Transposing Termination Properties in Higher Order Rewriting

Cynthia Kop kop@few.vu.nl

Department of Computer Science
VU University Amsterdam, the Netherlands.

Abstract. In higher-order term rewriting, distinguished from first-order term rewriting by the presence of bound variables and often also a type discipline, a plethora of different frameworks is used. Consequently, proofs of termination properties derived in one framework often have to be redone for others. In this paper we study the most common frameworks to see how systems in one formalism can be transformed into another, without losing termination properties.

1 Introduction

The last years have seen a rise in the interest in higher-order rewriting, especially the field of termination. Although this area is still far behind its first-order counterpart, various techniques for proving termination have been developed, such as monotone algebras [14], path orderings [4,3] and dependency pairs [16,10,9]. Since 2010 the termination competition [18] has a higher-order category.

However, a persistent problem is the lack of a fixed standard. There are several formalisms, each dealing with the higher-order aspect in a different way, along with variations and restrictions. Because of the differences in what is allowed, results in one formalism do not trivially, or at all, carry over to another. As such it is difficult to reuse results in a slightly different context, which necessitates a lot of double work. Consider for example the original HORPO, first defined for Algebraic Functional Systems in [4] and adapted for Pattern HRSs in [15]. Later extensions of HORPO can not yet be used for HRSs.

This became painfully clear in our endeavour to develop a termination tool. The question what formalism to support is essential from the start. Should we implement a tool for Nipkow's Pattern HRSs, then we cannot handle a system with a rule $\lambda x.Z \cdot x \Rightarrow Z$, and the tool would give a possibly false positive on a system with two rules $f \cdot 0 \Rightarrow g \cdot (\lambda x.0)$ and $g \cdot Z \Rightarrow Z \cdot (f \cdot 0)$ – both of which are valid programs in Jouannaud's Algebraic Functional System formalism. On the other hand, a tool for AFSs could not handle a system with rules like $f \cdot (\lambda x.g \cdot (Z \cdot x)) \Rightarrow Z \cdot 0$. Even if we accept that we cannot support everything, the price for any choice is substantial: for example, the latest version of a recursive path ordering [3] is only developed for AFSs, while the main developments in dependency pairs [10] have been defined for HRSs.

As we will see in this paper, however, the situation is not as grim as it seems at first. Each of the formalisms can be embedded, at least in part, in the

others. We will discuss transformations between the formalisms which preserve termination properties. Importantly, we aim to preserve the structure of rules as much as possible: the translation of a simple rule in formalism A should not be some hideously complicated monster in formalism B. This paper focusses on termination and thus ignores other interesting topics such as confluence. In addition, in discussing Nipkow's HRSs we restrict attention to *pattern* HRSs.

2 Core concepts

We first discuss some concepts which appear in many forms of higher-order term rewriting. In Section 3 we will use these concepts to discuss several formalisms.

2.1 First-order Term Rewriting

In first-order term rewriting, *terms* are built from an infinite set of variables $\mathcal{V}$ and a signature $\mathcal{F}$ of function symbols f, each with an arity n (denoted $ar(f) = n$), by the following grammar:

$$\mathbb{T} = x \mid f(\mathbb{T}^n) \qquad (x \in \mathcal{V},\ f \in \Sigma,\ ar(f) = n)$$

A *substitution* is a mapping $\gamma = [x_1 := s_1, \ldots, x_n := s_n]$ and the application $t\gamma$ of a substitution γ on a term t is obtained by placewise replacing each x_i in t by s_i. A *context* is a term with a special symbol $\Box$ in it, denoted $C[]$. The term $C[s]$ is $C[]$ with the $\Box$ symbol replaced by s.

A *first-order rewrite rule* is a pair $l \Rightarrow r$ such that all variables occurring in r also occur in l. A set of rewrite rules $\mathcal{R}$ generates the rewrite relation $\Rightarrow_{\mathcal{R}}$ by:

$$C[l\gamma] \Rightarrow_{\mathcal{R}} C[r\gamma] \quad (l \Rightarrow r \in \mathcal{R},\ \gamma \text{ a substitution},\ C[] \text{ a context})$$

Example 1. Consider the system with $\mathcal{F} = \{\mathtt{0}, \mathtt{s}, \mathtt{add}\}$ such that $ar(\mathtt{0}) = 0$, $ar(\mathtt{s}) = 1$ and $ar(\mathtt{add}) = 2$, and $\mathcal{R} = \{\mathtt{add}(x, 0) \Rightarrow x,\ \ \mathtt{add}(x, \mathtt{s}(y)) \Rightarrow \mathtt{s}(\mathtt{add}(x, y))\}$. Then $\mathtt{add}(\mathtt{s}(\mathtt{0}), \mathtt{s}(\mathtt{0})) \Rightarrow_{\mathcal{R}} \mathtt{s}(\mathtt{add}(\mathtt{s}(\mathtt{0}), \mathtt{0})) \Rightarrow_{\mathcal{R}} \mathtt{s}(\mathtt{s}(\mathtt{0}))$; thus we see, $1 + 1 = 2$.

2.2 Simple Types

Higher-order term rewriting commonly adds types and binders to first-order term rewriting. Most commonly *simple types* are used. Given a set of *base types* $\mathcal{B}$, the set of simple types is defined by the grammar:

$$\mathcal{T} = \iota \mid \mathcal{T} \to \mathcal{T} \qquad (\iota \in \mathcal{B})$$

A type of the form $\sigma \to \tau$ is called *functional*. Types are written as σ, τ, ρ and base types as ι, κ. The $\to$ associates to the right. A *type declaration* is an expression of the form $(\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$ with $\sigma_1, \ldots, \sigma_n, \tau \in \mathcal{T}$; such a declaration is said to have arity n. Type declarations are not types, but are used for typing purposes. A type declaration $() \longrightarrow \tau$ is usually just denoted by τ.

2.3 The Lambda-calculus

In the lambda-calculus, terms are built from a set $\mathcal{V}$ of variables, using λ-abstraction and application, defined recursively by the following grammar:

$$\mathbb{T} = x \mid \mathbb{T} \cdot \mathbb{T} \mid \lambda x.\mathbb{T} \qquad (x \in \mathcal{V})$$

The $\lambda x.s$ construct *binds* the variable x in s. Term equality is modulo renaming of variables bound by a λ (α-conversion), that is, $\lambda x.s = \lambda y.t$ if $s[x := y] = t$. Consequently, we can always assume variables in a binder to be fresh. The set $FVar(s)$ consists of all variables occurring in s which are not bound by a λ. We say s is *closed* if $FVar(s) = \emptyset$. Brackets are omitted where possible, considering application left-associative; a term $s \cdot t \cdot u$ should be read as $(s \cdot t) \cdot u$.

Substitution is defined as before, but cannot affect bound variables: a construct $s\gamma$ assumes that bound variables in s are renamed so they do not occur in domain or range of γ. However, a context $C[s]$ might bind variables in s.

Terms in the lambda-calculus are rewritten using the β-reduction rule:

$$C[(\lambda x.s) \cdot t] \Rightarrow_\beta C[s[x := t]]$$

The lambda-calculus is not terminating, as is demonstrated by the term $\omega \cdot \omega$, where $\omega = \lambda x.x \cdot x$; this term reduces in one step to itself. However, when you add types to λ-terms, termination is guaranteed. That is, assigning a simple type to all variables in $\mathcal{V}$ (notation: $x : \sigma \in \mathcal{V}$), a λ-term s is terminating if we can derive $s : \sigma$ for some type σ using the following clauses:

$$\begin{array}{ll} x : \sigma & \text{if } x : \sigma \in \mathcal{V} \\ s \cdot t : \tau & \text{if } s : \sigma \to \tau \text{ and } t : \sigma \\ \lambda x.s : \sigma \to \tau & \text{if } x : \sigma \in \mathcal{V} \text{ and } s : \tau \end{array}$$

We say a term is in β-normal form if it cannot be further rewritten using $\Rightarrow_\beta$. The $\Rightarrow_\beta$ relation is terminating on all typed terms and gives unique normal forms, even if typed function symbols are added to the term formation.

η-expansion The relation of *η-expansion*, $\hookrightarrow_\eta$, is defined as follows: $C[s] \hookrightarrow_\eta C[\lambda x_\sigma.s \cdot x_\sigma]$ if $s : \sigma \to \tau$ and the following conditions are satisfied:

1. x_σ is a fresh variable;
2. s is not an abstraction;
3. s in $C[s]$ is not the left-hand side of an application.

Every term s has a unique *η-long form* $s{\uparrow^\eta}$ which can be found by applying $\hookrightarrow_\eta$ until it is no longer possible.

3 Various Formalisms

With these preparations, we discuss the most common formalisms of higher-order term rewriting.

3.1 Combinatory Reduction Systems

The first formalism for general higher order rewriting was Aczel's *Contraction Schemes* [1]. This definition was generalised in 1980 by Klop to *Combinatory Reduction Systems* [5], although Contraction Schemes were implicitly typable whereas CRSs are not. We discuss the definition of CRSs as given in [6].

Basic Definition CRSs extend first-order term rewriting with *meta-variables* and a form of *developments*. Formally, given a set $\mathcal{V}$ of variables, a set $\mathcal{M}$ of meta-variables (denoted Z, X, Y) and a signature $\mathcal{F}$ of function symbols, the set of meta-terms is given by the following grammar:

$$\mathbb{T} = x \mid \lambda x.\mathbb{T} \mid f(\mathbb{T}^n) \mid Z(\mathbb{T}^m) \qquad x \in \mathcal{V},\ f \in \Sigma,\ Z \in \mathcal{M},\ ar(f) = n,\ ar(Z) = m$$

A meta-term is *closed* if it has no free variables, even if it does contain meta-variables. Note that the λ only binds variables; α-conversion works as before.

Terms are meta-terms without meta-variables. Substitution is defined as before (with the assumption that bound variables are renamed to not cause any problems), and additionally *meta-substitution* is defined. A meta-substitution is a map $\gamma = [Z_1 := s_1, \ldots, Z_n := s_n]$ where s_i has the form $\lambda x_1 \ldots x_m.t_i$ if $ar(Z_i) = m$, and the application of a meta-substitution on a meta-term is defined recursively as follows:

$$\begin{array}{rl} f(s_1,\ldots,s_n)\gamma = f(s_1\gamma,\ldots,s_n\gamma) & \text{if } f \in \Sigma \\ Z(s_1,\ldots,s_m)\gamma = t[x_1 := s_1\gamma,\ldots,x_m := s_m\gamma] & \text{if } \gamma(Z) = \lambda x_1 \ldots x_m.t \\ x\gamma = x & \text{if } x \in \mathcal{V} \end{array}$$

Applying a meta-substitution γ on a meta-term s is only defined if all meta-variables occurring in s are in $\mathrm{dom}(\gamma)$. In the second clause a normal substitution is used as well as recursive meta-substitution. A context is a term (not meta-term!) with a special $\Box$ symbol.

A rewrite rule is a pair $l \Rightarrow r$ of closed meta-terms, such that all meta-variables occurring in r also occur in l, and moreover l is a *pattern* of the form $f(l_1, \ldots, l_n)$; a *pattern* is a meta-term where in all meta-variable occurrences $Z(s_1, \ldots, s_n)$, all s_i are different bound variables. A set of rules $\mathcal{R}$ generates a rewrite relation on terms (not meta-terms!) as before:

$$C[l\gamma] \Rightarrow_{\mathcal{R}} C[r\gamma] \quad (l \Rightarrow r \in \mathcal{R},\ \gamma \text{ a meta–substitution},\ C[] \text{ a context})$$

Example 2. The untyped lambda-calculus can be seen as a CRS with a single symbol `app` of arity 2 and a rule $\mathtt{app}(\lambda x.Z(x), X) \Rightarrow Z(X)$.

Example 3. As a running example we will use the system `map` which has nullary symbols `nil`, `0`, unary symbol `s` and binary symbols `cons`, `map`, and rules:

$$\begin{array}{ll} \mathtt{map}(\lambda x.Z(x), \mathtt{nil}) & \Rightarrow \mathtt{nil} \\ \mathtt{map}(\lambda x.Z(x), \mathtt{cons}(X, Y)) & \Rightarrow \mathtt{cons}(Z(X), \mathtt{map}(\lambda x.Z(x), Y)) \end{array}$$

`nil` represents an empty list and `cons` the list constructor. A short reduction:

$$\begin{aligned}&\mathtt{map}(\lambda x.x, \mathtt{cons}(\mathtt{s}(0), \mathtt{nil})\\ &\quad \Rightarrow_{\mathcal{R}} \mathtt{cons}(\mathtt{s}(0), \mathtt{map}(\lambda x.x, \mathtt{nil}))\\ &\quad \Rightarrow_{\mathcal{R}} \mathtt{cons}(\mathtt{s}(0), \mathtt{nil})\end{aligned}$$

3.2 Inductive Data Type Systems

Combinatory Reduction Systems are extended with simple types in [2]. In these *Inductive Data Type Systems*, we assume given a set of typed variables $\mathcal{V}$, a set of meta-variables $\mathcal{M}$ each equipped with a type declaration and a set of function symbols f also each equipped with a type declaration; we assume $\mathcal{V}$ and $\mathcal{M}$ contain infinitely many (meta-)variables of each type. Meta-terms are formed as in CRSs, but with a type restriction. Formally, meta-terms are all expressions s for which we can derive $s : \sigma$ for some type σ using the following clauses:

$$\begin{array}{ll} x : \sigma & \text{if } x : \sigma \in \mathcal{V} \\ \lambda x.s : \sigma \to \tau & \text{if } x : \sigma \in \mathcal{V} \text{ and } s : \tau \\ f(s_1, \ldots, s_n) : \tau & \text{if } f : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau \in \mathcal{F},\ s_1 : \sigma_1, \ldots, s_n : \sigma_n \\ Z(s_1, \ldots, s_n) : \tau & \text{if } Z : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau \in \mathcal{M},\ s_1 : \sigma_1, \ldots, s_n : \sigma_n \end{array}$$

As before, a term in an IDTS is a meta-term without meta-variables, a pattern is a meta-term where all meta-variable occurrences have the form $Z(x_1, \ldots, x_n)$ with all x_i distinct bound variables. Substitution and meta-substitution are as before, with the restriction that they respect types. That is, $\gamma(x) : \sigma$ if $x : \sigma \in \mathcal{V}$ and $\gamma(Z) : \sigma_1 \to \ldots \to \sigma_n \to \tau$ if $Z : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau \in \mathcal{M}$. A context $C[]$ is a term containing some symbol $\Box_\sigma$ of type σ, and $C[s]$ is defined if $s : \sigma$.

Rewrite rules are pairs $l \Rightarrow r$ of closed meta-terms of the same type, such that l is a pattern of the form $f(l_1, \ldots, l_n)$, and given a set of rules $\mathcal{R}$, the rewrite relation $\Rightarrow_{\mathcal{R}}$ is defined as $C[l\gamma] \Rightarrow_{\mathcal{R}} C[r\gamma]$.

Example 4. The typed lambda-calculus can be encoded as an IDTS with for all types σ, τ a symbol $\mathtt{app}_{(\sigma \to \tau \times \sigma) \longrightarrow \tau}$ and a rule $\mathtt{app}(\lambda x.Z(x), X) \Rightarrow Z(X)$ of matching type. Note that we need infinitely many (similar) symbols and rules.

Example 5. The system `map` from Example 3 can be typed, using for instance a type declaration $(\mathtt{nat} \to \mathtt{nat} \times \mathtt{natlist}) \longrightarrow \mathtt{natlist}$ for `map`.

Although types are not explicitly mentioned in their definition, Aczel's Contraction Schemes [1] correspond with second-order IDTs, with a single base type. That is, choosing $\mathcal{B} = \{\mathtt{o}\}$ and defining $\sigma_0 := \mathtt{o},\ \sigma_{n+1} := \mathtt{o} \to \sigma_n$, a contraction scheme is an IDTS where all function symbols have a type declaration $(\sigma_{k_1} \times \ldots \times \sigma_{k_m}) \longrightarrow \mathtt{o}$ and some restrictions on the rules are satisfied.

3.3 Higher-order Rewriting Systems

Next, let us discuss Nipkow's *Higher-order Rewriting Systems*, a simply typed formalism where terms are defined modulo β and η. HRSs were first introduced in [13]. Following Wolfram [19] the restrictions on the rules were dropped in [11].

An HRS directly extends lambda-terms with typed function symbols and rules. Formally, assume given a set $\mathcal{F}$ of typed function symbols and a set $\mathcal{V}$ of typed variables which contains countably many variables of each type. *Pre-terms* over $\mathcal{F}, \mathcal{V}$ are those expressions s for which we can derive $s : \sigma$ using the clauses:

$$\begin{array}{ll} a : \sigma & \text{if } a : \sigma \in \mathcal{V} \cup \mathcal{F} \\ s \cdot t : \tau & \text{if } s : \sigma \to \tau \text{ and } t : \sigma \\ \lambda x.s : \sigma \to \tau & \text{if } x : \sigma \in \mathcal{V} \text{ and } s : \tau \end{array}$$

Note that this only differs from the definition of typed lambda-terms by typed function symbols as well as variables. As in the lambda-calculus, we consider application left-associative and omit unnecessary brackets. A *term* is a pre-term in η-long β-normal form. Every pre-term s corresponds with a unique term $s\updownarrow_\beta^\eta$.

Using the normal definition of substitution, a type-respecting mapping $[x_1 := s_1, \ldots, x_n := s_n]$, the result of applying a substitution γ on a term t is the *pre-term* t with all occurrences of some x_i replaced by s_i. A context is a *term* with a typed $\Box_\sigma$ symbol occurring in it; if σ is a base type then the pre-term $C[s]$ obtained by replacing $\Box_\sigma$ by some term of the same type is also a term.

