{u}$ for suitable $\gamma, \vec{u}$, which holds immediately (choosing γ empty and $\vec{u} = \vec{c}$) if $D[]$ is the empty context and by the induction hypothesis otherwise.

Since s has base type the only other form $C[]$ can have is $v \cdot D[] \cdot w_1 \cdots w_m$ (for some choice of v, possibly an application), and again it is clear that $\mu(C[]) = \mu(D[] \cdot \vec{c}) + 1$. If $D[]$ is the empty context we have $s \trianglerighteq^! t \cdot \vec{c}$ immediately, otherwise the induction hypothesis gives that $D[t] \cdot \vec{c} \rhd^! t\gamma \cdot \vec{u}$ for suitable $\gamma, \vec{u}$ as required. ◀

The following was also stated without proof:

▶ **Lemma 1.12.** *If $\succ \cup \succeq$ contains $\rhd^!$ and $f(x) \succeq f^\#(\vec{x})$ for all $f \in \mathcal{D}$, then $\geq$ satisfies the limited subterm property.*

Proof. We must see that for all x, s, t and $s \trianglerighteq u$ with u not an abstraction or a variable, $\exists\gamma[(\lambda x.\, s) \cdot t \geq u^\#\gamma[x := t]]$. Now, if s has base type and $s = u$ then $s \succeq u$ by reflexivity. Otherwise $s \cdot y_1 \cdots y_n \rhd u$, and therefore by Lemma 1.11 there are $\gamma, \vec{v}$ such that $s \cdot y_1 \cdots y_n \rhd^! u\gamma \cdot \vec{v}$, so by assumption $s \cdot \vec{y}\ (\succ \cup \succeq)\ u\gamma \cdot \vec{v}$. Since $\succeq$ contains `beta` and is monotonous, and because $\succ \cup \succeq$ is stable, $(\lambda x.\, s) \cdot t \cdot \vec{y} \succeq \cdot (\succ \cup \succeq)\ (u\gamma \cdot \vec{v})[x := t] = u\gamma[x := t] \cdot (\vec{v}[x := t]) =: u\gamma[x := t] \cdot \vec{w}$.

So either way, $s(\succeq \cup \succeq \cdot \succ)u\gamma[x := t]$. Because always $f(\vec{x}) \succeq f^\#(\vec{x})$ and by stability and reflexivity $u \succeq u^\#$, so by stability and monotonicity $u\gamma[x := t] \cdot \vec{w} \succeq u^\#\gamma[x := t] \cdot \vec{w}$. Since, by compatibility $\succeq \cdot \succ \cdot \succeq$ is always included either in $\succeq \cdot \succ$ or in $\succ \cdot \succeq$ (by compatibility) we thus have $(\lambda x.\, s) \cdot t \geq u^\#\gamma[x := t]$. ◀

Proof of Theorem 6.7. Let $(\succ, \succeq)$ be a reduction pair satisfying the requirements in the Theorem, and let $(>, \geq, \geq_1)$ be the associated reduction triple. By Theorem 6.2 D is non-looping if:

1. $l > p$ for all $l \rightsquigarrow p \in D_1$, which is satisfied if $\overline{l} \succ \overline{p}$;
2. $l \geq p$ for all $l \rightsquigarrow p \in D_2$, which is certainly satisfied if $\overline{l} \succeq \overline{p}$;
3. $l \geq_1 r$ for all $l \to r \in \mathcal{R}$, which is satisfied if $l \succeq r$;
4. either D is non-collapsing, or $\geq$ satisfies the limited subterm property, which by Lemma 1.12 holds if the conditions of the Theorem are satisfied.

◀

A.6 Proofs of Section 7

First we note some properties of FS and FR which are reasonably obvious, but nevertheless should be stated and proved:

▶ **Lemma 1.13.** *The following statements hold:*
1. *If $FS_n(s) \subseteq FS_m(t)$, then $FR_n(s) \subseteq FR_m(t)$.*
2. *$FR_n(s)$ is included in $FR_{n+1}(s)$.*
3. *If $Symb(s) \subseteq FS_k(t)$, then for any N, $FS_N(s) \subseteq FS_{N+k}(t)$.*
4. *If $l \to r \in FR_k(s)$, then $FR_N(l) \subseteq FR_{N+k+1}(s)$*
5. *For any n: $\to^*_{FR_n(s)}$ is included in $\to^*_{\mathcal{R}}$.*

