Simplifying Algebraic Functional Systems

Cynthia Kop kop@few.vu.nl

Department of Computer Science
VU University Amsterdam, the Netherlands.

Abstract. A popular formalism of higher order rewriting, especially in the light of termination research, are the *Algebraic Functional Systems* (AFSs) defined by Jouannaud and Okada. However, the formalism is very permissive, which makes it hard to obtain results; consequently, techniques are often restricted to a subclass. In this paper we study termination-preserving transformations to make AFS-programs adhere to a number of standard properties. This makes it significantly easier to obtain general termination results.

Key words: higher order term rewriting, algebraic functional systems, termination, transformations, currying, η-expansion

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 [11], path orderings [4,1] and dependency pairs [12,9,8]. Since 2010 the annual termination competition [13] 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, as well as variations and restrictions. Each style has different applications it models better, so it is hard to choose one over the others. Because of the differences, results in one formalism do not, in general, carry over to the next.

Consequently, when the topic of a higher order termination competition was brought up last year, the first question was: "What formalism?" Even having settled on monomorphic Algebraic Functional Systems, neither of the participating groups could immediately deal with the other's benchmarks, since one group used functional syntax and the other applicative.

In this paper we seek to alleviate this situation by studying transformations of Algebraic Functional Systems. AFSs, which were introduced as a modelling of functional programming languages, are an often recurring format in higher-order termination research, although most of the time they appear with some restrictions. Here we study an unrestricted version, with polymorphic types. After transforming an AFS it will satisfy a number of pleasant properties, such as β-normality and possibly η-normality, and we can freely swap between applicative and functional notation. The transformations are designed to have as little impact as possible and to preserve both termination and non-termination.

The aim of this work is to simplify termination research and tools for AFS. Without changing the syntactical freedom of the formalism, most unwelcome intricacies of the syntax can be eliminated in the input module of a tool. Techniques which are defined on a restricted subclass of AFSs (for example when rules are assumed to be η-normal) become generally applicable.

We will first present the definition of (polymorphic) AFSs. Section 3 sketches the problems we aim to solve; Sections 4–8 discuss various transformations. In Section 9 we will say a few words about generalising existing results.

2 Preliminaries

Algebraic Functional Systems (AFSs) were first defined in [3], but we follow (roughly) the more common definitions of [4]. Rather than using type declarations for variables in an environment, we annotate variables with their types directly in terms. This avoids the need to keep track of an environment.

Types Given a set of *type constructors* $\mathcal{B}$, each with a fixed arity $ar(b)$, and a set of *type variables* $\mathcal{A}$, the set of *polymorphic types* is defined by the grammar:

$$\mathcal{T} = \alpha \mid b(\mathcal{T}^n) \mid \mathcal{T} \to \mathcal{T} \qquad (\alpha \in \mathcal{A},\ b \in \mathcal{B},\ ar(b) = n)$$

A *monomorphic* type does not contain type variables. We assume at least one type constructor has arity 0, so monomorphic types exist. A type $\sigma \to \tau$ is *functional*, and a type $b(\sigma_1, \ldots, \sigma_n)$ is a *data type*. Types are written as σ, τ, ρ, data types as ι, κ and type variables as $\alpha, \varepsilon, \omega$. The $\to$ operator associates to the right. A *type declaration* is an expression of the form $(\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$ with $\sigma_1, \ldots, \sigma_n, \tau \in \mathcal{T}$. A type declaration $() \longrightarrow \tau$ is usually just denoted τ. For any type σ, let $FTVar(\sigma)$ be the set of type variables occurring in σ.

Example 1. Examples of monomorphic types are $\mathtt{nat}$, $\mathtt{nat} \to \mathtt{bool}$, and $\mathtt{list}(\mathtt{nat})$, and an example of a non-monomorphic type is $\alpha \to \mathtt{list}(\alpha)$.