A rewrite rule is a pair $l \Rightarrow r$ of terms which share the same base type, such that $FVar(r) \subseteq FVar(l)$ and l has the form $f \cdot l_1 \cdots l_n$. The rewrite relation on terms is given by:

$$C[l\gamma\updownarrow_\beta^\eta] \Rightarrow_\mathcal{R} C[r\gamma\updownarrow_\beta^\eta] \quad (l \Rightarrow r \in \mathcal{R},\ \gamma \text{ a substitution},\ C[] \text{ a context})$$

Unfortunately, this rewrite relation is in general not decidable. Hence attention is commonly restricted to the class of *pattern* HRSs, where the left-hand sides of rules are "patterns". A term l is a pattern if for every subterm $x \cdot t_1 \cdots t_n$ with x occurring free in l and $n > 0$, the t_i are the η-long forms of different bound variables. Patterns are defined by Miller [12], who proves that unification (and hence matching) modulo β is decidable for patterns. This restriction is very similar to patterns in CRSs and IDTSs. We will refer to Pattern HRSs as *PRSs*.

Example 6. The typed lambda-calculus can be encoded as a PRS with symbols $\mathtt{app}_{(\sigma\to\tau)\to\sigma\to\tau)}$ for all types σ, τ, and corresponding rules $\mathtt{app} \cdot (\lambda x.y \cdot x) \cdot z \Rightarrow y \cdot z$. Note that $\lambda x.y \cdot x$ in the left-hand side also matches on, for instance, $\lambda x.g(x, x)$, since this can be written as $\lambda x.((\lambda z.g(z, z)) \cdot x)\updownarrow_\beta^\eta$.

Example 7. The typed `map` system from Example 5 can be represented as a PRS with symbols $0 : \mathtt{nat}$, $\mathtt{s} : \mathtt{nat} \to \mathtt{nat}$, $\mathtt{map} : (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{natlist} \to \mathtt{natlist}$, $\mathtt{nil} : \mathtt{natlist}$ and $\mathtt{cons} : \mathtt{nat} \to \mathtt{natlist} \to \mathtt{natlist}$, and rules:

$$\begin{array}{ll} \mathtt{map} \cdot (\lambda x.F \cdot x) \cdot \mathtt{nil} & \Rightarrow \mathtt{nil} \\ \mathtt{map} \cdot (\lambda x.F \cdot x) \cdot (\mathtt{cons} \cdot y \cdot z) & \Rightarrow \mathtt{cons} \cdot (F \cdot y) \cdot (\mathtt{map} \cdot (\lambda x.F \cdot x) \cdot z) \end{array}$$

An example reduction:

$$\begin{array}{ll} & \mathtt{map} \cdot (\lambda x.x) \cdot (\mathtt{cons} \cdot (\mathtt{s} \cdot 0) \cdot \mathtt{nil}) \\ \Rightarrow_\mathcal{R} & (\mathtt{cons} \cdot ((\lambda x.x) \cdot (\mathtt{s} \cdot 0)) \cdot (\mathtt{map} \cdot (\lambda x.x) \cdot \mathtt{nil}))\updownarrow_\beta^\eta \\ = & \mathtt{cons} \cdot (\mathtt{s} \cdot 0) \cdot (\mathtt{map} \cdot (\lambda x.x) \cdot \mathtt{nil}) \\ \Rightarrow_\mathcal{R} & \mathtt{cons} \cdot (\mathtt{s} \cdot 0) \cdot (\mathtt{nil}\updownarrow_\beta^\eta) \\ = & \mathtt{cons} \cdot (\mathtt{s} \cdot 0) \cdot \mathtt{nil} \end{array}$$

3.4 Algebraic Functional Systems

Finally, we turn to Algebraic Functional Systems, as defined in [4] but with simple types, as this is how they are commonly used (using type instantiation, every polymorphic system corresponds to a monomorphic system anyway).

An AFS adds functions and rules to typed lambda-calculus. Given an infinite set of typed variables and a set of function symbols, each equipped with a type declaration, terms are expressions s such that $s : \sigma$ can be derived using:

$$\begin{array}{ll} x : \sigma : \sigma & \text{if } x \in \mathcal{V} \text{ and } \sigma \text{ a type} \\ f(s_1, \ldots, s_n) : \tau & \text{if } f : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau \text{ and } s_1 : \sigma_1, \ldots, s_n : \sigma_n \\ \lambda x.s : \sigma \to \tau & \text{if } x : \sigma \in \mathcal{V} \text{ and } s : \tau \\ s \cdot t : \tau & \text{if } s : \sigma \to \tau \text{ and } t : \sigma \end{array}$$

As usual, term equality is modulo α-conversion. As before, substitutions are type-respecting finite mappings $[x_1 := s_1, \ldots, x_n := s_n]$ which assign variables to terms, and applying a substitution on a term replaces x_i by s_i everywhere. A context $C[]$ is a term with a typed symbol $\Box_\sigma$ in it.

A rewrite rule is a pair $l \Rightarrow r$ of terms with the same type, such that $FVar(r) \subseteq FVar(l)$, and a set of rules $\mathcal{R}$ induces a rewrite relation:

$$\begin{array}{rl} C[l\gamma] \Rightarrow_{\mathcal{R}} C[r\gamma] & \text{for } l \Rightarrow r \in \mathcal{R},\ \gamma \text{ a substitution} \\ C[(\lambda x.s) \cdot t] \Rightarrow_{\mathcal{R}} C[s[x := t]] & \end{array}$$

So β-reduction is always added as a separate step.

Example 8. Since $\Rightarrow_\beta$ is added as an explicit rewrite step in every AFS, the typed lambda-calculus does not need to be encoded; it is the AFS with $\mathcal{R} = \emptyset$.

Example 9. The running example `map` can be represented as an AFS with function symbols $0 : \mathtt{nat}$, $\mathtt{s} : (\mathtt{nat}) \longrightarrow \mathtt{nat}$, $\mathtt{map} : ((\mathtt{nat} \to \mathtt{nat}) \times \mathtt{natlist}) \longrightarrow \mathtt{natlist}$, $\mathtt{cons} : (\mathtt{nat} \times \mathtt{natlist}) \longrightarrow \mathtt{natlist}$ and the following rules:

$$\begin{array}{ll} \mathtt{map}(F, \mathtt{nil}) & \Rightarrow \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, y)) & \Rightarrow \mathtt{cons}(F \cdot x, \mathtt{map}(F, y)) \end{array}$$

An example reduction:

$$\begin{array}{l} \mathtt{map}(\lambda x.x, \mathtt{cons}(\mathtt{s}(0), \mathtt{nil})) \\ \quad \Rightarrow_{\mathcal{R}} \mathtt{cons}((\lambda x.x) \cdot \mathtt{s}(0), \mathtt{map}(\lambda x.x, \mathtt{nil})) \\ \quad \Rightarrow_{\beta} \mathtt{cons}(\mathtt{s}(0), \mathtt{map}(\lambda x.x, \mathtt{nil})) \\ \quad \Rightarrow_{\mathcal{R}} \mathtt{cons}(\mathtt{s}(0), \mathtt{nil}) \end{array}$$

In the second step, we indicated that a β-step was used for clarity; it could also just have been written as $\Rightarrow_{\mathcal{R}}$.

Although there are no direct restrictions on rule formation, [8] gives us some assumptions we may safely make:

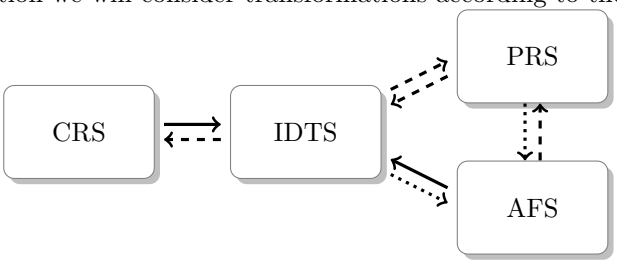
Lemma 1 (Result from [8]). *A set of AFS-rules can always be transformed (without affecting termination) so as to satisfy one or more of the following requirements:*

1. *the left-hand sides of rules are not abstractions*
2. *the left-hand sides of rules have no subterms of the form $x \cdot s$ with x a free variable*
3. *the left-hand sides of rules are not variables*
4. *both sides of the rules are β-normal*

In addition, if these requirements are satisfied, presenting the rules in applicative notation has no effect on termination.

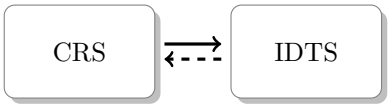
4 Transformations

Let us consider, now, how all these formalisms compare to each other. In this section we will consider transformations according to the following graph:



A solid arrow indicates that systems in the first formalism can be represented in the second without affecting termination. A dashed arrow indicates that systems in the first can be represented in the second without losing non-termination (but termination may be lost). A dotted arrow indicates that *some* systems in the first can be represented in the second without losing non-termination.

4.1 CRSs and IDTSs



Due to the untyped nature of CRSs, they are not overly interesting for general termination research; many intuitively innocent systems are non-terminating. For instance, the simple `map` example admits an infinite reduction: defining $\omega := \lambda x.\mathtt{map}(x, \mathtt{cons}(x, \mathtt{nil}))$ and $\Omega := \mathtt{map}(\omega, \mathtt{cons}(\omega, \mathtt{nil}))$ we have $\Omega \Rightarrow_{\mathcal{R}} \mathtt{cons}(\Omega, \mathtt{map}(\omega, \mathtt{nil}))$. In fact, any rule where the right-hand side contains a subterm $Z(X)$ is problematic. Nevertheless, we can say a few words on how termination relates to other systems, in particular IDTSs.

To start, note that the definitions of CRSs and IDTSs correspond; every IDTS is also a CRS, but with restrictions on the term formation. Thus, an IDTS is terminating if the CRS it corresponds to is. This provides the dashed arrow. A weak inclusion indeed, since termination will commonly be lost by dropping type

restrictions. For the other direction, we cannot just add types; both $f(0)$ and $f(\lambda x.0)$ are valid terms – not both can be typable. We can, however, encode CRS terms. To this end we will use a special symbol `flat` to "flatten" abstractions.

Transformation 1 *(Transforming a CRS into an IDTS)* Given a CRS $(\mathcal{F}, \mathcal{R})$, let $\mathcal{B} = \{\mathtt{o}\}$ and define type declarations $\sigma_n = (\mathtt{o} \times \ldots \times \mathtt{o}) \longrightarrow \mathtt{o}$ with n occurrences of $\mathtt{o}$ before the $\longrightarrow$. Let $\mathcal{F}^{\mathtt{flat}} = \{\mathtt{flat} : (\mathtt{o} \to \mathtt{o}) \longrightarrow \mathtt{o}\} \cup \{f' : \sigma_n | f \in \Sigma,\ ar(f) = n\}$ and assume every CRS-variable x corresponds with an IDTS-variable x' of type $\mathtt{o}$, and every meta-variable Z of arity n with an IDTS-meta-variable with type declaration σ_n. Let φ be the function mapping CRS-style (meta-)terms to IDTS-style (meta-)terms as follows:

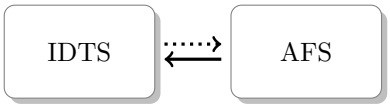
$$\begin{aligned}
\varphi(x) &= x' \ (x \text{ a variable}) \\
\varphi(f(s_1, \ldots, s_n)) &= f'(\varphi(s_1), \ldots, \varphi(s_n)) \ (f \in \mathcal{F}, ar(f) = n) \\
\varphi(Z(s_1, \ldots, s_n)) &= Z'(\varphi(s_1), \ldots, \varphi(s_n)) \ (Z \in \mathcal{M}, ar(Z) = n) \\
\varphi(\lambda x.s) &= \mathtt{flat}(\lambda x'.\varphi(s))
\end{aligned}$$

Let $\mathcal{R}^{\mathtt{CI}} := \{\varphi(l) \Rightarrow \varphi(r) | l \Rightarrow r \in \mathcal{R}\}$.

Theorem 1. *The CRS $(\mathcal{F}, \mathcal{R})$ is terminating if and only if the IDTS $(\mathcal{F}^{\mathtt{CI}}, \mathcal{R}^{\mathtt{CI}})$ is terminating.*

Proof. It is not hard to see that if $s \Rightarrow_{\mathcal{R}} t$ then also $\varphi(s) \Rightarrow_{\mathcal{R}^{\mathtt{CI}}} \varphi(t)$; thus, if the IDTS is terminating, so is the CRS. For the other direction, if $s \Rightarrow_{\mathcal{R}^{\mathtt{CI}}} t$ it is not hard to derive that also $\psi(s) \Rightarrow_{\mathcal{R}} \psi(t)$, where $\psi(u)$ is obtained from u by dropping types and replacing occurrences of $\mathtt{flat}(v)$ by v.

4.2 IDTSs and AFSs



Next, let us consider how IDTSs and AFSs relate. Both systems have a form of typing and the same way of constructing functional terms. However, there are a number of fundamental differences. In IDTSs, terms are purely functional; there is no application operator. Therefore it is not, in general, possible to form terms of arbitrary types. This may have a crucial impact on termination. Consider, for example, the following IDTS:

$$\mathcal{F} = \left\{ \begin{array}{l} a : \mathtt{nat} \\ g : (\mathtt{nat} \times \mathtt{nat}) \longrightarrow \mathtt{nat} \\ f : (\mathtt{nat} \times \mathtt{nat}) \longrightarrow \mathtt{bool} \\ h : (\mathtt{nat}) \longrightarrow \mathtt{bool} \end{array} \right\} \quad \mathcal{R} = \left\{ \begin{array}{l} f(a, g(\lambda xy.Z(x, y))) \\ \Rightarrow h(Z(a, g(\lambda xy.Z(x, y)))) \end{array} \right\}$$

With some inductive reasoning, we can see that a purely functional term of type `nat` cannot contain the symbol f anywhere and therefore cannot be reduced; consequently there is no reduction of more than one step. However, if

we add application to the system, we obtain an infinite reduction by instantiating $Z(x,y)$ with $z_{\mathtt{bool}\to\mathtt{nat}} \cdot f(x,y)$:

$$\begin{aligned}
&f(a, g(\lambda xy.z \cdot f(x,y)))\\
&\Rightarrow_{\mathcal{R}} h(z \cdot \underline{f(a, g(\lambda xy.z \cdot f(x,y)))})\\
&\Rightarrow_{\mathcal{R}} h(z \cdot h(z \cdot \underline{f(a, g(\lambda xy.z \cdot f(x,y)))}))\\
&\Rightarrow_{\mathcal{R}} \dots
\end{aligned}$$

Even when there is only one base type, the lack of application might make a fundamental difference, if the function symbols have an arity at most one:

$$\Sigma = \left\{ \begin{array}{l} 0 : \mathtt{nat} \\ f : (\mathtt{nat} \times \mathtt{nat}) \longrightarrow \mathtt{nat} \\ g : (\mathtt{nat} \times \mathtt{nat}) \longrightarrow \mathtt{nat} \end{array} \right\} \quad \mathcal{R} = \left\{ \begin{array}{l} f(\lambda xy.Z(x,y)) \Rightarrow Z(g(\lambda xy.Z(x,y)), 0) \\ g(\lambda xy.Z(x,y)) \Rightarrow Z(0, f(\lambda xy.Z(x,y))) \end{array} \right\}$$

The reason why this example is troublesome is that, due to the limited arity of function symbols, any term can only have one "leaf" symbol, something which is not the case when application is permitted. It takes a bit of reasoning to see that this system is indeed terminating. Intuitively, if the leaf symbol of a term s is not a bound variable, then any rewrite step decreases the size of the term. If it *is* a bound variable x, then there are two possible non-decreasing steps:

- $s = C[f(\lambda xy.s')] \Rightarrow_{\mathcal{R}} C[s'[x := g(\lambda xy.s')]]$
- $s = C[g(\lambda yx.s')] \Rightarrow_{\mathcal{R}} C[s'[x := f(\lambda yx.s')]]$

Here, y is ignored because it can't occur in s' when s' already contains x. After this step, the occurrence of x in s' is still the only leaf in the term, but now is no longer bound in one of the above contexts. Thus, any further step decreases the size of the term, so reductions must terminate. However, if we add application there is a trivial counterexample for termination: instantiate $Z(x,y)$ by $z \cdot x \cdot y$. Then:

$$\begin{aligned}
&\underline{f(\lambda xy.z \cdot x \cdot y)}\\
&\Rightarrow_{\mathcal{R}} z \cdot g(\lambda xy.z \cdot x \cdot y) \cdot 0\\
&\Rightarrow_{\mathcal{R}} z \cdot (z \cdot 0 \cdot \underline{f(\lambda xy.z \cdot x \cdot y)}) \cdot 0\\
&\Rightarrow_{\mathcal{R}} \dots
\end{aligned}$$

Another issue is that AFS-rules do not use meta-variables. An AFS has no way to express a rule like this example (copied from [2]):

$$\mathtt{d}(\lambda x.\mathtt{sin}(Z(x))) \Rightarrow \lambda x.(\mathtt{d}(\lambda y.Z(y))) \times \mathtt{cos}(Z(x))$$

However, all this relates to problems embedding IDTSs into AFSs. For the other direction, it is easy enough, if we are willing to accept an infinite system.