Proof.
(1) holds because $FR_k(u)$ is entirely defined by symbols in $FS_k(u)$: given $\mathcal{R}$ and $\langle f, \sigma\rangle$ we could define a set of rules $A_{f,\sigma}$ such that $FR_k(u) = \bigcup_{\langle f,\sigma\rangle \in FS_k(u)} A_{f,\sigma}$ (there may well be some overlap between different $A_{f,\sigma}$). Having this, it is evident that $FR_n(s) \subseteq FR_m(t)$ when $FS_n(s) \subseteq FS_m(t)$.
(2) is a consequence of (1): $FS_n(s) \subseteq FS_{n+1}(s)$ by the recursive definition.
(3) is proved with induction over N. If $N = 0$, the assumption provides what we need immediately. If we know the statement holds for given N, note that $FS_{N+1}(s)$ is the union of $FS_N(s)$ and $FS_0(l)$ for $l \to r \in FR_N(s)$. By the induction hypothesis also $FS_N(s) \subseteq FS_{N+k}(t)$ and therefore by 1) $FR_N(s) \subseteq FR_{N+k}(t)$. As such, each symbol added to FS_{N+1} is also added to FS_{N+k+1}.
(4) can be derived from (1) and (3): if $l \to r \in FR_k(s)$ then $Symb(l) \in FS_{k+1}(s)$, so by (3) also $FS_N(l) \subseteq FS_{N+k+1}(s)$. By (3), then, $FS_N(l) \subseteq FR_{N+k+1}(s)$ as required.
For **(5)** we first observe that whenever $l \to r \in \mathcal{R}$ and $x_1, \dots, x_n$ variables such that $l \cdot \vec{x}$ is well-typed, we also have $l \cdot \vec{x} \to_{\mathcal{R}} r\ \vec{x}$, by monotonicity of $\to_{\mathcal{R}}$. Since $\to^*_{\mathcal{R}}$ is transitive also the multistep relation $\to^*_{FR_n(s)}$ is included in $\to^*_{\mathcal{R}}$. ◀

Let $FR_\infty(s) = \bigcup_{n\in\mathbb{N}} FR_n(s)$. The following was stated without proof:

▶ **Lemma 1.14.** *In a system with finitely many rules we can always find N such that $FR_\infty(s) = FR_N(s)$.*

Proof. In a system with finitely many rules there are only finitely many different pairs $l \cdot \vec{x} \to r \cdot \vec{x}$ (counting rules equal if they are equal under renaming of variables). All rules in any $FR_n(s)$ have this form. Therefore, eventually $FR_N(s) = FR_{N+1}(s)$ and from that point on it is easy to see that $FR_{N+K}(s)$ stays the same. ◀

Proof of Lemma 7.4. Given a terminating term s, a term l and substitution γ such that $s \to^*_{\mathcal{R}} l\gamma$, we must show there is a substitution δ such that $\delta \to^*_{\mathcal{R}} \gamma$ and $s \to^*_{FR(l)} l\delta$. If $FR_\infty(l)$ contains any rule $l' \to r'$ with l' not simple or l itself is not simple this is obvious, because $FR(l) = \mathcal{R}$, so $\delta := \gamma$ suffices. Thus we can safely assume that l is simple, and

(**) *l' is simple for any $l' \to r' \in FR_\infty(l)$.*

For a given set of variables X, we say a term s is *simple using* X if s has no subterms $x \cdot \vec{t}$ with $x \in X$ nor any subterms $\lambda y.\, t$ with $FV(t) \cap X \neq \emptyset$. A simple term l is simple using $FV(l)$.

It suffices to find δ such that $s \to^*_{FR_\infty} l\delta$ and $\delta \to^*_{\mathcal{R}} \gamma$. We can additionally assume that $\mathsf{dom}(\gamma)$ contains no variables not occurring in l, as we could simply say $\gamma = \gamma_1 \cup \gamma_2$ with $l\gamma = l\gamma_1$, find suitable δ_1 with the procedure below, and take $\delta := \delta_1 \cup \gamma_2$.

Thus, let $\mathsf{dom}(\gamma) \subseteq X \subseteq FV(l)$ for l a simple term using X such that (**) holds, and assume $s \to^*_{\mathcal{R}} l\gamma$. Performing induction on s, ordered by $\to_{\mathcal{R}} \cup \rhd$, we will find some N and

substitution δ such that $s \to^*_{FR_N(l)} l\delta$ and $\delta \to^*_{\mathcal{R}} \gamma$. First suppose l is a variable in $\mathsf{dom}(\gamma)$, so $\mathsf{dom}(\gamma) = X = FV(l)$; choosing $\delta := \{l \mapsto s\}$ and $n = 0$ we are done. Otherwise, noting that l is simple using X, l is either an abstraction not containing any variables $x \in X$, a term $x \cdot l_1 \cdots l_n$ with $x \notin X$ (so also $x \notin \mathsf{dom}(\gamma)$!) or a term $f(l_1, \ldots, l_m) \cdot l_{m+1} \cdots l_n$.

We alter the derivation $s \to^*_{\mathcal{R}} l\gamma$ a bit. For now (only in this proof), let us call a step $u\chi \cdot t_0 \cdots t_n \to_{\mathcal{R}} v\chi \cdot t_0 \cdots t_n$ (with $u \to v \in \mathcal{R}$) *unpleasant* if v is an abstraction. Unpleasant steps can be avoided, because the rules have been "completed" in Section 4. Suppose $s \to^*_{\mathcal{R}} u\chi \cdot \vec{t} \to_{\mathcal{R}} v\chi \cdot \vec{t} \to^*_{\mathcal{R}} l\gamma$ and the reduction $s \to^*_{\mathcal{R}} u\chi \cdot \vec{t}$ does not use any (headmost) unpleasant steps. Since $l\gamma$ is not headed by a beta-redex there must be a headmost β-step in the next reduction, $v\chi \cdot \vec{t} = (\lambda x.\, w) \cdot t_0 \cdots t_n \to^*_{\mathcal{R}} (\lambda x.\, w') \cdot t'_0 \cdots t'_n \to_\beta w'[x := t'_0] \cdots t'_n \to^*_{\mathcal{R}} l\gamma$. But then also $u\chi \cdot t_0 \cdots t_n \to_{\mathcal{R}} w[x := t_0] \cdot t_1 \cdots t_n \to^*_{\mathcal{R}} l\gamma$. Using induction first over s (by $\to_{\mathcal{R}}$), then over the depth of the redex in s (if $s = u$ the choice of a different rule lowers the depth by one) this procedure always leads to a reduction $s \to^*_{\mathcal{R}} l\gamma$ without any unpleasant steps.