A *type substitution* θ is a finite mapping $[\alpha_1 := \sigma_1, \ldots, \alpha_n := \sigma_n]$; $\mathrm{dom}(\theta) = \{\alpha_1, \ldots, \alpha_n\}$, and $\tau\theta$ is the result of replacing all α_i in τ by σ_i (with τ a type or type declaration). We say $\sigma \geq \tau$ if $\tau = \sigma\theta$ for some type substitution θ. For example, $\alpha \geq \alpha \geq \alpha \to \varepsilon \geq \mathtt{nat} \to \mathtt{nat}$, but not $\alpha \to \alpha \geq \mathtt{nat} \to \mathtt{bool}$. Substitutions θ, χ *unify* types σ, τ if $\sigma\theta = \tau\chi$[1]. We will use the following lemma:

Lemma 1 (most general unifiers). *If σ, τ are unifiable, there exist type substitutions θ, χ which unify σ, τ such that for any unifying substitutions θ', χ', there is a type substitution d such that $\theta'_{\restriction FTVar(\sigma)} = d \circ \theta$ and $\chi'_{\restriction FTVar(\tau)} = d \circ \chi$.*

The type substitutions θ, χ in this Lemma are called *most general unifiers*. The composition $d \circ \theta$ is defined as $[\alpha := d(\theta(\alpha)) | \alpha \in \mathrm{dom}(\theta)]$, and $\theta'_{\restriction FTVar(\sigma)}$ is defined as $[\alpha := \theta'(\alpha) | \alpha \in \mathrm{dom}(\theta') \cap FTVar(\sigma)]$.

[1] Note that this definition of unifiers considers the type variables in σ and τ as different entities: the types α and $b(\alpha)$ are unified by $\theta = [\alpha := b(\varepsilon)]$ and $\chi = [\alpha := \varepsilon]$.

Terms Given a set $\mathcal{F}$ of *function symbols*, each equipped with a type declaration (notation f_σ), and an infinite set $\mathcal{V}$ of *variables*, the set of *terms* consists of those expressions for which we can derive $s : \sigma$ for some type σ, using the clauses:

$$\begin{array}{ll}
\texttt{(var)}\ x_\sigma : \sigma & \text{if } x \in \mathcal{V} \text{ and } \sigma \text{ a type} \\
\texttt{(fun)}\ f_\sigma(s_1, \ldots, s_n) : \tau & \text{if } f_\rho \in \mathcal{F} \text{ and } \rho \geq \sigma = (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau \\
 & \quad \text{and } s_1 : \sigma_1, \ldots, s_n : \sigma_n \\
\texttt{(abs)}\ \lambda x_\sigma.s : \sigma \to \tau & \text{if } x \in \mathcal{V},\ \sigma \text{ a type and } s : \tau \\
\texttt{(app)}\ s \cdot t : \tau & \text{if } s : \sigma \to \tau \text{ and } t : \sigma
\end{array}$$

Moreover, variables must have a unique type in s: if for $x \in \mathcal{V}$ both x_σ and x_τ occur in s then $\sigma = \tau$. The abstraction operator λ binds occurrences of a variable as in the λ-calculus; term equality is modulo renaming of variables bound by an abstraction operator (α-conversion). Write $FVar(s)$ for the set of variables in s not bound by a λ. The $\cdot$ operator for application associates to the left. To maintain readability, we will regularly omit explicit type notation in function symbols and variables, and just assume the most general possible type.

Example 2. As a running example we will use the system $\mathcal{F}_{\mathtt{map}}$ with symbols:

$$\left\{ \begin{array}{lll} \mathtt{map}_{((\alpha\to\alpha)\times\mathtt{list}(\alpha))\longrightarrow\mathtt{list}(\alpha)}, & \mathtt{op}_{(\omega\to\varepsilon\times\alpha\to\omega)\longrightarrow\alpha\to\varepsilon}, & \mathtt{nil}_{\mathtt{list}(\alpha)}, \ \mathtt{0}_{\mathtt{nat}}, \\ \mathtt{cons}_{(\alpha\times\mathtt{list}(\alpha))\longrightarrow\mathtt{list}(\alpha)}, & \mathtt{pow}_{(\alpha\to\alpha\times\mathtt{nat})\longrightarrow\alpha\to\alpha}, & \mathtt{s}_{(\mathtt{nat})\longrightarrow\mathtt{nat}} \end{array} \right\}$$