Transformation 2 *(Transforming an AFS into an IDTS)* Given an AFS $(\mathcal{F}, \mathcal{R})$, let $\mathcal{F}^{\mathtt{AI}} = \mathcal{F} \cup \{\mathtt{app}^{\sigma,\tau} : (\sigma \to \tau \times \sigma) \longrightarrow \tau | \sigma, \tau \in \mathcal{T}\}$ and $\mathcal{R}^{\mathtt{AI}} = \{\varphi(l \Rightarrow r) | l \Rightarrow r \in \mathcal{R}\} \cup \{\mathtt{app}^{\sigma,\tau}(\lambda x.Z(x), X) \Rightarrow Z(X) | \sigma, \tau \in \mathcal{T}\}$. Here, $\phi(l \Rightarrow r)$ replaces every free variable $x : \sigma$ in l and r by a corresponding meta-variable

$Z : () \longrightarrow \sigma$, and replaces applications $s \cdot t$ with $s : \sigma \to \tau$ by $\mathtt{app}^{\sigma,\tau}(s,t)$.

AFS-terms over $\mathcal{F}$ corespond one-on-one to IDTS-terms over $\mathcal{R}^{\mathtt{AI}}$ and a step in one system corresponds with a step in the other. Consequently:

Theorem 2. *$(\mathcal{F},\mathcal{R})$ is a terminating AFS if and only if $(\mathcal{F}^{\mathtt{AI}},\mathcal{R}^{\mathtt{AI}})$ is a terminating IDTS.*

The other direction is more difficult, for all the reasons given above. However, we can obtain a *partial* embedding. If we are willing to surrender completeness, we can transform IDTSs of a certain form into an AFS which is non-terminating if the original system is.

We say a meta-term s has *simple meta-applications* if all meta-variables in s occur in a form $\lambda x_1 \ldots x_n.Z(x_1,\ldots,x_n)$. Thus, a meta-term $\mathtt{map}(\lambda x.Z(x),\mathtt{cons}(X,Y))$ has simple meta-applications, whereas $\mathtt{d}(\lambda x.\mathtt{sin}(Z(x)))$ does not.

Transformation 3 *(Transforming an IDTS with simple meta-applications into an AFS)* Given an IDTS $(\mathcal{F},\mathcal{R})$ such that for all $l \Rightarrow r \in \mathcal{R}$, the left-hand side l has simple meta-applications. Choose for all meta-variables $Z : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$ a corresponding variable $y_Z : \sigma_1 \to \ldots \to \sigma_n \to \tau$, and define for any IDTS-meta-term s the AFS-term $\varphi(s)$ as follows: every occurrence $\lambda x_1 \ldots x_n.Z(x_1,\ldots,x_n)$ $(n \geq 0)$ is replaced by y_Z, and every remaining sub-meta-term $Z(s_1,\ldots,s_n)$ is replaced by $y_Z \cdot s_1 \cdots s_n$. Define $\mathcal{R}^{\mathtt{IA}} = \{\varphi(l) \Rightarrow \varphi(r) | l \Rightarrow r \in \mathcal{R}\}$.

As an example, Transformation 3 transforms the IDTS-rule:

$$\mathtt{map}(\lambda x.Z(x),\mathtt{cons}(X,Y)) \Rightarrow \mathtt{cons}(Z(X),\mathtt{map}(\lambda x.Z(x),Y))$$

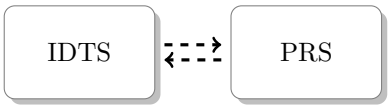
into the AFS-rule:

$$\mathtt{map}(y_Z,\mathtt{cons}(y_X,y_Y)) \Rightarrow \mathtt{cons}(y_Z \cdot y_X,\mathtt{map}(y_Z,y_Y))$$

Theorem 3. *If the AFS $(\mathcal{F},\mathcal{R}^{\mathtt{IA}})$ is terminating, then the IDTS $(\mathcal{F},\mathcal{R})$ is terminating.*

Proof. Every term in the IDTS over $\mathcal{F}$ is also a term in the corresponding AFS, and if there is an infinite reduction in the latter, there is an infinite reduction in the former because $s \Rightarrow_{\mathcal{R}} t$ implies $s \Rightarrow_{\mathcal{R}^{\mathtt{IA}}} \cdot \Rightarrow_\beta^* t$. This is because, if $s = C[l\gamma]$ and $t = C[r\gamma]$ for some rule $l \Rightarrow r \in \mathcal{R}$, and $l' \Rightarrow r' = \varphi(l \Rightarrow r)$, then $s = C[l'\delta] \Rightarrow_{\mathcal{R}^{\mathtt{IA}}} C[r'\delta] \Rightarrow_\beta^* C[r\gamma]$. Here, if $\gamma = [Z_1 := u_1,\ldots,Z_n := u_n]$, then $\delta = [y_{Z_1} := u_1,\ldots,y_{Z_n} := u_n]$.

4.3 IDTSs and PRSs



Now, consider pattern HRSs. In this formalism rewriting is modulo β and η, which (as we shall see) in practice means that terms are η-long, and after every reduction a term is β-normalised. Here, too, the presence of application allows formation of complex terms of arbitrary types, something which cannot be done in IDTSs.

HRSs can be embedded into IDTSs in a somewhat way as AFSs: we have to add special application symbols and rules to reduce them. However, we also have to deal with more advanced matching, and the fact that IDTS-terms are entirely applicative.

Transformation 4 *(Transforming a PRS into an IDTS)* Given a PRS $(\mathcal{F}, \mathcal{R})$, let $\mathcal{F}^{\mathtt{func}} = \{f' : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \iota | f : \sigma_1 \to \ldots \to \sigma_n \to \iota \in \mathcal{F} | \iota \in \mathcal{B}\}$ and $\mathcal{F}^{\mathtt{PI}} = \mathcal{F}^{\mathtt{func}} \cup \{\mathtt{app}^\tau | (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \iota | \tau = \sigma_1 \to \ldots \to \sigma_n \to \iota \in \mathcal{T}\}$. For all variables $x : \sigma_1 \to \ldots \sigma_n \to \iota$ choose a uniquely corresponding IDTS-meta-variable $Z_x : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \iota$ (note that ι always indicates a base type). Now define a function φ mapping HRS-terms to IDTS-terms as follows: given a set S of variables,

$$
\begin{array}{lll}
\varphi_S(\lambda x.s) & = \lambda x.\varphi_S(s) & \\
\varphi_S(f \cdot s_1 \cdots s_n) & = f'(\varphi_S(s_1), \ldots, \varphi_S(s_n)) & f \in \mathcal{F} \\
\varphi_S(x \cdot s_1 \cdots s_n) & = \mathtt{app}^\sigma(x, \varphi_S(s_1), \ldots, \varphi_S(s_n)) & x : \sigma \in \mathcal{V} \setminus S \\
\varphi_S(x \cdot y_1 \uparrow^\eta \cdots y_n \uparrow^\eta) & = Z_x(y_1, \ldots, y_n) & x \in S,\ y_1, \ldots, y_n \in \mathcal{V} \setminus S \\
\varphi_S(x \cdot s_1 \cdots s_n) & = Z_x(\psi_S(s_1), \ldots, \psi_S(s_n)) & \text{otherwise}
\end{array}
$$

$$
\begin{array}{l}
\text{Let } \mathcal{R}^{\mathtt{PI}} := \{\varphi_{FVar(l)}(l) \Rightarrow \varphi_{FVar(l)}(r) | l \Rightarrow r \in \mathcal{R}\} \cup \\
\qquad \{\mathtt{app}^\sigma(\lambda x_1 \ldots x_n.Z(x_1, \ldots, x_n), X_1, \ldots, X_n) \Rightarrow \\
\qquad\quad Z(X_1, \ldots, X_n) \mid \sigma = \sigma_1 \to \ldots \to \sigma_n \to \iota \in \mathcal{T},\ n > 0\}
\end{array}
$$

The φ_S function in this transformation does three things:

- an application headed by a function symbol, $f \cdot s_1 \cdots s_n$, is mapped to a functional term $f(s_1, \ldots, s_n)$;
- an application headed by a variable symbol, $x \cdot s_1 \cdots s_n$, is mapped to a special functional term $\mathtt{app}(x, s_1, \ldots, s_n)$; if x is intantiated by an abstraction, such a term can be "β-reduced" by corresponding rules;
- in rules, the free variables used for matching are replaced by meta-variables, in such a way that PRS-patterns are mapped to IDTS-patterns.

Theorem 4. *If the IDTS $(\mathcal{F}^{\mathtt{PI}}, \mathcal{R}^{\mathtt{PI}})$ is terminating, then so is the PRS $(\mathcal{F}, \mathcal{R})$.*

Proof. Write $\Rightarrow_{\mathtt{beta}}$ for a reduction $C[\mathtt{app}(\lambda \boldsymbol{x}.s, \boldsymbol{t})] \Rightarrow C[s[\boldsymbol{x} := \boldsymbol{t}]]$. Using induction on the pre-term $s[\boldsymbol{x} := \boldsymbol{t}]$ ordered by the union of $\Rightarrow_\beta$ and subterm reduction, it follows easily that always $\varphi_\emptyset(s[\boldsymbol{x} := \boldsymbol{t}]) \Rightarrow^*_{\mathtt{beta}} \varphi_\emptyset(s)[\boldsymbol{x} := \varphi_\emptyset(\boldsymbol{t})]$. Having this, induction on the form of s shows that $\varphi_S(s)[\boldsymbol{Z_x} := \boldsymbol{t}] \Rightarrow^*_{\mathtt{beta}} \varphi_\emptyset(s[\boldsymbol{x} := \boldsymbol{t}])$ if $S \supseteq FVar(s)$ and this is even an equality if s is a pattern. Consequently, $s \Rightarrow_{\mathcal{R}} t$ implies $\varphi_\emptyset(s) \Rightarrow^+_{\mathcal{R}^{\mathtt{PI}}} \varphi_\emptyset(t)$.

It should be noted that IDTS-terms over $\mathcal{F}^{\mathtt{PI}}$ correspond with *pre-terms* over $\mathcal{F}$, not just terms. This is the reason why Theorem 4 is only one-way: non-termination may appear by reducing terms which are not "β-normal". An example of a terminating PRS, with a corresponding non-terminating IDTS, is:

$$\begin{aligned} \mathtt{h} \cdot 0 &\Rightarrow \mathtt{g} \cdot (\lambda xy.x \cdot (\mathtt{h} \cdot y)) \qquad (x : \mathtt{nat} \to \mathtt{nat}) \\ \mathtt{g} \cdot (\lambda xy.Z \cdot (\lambda z.Z \cdot z) \cdot y) &\Rightarrow Z \cdot (\lambda z.0) \cdot 0 \end{aligned}$$

$$\begin{aligned} \mathtt{h}'(0') &\Rightarrow \mathtt{g}'(\lambda xy.\mathtt{app}^{\mathtt{nat}\to\mathtt{nat}}(x, \mathtt{h}'(y))) \\ \mathtt{g}'(\lambda xy.Z(x,y)) &\Rightarrow Z(\lambda z.0', 0') \\ \mathtt{app}^{\sigma}(\lambda \boldsymbol{x} Z(\boldsymbol{x}).., \boldsymbol{X}) &\Rightarrow Z(\boldsymbol{X}) \qquad (\text{all } \sigma) \end{aligned}$$

The IDTS-version is non-terminating because $\mathtt{h}'(0')$ reduces in two steps to $\mathtt{app}^{\mathtt{nat}\to\mathtt{nat}}(\lambda z.0', \mathtt{h}'(0'))$, which contains the original term. In the original PRS, the pre-term corresponding to this term is immediately β-reduced, which causes the problematic subterm $\mathtt{h}'(0')$ to disappear.

In this case the problem is somewhat obvious: the rules introduce the β-redex. But what if the system is second-order? Could this still happen? Might we have an equivalence in *some* cases?

Unfortunately, it is rather hard to give any kind of condition for β-reduction not to be needed. Consider for example the following IDTS:

$$\begin{aligned} \mathtt{f}_1(Z) &\Rightarrow \mathtt{f}_2(Z, Z) \\ \mathtt{f}_2(\mathtt{a}, Z) &\Rightarrow \mathtt{f}_3(Z) \\ \mathtt{g}_1(\lambda x.\mathtt{f}_3(Z(x))) &\Rightarrow Z(\mathtt{hide}(\lambda x.\mathtt{f}_3(Z(x)))) \\ \mathtt{unhide}(\mathtt{hide}(\lambda x.Z(x))) &\Rightarrow \mathtt{g}_2(\lambda x.Z(x)) \\ \mathtt{g}_2(\lambda x.\mathtt{f}_3(Z(x))) &\Rightarrow \mathtt{g}_1(\lambda x.\mathtt{f}_1(Z(x))) \\ \mathtt{app}^{\sigma}(\lambda \boldsymbol{x} Z(\boldsymbol{x}).., \boldsymbol{X}) &\Rightarrow Z(\boldsymbol{X}) \quad (\text{all } \sigma) \end{aligned}$$

This (admittedly highly artificial) system can be obtained as a translation of a PRS, and has the property that any term that is `beta`-normal can only be reduced to other `beta`-normal terms. Yet non-termination is caused by the possibility of non-β-normal terms. Let $\chi[y] := \mathtt{app}^{\mathtt{nat}\to\mathtt{nat}}(\lambda x.\mathtt{a}, \mathtt{unhide}(x))$. Then:

$$\begin{array}{lll} & \mathtt{g}_1(\lambda y.\mathtt{f}_1(\chi[y])) & \Rightarrow_{\mathcal{R}} \mathtt{g}_1(\lambda y.\mathtt{f}_2(\chi[y], \chi[y])) \\ \Rightarrow_{\mathcal{R}} & \mathtt{g}_1(\lambda y.\mathtt{f}_2(\mathtt{a}, \chi[y])) & \Rightarrow_{\mathcal{R}} \mathtt{g}_1(\lambda y.\mathtt{f}_3(\chi[y])) \\ \Rightarrow_{\mathcal{R}} & \chi[\mathtt{hide}(\lambda y.\mathtt{f}_3(\chi[y]))] & = \quad \mathtt{app}^{\mathtt{nat}\to\mathtt{nat}}(\lambda x.\mathtt{a}, \mathtt{unhide}(\mathtt{hide}(\lambda y.\mathtt{f}_3(\chi[y])))) \\ \Rightarrow_{\mathcal{R}} & \mathtt{app}^{\mathtt{nat}\to\mathtt{nat}}(\lambda x.\mathtt{a}, \mathtt{g}_2(\lambda y.\mathtt{f}_3(\chi[y]))) & \Rightarrow_{\mathcal{R}} \mathtt{app}^{\mathtt{nat}\to\mathtt{nat}}(\lambda x.\mathtt{a}, \underline{\mathtt{g}_1(\lambda y.\mathtt{f}_1(\chi[y]))}) \end{array}$$

The crux of the example is that due to the `beta`-rule there is a term that reduces to `a` but also has a subterm $\mathtt{unhide}(y)$; no counterpart on terms without such a redex exists.

These examples show that while termination results on IDTSs can be used for PRSs, the translation may occasionally lose termination – in fact, a PRS is terminating if and only if the corresponding IDTS is terminating with the reduction strategy "reduce subterms headed by some $\mathtt{app}^{\sigma}$ first, if possible".

Let us consider the other direction. As indicated in the diagram, we can embed IDTSs into PRSs, although again we may lose termination.

Transformation 5 *(Transforming an IDTS into a PRS)* Given an IDTS $(\mathcal{F}, \mathcal{R})$. For any given type declaration $\sigma = (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$, let σ' be the type $\sigma_1 \to \ldots \to \sigma_n \to \tau$, and let $\mathcal{F}^{\mathtt{IP}} := \{f' : \sigma' | f : \sigma \in \mathcal{F}\}$. For all meta-variables $Z : \sigma$, let $x_Z : \sigma'$ be a variable of corresponding type. Now, for a given meta-term s, let $\varphi(s)$ be the PRS-pre-term defined as follows:

$$\begin{aligned} \varphi(x) &= x \\ \varphi(f(s_1, \ldots, s_n)) &= f' \cdot \varphi(s_1) \cdots \varphi(s_n) \\ \varphi(\lambda x.s) &= \lambda x.\varphi(s) \\ \varphi(Z(s_1, \ldots, s_n)) &= x_Z \cdot \varphi(s_1) \cdots \varphi(s_n) \end{aligned}$$

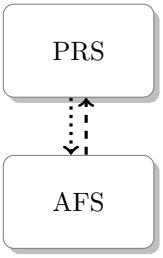
We define $\mathcal{R}^{\mathtt{IP}} := \{\varphi(l) \cdot x_1 \cdots x_n{\uparrow^\eta} \Rightarrow \varphi(r) \cdot x_1 \cdots x_n{\uparrow^\eta} \mid l \Rightarrow r \in \mathcal{R},\ l : \sigma_1 \to \ldots \to \sigma_n \to \iota,\ \iota \in \mathcal{B},\ x_1, \ldots, x_n \text{ fresh}\}$.

Theorem 5. *If the PRS $(\mathcal{F}^{\mathtt{IP}}, \mathcal{R}^{\mathtt{IP}})$ is terminating, then so is the IDTS $(\mathcal{F}, \mathcal{R})$.*

Proof. Write $\varphi^\uparrow(s)$ for $\varphi(s){\uparrow^\eta}$. With induction on the form of s it is easy to see that $\varphi^\uparrow(s)[\boldsymbol{x} := \varphi^\uparrow(\boldsymbol{t})] = \varphi^\uparrow(s[\boldsymbol{x} := \boldsymbol{t}])$, and $\varphi^\uparrow(s)[\boldsymbol{x_Z} := \varphi^\uparrow(\boldsymbol{t})] = \varphi^\uparrow(s[\boldsymbol{x} := \boldsymbol{t}])$. As a consequence, it follows that $\varphi^\uparrow(s) \Rightarrow_{\mathcal{R}^{\mathtt{IP}}} \varphi^\uparrow(t)$ whenever $s \Rightarrow_{\mathcal{R}} t$, so any infinite reduction in the latter system leads to an infinite reduction in the former.