Now we pay special attention to *variable-focussed* steps. If $u\chi \cdot t_1 \cdots t_n \to_{\mathcal{R}} v\chi \cdot t_1 \cdots t_n$ (with $u \to v \in \mathcal{R}$) and $head(v) \in \mathcal{V}$ this is a so-called variable-focussed step (we only use this terminology in this proof). Suppose $s \to^*_{\mathcal{R}} u\chi \cdot \vec{t} \to_{\mathcal{R}} v\chi \cdot \vec{t} \to^*_{\mathcal{R}} l\gamma$ with $head(v)$ a variable and the reduction $s \to^*_{\mathcal{R}} u\chi$ not containing any headmost variable-focussed steps. Then note that $v : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \sigma$ where $l : \sigma$; whether l is an abstraction, headed by a bound variable or headed by a functional term, there is some $f \in \mathcal{F} \cup \{ABS, VAR\}$ such that $\langle f, \sigma \rangle \in FS_0(l)$. Therefore $l' := u \cdot x_1 \cdots x_n \to v \cdot x_1 \cdots x_n =: r' \in FR_0(l)$. By (**) l' is simple and by Lemma 1.13(4) $FR_m(l') \subseteq FR_{m+1}(l)$, so (**) also holds for l'. Let $\gamma' := \chi \cup [x_1 := t_1, \ldots, x_n := t_n]$. Now suppose we can find N', δ' such that $s \to^*_{FR_{N'}(l')} l'\delta'$ and $\delta' \to^*_{\mathcal{R}} \gamma'$. Then we know $s \to^*_{FR_{N'+1}(l)} l'\delta' \to_{FR_0(l)} r'\delta' \to^*_{\mathcal{R}} r'\gamma' \to^*_{\mathcal{R}} l\gamma$. Since $\to^*_{FR_k(l)}$ is included in $\to^*_{\mathcal{R}}$ for any k by Lemma 1.13(5) we can apply the induction hypothesis on the reduction $r'\delta' \to^*_{\mathcal{R}} l\gamma$ to find M and δ such that $r'\delta' \to^*_{FR_M(l)} l\delta$ and $\delta \to^*_{\mathcal{R}} \gamma$. By Lemma 1.13(2) we can then conclude $s \to^*_{FR_k(l)} l\delta$, where $k = max(N' + 1, M)$.
What remains to be seen is only that such N', δ' really exist. We can safely assume that γ' has domain $FV(l')$ and take $X' = FV(l')$ as well. Then l' is simple using X', (**) holds, and $s \to^*_{\mathcal{R}} l'\gamma'$, so also $s \to^*_{\mathcal{R}} l'\gamma'$. We will go on with l', γ', X' instead of l, γ, X.

The gain of the procedure detailed above is that the reduction $s \to^*_{\mathcal{R}} l\gamma$ can now be assumed not to have any headmost unpleasant or variable-focussed steps. If s itself is headed by a beta-redex, then as explained above eventually a β-step is done: $s = (\lambda x.\, t) \cdot u \cdot v_1 \cdots v_n \to^*_{\mathcal{R}} (\lambda x.\, t') \cdot u' \cdot v'_1 \cdots v'_n \to^*_\beta t'[x := u'] \cdot v'_1 \cdots v'_n \to^*_{\mathcal{R}} l\gamma$. We might as well have done the β-step immediately, and the induction hypothesis provides N and suitable δ such that $t[x := u] \cdot \vec{v} \to^*_{FR_N(l)} l\delta$; since $\to_\beta$ is included in $\to_{FR_N(l)}$ this completes the induction step.

Thus, we can safely assume that s is not headed by a beta-redex and $s \to^*_{\mathcal{R}} l\gamma$ without any headmost unpleasant or variable-focussed steps, so no headmost beta-redex is ever created either. Consequently, either $s \to^*_{\mathcal{R}} l\gamma$ purely by internal steps, or whenever $s \to^*_{\mathcal{R}} l'\gamma' \cdot \vec{t} \to_{\mathcal{R}} r'\gamma' \cdot \vec{t} \to^*_{\mathcal{R}} l\gamma$ the head of r' is a functional term $f(\vec{u})$. Our aim is still to find some N and substitution δ such that $s \to^*_{FR_n(l)} l\delta$ and $\delta \to^*_{\mathcal{R}} \gamma$. We perform a second induction on the length of the reduction. Consider the form of l.