An example term in this system is $\mathtt{map}(\lambda x.\mathtt{s}(x), \mathtt{cons}(\mathtt{0}, \mathtt{nil}))$. Since type annotations have been omitted, they should be imagined as general as possible to keep the term well-typed, so $\mathtt{cons}$, for instance, would be $\mathtt{cons}_{(\mathtt{nat}\times\mathtt{list}(\mathtt{nat}))\longrightarrow\mathtt{list}(\mathtt{nat})}$.

We extend type substitutions and $\geq$ to terms in the obvious way, with a type substitution θ replacing α by $\theta(\alpha)$ in all type denotations in the term.

Example 3. Using the symbols of Example 2, $\mathtt{op}_{(\alpha\to\varepsilon\times\varepsilon\to\alpha)\longrightarrow\varepsilon\to\varepsilon}(F_{\alpha\to\varepsilon}, G_{\varepsilon\to\alpha})[\alpha := \varepsilon, \varepsilon := \mathtt{nat}] = \mathtt{op}_{(\varepsilon\to\mathtt{nat}\times\mathtt{nat}\to\varepsilon)\longrightarrow\mathtt{nat}\to\mathtt{nat}}(F_{\varepsilon\to\mathtt{nat}}, G_{\mathtt{nat}\to\varepsilon})$.

A (term) substitution is the homomorphic extension of a mapping $[x^1_{\sigma_1} := s_1, \ldots, x^n_{\sigma_n} := s_n]$, where each $s_i : \sigma_i$. Substitutions are denoted $\gamma, \delta, \ldots$, the result of applying a substitution $s\gamma$. A substitution can not affect bound variables; applying a substitution $(\lambda x_\sigma.s)\gamma$ assumes x occurs neither in domain nor range of γ (a safe assumption since we can rename bound variables). A *context* is a term C containing a special symbol $\Box_\sigma$. The result of replacing $\Box_\sigma$ in C by a term s of type σ is denoted $C[s]$. Here, s may contain variables bound by C.

β and η $\Rightarrow_\beta$ is the monotonic relation generated by $(\lambda x.s) \cdot t \Rightarrow_\beta s[x := t]$. This relation is strongly normalising and has unique normal forms.

For a given set V of variables, we define restricted η-expansion: $C[s] \hookrightarrow_{\eta,V} C[\lambda x_\sigma.s \cdot x_\sigma]$ if $s : \sigma \to \tau$, x is fresh and the following conditions are satisfied:

1. s is neither an abstraction nor a variable in V
2. s in $C[s]$ is not the left part of an application.

By (2), s is not expanded if it occurs in a subterm of the form $s\ t_1 \cdots t_n$; (1) and (2) together guarantee that $\hookrightarrow_{\eta,V}$ terminates. Therefore every term s has a unique η, V*-long form* $s{\uparrow}^{\eta}_{V}$ which can be found by applying $\hookrightarrow_{\eta,V}$ until it is no longer possible. We say a term s is in η-long form if $s = s{\uparrow}^{\eta} = s{\uparrow}^{\eta}_{\emptyset}$.

Rules and Rewriting An AFS consists of an alphabet $\mathcal{F}$ and a (finite or countably infinite) set $\mathcal{R}$ of *rules*. Rules are tuples $l \Rightarrow r$ where l and r are terms of the same type such that all variables and type variables in r also occur in l. The relation $\Rightarrow_{\mathcal{R}}$ induced by $\mathcal{R}$ is the monotonic relation generated by the β-rule and: $l\theta\gamma \Rightarrow_{\mathcal{R}} r\theta\gamma$ if $l \Rightarrow r \in \mathcal{R}$, θ is a type substitution and γ a substitution.