The examples in Section 4.2 demonstrate that the implication in the other direction does not hold.

4.4 PRSs and AFSs



Finally, let us consider transformations between HRSs and AFSs. This is not entirely necessary: we could transform one into the other by going through IDTSs. The reason to consider a separate embedding is to make the result slightly simpler: both AFSs and HRSs use application natively, while in IDTSs it must be simulated with a function symbol. Consequently, if an AFS with a rule $l \Rightarrow x \cdot y_1 \cdot y_2 \cdot y_3$ is embedded into an IDTS, the right-hand side becomes $\mathtt{app}(\mathtt{app}(\mathtt{app}(x, y_1), y_2), y_3)$; a direct transformation into an HRS would have a right-hand side more like $\mathtt{app}(x, y_1, y_2, y_3)$.

To business. In [8] it has been demonstrated that an AFS may be η-expanded without losing non-termination (although termination may sometimes be lost). Defining $s{\uparrow^\eta_V}$ as the normal form of s under η-expansion, where variables in V are not expanded, [8, Theorem 7] gives:

Lemma 2 (η-expanding AFS-rules). *Let $(\mathcal{F}, \mathcal{R})$ be an AFS that satisfies requirements 1–4 from Lemma 1, and which is presented in applicative form (so function symbols take no arguments).*

Let $\mathcal{R}^{\uparrow}$ be the set consisting of rules $\{(l \cdot x_1 \cdots x_n)\uparrow^{\eta}_{V} \Rightarrow (r \cdot x_1 \cdots x_n)\uparrow^{\eta}_{V}$ for $l \Rightarrow r \in \mathcal{R},\ l : \sigma_1 \to \ldots \to \sigma_n \to \iota$ with ι a data type, all $x_i : \sigma_i$ fresh variables and $V := FVar(l) \cup \{x_1, \ldots, x_n\}$.

Then $(\mathcal{F}, \mathcal{R})$ is terminating if there is no infinite $\Rightarrow_{\mathcal{R}^{\uparrow}}$ reduction on η-long terms over $\mathcal{F}$.

As an example of what this theorem gives, consider an AFS with rules:

$$\begin{aligned} \mathtt{repeat}(Z, \mathtt{0}) &\Rightarrow \lambda x.x \\ \mathtt{repeat}(Z, \mathtt{s}(x)) &\Rightarrow \mathtt{op}(Z, \mathtt{repeat}(Z, x)) \end{aligned}$$

Taking the applicative form and η-expanding this we obtain a system with rules:

$$\begin{aligned} \mathtt{repeat} \cdot Z \cdot \mathtt{0} \cdot y &\Rightarrow (\lambda x.x) \cdot y \\ \mathtt{repeat} \cdot Z \cdot (\mathtt{s} \cdot x) \cdot y &\Rightarrow \mathtt{op} \cdot Z \cdot (\lambda z.\mathtt{repeat} \cdot Z \cdot x \cdot z) \cdot y \end{aligned}$$

Transformation 6 *(Transforming an applicative η-long AFS into a PRS)* Given an applicative η-long AFS $(\mathcal{F}, \mathcal{R})$ as formed in Lemma 2, let $\mathcal{F}^{\mathtt{AP}} = \mathcal{F} \cup \{\mathtt{app}^{\sigma} : \sigma \to \sigma \mid \sigma \in \mathcal{T},\ \sigma \text{ composed}\}$. For any term s, let $\varphi(s)$ be inductively defined as follows:

$$\begin{array}{lll} \varphi(x) & = x\uparrow^{\eta} & (x \in \mathcal{V}) \\ \varphi(x \cdot s_0 \cdots s_n) & = \mathtt{app}^{\sigma} \cdot \varphi(x), \varphi(s_0), \ldots, \varphi(s_n) & (x : \sigma \in \mathcal{V}) \\ \varphi(\lambda x.s) & = \lambda x.\varphi(s) & \\ \varphi((\lambda x.s) \cdot t_0 \cdots t_n) & = \mathtt{app}^{\sigma} \cdot (\lambda x.\varphi(s)) \cdot \varphi(t_0) \cdots \varphi(t_n) & \lambda x.s : \sigma \\ \varphi(f \cdot s_0 \cdots s_n) & = f \cdot \varphi(s_0) \cdots \varphi(s_n) & f \in \mathcal{F} \end{array}$$

$$\begin{aligned} \text{Define } \mathcal{R}^{\mathtt{AP}} = \{&\varphi(l) \Rightarrow \varphi(r) \mid l \Rightarrow r \in \mathcal{R}\} \cup \\ \{&\mathtt{app}^{\sigma \to \iota} \cdot x \cdot y\uparrow^{\eta} \Rightarrow x \cdot y\uparrow^{\eta} \mid \sigma \in \mathcal{T}, \iota \in \mathcal{B}\} \cup \\ \{&\mathtt{app}^{\sigma \to \tau} \cdot x \cdot y \cdot \boldsymbol{z}\uparrow^{\eta} \Rightarrow \mathtt{app}^{\tau} \cdot (x \cdot y) \cdot \boldsymbol{z}\uparrow^{\eta} \mid \sigma, \tau \in \mathcal{T},\ \tau \notin \mathcal{B}\} \end{aligned}$$

It is easy to see that any AFS-term which is in η-long form except (possibly) for some variables, is mapped to a PRS-term. Moreover, any reduction step is preserved, and every PRS-term uniquely corresponds with an η-long AFS-term. We obtain:

Theorem 6. *The (applicative, η-long) AFS $(\mathcal{F}, \mathcal{R})$ is terminating if and only if the corresponding PRS $(\mathcal{F}, \mathcal{R}^{\mathtt{AP}})$ is.*

The other direction, embedding a PRS into the AFS formalism, brings nothing new. Going via IDTSs, a PRS can be transformed into an AFS, without losing non-termination, as long as the free variables x in the left-hand sides of the rules only occur in the form $\lambda y_1 \ldots y_n.x \cdot y_1\uparrow^{\eta} \cdots y_n\uparrow^{\eta}$ (with $n \geq 0$). However, this transformation introduces an infinite number of symbols $\mathtt{app}^{\sigma}$ and

rules corresponding with them; in AFSs this is unnecessary, because application is already part of the formalism.

Transformation 7 *(Transforming a PRS with simple free variables into an AFS)* Given a PRS $(\mathcal{F}, \mathcal{R})$ such that for all $l \Rightarrow r \in \mathcal{R}$ any free variables x in l occur only in the form $\lambda y_1 \ldots y_n.x \cdot y_1{\uparrow^\eta} \cdots y_n{\uparrow^\eta}$ (with $n \geq 0$). Let $\mathcal{F}^{\mathtt{PA}} = \{f' : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \iota \mid f : \sigma_1 \to \ldots \to \sigma_n \to \iota \in \mathcal{F},\ \iota \in \mathcal{B}\}$, and for any PRS-term s and set of variables V, let $\varphi_V(s)$ be the AFS-term s with all occurrences of some $\lambda x_1 \ldots x_n.y \cdot x_1{\uparrow^\eta} \cdots x_n{\uparrow^\eta}$ with $y \in V$ replaced by just y. Define $\mathcal{R}^{\mathtt{PA}} = \{\varphi_{FVar(l)}(l) \Rightarrow \varphi_{FVar(r)} \mid l \Rightarrow r \in \mathcal{R}\}$.

Theorem 7. *The PRS $(\mathcal{F}, \mathcal{R})$ is terminating if and only if the AFS $(\mathcal{F}^{\mathtt{PA}}, \mathcal{R}^{\mathtt{PA}})$ is terminating using a β-first reduction strategy.*

A β-first reduction strategy means that a term is reduced with a β-step if this is possible.

5 Case Study: Static Dependency Pairs

To see how these transformations work in practice, let us consider a recent result: the static dependency pairs approach. Static Dependency Pairs were defined on Nipkow's HRSs in [10], and extended to use argument filterings and usable rules in [17]. We will discuss only a simplified version of the first definition.

Given an HRS $(\mathcal{F}, \mathcal{R})$, the set $\mathcal{D}$ of *defined symbols* consists of those $f \in \mathcal{F}$ for which a rule $l \Rightarrow r$ exists with $l = f \cdot l_1 \cdots l_n$. Let $\mathcal{F}^\# := \mathcal{F} \cup \{f^\# : \sigma \mid f : \sigma \in \mathcal{D}\}$.

A PRS-rule $l \Rightarrow r$ is called *plain function passing* (PFP) if any higher-order variables occur at an argument position of l. That is, writing $l = f \cdot l_1 \cdots l_n$, if $x \in FVar(l)$ has composed type, then $x{\uparrow^\eta} = l_i$ for some i. *(This definition is a bit simplified from the original in [10], because we restrict to patterns.)*

A pair of terms $l = f \cdot l_1 \cdots l_n$ and r generate a set of *dependency pairs* as follows:

$$\begin{array}{ll}
\mathtt{DP}(f \cdot l_1 \cdots l_n, \lambda x.r) & = \mathtt{DP}(f \cdot l_1 \cdots l_n, r) \\
\mathtt{DP}(f \cdot l_1 \cdots l_n, g \cdot r_1 \cdots r_m) & = \{f^\# \cdot l_1 \cdots l_n \rightsquigarrow g^\# \cdot r_1 \cdots r_m\} \cup \\
 & \qquad \bigcup_{1 \leq k \leq n} \mathtt{DP}(f \cdot l_1 \cdots l_n, r_i) \quad (g \in \mathcal{D}) \\
\mathtt{DP}(f \cdot l_1 \cdots l_n, a \cdot r_1 \cdots r_m) & = \bigcup_{1 \leq k \leq n} \mathtt{DP}(f \cdot l_1 \cdots l_n, r_i) \quad (a \in (\mathcal{F} \setminus \mathcal{D}) \cup \mathcal{V})
\end{array}$$

A set of plain function passing rules $\mathcal{R}$ generates the set of dependency pairs $\mathtt{DP}(\mathcal{R}) = \bigcup_{l \Rightarrow r \in \mathcal{R}} \mathtt{DP}(l, r)$.

The *dependency graph* is a graph with the pairs in $\mathtt{DP}(\mathcal{R})$ as nodes, and an edge from node $l \rightsquigarrow r$ to node $l' \rightsquigarrow r'$ if there are substitutions γ, δ such that $r\gamma \Rightarrow_{\mathcal{R}}^* l'\delta$. This graph is in general not computable, so it is common to work with approximations.

A *reduction pair* is a pair $(>, \geq)$ of a strict order and a quasi-order on terms over $\mathcal{F}^\#$, such that $>$ is well-founded, $\geq$ is monotonic, either $> \cdot \geq$ is included

in $>$ or $\geq \cdot >$ is, and $>$ and $\geq$ are both stable: if l is a pattern and $l > r$ or $l \geq r$, then for any substitution γ also $l\gamma > r\gamma$ or $l\gamma \geq r\gamma$ respectively[1].

Given a set of PFP rules $\mathcal{R}$, whose dependency graph has no infinite path. Suppose that, for every cycle on the graph with nodes D, there is a reduction pair $(>, \geq)$ such that $l \geq r$ for all $l \Rightarrow r \in \mathcal{R}$, $l > r$ for at least one pair $l \rightsquigarrow r \in D$, and $l \geq r$ for all other pairs $l \rightsquigarrow r \in D$. Then $\Rightarrow_{\mathcal{R}}$ is terminating.

Now let us consider how we can transpose this termination method to Algebraic Functional Systems. Given any set of AFS-rules $\mathcal{R}$ which satisfies the requirements from Lemma 1, η-expand the rules and present the result in functional form (functional and applicative notation are equivalent when the requirements from lemma 1 are satisified, and functional form is often easier to work with). We say the system is PFP if, in all rules $f(l_1, \ldots, l_n) \Rightarrow r$, each l_i either is a variable, or does not contain any free variables of functional type. Defined symbols are those f such that a rule $f(l_1, \ldots, l_n) \Rightarrow r$ exists, and $\mathcal{F}^\# = \mathcal{F} \cup \{f^\# \ : \sigma |\ f : \sigma \in \mathcal{D}\}$. The set of dependency pairs of a set of rules is given by $\mathtt{DP}(\mathcal{R}) = \bigcup_{l \Rightarrow r \in \mathcal{R}} \mathtt{DP}(l, r)$, where $\mathtt{DP}(l, r)$ is inductively defined as follows:

$$
\begin{array}{ll}
\mathtt{DP}(f(l_1, \ldots, l_n), \lambda x.r) & = \mathtt{DP}(f(l_1, \ldots, l_n), r) \\
\mathtt{DP}(f(l_1, \ldots, l_n), g(r_1, \ldots, r_m)) & = \{f^\#(l_1, \ldots, l_n) \rightsquigarrow g^\#(r_1, \ldots, r_m)\} \cup \\
 & \quad \bigcup_{1 \leq i \leq m} \mathtt{DP}(f(\boldsymbol{l}), r_i) \quad (g \in \mathcal{D}) \\
\mathtt{DP}(f(l_1, \ldots, l_n), g(r_1, \ldots, r_m)) & = \bigcup_{1 \leq i \leq m} \mathtt{DP}(f(\boldsymbol{l}), r_i) \quad (g \in \mathcal{F} \setminus \mathcal{D}) \\
\mathtt{DP}(f(l_1, \ldots, l_n), x \cdot r_1 \cdots r_m) & = \bigcup_{1 \leq i \leq m} \mathtt{DP}(f(\boldsymbol{l}), r_i) \quad (x \in \mathcal{V})
\end{array}
$$

The dependency graph of $\mathcal{R}$ and reduction pairs are defined exactly as with HRSs. Now we have:

Theorem 8. *Given a set of PFP AFS-rules $\mathcal{R}$, whose dependency graph has no infinite path. Suppose that, for every cycle on the graph with nodes D, there is a reduction pair $(>, \geq)$ such that (1) $l \geq r$ for all $l \Rightarrow r \in \mathcal{R}$, (2) $l > r$ for at least one $l \rightsquigarrow r \in \mathtt{DP}(\mathcal{R})$ and (3) $l \geq r$ for all other pairs $l \rightsquigarrow r \in \mathtt{DP}(\mathcal{R})$. Then $\Rightarrow_{\mathcal{R}}$ is terminating.*

Proof. Consider Transformation 6 and the static dependency pair approach for PRSs. Evidently the transformation of a PFP AFS is a PFP PRS, and its dependency pairs are exactly the counterparts of the newly defined dependency pairs, and the pairs $\mathtt{app}^{\sigma \to \tau \#}(\boldsymbol{x}) \rightsquigarrow \mathtt{app}^{\tau \#}(\boldsymbol{x})$. It should be noted, however, that there is no cycle or infinite path involving any of the $\mathtt{app}^\#$ dependency pairs: from the pair whose left-hand side is headed by $\mathtt{app}^{\sigma \#}$ there is only an edge to a pair whose left-hand side is headed by $\mathtt{app}^{\tau \#}$ of a smaller type.

Thus, termination follows if requirement (1–3) are satisfied for all cycles in the graph, and since all nodes on a cycle correspond with an AFS-dependency

[1] In fact, the original definition requires $>$ and $\geq$ to be fully closed under substitution. This is a significant restriction: for instance $x \cdot s > x \cdot t$ would imply $u > u$ by instantiating x with $\lambda y.u$. Since we restrict to pattern HRSs, the restriction as given here suffices.

pair (and functional and applicative notation can reasonably be swapped), this holds if the requirements in the Theorem hold.

In a similar way, results like the Computability Path Ordering for AFSs [3] and monotone algebras for HRSs [14] can be transposed to to other formalisms.

6 A Joint Formalism?

With the transformations from Section 4, we can use methods from all common formalisms and use them in the others, often to great effect (as we hope was demonstrated in Section 5). Thus, it is not all that harmful to just choose the most convenient formalism to derive a new result in.

However, we started out from the viewpoint of building a termination tool, which should be able to deal with a variety of systems and use techniques regardless of the formalism they were defined for. Have we reached this point?

Almost. We could choose any of the typed formalisms and transform given systems of other formalisms into it, if they satisfy the requirements. Although this would sometimes lead to an infinite number of rules, they are all very similar rules, which a tool should be able to deal with without too much problems.

In WANDA[7], our termination tool, we chose *applicative IDTSs*, IDTSs with application and β-reduction added (equivalently, AFSs with meta-variables added). Since application and β-reduction could be implemented as function symbols and rules, this brings nothing new (but avoids the need for an infinite system). Noting Lemma 1 and the pattern restriction on PRSs, we can freely assume:

- both sides of AIDTS-rules are β-normal;
- left-hand sides of AIDTS-rules are patterns with a form $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$;
- left-hand sides of AIDTS-rules do not have a subterm $Z(\boldsymbol{x}) \cdot s$.

And thus, results like Lemma 2 extend (for the right definition of η-expansion – a complete discussion of this is beyond the scope of this paper). Results for AFSs can be used on AIDTS with simple meta-variables, results for PRSs can be transposed using Transformation 5, or by making a transformation more like Transformation 6. Meanwhile, systems in all of the formalisms can be represented as an AIDTS:

- an AFS is terminating if and only if the corresponding applicative IDTS is terminating (Theorem 2);
- a PRS is terminating if and only if the corresponding applicative IDTS is terminating with a β-first reduction strategy (this proof is almost exactly the same as that of Theorem 7);
- an IDTS is terminating if and only if the corresponding applicative IDTS (which has the same rules) is terminating on purely functional terms.