If l is an abstraction $\lambda x.\, l'$, then $X = FV(l') \cap X = \emptyset$, so γ is the empty substitution and we are actually trying to prove $s \to^*_{FR_N(l)} l$ for some N. If also $s = \lambda x.\, s'$ then we apply the $\triangleright$ part of the first induction hypothesis (which we can do, because still l' is simple using $X = \emptyset$, and (**) holds because $FS_0(l') \subseteq FS(l)$ and therefore $FR_n(l') \subseteq FR_n(l)$ for any n by Lemma 1.13(1)). We see that $s' \to^*_{FR_N(l')} l'$ for some N, and since $FR_N(l') \subseteq FR_N(l)$ therefore $s \to^*_{FR_N(l)} l = l\gamma$. Alternatively, if s is not an abstraction, there must be a topmost

step reducing to an abstraction. Since there would be no headmost (and thus also no topmost) β-steps in the reduction this can only be the case if the reduction has the form $s \to^*_{\mathcal{R}} u\chi \to_{\mathcal{R}} v\chi \to^*_{\mathcal{R}} l\gamma$ with $u \to v \in \mathcal{R}$ and v an abstraction. But v has the same type σ as l, and since $\langle ABS, \sigma\rangle \in FS_0(l)$ therefore $u \to v \in FR_0(l)$. Thus (**) still holds for u and u is simple over $FV(u) = \mathsf{dom}(\chi) =: X'$. We can apply the second induction hypothesis to find M and a substitution ϵ such that $s \to^*_{FR_M(u)} u\epsilon$ and $\epsilon \to^*_{\mathcal{R}} \chi$, and noting that $FR_M(u) \subseteq FR_{M+1}(l)$ by Lemma 1.13(4) also $s \to^*_{FR_{M+1}(l)} u\epsilon \to_{FR_0(l)} v\epsilon \to^*_{\mathcal{R}} v\chi \to^*_{\mathcal{R}} l\gamma$. Now the first induction hypothesis provides suitable K, δ such that $v\epsilon \to^*_{FR_K(l)} l\delta$ and $\delta \to^*_{\mathcal{R}} \gamma$. Defining $N = max(M+1, K)$, we have $s \to^*_{FR_N(l)} l\delta$ as required, by Lemma 1.13(2).

Alternatively, l either has the form $x \cdot l_1 \cdots l_m$ with x a variable not in $\mathsf{dom}(\gamma)$, or $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$. Since all headmost steps in the reduction $s \to^*_{\mathcal{R}} l\gamma$ use a rule $u \to f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$, the first form, $l = x \cdot l_1 \cdots l_m$ is only possible if this form was present from the beginning: $s = x \cdot s_1 \cdots s_n$ and each $s_i \to^*_{\mathcal{R}} l_i\gamma$. Note that l is simple using X, so it is linear over $X \supseteq \mathsf{dom}(\gamma)$. We can define $X_i = X \cap FV(l_i)$ and $\gamma_i := \gamma_{\restriction X_i}$ for $1 \leq i \leq m$ to have a number of substitutions on disjunct domains. $FS(l_i) \subseteq FS(l)$ for all i so by Lemma 1.13(1, 3) all $FR_k(l_i) \subseteq FR_k(l)$. Hence (**) applies on l_i too, and we can apply the induction hypothesis to get $N_1, \ldots, N_m$ and $\delta_1, \ldots, \delta_m$ such that each $s_i \to^*_{FR_{N_i}(l)} l_i\delta_i$ and $\delta_i \to^*_{\mathcal{R}} \gamma_i$. Define $N := max(N_1, \ldots, N_m)$ and $\delta := \delta_1 \cup \ldots \cup \delta_m$ (which is well-defined because all γ_i and therefore also all δ_i have disjunct domains, and has the same domain as γ because $\mathsf{dom}(\gamma) \subseteq FV(l)$). Then $\delta \to^*_{\mathcal{R}} \gamma$ and by Lemma 1.13(2) also $s \to^*_{FR_N(l)} l\delta$. If l has the form $f(l_1, \ldots, l_k) \cdot l_{n+1} \cdots l_m$ and $s \to^*_{in} l\gamma$ we equally complete with the $\triangleright$ part of the first induction hypothesis.

What remains is the case $s \to^*_{\mathcal{R}} u\chi \cdot t_1 \cdots t_n \to_{\mathcal{R}} v\chi \cdot t_1 \cdots t_n \to^*_{in} l\gamma$. We can write $v = f(v_1, \ldots, v_m) \cdot v_{m+1} \cdots v_k : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \sigma$ and $l = f(l_1, \ldots, l_m) \cdot l_{m+1} \cdots l_{k+n} : \sigma$. Therefore $\langle f, \sigma\rangle \in FS_0(l)$ and thus $u \cdot x_1 \cdots x_n \to v \cdot x_1 \cdots x_n \in FR_0(l)$. Define $l' := u \cdot \vec{x}$ and $\gamma' := \chi \cup [x_1 := t_1, \ldots, x_n := t_n]$. As we did before (in the case of a head step reducing to an abstraction), we apply the second part of the induction hypothesis to find find M, δ' such that $s \to^*_{FR_M(l)} l'\gamma'$ and then the first part of the induction hypothesis to find a suitable K, δ such that $r'\delta' \to^*_{FR_K(l)} l\delta$; as before, we choose $N := max(M, K)$. ◀