However, it is not our intention to promote yet another formalism; applicative IDTSs are a convenient choice to represent all existing formalisms in a tool, but it is unclear whether they have further merit; for example, we did not study critical pairs in AIDTSs. Tool designers could also, for instance, choose a form of PRSs with additional "weak" application symbols $\mathtt{app}^\sigma$.

7 Conclusions

In this paper, we have seen how systems defined in any of the common formalisms can be transformed into any of the others. We have seen an example of how this can be used, by transposing the static dependency pair approach to AFSs. In addition, we have laid some foundations for general termination tools which may deal with systems in all the common formalisms, rather than just one.

References

1. P. Aczel. A general Church-Rosser theorem. University of Manchester, jul 1978.
2. F. Blanqui. Termination and confluence of higher-order rewrite systems. In *RTA 2000*, volume 1833 of *LNCS*, Norwich, United Kingdom, 2000.
3. F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: The end of a quest. In *CSL 2008*, volume 5213 of *LNCS*, pages 1–14, Bertinoro, Italy, July 2008. Springer.
4. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *LICS 1999*, pages 402–411, Trento, Italy, July 1999.
5. J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, The Netherlands, 1980. PhD Thesis.
6. J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1-2):279 – 308, dec 1993.
7. C. Kop. Wanda. http://www.few.vu.nl/ kop/code.html.
8. C. Kop. Simplifying algebraic functional systems. In *CAI 2011*, June 2011. To Appear.
9. C. Kop and F. van Raamsdonk. Higher order dependency pairs for algebraic functional systems. In *RTA 2011*, June 2011. To Appear.
10. 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.
11. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.
12. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
13. T. Nipkow. Higher-order critical pairs. In *LICS 1991*, pages 342–349, Amsterdam, The Netherlands, July 1991.
14. J.C. van de Pol. *Termination of Higher-order Rerwite Systems*. PhD thesis, University of Utrecht, 1996.

15. F. van Raamsdonk. On termination of higher-order rewriting. In A. Middeldorp, editor, *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA 2001)*, volume 2051 of *LNCS*, pages 261–275, Utrecht, The Netherlands, May 2001.
16. 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.
17. 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.
18. Wiki. Termination portal. http://www.termination-portal.org/.
19. D. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, United Kingdom, 1993.

A Extended Proofs

A.1 Theorem 1

Lemma 3. *$\varphi(s\gamma) = \varphi(s)\gamma^\varphi$ for all CRS-terms s and substitutions γ.*

Here, $\gamma^\varphi = [x' := \varphi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$.

Proof. By induction on the form of s.

If s is a variable not in the domain of γ, then $s' \notin \mathrm{dom}(\gamma^\varphi)$; thus, $\varphi(s\gamma)\varphi(s) = s' = s'\gamma^\varphi = \varphi(s)\gamma^\varphi$.

If s is a variable in the domain of γ, then $\varphi(s\gamma) = \varphi(\gamma(t)) = \gamma^\varphi(s') = \varphi(s)\gamma^\varphi$.

If $s = \lambda x.t$ (with $x \notin \mathrm{dom}(\gamma)$), then $\varphi(s\gamma) = \mathtt{flat}(\lambda x'.\varphi(s\gamma)) =$ (by the induction hypothesis) $\mathtt{flat}(\lambda x'.\varphi(s)\gamma^\varphi) = \mathtt{flat}(\lambda x'.\varphi(s))\gamma^\varphi = \varphi(s)\gamma^\varphi$.

If $s = f(s_1, \ldots, s_n)$ with $f \in \mathcal{F}$, then $\varphi(s\gamma) = f'(\varphi(s_1\gamma), \ldots, \varphi(s_n\gamma)) =$ (by the induction hypothesis) $f'(\varphi(s_1)\gamma^\varphi, \ldots, \varphi(s_n)\gamma^\varphi) = f'(\varphi(s_1), \ldots, \varphi(s_n))\gamma^\varphi = \varphi(s)\gamma^\varphi$.

Lemma 4. *$\varphi(s\gamma) = \varphi(s)\gamma^\varphi$ for all CRS-meta-terms s and meta-substitutions γ whose domain contains all meta-variables occurring in s.*

Proof. By induction on the form of s.

If s is a variable (so not in the domain of γ), an abstraction or a function application, we proceed exactly as in Lemma 3.

Otherwise, $s = Z(s_1, \ldots, s_n)$ for some n, and we can write $\gamma(Z) = \lambda x_1 \ldots x_n.t$ (since by assumption $Z \in \mathrm{dom}(\gamma)$). Thus, $\varphi(s\gamma) = \varphi(t[x_1 := s_1\gamma, \ldots, x_n := s_n\gamma])$, which by Lemma 3 is equal to $\varphi(t)[x'_1 := \varphi(s_1\gamma), \ldots, x'_n := \varphi(s_n\gamma)]$. Using the induction hypothesis, $\varphi(s_i\gamma) = \varphi(s_i)\gamma^\varphi$, so this term is exactly $\varphi(Z'(\varphi(s_1), \ldots, \varphi(s_n)))\gamma^\varphi$, that is, $\varphi(s')\gamma^\varphi$.

Lemma 5. *If $s \Rightarrow_\mathcal{R} t$ then $\varphi(s) \Rightarrow_{\mathcal{R}^\flat} \varphi(t)$.*

Proof. By induction on the form of s.

If $s = l\gamma$ and $t = r\gamma$ for some rule $l \Rightarrow r \in \mathcal{R}$ and meta-substitution γ, then $\varphi(s) = \varphi(l)\gamma^\varphi \Rightarrow_{\mathcal{R}^\flat} \varphi(r)\gamma^\varphi = \varphi(r\gamma)$ by Lemma 4.

If $s = f(s_1, \ldots, s_i, \ldots, s_n)$ and $t = f(s_1, \ldots, s'_i, \ldots, s_n)$ and $s_i \Rightarrow_\mathcal{R} s'_i$, then $\varphi(s) = f'(\varphi(s_1), \ldots, \varphi(s_i), \ldots, \varphi(s_n)) \Rightarrow_{\mathcal{R}^\flat} f'(\varphi(s_1), \ldots, \varphi(s'_i), \ldots, \varphi(s_n)) = \varphi(t)$ by the induction hypothesis.

If $s = \lambda x.u$ and $t = \lambda x.v$ and $u \Rightarrow_\mathcal{R} v$, then $\varphi(s) = \lambda x'.\varphi(u) \Rightarrow_{\mathcal{R}^{\mathrm{CI}}} \lambda x'.\varphi(v) = \varphi(t)$ by the induction hypothesis.

Definition 1. *There are countably many variables both for CRSs and for IDTSs; thus we can choose, for every IDTS-variable x, a CRS-variable y_x. Similarly, we can assign to every IDTS-meta-variable a CRS-meta-variable of the same arity. Having done this, we can formally define an inverse to φ in the form of ψ, defined as follows:*

$$
\begin{array}{ll}
\psi(x) & = y_x \quad (x \text{ a variable}) \\
\psi(f'(s_1,\ldots,s_n)) & = f(\psi(s_1),\ldots,\psi(s_n))) \\
\psi(\mathtt{flat}(s)) & = \psi(s) \\
\psi(Z(s_1,\ldots,s_n)) & = X_Z(\psi(s_1),\ldots,\psi(s_n)) \\
\psi(\lambda x.s) & = \lambda y_x.\psi(s)
\end{array}
$$

Lemma 6. *$\psi(s\gamma) = \psi(s)\gamma^\psi$ for any term s and substitution γ, where $\gamma^\psi = [y_x := \psi(\gamma(x))|x \in \mathrm{dom}(\gamma)]$.*

Proof. By induction on s.

If s is a variable not in the domain of γ, then y_s is not in the domain of γ^ψ. Thus, $\psi(s\gamma) = \psi(s) = y_s = y_s\gamma^\psi = \psi(s)\gamma^\psi$.

If s is a variable in the domain of γ, then $\psi(s\gamma) = \psi(\gamma(s)) = \gamma^\psi(y_s) = \psi(s)\gamma^\psi$.

If $s = \lambda x.t$ (with $x \notin \mathrm{dom}(\gamma)$), then $\psi(s\gamma) = \lambda y_x.\psi(s\gamma)) =$ (by the induction hypothesis) $\lambda y_x.\psi(s)\gamma^\psi = (\lambda y_x.\psi(s))\gamma^\psi = \psi(s)\gamma^\psi$.

If $s = f'(s_1,\ldots,s_n)$ with $f \in \mathcal{F}^{\mathtt{flat}}$, then $\psi(s\gamma) = f(\psi(s_1\gamma),\ldots,\psi(s_n\gamma)) =$ (by the induction hypothesis) $f(\psi(s_1)\gamma^\psi,\ldots,\psi(s_n)\gamma^\psi) = f(\psi(s_1),\ldots,\psi(s_n))\gamma^\psi = \psi(s)\gamma^\psi$.

If $s = \mathtt{flat}(u)$, then $\psi(s\gamma) = \psi(\mathtt{flat}(u\gamma)) = \psi(u\gamma) =$ (by the induction hypothesis) $\psi(u)\gamma^\psi = \psi(\mathtt{flat}(u))\gamma^\psi = \psi(s)\gamma^\psi$.

Lemma 7. *Let s be a CRS-meta-term, γ a meta-substitution whose domain contains all meta-variables Z' for Z occurring in s, and χ a substitution whose domain contains all variables which occur freely in s. We assume* $\mathrm{dom}(\chi)$ *contains no variables occurring somewhere in* $\mathrm{ran}(\gamma)$.
Let δ_γ be the CRS-meta-substitution $[Z := \psi(\gamma(Z'))|Z' \in \mathrm{dom}(\gamma)]$ and let δ_χ be the substitution $[x := \psi(\chi(x'))|x' \in \mathrm{dom}(\chi)]$.
Then $\psi(\varphi(s)\gamma\chi) = s\delta_\gamma\delta_\chi$.

Proof. By induction on the form of s.

If s is a variable x, so $\varphi(s) = x'$, then note that χ contains x' by assumption (and γ has no effect). Thus, $\psi(\varphi(s)\gamma\chi) = \psi(\chi(x')) = \delta_\chi(x) = s\delta_\gamma\delta_\chi$.

If $s = \lambda x.t$, then $\psi(\varphi(s)\gamma\chi) = \psi(\mathtt{flat}(\lambda x'.\varphi(t))\gamma\chi) = \psi(\mathtt{flat}(\lambda x'.\varphi(t)\gamma\chi)) = \psi(\lambda x'.\varphi(t)\gamma\chi) = \lambda y_{x'}.\psi(\varphi(t)\gamma\chi)$. Defining $\chi' := \chi \cup [x' := x']$, we can apply the induction hypothesis to see that $\lambda y_{x'}.\psi(\varphi(t)\gamma\chi) = \lambda y_{x'}.\psi(\varphi(t)\gamma\chi') = \lambda y_{x'}.t\delta_\gamma\delta_{\chi'} = \lambda y_{x'}.t\delta_\gamma\delta_\chi[x := \psi(\chi'(x'))] = \lambda y_{x'}.t\delta_\gamma\delta_\chi[x := y_{x'}]$. Using α-conversion this is equal to $\lambda x.t\delta_\gamma\delta_\chi$ as required.

If $s = f(s_1,\ldots,s_n)$, then $\psi(\varphi(s)\gamma\chi) = \psi(f'(\varphi(s_1),\ldots,\varphi(s_n))\gamma\chi)$, which equals $f(\psi(\varphi(s_1)\gamma\chi),\ldots,\psi(\varphi(s_n)\gamma\chi))$. Applying the induction hypothesis on the s_i, this equals $f(s_1\delta_\gamma\delta_\chi,\ldots,s_n\delta_\gamma\delta_\chi) = s\delta_\gamma\delta_\chi$ as required.

Finally, if $s = Z(s_1,\ldots,s_n)$, then let $\gamma(Z) = \lambda x'_1\ldots x'_n.t$ (using α-conversion we can choose fresh $x_1,\ldots,x_n$ such that $\gamma(Z)$ has this form). In this case, $\psi(\varphi(s)\gamma\chi) = \psi(Z'(\varphi(s_1),\ldots,\varphi(s_n))\gamma\chi) = \psi(t[x'_1 := \varphi(s_1)\gamma,\ldots,x'_n := \varphi(s_n)\gamma]\chi)$. By the assumption that $\mathrm{dom}(\chi)$ contains no variables occurring in $\mathrm{ran}(\gamma)$, this can be rewritten to $\psi(t[x'_1 := \varphi(s_1)\gamma\chi,\ldots,x'_n := \varphi(s_n)\gamma\chi])$, which by Lemma

7 is equal to $\psi(t)[y_{x'_1} := \psi(\varphi(s_1)\gamma\chi), \ldots, y_{x'_n} := \psi(\varphi(s_n)\gamma\chi)]$, and by the induction hypothesis on the s_i to $\psi(t)[y_{x'_1} := s_1\delta_\gamma\delta_\chi, \ldots, y_{x'_n} := s_n\delta_\gamma\delta_\chi]$. On the other hand, consider $s\delta_\gamma\delta_\chi$ (**). We have $\delta_\gamma(Z) = \lambda y_{x'_1} \ldots y_{x'_n}.\psi(t)$ and therefore $s\delta_\gamma\delta_\chi = \psi(t)[y_{x'_1} := s_1\delta_\gamma, \ldots, y_{x'_1} := s_n\delta_\gamma]\delta_\chi = \psi(t)[y_{x'_1} := s_1\delta_\gamma\delta_\chi, \ldots, y_{x'_1} := s_n\delta_\gamma\delta_\chi]$, the same as (**).

Lemma 8. *If $s \Rightarrow_{\mathcal{R}^\flat} t$, then $\psi(s) \Rightarrow_{\mathcal{R}} \psi(t)$.*

Proof. By induction on the size of s.

If $s = \varphi(l)\gamma$ and $t = \varphi(r)\gamma$ for some rule $l \Rightarrow r \in \mathcal{R}$ and IDTS-meta-substitution γ, then by Lemma 7 (applied with empty χ, since rules are closed by definition), $\psi(s) = l\delta_\gamma \Rightarrow_{\mathcal{R}} r\delta_\gamma = \psi(t)$.

If $s = f''(s_1, \ldots, s_i, \ldots, s_n)$ and $t = f'(s_1, \ldots, s'_i, \ldots, s_n)$ and $s_i \Rightarrow_{\mathcal{R}^{\mathtt{CI}}} s'_i$, then $\psi(s) = f(\psi(s_1), \ldots, \psi(s_i), \ldots, \psi(s_n)) \Rightarrow_{\mathcal{R}} f(\psi(s_1), \ldots, \psi(s'_i), \ldots, \psi(s_n)) = \psi(t)$ by the induction hypothesis.

If $s = \mathtt{flat}(u)$ and $t = \mathtt{flat}(v)$ and $u \Rightarrow_{\mathcal{R}^{\mathtt{CI}}} v$, then $\psi(s) = \psi(u) \Rightarrow_{\mathcal{R}} \psi(v) = \psi(t)$ by the induction hypothesis.

If $s = \lambda x.u$ and $t = \lambda x.v$ and $u \Rightarrow_{\mathcal{R}^{\mathtt{CI}}} v$, then $\psi(s) = \lambda y_x.\psi(u) \Rightarrow_{\mathcal{R}} \lambda y_x.\psi(v) = \psi(t)$ by the induction hypothesis.

The proof of Theorem 1 is a comination of Lemmas 5 and 8.

A.2 Theorem 2

Definition 2.
Let φ map an AFS-term over $\mathcal{F}$ to an IDTS-term over $\mathcal{F}^{\mathtt{AI}}$, as follows:

$$\begin{array}{lll} \varphi(x) & = x & (x \text{ a variable}) \\ \varphi(\lambda x.s) & = \lambda x.\varphi(s) & \\ \varphi(f(s_1, \ldots, s_n)) & = f(\varphi(s_1), \ldots, \varphi(s_n)) & \\ \varphi(s \cdot t) & = @^{\sigma,\tau}(\varphi(s), \varphi(t)) & (s : \sigma \to \tau) \end{array}$$

Let ψ map an AIDTS-term over $\mathcal{F}^{\mathtt{AI}}$ to an AFS-term over $\mathcal{F}$, as follows:

$$\begin{array}{lll} \psi(x) & = x & (x \text{ a variable}) \\ \psi(\lambda x.s) & = \lambda x.\psi(s) & \\ \psi(f(s_1, \ldots, s_n)) & = f(\psi(s_1), \ldots, \psi(s_n)) & (f \in \mathcal{F}) \\ \psi(@^{\sigma,\tau}(s,t)) & = \psi(s) \cdot \psi(t) & \end{array}$$

Moreover, let $\varphi(l \Rightarrow r)$ be the rule $\varphi(l)' \Rightarrow \varphi(r)'$, where s' is s with all variables x replaced by a corresponding meta-variable Z_x.

Lemma 9. *For all AFS-terms s, $\psi(\varphi(s)) = s$.*

Proof. A trivial induction on the size of s.

Lemma 10. *$\varphi(s\gamma) = \varphi(s)\gamma^\varphi$, where $\gamma^\varphi = [x := \varphi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$.*

Proof. A trivial induction on the size of s.

Lemma 11. *$\psi(s\gamma) = \psi(s)\gamma^\psi$, where $\gamma^\psi = [x := \psi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$.*

Proof. A trivial induction on the size of s.

Lemma 12. *For all AFS-terms s, t, if $s \Rightarrow_{\mathcal{R}} t$, then $\varphi(s) \Rightarrow_{\mathcal{R}^{\mathtt{AI}}} \varphi(t)$.*

Proof. By induction on the size of s; the only interesting cases are the base cases, s reduces by a topmost step.