Proof of Lemma 7.5. We must see that if $\mathcal{R}$ is non-terminating, then there is an infinite dependency chain over $\mathsf{DP}(\mathcal{R})$ such that $t_i \to^*_{FR(l_{i+1}),in} s_{i+1}$ (rather than just $t_i \to^*_{\mathcal{R},in} s_{i+1}$). We use the same approach as in the proof of Theorem 4.5. Given any non-terminating term, let u_{-1} be an MNT subterm. Now, for any $i \in \mathbb{N}$, let u_i be a given MNT term, and consider an infinite reduction starting in u_i. Then u_i is either headed by a β-redex, or has the form $f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$ with $f \in \mathcal{D}$. In the first case we proceed exactly as done in the proof of Theorem 4.5, which gives $s_{i+1} = t_i$. In the second case, $u_i = f(v_1, \ldots, v_n) \cdot v_{n+1} \cdots v_m$, only finitely many steps in the v_j can be taken before a head step appears in any infinite reduction, say $u_i \to^*_{in} f(v'_1, \ldots, v'_n) \cdot v'_{n+1} \cdots v'_m = l\gamma \cdot v'_{j+1} \cdots v'_m$ with $j \leq m$ and $r\gamma \cdot v'_{j+1} \cdots v'_m$ still non-terminating. As explained before, we can always find such a rule that either $m = j$ or r is not an abstraction. Define $l' := l \cdot x_{j+1} \cdots x_n$ and $r' := r \cdot x_{j+1} \cdots x_n$ for fresh $x_{j+1} \cdots x_n$. If $FR(l') = \mathcal{R}$ let $\delta = \gamma \cup [x_{j+1} := v_{j+1}, \ldots, x_m := v_m]$; then certainly $u_i \to^*_{FR(l'),in} l'\delta$ and $r'\delta$ is still non-terminating (since it reduces to $r\gamma \cdot v'_{j+1} \cdots v'_m$). Otherwise l' must be simple, so it is left-linear; write $l' = f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_j \cdot x_{j+1} \cdots x_m$. Assuming (w.l.o.g.) that $\mathsf{dom}(\gamma) = FV(l)$ we can define $\gamma_i = \gamma_{\restriction FV(l_k)}$ for $1 \leq k \leq j$ and have $\gamma = \gamma_1 \cup \ldots \cup \gamma_j$ and all γ_k have disjunct domains. Since u_i is MNT all v_k are terminating, so by Lemma 7.4 we can find substitution δ_k on the same domain as γ_k such that $v_k \to^*_{FR(l_i)} l_i\delta_k$ and all $\delta_k(x) \to^*_{\mathcal{R}} \gamma_k(x)$. By Lemma 1.13(1,3) $FR(l_i) \subseteq FR(l')$, so defining $\delta := \delta_1 \cup \ldots \cup \delta_j \cup [x_{j+1} := v_{j+1}, \ldots, x_m := v_m]$ we have $u_i \to^*_{FR(l'),in} l'\delta$ for all $1 \leq i \geq m$ and $\delta \to^*_{\mathcal{R}} \gamma \cup [x_{j+1} := v'_{j+1}, \ldots, x_m := v'_{j+1}]$,

so $r'\delta$ is non-terminating.

If $m > j$ then in either of the two cases above $r'\delta$ is MNT (as discussed in the original proof), so $l' \rightsquigarrow r'$ is a dependency pair and we can define $u_{i+1} := r'\delta$ and $\rho_{i+1}, s_{i+1}, t_{i+1} := l' \rightsquigarrow r',\ l'\delta,\ r'\delta$. If $m = j$, so $l = l', r = r'$, let $p := p'[\vec{y} := \vec{c}]$ with $r \trianglerighteq p'$, $FV(p')\backslash FV(r) = \{\vec{y}\}$ and p' minimal such that $p\delta$ is still non-terminating. As discussed before, $p\delta$ is MNT, and $l^\# \rightsquigarrow p^\#$ is a dependency pair. Choose $u_{i+1} := p\gamma$ and define $\rho_{i+1}, s_{i+1}, t_{i+1} := l^\# \rightsquigarrow p^\#, l^\#\delta, p^\#\delta$. The chain constructed in this way is an infinite dependency chain satisfying the requirement in the Lemma. ◀

Proof of Theorem 7.6. Note: this Theorem introduces a new definition of "non-looping". First we see: if D is non-looping, then there is no usable dependency chain with all pairs in D. This holds with induction over the derivation that D is non-looping: evidently there is no usable chain over $\emptyset$ (since any usable chain is a dependency chain), and the existence of such a split and ordering implies there cannot be a usable chain, in the same way as Theorem 6.2 was proved but noting that, since only rules in $FR(D)$ are used in every step, $l \geq_1 r$ is only required for these rules. ◀

A.7 Proofs of Section 8

▶ **Lemma 1.15** (Examples of Weakly Monotonic Functionals). *The following functions are weakly monotonic.*

1. *For any element $n \in A_\iota$ and $\sigma = \tau_1 \Rightarrow \ldots \Rightarrow \tau_k \Rightarrow \iota$ the function n_σ.*
2. *Any typing of the function $\boldsymbol{\lambda} f.f(\vec{0})$.*
3. *Any $\max_{\sigma,\iota}$.*

Proof. (1) is Lemma 4.2.2(1) in [21]. (2) is an immediate consequence of (1) and Lemma 4.1.6 in [21], writing $\boldsymbol{\lambda} f.f(\vec{0}) = [\![\lambda F.\, F \cdot \mathsf{o}_1 \cdots \mathsf{o}_n]\!]_{\mathcal{J}}$ where $\mathcal{J}_{\mathsf{o}_i} = 0_{\sigma_i}$. (3) holds by induction on σ, noting that $\max_{\iota,\kappa}$ is certainly weakly monotonic, and using once more Lemma 4.1.6 in [21] for the induction step. ◀

▶ **Lemma 1.16** (Property of max). $\max_\sigma(f, n) \sqsupseteq_{wm} f, n_\sigma$

Proof. By induction on σ. If σ is a base type, then $\max_\sigma(f, n) \sqsupseteq_{wm} f, n$ by the assumption on $\vee$. If $\sigma = \tau \Rightarrow \tau'$, then $\max_\sigma(f, n) = \boldsymbol{\lambda} x.\max(f(x), n)$, which by induction $\sqsupseteq_{wm}$ both $\boldsymbol{\lambda} x.f(x) = f$ and $\boldsymbol{\lambda} x.n_\tau = n_\sigma$. ◀

▶ **Definition 1.17** (Translation). Introduce a new symbol $@_{\sigma,\tau} : (\sigma \Rightarrow \tau) \Rightarrow \sigma \Rightarrow \tau$ for all types σ, τ and let $f' : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \tau$ be the flattened version of $f : (\sigma_1 \times \ldots \times \sigma_n) \Rightarrow \tau$. For any term s, let its lengthened version $[s]$ be defined as follows: $[s] = s$ if $s : \iota$ and $[s] = \lambda x.\,[s \cdot [x]]$ if $s : \sigma \Rightarrow \tau$ and x a fresh variable. Now define the following translation of AFS-terms to HRS-terms:

$$\begin{array}{rcl} \phi(x) & = & [x] \\ \phi(f(s_1, \ldots, s_n)) & = & [f' \cdot \phi(s_1) \cdots \phi(s_n)] \\ \phi(\lambda x.\, s) & = & \lambda x.\, \phi(s) \\ \phi(s \cdot t) & = & [@(\phi(s), \phi(t))] \end{array}$$

Then evidently $\phi(s)$ is in long β/η-normal form for all s, and thus can be seen as an HRS-term. Assuming $\mathcal{J}_{@_{\sigma,\tau}} = \boldsymbol{\lambda} f.\boldsymbol{\lambda} n.\max_\tau(f(n), n(\vec{0}))$ the definition of $[\![\cdot]\!]$ as given in Definition 8.2 corresponds with the definition in [21], as the following Lemmas show:

▶ **Lemma 1.18.** *Using Pol's interpretations, $[\![s]\!] = [\![[s]]\!]$.*

Proof. By induction on the type of s; obvious for base types, and if $s : \sigma \Rightarrow \tau$ then $[\![\phi([s])]\!] = [\![\phi(\lambda x.\,[s\cdot[x]])]\!] = \boldsymbol{\lambda} n.[\![[s\cdot[x]]]\!]_{x:=n} =$ (IH) $\boldsymbol{\lambda} n.[\![s\cdot[x]]\!]_{x:=n} = \boldsymbol{\lambda} n.[\![s]\!]_{x:=n}([\![[x]]\!]_{x:=n}) = \boldsymbol{\lambda} n.[\![s]\!]([\![[x]]\!]_{x:=n}) =$ (IH) $\boldsymbol{\lambda} n.[\![s]\!]([\![x]\!]_{x:=n}) = \boldsymbol{\lambda} n.[\![s]\!](n) = [\![s]\!]$, since functions are extensional. ◀

▶ **Lemma 1.19.** *If $\mathcal{J}'_{f'} = \mathcal{J}_f$ for all f, then $[\![s]\!]_{\mathcal{J},\alpha}$ from Definition 8.2 equals $[\![\phi(s)]\!]_{\alpha,\mathcal{J}'}$ from [21].*

Proof. With induction on the size of s. The case where s is a variable is immediate, if s is an abstraction we use the induction hypothesis. If $s = f(s_1,\dots,s_m)$ then $[\![s]\!] = \mathcal{J}_f([\![s_1]\!],\dots,[\![s_m]\!]) = \mathcal{J}_f([\![s_1]\!])\dots([\![s_m]\!]) =$ (IH) $\mathcal{J}_f([\![\phi(s_1)]\!])\dots([\![\phi(s_m)]\!]) =$ (Lemma 1.18, and because $\mathcal{J}'_{f'} = \mathcal{J}_f$) $[\![\phi(s)]\!]$. If $s = t\cdot u$ then $[\![s]\!] = \max([\![t]\!](\llbracket u\rrbracket), \llbracket u\rrbracket(\vec{0})) = \mathcal{J}'_{@}([\![t]\!], \llbracket u\rrbracket) =$ (IH) $[\![@(\phi(t),\phi(s))]\!] =$ (Lemma 1.18) $[\![\phi(s)]\!]$ as required. ◀

We continue with some additional properties of the interpretation and the max function, which will be necessary for the proof of Theorem 8.4.