If $s = (\lambda x.u) \cdot v$ and $t = u[x := v]$, let $\lambda x.u : \sigma \to \tau$. Then $\varphi(s) = \mathtt{app}^{\sigma,\tau}(\lambda x.\varphi(u), \varphi(v))$. By the rule $\mathtt{app}^{\sigma,\tau}(\lambda x.Z(x), X) \Rightarrow Z(X)$ this reduces to $\varphi(u)[x := \varphi(v)]$, which by Lemma 10 equals $\varphi(u[x := v]) = \varphi(t)$.

If $s = l\gamma$ and $t = r\gamma$ for some rule $l \Rightarrow r \in \mathcal{R}$, then by Lemma 10 also $\varphi(s) = \varphi(l)\gamma^\varphi$ and $\varphi(t) = \varphi(r)\gamma^\varphi$. Defining $\gamma'^\varphi := [Z_x := \gamma^\varphi(x) | x \in \mathrm{dom}(\gamma)]$ it is thus evident that $\varphi(s) = \varphi(l)'\gamma'^\varphi$ and $\varphi(t) = \varphi(r)'\gamma'^\varphi$. Since $\varphi(l)' \Rightarrow \varphi(r)' \in \mathcal{R}^{\mathtt{AI}}$ we are done.

Lemma 13. *For all IDTS-terms s, t, if $s \Rightarrow_{\mathcal{R}^{\mathtt{AI}}} t$, then $\psi(s) \Rightarrow_{\mathcal{R}} \psi(t)$.*

Proof. By induction on the size of s; the only interesting cases are when the reduction is topmost.

If $s = \varphi(l)'\gamma$ and $t = \varphi(r)'\gamma$ for $l \Rightarrow r \in \mathcal{R}$ and a meta-substitution γ, then we also have $s = \varphi(l)\delta$ and $t = \varphi(r)\delta$, for $\delta = [x := \gamma(Z_x) | Z_x \in \mathrm{dom}(\gamma)]$ (it is safe to assume that $\mathrm{dom}(\gamma)$ contains exactly those meta-variables occurring in $\varphi(l)'$, which all have this form). But then, $\psi(s) = \psi(\varphi(l))\delta^\psi$ by Lemma 11, $= l\delta^\psi$ by Lemma 9, which reduces to $r\delta^\psi$ and, by Lemmas 9 and 11, is equal to $\psi(\varphi(l)\delta) = \psi(t)$ again.

If $s = l\gamma$ and $t = r\gamma$ for $l \Rightarrow r$ one of the rules $\mathtt{app}^{\sigma,\tau}(\lambda x.Z(x), X)$, then $s = \mathtt{app}^{\sigma,\tau}(\lambda x.u, v)$ and $t = u[x := v]$. Thus, $\psi(s) = (\lambda x.\psi(u)) \cdot \psi(v) \Rightarrow_\beta \psi(u)[x := \psi(v)]$, which by Lemma 11 equals $\psi(u[x := v]) = \psi(t)$.

Theorem 2 is a combination of Lemmas 12 and 13.

A.3 Theorem 3

Definition 3. *Choose for every meta-variable $Z : (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$ a variable $y_Z : \sigma_1 \to \ldots \to \sigma_n \to \tau$. The translation from an IDTS-meta-term to an AFS-term is inductively defined as follows:*

$\varphi(x)$	$= x$	x a variable
$\varphi(\lambda x.s)$	$= \lambda x.\varphi(s)$	$s \neq \lambda z_2 \ldots z_n.Z(x, z_2, \ldots, z_n)$
$\varphi(f(s_1, \ldots, s_n))$	$= f(\varphi(s_1), \ldots, \varphi(s_n))$	
$\varphi(\lambda x_1 \ldots x_n.Z(x_1, \ldots, x_n))$	$= y_Z$	
$\varphi(Z(s_1, \ldots, s_n)$	$= y_Z \cdot \varphi(s_1) \cdots \varphi(s_n)$	

Given a set of IDTS-rules $\mathcal{R}$, let $\mathcal{R}^{\mathtt{IA}} = \{\varphi(l) \Rightarrow \varphi(r) | l \Rightarrow r \in \mathcal{R}\}$

Lemma 14. *If l is a closed IDTS-meta-term with simple meta-applications, γ is a meta-substitution whose domain contains all meta-variables in l, and δ is the substitution $[y_Z := \gamma(Z) | Z \in \mathrm{dom}(\gamma)]$, then $l\gamma = \varphi(l)\delta$.*

Proof. Dropping the "closed" requirement, we prove that this holds if the free variables in l are not in $\mathrm{dom}(\delta)$. This certainly holds for closed terms, and using α-conversion we can maintain this property during an induction on the form of l.

If l is a variable, then $l\gamma = l$ and (since $\varphi(l) = l$ and $\mathrm{dom}(\delta)$ does not contain l) also $\varphi(l)\delta = l$.

If $l = \lambda x.l'$ with l' not of the form $\lambda z_2 \dots z_n.Z(x, z_2, \dots, z_n)$, then $l\gamma = \lambda x.l'\gamma$, which by the induction hypothesis equals $\lambda x.\varphi(l')\delta = \varphi(l)\delta$.

If $l = f(l_1, \dots, l_n)$ then $l\gamma = f(l_1\gamma, \dots, l_n\gamma) =$ (IH) $f(\varphi(l_1)\delta, \dots, \varphi(l_n)\delta) = \varphi(l)\delta$.

If $l = \lambda x_1 \dots x_n.Z(x_1, \dots, x_n)$, then consider $\gamma(Z)$. This must have the form $\lambda z_1 \dots z_n.s$ for some term s, so $l\gamma = \lambda x_1 \dots x_n.s[x_1 := z_1, \dots, x_n := z_n] = \lambda z_1 \dots z_n.s$ (using α-conversion). So $l\gamma = \gamma(Z) = \delta(y_Z) = \varphi(l)\delta$.

Since meta-variables in a simple meta-application occur only in this form, there are no other cases.

Lemma 15. *If s is a closed IDTS-meta-term, γ is a meta-substitution whose domain contains all meta-variables in s and δ is the substitution $[y_Z := \gamma(Z) | Z \in \mathrm{dom}(\gamma)]$, then $\varphi(s)\delta \Rightarrow_\beta^* s\gamma$.*

Proof. We again perform induction on the form of s, changing the "closed" requirement into "all variables occurring freely in s do not occur in $\mathrm{dom}(\delta)$.

The cases where s is a variable, a function symbol, and both cases where s is an abstraction are exactly as in Lemma 14, using $\Rightarrow_\beta^*$ in the induction step instead of equality (and noting that $\Rightarrow_\beta^*$ is both reflexive and transitive).

Consider the only remaining case, $s = Z(s_1, \dots, s_n)$. Let $\gamma(Z) = \lambda x_1 \dots x_n.t$. Then $s\gamma = t[x_1 := s_1, \dots, x_n := s_n]$. On the other hand, $\varphi(s)\delta = (y_Z \cdot \varphi(s_1) \cdots \varphi(s_n))\delta = (\lambda x_1 \dots x_n.t) \cdot \varphi(s_1)\delta \cdots \varphi(s_n)\delta$, which by the induction hypothesis on the s_i equals $(\lambda x_1 \dots x_n.t) \cdot s_1\gamma \cdots s_n\gamma$. This term $\Rightarrow_\beta$-reduces in n steps to the required $t[x_1 := s_1\gamma, \dots, x_n := s_n\gamma]$.

Theorem 3 follows from Lemmas 14 and 15: if s is an IDTS-term over $\mathcal{F}$, it is also an AFS-term over $\mathcal{F}$, and if $s \Rightarrow_\mathcal{R} t$, then there is a rule $l \Rightarrow r$, context C and meta-substitution γ such that $s = C[l\gamma]$ and $t = C[r\gamma]$. But by these two Lemmas, $s = C[\varphi(l)\delta] \Rightarrow_{\mathcal{R}^{\mathrm{IA}}} C[\varphi(r)\delta] \Rightarrow_\beta^* C[r\gamma]$.

A.4 Theorem 4

In this section, let $\varphi := \varphi_\emptyset$, and given a PRS-substitution γ, let $\gamma^\varphi := [x := \varphi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$ and

Lemma 16. *If l is a PRS-pattern, then $\varphi_{FVar(l)}(l)$ is a pattern and if γ is a substitution whose domain contains all variables occurring freely in l, then $l\gamma = \varphi_{FVar(l)}(l)\delta_\gamma$, where δ_γ is the IDTS-meta-substitution $[Z_x := \varphi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$.*

Proof. Let S be a set of variables, and suppose that l is a PRS-term such that for any subterm $x \cdot l_1 \cdots l_n$ of l with $x \in S$, all l_i are (η-long forms of) different variables not in S (this holds if l is a PRS-pattern and $S = FVar(l)$). We can also assume $\mathrm{dom}(\gamma) = S$, and that none of the variables which occur in l but are not in S occur in the range of γ.

Now we prove: $\varphi_S(l)$ is a meta-term where in all subterms of the form $Z(s_1, \ldots, s_n)$, all s_i are difference variables not in S (and possibly bound in l); moreover, $\varphi(l\gamma) = \varphi_S(l)\delta_\gamma$. Note that if $S = FVar(l)$, the first of these statements implies that $\varphi_S(l)$ is a pattern (since φ does not create additional variables and does not cause bound variables to become free).

We prove this with induction on the form of l.

First suppose $l = x \cdot s_1 \cdots s_n$ with $x \in S$, so the s_i are the η-long forms of different variables not in S (so also not in $\mathrm{dom}(\gamma)$), say $s_i = y_i{\uparrow^\eta}$. Then $\varphi_S(l) = Z_x(y_1, \ldots, y_n)$, which satisfies the requirements. Let $\gamma(x) = \lambda y_1 \ldots y_n.t$ (we can always assume this form, using α-conversion); $l\gamma = \gamma(x) \cdot s_1\gamma \cdots s_n\gamma{\downarrow^\eta_\beta} = \gamma(x) \cdot y_1 \cdots y_n \,{\downarrow^\eta_\beta}$ (because the y_i are η-equal to s_i and do not occur in γ), $= (\lambda \boldsymbol{y}.t) \cdot \boldsymbol{y}\,{\downarrow^\eta_\beta} = t\,{\downarrow^\eta_\beta} = t$ because $\gamma(x)$ and hence t is a term, and thus already in long β/η-normal form. On the other hand, $Z_x(y_1, \ldots, y_n)\delta_\gamma = \varphi(t)[y_1 := y_1, \ldots, y_n := y_n] = \varphi(t)$ as required.

Next, suppose $l = x \cdot l_1 \cdots l_n$ with $x \notin S$; let $x : \sigma$. Then $\varphi_S(l) = \mathtt{app}^\sigma(x, \varphi_S(l_1), \ldots, \varphi_S(l_n))$, which satisfies the first requirement by the induction hypothesis on the l_i. As for the second requirement, since $\mathrm{dom}(\gamma) = S \not\ni x$, we have $l\gamma = x \cdot l_1\gamma \cdots l_n\gamma$, and therefore $\varphi(l\gamma) = \mathtt{app}^\sigma(x, \varphi(l_1\gamma), \ldots, \varphi(l_n\gamma))$, which by the induction hypothesis equals $\mathtt{app}^\sigma(x, \varphi_S(l_1)\delta_\gamma, \ldots, \varphi_S(l_n)\delta_\gamma) = \varphi_S(l)\delta_\gamma$.

Similarly, when $l = f \cdot l_1 \cdots l_n$, then $\varphi_S(l) = f'(\varphi_S(l_1), \ldots, \varphi_S(l_n))$ which satisfies the first requirement by the induction hypothesis. As for the second, $\varphi(l\gamma) = \varphi(f \cdot l_1\gamma \cdots l_n\gamma) = f'(\varphi(l_1\gamma), \ldots, \varphi(l_n\gamma))$ = (induction hypothesis) $f'(\varphi_S(l_1)\delta_\gamma, \ldots, \varphi_S(l_n)\delta_\gamma) = \varphi_S(l)\delta_\gamma$.

Finally, let $l = \lambda x.l'$, so $\varphi_S(l) = \lambda x.\varphi_S(l')$, which satisifies the first requirement because (by induction $\varphi_S(l')$ does. Also, $\varphi(l\gamma) = \lambda x.\varphi(l'\gamma) = \lambda x.\varphi_S(l')\delta_\gamma = \varphi_S(l)\delta_\gamma$ by the induction hypothesis.

Lemma 16 both justifies Transformation 4 (the resulting meta-term pairs $\varphi_{FVar(l)}(l) \Rightarrow \varphi_{FVar(l)}(r)$ are actually rules), and gives a start to the proof of Theorem 4.

Lemma 17. *For a PRS-substitution γ, let $\gamma^\varphi := [x := \varphi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$. For any PRS-term s and PRS-substitution γ: $\varphi(s)\gamma^\varphi \Rightarrow^*_{\mathtt{beta}}$.*

Here, $\Rightarrow_{\mathtt{beta}}$ indicates a reduction with one of the rules whose root symbol is some $\mathtt{app}^\sigma$.

Proof. By induction on the pre-term $s\gamma$, ordered by the union of $\Rightarrow_\beta$ and subterm steps.

If $s = x \cdot s_1 \cdots s_n$ with $x \in \mathrm{dom}(\gamma)$, let $\gamma(x) = \lambda y_1 \ldots y_n.t$. Then $\varphi(s)\gamma^\varphi = \mathtt{app}^\sigma(x, \varphi(s_1), \ldots, \varphi(s_n))\gamma^\varphi = \mathtt{app}^\sigma(\varphi(\gamma(x)), \varphi(s_1)\gamma^\varphi, \ldots, \varphi(s_n)\gamma^\varphi)$, which by the induction hypothesis $\Rightarrow^*_{\mathtt{beta}} \mathtt{app}^\sigma(\varphi(\gamma(x)), \varphi(s_1\gamma), \ldots, \varphi(s_n\gamma)) = \mathtt{app}^\sigma(\lambda y_1$

$\ldots y_n.\varphi(t), \varphi(s_1\gamma), \ldots, \varphi(s_n\gamma)) \Rightarrow_{\mathtt{beta}} \varphi(t)[y_1 := \varphi(s_1\gamma), \ldots, y_n := \varphi(s_n\gamma)]$. We can apply the $\Rightarrow_\beta$-part of the induction hypothesis to see that this term $\Rightarrow^*_{\mathtt{beta}}$ $\varphi(t[y_1 := s_1\gamma, \ldots, y_n := s_n\gamma]) = \varphi(s\gamma)$.

Next, if $s = x \cdot s_1 \cdots s_n$ with x a variable not in $\mathrm{dom}(\gamma)$, then we only need the subterm part of the induction hypothesis: $\varphi(s)\gamma^\varphi = \mathtt{app}^\sigma(x, \varphi(s_1)\gamma^\varphi, \ldots, \varphi(s_n)\gamma^\varphi) \Rightarrow^*_{\mathtt{beta}}$ $\mathtt{app}^\sigma(x, \varphi(s_1\gamma), \ldots, \varphi(s_n\gamma)) = \varphi(s\gamma)$ (note, after all, that γ^φ has the same domain as γ).

If $s = f \cdot s_1 \cdots s_n$ with $f \in \mathcal{F}$, then we also just need the subterm part of the induction hypothesis: $\varphi(s)\gamma^\varphi = f'(\varphi(s_1)\gamma^\varphi, \ldots, \varphi(s_n)\gamma^\varphi) \Rightarrow^*_{\mathtt{beta}} f'(\varphi(s_1\gamma), \ldots, \varphi(s_n\gamma)) = \varphi(s\gamma)$.

Finally, if $s = \lambda x.t$ we again complete with the induction hypothesis: $\varphi(s)\gamma^\varphi = \lambda x.\varphi(t)\gamma^\varphi \Rightarrow^*_{\mathtt{beta}} \lambda x.\varphi(t\gamma) = \varphi(s\gamma)$.

Lemma 18. *For any PRS-term s and substitution γ on domain $S \supseteq FVar(s)$: $\varphi_S(s)\delta_\gamma \Rightarrow^*_{\mathtt{beta}} \varphi(s\gamma)$.*

Proof. As before, to be able to do induction we need a slightly different statement: fixing a substitution γ on domain S, let s be a PRS-term such that none of the variables which occurr freely in s but are not in S occur in the range of γ; we prove that $\varphi_S(s)\delta_\gamma \Rightarrow^*_{\mathtt{beta}} \varphi(s\gamma)$ under these conditions.

We prove this by induction on the form of s. The cases where s is headed by a variable not in S or a function symbol, and where s is an abstraction, are exactly as in Lemma 17.

The only remaining case is when $s = x \cdot s_1 \cdots s_n$ with $x \in S$; we may write $\gamma(x) = \lambda x_1 \ldots x_n.t$. In this case $\varphi_S(s)\delta_\gamma = Z_x(\varphi_S(s_1), \ldots, \varphi_S(s_n))\delta_\gamma = \varphi(t)[x_1 := \varphi_S(s_1)\delta_\gamma, \ldots, x_n := \varphi_S(s_n)\delta_\gamma]$. By the induction hypothesis on the s_i this reduces to $\varphi(t)[x_1 := \varphi(s_1\gamma), \ldots, s_n\gamma]$, and by Lemma 17 it reduces further to $\varphi(t[x_1 := s_1\gamma, \ldots, x_n := s_n\gamma]) = \varphi(s\gamma)$.

Lemma 19. *For any PRS-terms s, t, if $s \Rightarrow_{\mathcal{R}} t$ then $\varphi(s) \Rightarrow^+_{\mathcal{R}^{\mathtt{PI}}} \varphi(t)$.*

Proof. By induction on the size of s.