▶ **Lemma 1.20.** *Let $\mathcal{J}$ be an interpretation function which maps all c_x to 0_σ if $x : \sigma$. Then for all s and valuations α: $[\![s\cdot c_{x_1}\cdots c_{x_n}]\!]_{\mathcal{J},\alpha} = [\![s]\!]_{\mathcal{J},\alpha}(0_{\sigma_1},\dots,0_{\sigma_n})$.*

Proof. By induction on n; the base case is immediate. If $A := [\![s\cdot c_{x_1}\cdots c_{x_{n-1}}]\!]_{\mathcal{J},\alpha} = [\![s]\!]_{\mathcal{J},\alpha}(0_{\sigma_1},\dots,0_{\sigma_{n-1}})$, then $[\![s\cdot c_{x_1}\cdots c_{x_n}]\!]_{\mathcal{J},\alpha} = \max(A([\![c_{x_n}]\!]_{\mathcal{J},\alpha}), [\![c_{x_n}]\!]_{\mathcal{J},\alpha}(\vec{0})) =$ (by assumption) $\max(A(0_{\sigma_n}), 0) =$ (definition of $\vee$) $A(0_{\sigma_n}) = [\![s]\!]_{\mathcal{J},\alpha}(\vec{0})$. ◀

▶ **Lemma 1.21.** *If $s = t\cdot u_1\cdots u_n$ and has base type, then $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![u_i]\!]_{\mathcal{J},\alpha}(\vec{0})$ for each $1 \le i \le n$ and $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![t]\!]_{\mathcal{J},\alpha}([\![u_1]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha})$.*

Proof. By induction on k we see that $[\![t\cdot u_1\cdots u_k]\!]_{\mathcal{J},\alpha}([\![u_{k+1}]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha}) \sqsupseteq_{wm}$ each $[\![u_i]\!](\vec{0})$ for $i \le k$ and also $\sqsupseteq_{wm} [\![t]\!]([\![u_1]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha})$; for $k = n$ this gives the Lemma. For $k = 0$ the statement is evident (the first claim holds because there is no $1 \le i \le k$, the second because $\sqsupseteq_{wm}$ is reflexive). If $0 < k \le n$ and the statement holds for $k-1$, then note that $[\![t\cdot u_1\cdots u_k]\!]_{\mathcal{J},\alpha}([\![u_{k+1}]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha}) = \max([\![t\cdot u_1\cdots u_{k-1}]\!]_{\mathcal{J},\alpha}([\![u_k]\!]_{\mathcal{J},\alpha}), [\![u_k]\!]_{\mathcal{J},\alpha}(\vec{0}))($ $[\![u_{k+1}]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha})$. By Lemma 1.16 this term $\sqsupseteq_{wm}$ both $[\![t\cdot u_1\cdots u_{k-1}]\!]_{\mathcal{J},\alpha}([\![u_k]\!]_{\mathcal{J},\alpha})($ $[\![u_{k+1}]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha}) = [\![t\cdot u_1\cdots u_{k-1}]\!]_{\mathcal{J},\alpha}([\![u_k]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha})$ and also $\sqsupseteq_{wm}$ $(\boldsymbol{\lambda} x_{k+1}\cdots x_n.[\![u_k]\!]_{\mathcal{J},\alpha}(\vec{0}))([\![u_{k+1}]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha}) = [\![u_k]\!]_{\mathcal{J},\alpha}(\vec{0})$. This proves one part of the Lemma, the rest is given by the induction hypothesis. ◀

▶ **Lemma 1.22.** *Let $\mathcal{J}$ be a non-decreasing interpretation function which maps all c_x to 0_σ of the corresponding type. Suppose $s \trianglerighteq^! t$. Then $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![t]\!]_{\mathcal{J},\alpha}$ for all valuations α.*

Proof. By induction over the derivation of $s \trianglerighteq^! t$. The claim is evident if $s = t$, since $\sqsupseteq_{wm}$ is reflexive. If $s = (\lambda x.\,u)\cdot v\cdot w_1\cdots w_n$ and $u[x := v]\cdot\vec{w} \trianglerighteq^! t$, then by the induction hypothesis $[\![u[x := v]\cdot\vec{w}]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![t]\!]$, so by transitivity of $\sqsupseteq_{wm}$ we are done if $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![u[x := v]\cdot\vec{w}]\!]_{\mathcal{J},\alpha}$. By weak monotonicity of the function $K := \boldsymbol{\lambda} f.\boldsymbol{\lambda} n.\max(f(n), n(\vec{0}))$ (which holds by weak monotonicity of max and [21, Lemma 4.1.6]) this is certainly true if $[\![(\lambda x.\,u)\cdot v]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![u[x := v]]\!]_{\mathcal{J},\alpha}$. But this follows from the substitution Lemma (3.2.1) in [21]: $[\![(\lambda x.\,u)\cdot v]\!]_{\mathcal{J},\alpha} = \max([\![\lambda x.\,u]\!]_{\mathcal{J},\alpha}([\![v]\!]_{\mathcal{J},\alpha}), [\![v]\!]_{\mathcal{J},\alpha}(\vec{0})) \sqsupseteq_{wm} [\![\lambda x.\,u]\!]_{\mathcal{J},\alpha}([\![v]\!]_{\mathcal{J},\alpha})$ (by Lemma 1.21), $= (\boldsymbol{\lambda} n.[\![u]\!]_{\mathcal{J},\alpha[x\mapsto n]})([\![v]\!]_{\mathcal{J},\alpha}) = [\![u]\!]_{\mathcal{J},\alpha\circ[x:=v]}$ (where $\alpha\circ[x := v]$ is the valuation $\alpha[x := [\![v]\!]_{\mathcal{J},\alpha}]$), which by the substitution Lemma equals $[\![u[x := v]]\!]_{\mathcal{J},\alpha}$.