If $s = l\gamma$, then we can safely assume $\mathrm{dom}(\gamma)$ exactly contains the variables occurring freely in l, and thus $\varphi(s) = \varphi(l)\delta_\gamma$ by Lemma 16, $\Rightarrow_{\mathcal{R}^{\mathtt{PI}}} \varphi(r)\delta_\gamma$ by definition, $\Rightarrow^*_{\mathtt{beta}} \varphi(r\gamma) = \varphi(t)$ by Lemma 18.

In all other cases we immediately complete with the induction hypothesis.

Theorem 4 is an immediate consequence of Lemma 19.

A.5 Theorem 5

Write $\varphi^\uparrow(s)$ as shortened notation for $\varphi(s)\uparrow^\eta$.

First we should note that $\mathcal{R}^{\mathtt{IP}}$ is a well-defined PRS, that is, that the left-hand sides of the rules are patterns.

Lemma 20. *If l is an IDTS-pattern, then $\varphi(l) \cdot x_1 \cdots x_n\uparrow^\eta$ is a PRS-pattern for fresh variables $x_1, \ldots, x_n$.*

Proof. Note that $\varphi^\uparrow(l) = \lambda x_1 \dots x_n.(\varphi(l) \cdot x_1 \cdots x_n)\uparrow^\eta$ is a pattern if and only if $\varphi(l) \cdot x_1 \cdots x_n\uparrow^\eta$ is; thus it suffices to prove that $\varphi^\uparrow(l)$ is a PRS-pattern if l is an IDTS-pattern.

Since the only variables occurring freely in $\varphi(l)$ are the x_Z with Z occurring in l, $\varphi(l)$ is a pattern if all subterms of the form x_Z in $\varphi(l)$ occur only in the form $x_Z \cdot y_1 \cdots y_n$ with the y_i not any of the x_Z (using α-conversion we can assume the bound variables have no such form).

Thus, by induction on the size of l: if l is an IDTS-pattern, and no variables in l have the form x_Z, then for any occurrence of some $x_Z \cdot s_1 \cdots s_n$ in $\varphi^\uparrow(l)$, all s_i are the η-long forms of different bound variables.

If l is a variable (so not some x_Z), then $\varphi^\uparrow(l) = l\uparrow^\eta$, which contains no x_Z.

If $l = \lambda y.l'$, then $\varphi^\uparrow(l) = (\lambda y.\varphi(l'))\uparrow^\eta = \lambda y.(\varphi^\uparrow(l'))$, which by the induction hypothesis satisfies the requirement.

If $l = f(l_1, \dots, l_n)$, then $\varphi^\uparrow(l) = f' \cdot \varphi(l_1) \cdots \varphi(l_n)\uparrow^\eta = \lambda y_{n+1} \dots y_m.f' \cdot \varphi^\uparrow(l_1) \cdots \varphi^\uparrow(l_n) \cdot y_{n+1}\uparrow^\eta \cdots y_m\uparrow^\eta$, which satisfies the requirement if each of the $\varphi^\uparrow(l_i)$ do; this holds by the induction hypothesis.

Finally, if $l = Z(y_1, \dots, y_n)$, then all y_i are different variables which are not some x_Z by assumption. Thus, $\varphi^\uparrow(l) = x_Z \cdot y_1 \cdots y_n\uparrow^\eta = \lambda z_{n+1} \dots z_m.x_Z \cdot y_1\uparrow^\eta \cdots y_n\uparrow^\eta \cdot z_{n+1}\uparrow^\eta \cdots z_m\uparrow^\eta$, which indeed satisfies the requirement!

Then, the usual prerequisites: φ is preserved under substitution and meta-substitution. Given a substitution γ, let γ^φ be the substitution $[x := \varphi^\uparrow(\gamma(x))|x \in \mathrm{dom}(\gamma)]$ and for a meta-substitution γ let γ^φ be the substitution $[x_Z := \varphi^\uparrow(\gamma(Z))|Z \in \mathrm{dom}(\gamma)]$.

Lemma 21. *$\varphi^\uparrow(s)\gamma^\varphi = \varphi^\uparrow(s\gamma)$ for all IDTS-terms s and substitutions γ.*

Proof. Induction on the size of s.

If s is a variable $x \in \mathrm{dom}(\gamma)$, then $\varphi^\uparrow(s)\gamma^\varphi = x\uparrow^\eta\gamma^\varphi$, which modulo β/η is equal to $x\gamma^\varphi = \gamma^\varphi(x) = \varphi^\uparrow(\gamma(x)) = \varphi^\uparrow(s\gamma)$.

If s is a variable $x \notin \mathrm{dom}(\gamma)$, then $\varphi^\uparrow(s)\gamma^\varphi = x\uparrow^\eta\gamma^\varphi = x\uparrow^\eta = \varphi^\uparrow(s\gamma)$.

If s is an abstraction $\lambda x.t$, then $\varphi^\uparrow(s)\gamma^\varphi = \lambda x.\varphi^\uparrow(t)\gamma^\varphi$, which by the induction hypothesis equals $\lambda x.\varphi^\uparrow(t\gamma) = \varphi^\uparrow(s\gamma)$.

Finally, if $s = f(s_1, \dots, s_n)$ then $\varphi^\uparrow(s)\gamma^\varphi = (\lambda x_{n+1} \dots x_m.f' \cdot \varphi^\uparrow(s_1) \cdots \varphi^\uparrow(s_n) \cdot x_{n+1}\uparrow^\eta \cdots x_m\uparrow^\eta)\gamma^\varphi = \lambda x_{n+1} \dots x_m.f' \cdot \varphi^\uparrow(s_1)\gamma^\varphi \cdots \varphi^\uparrow(s_n)\gamma^\varphi \cdot x_{n+1}\uparrow^\eta \cdots x_m\uparrow^\eta$. Using the induction hypothesis on the s_i, this is equal to $\lambda x_{n+1} \dots x_m.f' \cdot \varphi^\uparrow(s_1\gamma) \cdots \varphi^\uparrow(s_n\gamma) \cdot x_{n+1}\uparrow^\eta \cdots x_m\uparrow^\eta = (f' \cdot \varphi(s_1\gamma) \cdots \varphi(s_n\gamma))\uparrow^\eta = \varphi^\uparrow(s)$.

Lemma 22. *$\varphi^\uparrow(s)\gamma^\varphi = \varphi^\uparrow(s\gamma)$ for all IDTS-meta-terms s and meta-substitutions γ whose domain contains all meta-variables in s (we assume that s does not freely contain any variables x_Z).*

Proof. By induction on the size of s; all steps can be copied from Lemma 21 except when $s = Z(s_1, \dots, s_n)$.

In this case, $\varphi^\uparrow(s)\gamma^\varphi = (\lambda y_{n+1} \dots y_m.x_Z \cdot \varphi^\uparrow(s_1) \cdots \varphi^\uparrow(s_n) \cdot y_{n+1}\uparrow^\eta \cdots y_m\uparrow^\eta)\gamma^\varphi$. Let $\gamma(Z) = \lambda z_1 \dots z_m.t$, so $\gamma^\varphi(x_Z) = \varphi^\uparrow(\gamma(Z)) = \lambda z_1 \dots z_m.\varphi^\uparrow(t)$. Thus, our term is β/η-equal to $\lambda y_{n+1} \dots y_m.\varphi^\uparrow(t)[z_1 := \varphi^\uparrow(s_1)\gamma^\varphi, \dots, z_m := \varphi^\uparrow(s_n)\gamma^\varphi] \cdot$

$y_{n+1}\uparrow^\eta \cdots y_m\uparrow^\eta$, which by the induction hypothesis equals $\lambda y_{n+1}\ldots y_m.\varphi^\uparrow(t)[z_1 := \varphi^\uparrow(s_1\gamma), \ldots, z_m := \varphi^\uparrow(s_n\gamma)]\cdot y_{n+1}\uparrow^\eta \cdots y_m\uparrow^\eta$. This, again, is β/η-equal to $\varphi^\uparrow(t)[z_1 := \varphi^\uparrow(s_1\gamma), \ldots, z_n := \varphi^\uparrow(s_n\gamma)]$, which by Lemma 22 is equal to $\varphi^\uparrow(t[z_1 := s_1\gamma, \ldots, z_m := s_n\gamma]) = \varphi^\uparrow(s\gamma)$ as required.

Lemma 23. *If $s \Rightarrow_\mathcal{R} t$ for PRS-terms s, t, then $\varphi^\uparrow(s) \Rightarrow_{\mathcal{R}^{\text{IP}}} \varphi^\uparrow(t)$.*

Proof. Induction on the size of s.

If $s = l\gamma$ and $t = r\gamma$, then $\varphi^\uparrow(s) = \varphi^\uparrow(l)\gamma^\varphi = (\lambda x_1 \ldots x_n.(\varphi(l)\cdot x_1 \cdots x_n)\uparrow^\eta)\gamma^\varphi = \lambda\boldsymbol{x}.(\varphi(l)\cdot\boldsymbol{x})\uparrow^\eta\delta$, where $\delta = \gamma^\varphi \cup [x_1 := x_1\uparrow^\eta, \ldots, x_n := x_n\uparrow^\eta]$. This term $\Rightarrow_{\mathcal{R}^{\text{IP}}}$-reduces to $\lambda\boldsymbol{x}.(\varphi(r)\cdot\boldsymbol{x})\uparrow^\eta\delta = (\lambda\boldsymbol{x}.\varphi(r)\cdot\boldsymbol{x}\uparrow^\eta)\gamma^\varphi = \varphi^\uparrow(r)\gamma^\varphi = \varphi^\uparrow(t)$ by Lemma 22.

The other cases are trivial uses of the induction hypothesis, included for completeness:

If $s = \lambda x.u$ and $t = \lambda x.v$ with $u \Rightarrow_\mathcal{R} v$, then $\varphi^\uparrow(s) = \lambda x.\varphi^\uparrow(u) \Rightarrow_{\mathcal{R}^{\text{IP}}} \lambda x.\varphi^\uparrow(v) = \varphi^\uparrow(t)$ by the induction hypothesis.

If $s = f(s_1, \ldots, s_i, \ldots, s_n)$ and $t = f(s_1, \ldots, s_i', \ldots, s_n)$ and $s_i \Rightarrow_\mathcal{R} s_i'$, then $\varphi^\uparrow(s) = \lambda\boldsymbol{x}.f'\cdot\varphi^\uparrow(s_1)\cdots\varphi^\uparrow(s_i)\cdots\varphi^\uparrow(s_n)\cdot\boldsymbol{x}\uparrow^\eta \Rightarrow_{\mathcal{R}^{\text{IP}}} \lambda\boldsymbol{x}.f'\cdot\varphi^\uparrow(s_1)\cdots\varphi^\uparrow(s_i')\cdots\varphi^\uparrow(s_n)\cdot\boldsymbol{x}\uparrow^\eta = \varphi^\uparrow(t)$ by the induction hypothesis.

Theorem 5 is an immediate consequence of Lemma 23.

A.6 Theorem 6

In this section we assume AFS-terms in an applicative form.

For V a set of variables, say a term s is in η, V-long form if:

$s = a\cdot s_1\cdots s_n : \iota$	and $s_1, \ldots, s_n$ in $\eta, V-$long form and $a \in \mathcal{F}\cup\mathcal{V}$
$s = (\lambda x.t_0)\cdot t_1\cdots t_n : \sigma$	and $t_0, \ldots, t_n$ in $\eta, V-$long form and $\sigma \in \mathcal{B}$ or $n = 0$
$s = x \in V$	

It should be clear that for all terms, $s\uparrow^\eta_V$ is in η, V-long form.

Lemma 24. *Let γ be a substitution with domain V; if l is an AFS-term which has no subterms $x\cdot s$ with $x \in \mathcal{V}$, and l is in η, V-long form and $l\gamma$ is in η-long form, then all $\gamma(x)$ with x occurring in l are in η-long form.*

Proof. By induction on the form of l.

If $l = x \in V$, then $l\gamma = \gamma(x)$, is η-long by assumption.

If $l = a\cdot l_1\cdots l_n$ with $a \in \mathcal{F}\cup\mathcal{V}$, note that by assumption either $a \notin V$, or $a \in V$ and $n = 0$. The latter case we just treated, in the former $l\gamma = a\cdot l_1\gamma\cdots l_n\gamma$; since all variables in V which occur in l must occur in one of the l_i, the lemma follows.

If $l = \lambda x.l'$, we can use α-conversion to guarantee that $x \notin V$; thus, $l\gamma = \lambda x.(l'\gamma)$ and the induction hypothesis guarantees that $\gamma(x)$ has the required form.

It is also evident that $\varphi(l)$ is a pattern if l has no subterms $x \cdot s$ with x a variable free in l. Given an η-long substitution γ, write $\gamma^\varphi = [x := \varphi(\gamma(x)) | x \in \mathrm{dom}(\gamma)]$; clearly, this is a PRS-substitution mapping variables to PRS-terms.

Lemma 25. *If $\gamma(x)$ is η-long for all x in its domain V, then for all η, V-long terms s: $\varphi(s\gamma) = \varphi(s)\gamma^\varphi$.*

Proof. By induction on the form of s.

If $s = x \in V$, then $\varphi(s\gamma) = \varphi(\gamma(x)) = \gamma^\varphi(x) =_{\beta/\eta} x\uparrow^\eta\gamma^\varphi = \varphi(s)\gamma^\varphi$.

If $s = a \cdot s_1 \cdots s_n$ with $a \in \mathcal{F} \cup \mathcal{V} \setminus V$, then $\varphi(s\gamma) = \varphi(a \cdot s_1\gamma \cdots s_n\gamma) = a \cdot \varphi(s_1\gamma) \cdots \varphi(s_n\gamma) =$ (by the induction hypothesis) $a \cdot \varphi(s_1)\gamma^\varphi \cdots \varphi(s_n)\gamma^\varphi = (a \cdot \varphi(s_1) \cdots \varphi(s_n))\gamma^\varphi = \varphi(s)\gamma^\varphi$.

If $s = \lambda x.t$ then $\varphi(s\gamma) = \lambda x.\varphi(t\gamma)$, which by induction equals $\lambda x.\varphi(t)\gamma^\varphi = \varphi(s)\gamma^\varphi$.

Finally, if $s = (\lambda x.t) \cdot s_0 \cdots s_n$ then $\varphi(s\gamma) = \varphi((\lambda x.t\gamma) \cdot s_0\gamma \cdots s_n\gamma) = (\lambda x.\varphi(t\gamma)) \cdot \varphi(s_1\gamma) \cdots \varphi(s_n\gamma)$, which by the induction hypothesis equals $(\lambda x.\varphi(t)\gamma^\varphi) \cdot \varphi(s_1)\gamma^\varphi \cdots \varphi(s_n)\gamma^\varphi = \varphi(s)\gamma^\varphi$.

Lemma 26. *Given an applicative, η-long AFS $(\mathcal{F}, \mathcal{R})$ which satisfies the requirements from Lemma 1 and η-long terms s, t, then $s \Rightarrow_\mathcal{R} t$ implies $\varphi(s) \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} \varphi(t)$.*

Proof. By induction on the form of s.

We consider first the base cases: β-reduction by a topmost step, β-reduction by a headmost step, and reduction by a rule (note that all rules are assumed to have base type).

If $s = (\lambda x.u) \cdot v$ and $t = u[x := v]$, then let $\lambda x.u : \sigma \to \tau$. Since $s : \tau$ is η-long by assumption, τ is a base type ι. Thus, $\varphi(s) = \mathtt{app}^{\sigma\to\iota} \cdot (\lambda x.\varphi(u)) \cdot \varphi(v) \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} (\lambda x.\varphi(u)) \cdot \varphi(v) =_\beta \varphi(u)[x := v]$, which by Lemma 25 equals $\varphi(u[x := v]) = \varphi(t)$.

If $s = (\lambda x.u) \cdot v \cdot w_0 \cdots w_n$ and $t = u[x := v] \cdot w_0 \cdots w_n$, let $\lambda x.u : \sigma \to \tau$; then $\tau \notin \mathcal{B}$. Therefore $\varphi(s) = \mathtt{app}^{\sigma\to\tau} \cdot (\lambda x.\varphi(u)) \cdot \varphi(v) \cdot \varphi(w_0) \cdots \varphi(w_n) \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} \mathtt{app}^\tau \cdot ((\lambda x.\varphi(u)) \cdot \varphi(v)) \cdot \varphi(w_0) \cdots \varphi(w_n) =_\beta \mathtt{app}^\tau \cdot \varphi(u)[x := \varphi(v)] \cdot \varphi(w_0) \cdots \varphi(w_n)$. By Lemma 25 this is equal to $\mathtt{app}^\tau \cdot \varphi(u[x := v]) \cdot \varphi(w_0) \cdots \varphi(w_n)$. Noting that u must be an abstraction (since s is η-long, and therefore so is u), this is just $\varphi(s)$.

If $s = l\gamma$ and $t = r\gamma$ for some rule $l \Rightarrow r \in \mathcal{R}$ and substitution γ, then we can safely assume $\mathrm{dom}(\gamma) = FVar(l)$. By Lemma 24 all $\gamma(x)$ are η-long. Thus, Lemma 25 gives that $\varphi(s) = \varphi(l)\gamma^\varphi \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} \varphi(r)\gamma^\varphi = \varphi(t)$.

The other cases follow trivially with the induction hypothesis. The various cases are worked out below for completeness.

If $s = \lambda x.u$ and $t = \lambda x.v$ and $u \Rightarrow_\mathcal{R} v$, then by the induction hypothesis $\varphi(s) = \lambda x.\varphi(u) \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} \lambda x.\varphi(v) = \varphi(t)$.