If $s = f(u_1,\dots,u_n)\cdot v_1\cdots v_m$ and $u_i\cdot\vec{c} \trianglerighteq^! t$, then by transitivity of $\sqsupseteq_{wm}$ and the induction hypothesis we are done if $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![u_i\cdot\vec{c}]\!]_{\mathcal{J},\alpha}$. By Lemma 1.21 $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm}$ $[\![f(\vec{u})]\!]_{\mathcal{J},\alpha}([\![v_1]\!]_{\mathcal{J},\alpha},\dots,[\![v_m]\!]_{\mathcal{J},\alpha}) = \mathcal{J}_f([\![u_1]\!]_{\mathcal{J},\alpha},\dots,[\![u_n]\!]_{\mathcal{J},\alpha},[\![v_1]\!]_{\mathcal{J},\alpha},\dots,[\![v_m]\!]_{\mathcal{J},\alpha})$. Noting

that $\mathcal{J}_f$ is weakly monotonic and that any $[\![w]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} 0_\sigma$, this term $\sqsupseteq_{wm} \mathcal{J}_f(0_{\sigma_1}, \ldots, [\![u_i]\!]_{\mathcal{J},\alpha}, \ldots, , 0_{\sigma_n}, 0_{\tau_1}, \ldots, 0_{\tau_m})$, which by non-decreasingness $\sqsupseteq_{wm} [\![u_i]\!]_{\mathcal{J},\alpha}(\vec{0}) =$ (by Lemma 1.20) $[\![u_i \cdot \vec{c}]\!]_{\mathcal{J},\alpha}$ as required.

If neither of those is the case, we can write $s = u \cdot v_0 \cdots v_n$ and $v_i \cdot \vec{c} \trianglerighteq^! t$. By Lemma 1.21 $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![v_i]\!]_{\mathcal{J},\alpha}(\vec{0})$, which by Lemma 1.20 $= [\![v_i \cdot \vec{c}]\!]_{\mathcal{J},\alpha}$. ◀

Proof of Theorem 8.4. $\succ$ is an ordering by its definition, and is well-founded because if $f \sqsupset_{wm} g$ then also $f(\vec{0}) \sqsupset_{wm} g(\vec{0})$, and we assumed that $\sqsupset_{wm}$ is a well-founded order in $\mathcal{A}$. $\succeq$ is a quasi-ordering by its definition, and is monotonic as we see with a simple case distinction; if $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![t]\!]_{\mathcal{J},\alpha}$ for all α, then:

- $[\![\lambda x.\, s]\!]_{\mathcal{J},\alpha} = \boldsymbol{\lambda} n.[\![s]\!]_{\mathcal{J},\alpha\circ[x:=n]} \sqsupseteq_{wm} \boldsymbol{\lambda} n.[\![t]\!]_{\mathcal{J},\alpha\circ[x:=n]} = [\![t]\!]_{\mathcal{J},\alpha}$.
- $[\![s \cdot u]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![t \cdot u]\!]_{\mathcal{J},\alpha}$ by weak monotonicity of $K := \boldsymbol{\lambda} f.\boldsymbol{\lambda} n.\, \max(f(n), n(\vec{0}))$.
- $[\![u \cdot s]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![u \cdot t]\!]_{\mathcal{J},\alpha}$ by weak monotonicity of K.
- $[\![f(\ldots, s, \ldots)]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![f(\ldots, t, \ldots)]\!]_{\mathcal{J},\alpha}$ by weak monotonicity of $\mathcal{J}_f$.

In addition, $\sqsupseteq_{wm}$ contains `beta` by Lemma 1.21, as we also saw in the first part of the proof of Lemma 1.22.

As for compatibility, if $s \succ t \succeq u$, then for all valuations α we have: $[\![s]\!]_{\mathcal{J},\alpha} = \boldsymbol{\lambda} n_1 \ldots n_k.f(n_1, \ldots, n_k) \sqsupset_{wm} [\![t]\!]_{\mathcal{J},\alpha} = \boldsymbol{\lambda} n_1 \ldots n_k.g(n_1, \ldots, n_k) \sqsupseteq_{wm} [\![u]\!]_{\mathcal{J},\alpha} = \boldsymbol{\lambda} n_1 \ldots n_k.h(n_1, \ldots, n_k)$; since $\sqsupset_{wm}$ and $\sqsupseteq_{wm}$ are compatible both ways ($\sqsupseteq_{wm}$ on $\mathcal{A}$ just being the reflexive closure of $\sqsupset_{wm}$), thus $f(n_1, \ldots, n_k) \succ h(n_1, \ldots, n_k)$, and $s \succ u$ follows.

Finally, stability. Here, we use the substitution Lemma ((3.2.1) in [21]): always $[\![u\gamma]\!]_{\mathcal{J},\alpha} = [\![u]\!]_{\mathcal{J},\alpha\circ\gamma}$, so if $[\![s]\!]_{\mathcal{J},\alpha} \sqsupset_{wm} [\![t]\!]_{\mathcal{J},\alpha}$ for all α, then this must also hold for a valuation of the form $\alpha \circ \gamma$. The same holds for $\sqsupseteq_{wm}$.

The subterm property is Lemma 1.22 and the marking property, $[\![f(\vec{x})]\!]_{\mathcal{J},\alpha} \sqsupseteq_{wm} [\![f^\#(\vec{x})]\!]_{\mathcal{J},\alpha}$ is immediate with the assumption $\mathcal{J}_f \sqsupseteq_{wm} \mathcal{J}_{f^\#}$. ◀