If $s = (\lambda x.u) \cdot w_0 \cdots w_n$ and $t = (\lambda x.v) \cdot w_0 \cdots w_n$ and $u \Rightarrow_\mathcal{R} v$ and $\lambda x.u : \sigma$, then by the induction hypothesis $\varphi(s) = \mathtt{app}^\sigma \cdot (\lambda x.\varphi(u)) \cdot \varphi(w_0) \cdots \varphi(w_n) \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} \mathtt{app}^\sigma \cdot (\lambda x.\varphi(v)) \cdot \varphi(w_0) \cdots \varphi(w_n) = \varphi(t)$.

If $s = a \cdot s_1 \cdots s_i \cdots s_n$ and $t = a \cdot s_1 \cdots t_i \cdots s_n$ and $s_i \Rightarrow_\mathcal{R} t_i$, with $a : \sigma$ an abstraction, variable or function symbol, let a' be either $\mathtt{app}^\sigma \cdot \varphi(a)$ if a is an

abstraction, or $a' = a$ otherwise. Then $\varphi(s) = a' \cdot \varphi(s_1) \cdots \varphi(s_i) \cdots \varphi(s_n) \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} a' \cdot \varphi(s_1) \cdots \varphi(t_i) \cdots \varphi(s_n) = \varphi(t)$.

Note that Theorem 6 uses *if and only if*. One direction is given by Lemma 26, for the other we need to be able to reverse the transformation. Define a transformation ψ from PRS-terms over $(\mathcal{F}^{\mathtt{AP}}, \mathcal{R}^{\mathtt{AP}})$ to AFS-terms as follows:

$$\begin{array}{lll} \psi(a \cdot s_1 \cdots s_n) & = a \cdot \psi(s_1) \cdots \psi(s_n) & (a \in \mathcal{F} \cup \mathcal{V}) \\ \psi(\mathtt{app}^\sigma \cdot s_0 \cdots s_n) & = \psi(s_0) \cdot \psi(s_1) \cdots \psi(s_n) & \\ \psi(\lambda x.s) & = \lambda x.\psi(s) & \end{array}$$

Lemma 27. *$\psi(s\gamma) = \psi(s)\gamma^\psi$ for all PRS-terms s.*

Proof. By induction on the pre-term $s\gamma$, ordered by the union of $\Rightarrow_\beta$ and the subterm relation.

If $s = \lambda x.t$, then $\psi(s\gamma) = \lambda x.\psi(t\gamma) =$ (IH) $\lambda x.(\psi(t)\gamma^\psi) = \psi(s)\gamma^\psi$.

If $s = \mathtt{app}^\sigma \cdot s_0 \cdots s_n$, then $\psi(s\gamma) = \psi(s_0\gamma) \cdot \psi(s_1\gamma) \cdots \psi(s_n\gamma) =$ (IH) $\psi(s_0)\gamma^\psi \cdot \psi(s_1)\gamma^\psi \cdots \psi(s_n)\gamma^\psi = (\psi(s_0) \cdot \psi(s_1) \cdots \psi(s_n))\gamma^\psi = \psi(s)\gamma^\psi$.

If $s = a \cdot s_1 \cdots s_n$ with $a \in \mathcal{F} \cup \mathcal{V} \setminus \mathrm{dom}(\gamma)$, then $\psi(s\gamma) = a \cdot \psi(s_1\gamma) \cdots \psi(s_n\gamma) =$ (IH) $a \cdot \psi(s_1)\gamma^\psi \cdots \psi(s_n)\gamma^\psi = \psi(s)\gamma^\psi$.

Finally, if $s = x \cdot s_1 \cdots s_n$ with $x \in \mathrm{dom}(\gamma)$, let $\gamma(x) = \lambda y_1 \ldots y_n.t$. Then $\psi(s\gamma) = \psi(t[y_1 := s_1\gamma, \ldots, y_n := s_n\gamma])$; by the induction hypothesis ($\Rightarrow_\beta$ part) this is equal to $\psi(t)[y_1 := \psi(s_1\gamma), \ldots, y_n := \psi(s_n\gamma)]$, which by the subterm part of the induction hypothesis equals $\psi(t)[y_1 := \psi(s_1)\gamma^\psi, \ldots, y_n := \psi(s_n)\gamma^\psi]$. Since $\psi(\gamma(x)) = \lambda y_1 \ldots y_n.\psi(t)$, this is exactly $\psi(s)\gamma^\psi$.

Lemma 28. *For an η, V-long AFS-term s and PRS-substitution γ with domain V: $\psi(\varphi(s)\gamma) = s\gamma^\psi{\uparrow^\eta}$.*

Proof. By induction on the form of s.

If $s = x \in V$, then $\psi(\varphi(s)\gamma) =_{\beta/\eta} \psi(x\gamma) = \gamma^\psi(x) = s\gamma^\psi$, which is η-long.

If $s = x \in \mathcal{V} \setminus V$, then $\psi(\varphi(s)\gamma) = \psi(x{\uparrow^\eta}) = x{\uparrow^\eta} = s\gamma^\psi{\uparrow^\eta}$.

If $s = f \cdot s_1 \cdots s_1$ with $f \in \mathcal{F}$, then $\psi(\varphi(s)\gamma) = \psi(f \cdot \varphi(s_1)\gamma \cdots \varphi(s_n)\gamma) = f \cdot \psi(\varphi(s_1)\gamma) \cdots \psi(\varphi(s_n)\gamma)$, which by the induction hypothesis equals $f \cdot s_1\gamma^\psi \cdots s_n\gamma^\psi = s\gamma^\psi = s\gamma^\psi{\uparrow^\eta}$.

If $s = x \cdot s_0 \cdots s_1$ with $x : \sigma \in \mathcal{V}$, then either $\varphi(x)\gamma = x{\uparrow^\eta}$ or $\varphi(x)\gamma = \gamma(x) = \gamma(x){\uparrow^\eta}$ is an abstraction. Thus, whether $x \in V$ or not, $\psi(\varphi(s)\gamma) = \psi(\mathtt{app}^\sigma \cdot \varphi(x)\gamma \cdot \varphi(s_1)\gamma \cdots \varphi(s_n)\gamma) = \psi(\varphi(x)\gamma) \cdot \psi(\varphi(s_1)\gamma) \cdots \psi(\varphi(s_n)\gamma)$. By the induction hypothesis, that is equal to $x\gamma^\psi \cdot s_1\gamma^\psi \cdots s_n\gamma^\psi = s\gamma^\psi$.

Finally, if $s = (\lambda x.t) \cdot u_0 \cdots u_n$ with $\lambda x.t : \sigma$, then $\psi(\varphi(s)\gamma) = \mathtt{app}^\sigma \cdot (\lambda x.\varphi(t)\gamma) \cdot \varphi(u_0)\gamma \cdots \varphi(u_n)\gamma = (\lambda x.\psi(\varphi(t)\gamma)) \cdot \psi(\varphi(u_0)\gamma) \cdots \psi(\varphi(u_n)\gamma)$, which by the induction hypothesis equals $(\lambda x.t\gamma^\psi) \cdot u_0\gamma^\psi \cdots u_n\gamma^\psi = s\gamma^\psi$.

Lemma 29. *For PRS-terms s, t, if $s \Rightarrow_{\mathcal{R}^{\mathtt{AP}}} t$ then $\psi(s) \Rightarrow_{\mathcal{R}} \psi(t)$.*

Proof. By induction on the form of s.

If $s = \varphi(l)\gamma$ and $t = \varphi(r)\gamma$ for some $l \Rightarrow r \in \mathcal{R}$, then by Lemma 28 $\psi(s) = l\gamma^\psi \Rightarrow_{\mathcal{R}} r\gamma^\psi = \psi(t)$.

If $s = \mathsf{app}^{\sigma\to\iota} \cdot (\lambda x.u) \cdot v$ and $t = u[x := v]$ then $\psi(s) = (\lambda x.\psi(u)) \cdot \psi(v) \Rightarrow_\beta \psi(u)[x := \psi(v)]$, which by Lemma 27 equals $\psi(u[x := v]) = \psi(t)$.

If $s = \mathsf{app}^{\sigma\to\tau} \cdot (\lambda x.u) \cdot v \cdot w_0 \cdots w_n$ and $t = \mathsf{app}^{\tau} \cdot u[x := v] \cdot w_0 \cdots w_n$, then $\psi(s) = (\lambda x.\psi(u)) \cdot \psi(v) \cdot \psi(w_0) \cdots \psi(w_n) \Rightarrow_\beta \psi(u)[x := \psi(v)] \cdot \psi(w_0) \cdots \psi(w_n) = \psi(t)$ again by Lemma 27.

What remains are the various cases for the induction hypothesis, each of which is entirely straightforward.

The proof of Theorem 6 is given by the combination of Lemmas 26 and 29.

A.7 Theorem 7

Lemma 30. *If l is a PRS-term and γ a β/η-normal substitution with domain V, and if variables y in V occur in l only in the form $\lambda x_1 \dots x_n.y \cdot x_1\uparrow^\eta \cdots x_n\uparrow^\eta$, then $\varphi_V(l)\gamma = l\gamma\updownarrow^\eta_\beta$.*

Proof. By induction on the form of l.

If $l = \lambda x_1 \dots x_n.y \cdot x_1\uparrow^\eta \cdots x_n\uparrow^\eta$ with $y \in V$, then $\varphi_V(l) = y$, so $\varphi_V(l)\gamma = \gamma(y) = l\gamma\updownarrow^\eta_\beta$ because γ is in β/η-normal form.

Otherwise, let $l = \lambda x_1 \dots x_n.a \cdot l_1 \cdots l_n$ with a either a function symbol or variable not in V. Then the l_i satisfy the requirements for the induction hypothesis, so $\varphi_V(l)\gamma = \lambda\boldsymbol{x}.a \cdot \varphi_V(l_1)\gamma \cdots \varphi_V(l_n)\gamma =$ (IH) $\lambda\boldsymbol{x}.a \cdot l_1\gamma\updownarrow^\eta_\beta \cdots l_n\gamma\updownarrow^\eta_\beta = (\lambda\boldsymbol{x}.a \cdot l_1 \cdots l_n)\gamma\updownarrow^\eta_\beta = l\gamma\updownarrow^\eta_\beta$.

Lemma 31. *If s is a PRS-term and γ a β/η-normal substitution, and V is a set of variables, then $\varphi_V(s)\gamma\updownarrow^\eta_\beta = s\gamma\updownarrow^\eta_\beta$.*

Proof. This holds because $\varphi_V(s) =_\eta s$ and β/η-equality is preserved under substitution).

Lemma 32. *If $\mathrm{dom}(\gamma) = V$ and s is in η, V-long form, but γ is an η-long substitution, then the AFS-term $s\gamma$ is η-long.*

Proof. With a trivial induction on the form of s. Write $s = \lambda x_1 \dots x_n.t$.

If $t = y \in V$, then $s\gamma = \boldsymbol{x}\gamma(y)$ is η-long because $\gamma(y)$ is.

If $t = y \cdot t_1 \cdots t_m$ with $y \in V$, then $s\gamma = \lambda\boldsymbol{x}.\gamma(y) \cdot t_1\gamma \cdots t_m\gamma$ is η-long because both $\gamma(y)$ is and all $t_i\gamma$ are by induction.

If $t = a \cdot t_1 \cdots t_m$ with a a variable not in V or a function symbol, then $s\gamma = \lambda\boldsymbol{x}.a \cdot t_1\gamma \cdots t_m\gamma$ is η-long because all $t_i\gamma$ are by induction.

Finally, if $t = (\lambda z.u) \cdot t_0 \cdots t_m$, then $s\gamma = \lambda\boldsymbol{x}.(\lambda z.u\gamma) \cdot t_0\gamma \cdots t_m\gamma$ is η-long because $u\gamma$ and all $t_i\gamma$ are.

Lemma 33. *If $s \Rightarrow_{\mathcal{R}} t$ then $s \Rightarrow_{\mathcal{R}^{\mathrm{PA}}} u \Rightarrow^*_\beta t$ (and thus $s \Rightarrow^+_{\mathcal{R}^{\mathrm{PA}},\beta-\mathrm{first}} t$).*

Proof. If $s \Rightarrow_{\mathcal{R}} t$ there is a rule $l \Rightarrow r$, a context C and a substitution γ such that $s = C[l\gamma\updownarrow^\eta_\beta]$ and $t = C[r\gamma\updownarrow^\eta_\beta]$. By Lemma 30 also $s = C[\varphi_{FVar(l)}(l)\gamma]$, which $\Rightarrow_{\mathcal{R}^{\mathrm{PA}}} C[\varphi_{FVar(l)}(r)\gamma] =: u$.

By Lemma 31 this has the same β/η-normal form as $r\gamma\updownarrow^\eta_\beta$, and since by lemma 32 $\varphi_{FVar(l)}(r)\gamma$ is already η-long, $\varphi_{FVar(l)}(r)\gamma\updownarrow^\eta_\beta = \varphi_{FVar(l)}(r)\gamma\downarrow_\beta$. Thus, $\varphi_{FVar(l)}(r)\gamma \Rightarrow^*_\beta r\gamma\updownarrow^\eta_\beta$, and therefore indeed $u \Rightarrow^*_\beta t$.

For the other direction, the proof strategy, we note the following. It has already been proved that an η-long set of rules (such as $\mathcal{R}^{\mathtt{PA}}$) is terminating if and only if it is terminating on η-long terms. The same holds if we employ a β-first reduction strategy:

Lemma 34. *$\Rightarrow_{\mathcal{R}^{\mathtt{PA}}}$ is terminating with a β-first reduction strategy if and only if $\Rightarrow_{\mathcal{R}^{\mathtt{PA}}}$ is terminating with a β-first reduction strategy on η-long terms.*

Proof. Naturally, if $\Rightarrow_{\mathcal{R}^{\mathtt{PA}}}$ is terminating using some strategy, it is terminating with that on specific classes of terms.

For the other direction, it is not hard to see (and in fact, this is how Lemma 2 was proven) that $s \Rightarrow_{\mathcal{R}^{\mathtt{PA}}} t$ implies $s{\uparrow^\eta} \Rightarrow_{\mathcal{R}^{\mathtt{PA}}} \cdot \Rightarrow_\beta^* t{\uparrow^\eta}$ (both for rule steps and β-steps). Thus, let a β-first infinite reduction be given, say $s_0 \Rightarrow_\beta^* t_0 \Rightarrow_\mathcal{R} s_1 \Rightarrow_\beta^* t_1 \ldots$ with all t_i completely β-normal. Then $s_0{\uparrow^\eta} \Rightarrow_\beta^* t_0{\uparrow^\eta} \Rightarrow_\mathcal{R} \cdot \Rightarrow_\beta^* s_1{\uparrow^\eta} \Rightarrow_\beta^* t_1{\uparrow^\eta} \ldots$ which is also an infinite β-first reduction.

Lemma 35. *If $\Rightarrow_{\mathcal{R}^{\mathtt{PA}}}$ is terminating on AFS-terms over $\mathcal{F}^{\mathtt{PA}}$ using a β-first reduction strategy, then $\Rightarrow_\mathcal{R}$ is terminating on PRS-terms over $\mathcal{F}$.*

Proof. Note that the rules in $\mathcal{R}^{\mathtt{PA}}$ have η-long form. Thus, $\Rightarrow_{\mathcal{R}^{\mathtt{PA}}}$ is terminating (using a β-first strategy) if and only if it is terminating (using a β-first strategy) on η-long terms by Lemma 34. Such a reduction has the form $s_0 \Rightarrow_\beta^* t_0 \Rightarrow_\mathcal{R} s_1 \Rightarrow_\beta^* t_1 \Rightarrow_\mathcal{R} \ldots$ with all s_i, t_i in η-long form and all t_i being β-normal.

We will show that $t_0 \Rightarrow_\mathcal{R} t_1 \Rightarrow_\mathcal{R} \ldots$ is an infinite PRS-reduction; since the t_i are indeed both η-long and β-normal this holds if, for all i, $t_i = C[l\gamma{\updownarrow_\beta^\eta}]$ and $C[r\gamma{\updownarrow_\beta^\eta}] = t_{i+1}$ for some $l \Rightarrow r \in \mathcal{R}$, substitution γ and context C.

Now, each t_i has the form $C[\varphi_{FVar(l)}(l)\gamma]$ for $l \Rightarrow r$ a rule in $\mathcal{R}$, γ a substitution and C a context, and $s_{i+1} = C[\varphi_{FVar(l)}(r)\gamma]$. Since l, r have base type, $\varphi_{FVar(l)}(l)\gamma$ is a β/η-normal term. We can safely assume that $\mathrm{dom}(\gamma) = FVar(l)$. Induction on the form of l (in the same way as Lemma 30, so using a set V to describe the pattern property of l) provides trivially that all $\gamma(x)$ are β/η-normal. Thus, in the original PRS, $t_i = C[l\gamma{\updownarrow_\beta^\eta}]$ by Lemma 31.

We are done if $t_{i+1} = s_{i+1}{\downarrow_\beta} = C[r\gamma\,{\updownarrow_\beta^\eta}]$. Since $s_{i+1} = C[\varphi_{FVar(l)}(r)\gamma]$ this holds if $\varphi_{FVar(l)}(r)\gamma\,{\downarrow_\beta} = r\gamma\,{\updownarrow_\beta^\eta}$. As $\varphi_{FVar(l)}(r)\gamma$ is already in η-long form by Lemma 32, $\varphi_{FVar(l)}(r)\gamma{\downarrow_\beta} = \varphi_{FVar(l)}(r)\gamma{\updownarrow_\beta^\eta}$, which by Lemma 31 equals $r\gamma{\updownarrow_\beta^\eta}$ as required.

Theorem 7 is a combination of Lemmas 33 and 35.