Example 4. Using the symbols $\mathcal{F}_{\mathtt{map}}$ from Example 2, let $R_{\mathtt{map}}$ be the set:

$$\left\{\begin{array}{ll} \mathtt{map}(F, \mathtt{nil}) & \Rightarrow \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, y)) & \Rightarrow \mathtt{cons}(F \cdot x, \mathtt{map}(F, y)) \\ \mathtt{pow}(F, 0) & \Rightarrow \lambda x.x \\ \mathtt{pow}(F, \mathtt{s}(x)) & \Rightarrow \mathtt{op}(F, \mathtt{pow}(F, x)) \\ \mathtt{op}(F, G) \cdot x & \Rightarrow F \cdot (G \cdot x) \\ \lambda x.F \cdot x & \Rightarrow F \end{array}\right\}$$

As before, types should be imagined as general as possible. The last rule, for example, should be read as $\lambda x_\alpha.F_{\alpha\to\kappa} \cdot x_\alpha \Rightarrow F_{\alpha\to\kappa}$. An example reduction:

$$\begin{array}{ll} \mathtt{map}(\underline{\mathtt{pow}(\lambda x.\mathtt{s}(\mathtt{s}(x)), \mathtt{s}(0))}, \mathtt{cons}(0, \mathtt{nil})) & \Rightarrow_{\mathcal{R}} \\ \underline{\mathtt{map}(\mathtt{op}(\lambda x.\mathtt{s}(\mathtt{s}(x)), \mathtt{pow}(\lambda x.\mathtt{s}(\mathtt{s}(x)), 0)), \mathtt{cons}(0, \mathtt{nil}))} & \Rightarrow_{\mathcal{R}} \\ \mathtt{cons}(\mathtt{op}(\lambda x.\mathtt{s}(\mathtt{s}(x)), \mathtt{pow}(\lambda x.\mathtt{s}(\mathtt{s}(x)), 0)) \cdot 0, \underline{\mathtt{map}(\ldots, \mathtt{nil})}) & \Rightarrow_{\mathcal{R}} \\ \mathtt{cons}(\mathtt{op}(\lambda x.\mathtt{s}(\mathtt{s}(x)), \underline{\mathtt{pow}(\lambda x.\mathtt{s}(\mathtt{s}(x)), 0)}) \cdot 0, \mathtt{nil}) & \Rightarrow_{\mathcal{R}} \\ \mathtt{cons}(\underline{\mathtt{op}(\lambda x.\mathtt{s}(\mathtt{s}(x)), \lambda y.y)} \cdot 0, \mathtt{nil}) & \Rightarrow_{\mathcal{R}} \\ \mathtt{cons}((\lambda x.\mathtt{s}(\mathtt{s}(x))) \cdot (\underline{(\lambda y.y) \cdot 0}), \mathtt{nil}) & \Rightarrow_{\beta} \\ \mathtt{cons}(\underline{(\lambda x.\mathtt{s}(\mathtt{s}(x))) \cdot 0}, \mathtt{nil}) & \Rightarrow_{\beta} \\ \mathtt{cons}(\mathtt{s}(\mathtt{s}(0)), \mathtt{nil}) & \end{array}$$

Note that the $\Rightarrow_\beta$ steps in this are also $\Rightarrow_{\mathcal{R}}$ steps since $\Rightarrow_\beta$ is included in $\Rightarrow_{\mathcal{R}}$ by definition; they are named separately for clarity.

3 Problems

The permissive nature of AFS-syntax makes it difficult to obtain general results. The first issue is the status of application. When extending first order results it is convenient to consider the $\cdot$ operator as a (polymorphic) binary function symbol. But this doesn't work very well with applicative systems, which have rules like $\mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot x \cdot y) \Rightarrow \mathtt{cons} \cdot (F \cdot x) \cdot (\mathtt{map} \cdot F \cdot y)$; AFSs generated from functional programs in e.g. Haskell will commonly have such a form. Due to the repeated occurrence of the $\cdot$ symbol, no version of a higher-order path ordering [4,1] can handle this system. To avoid this problem, we might consider

$\cdot$ as a stronger construction, much like function application; this is done in Nipkow's *Higher-Order Rewriting Systems* (HRSs) [10], where terms are built using only abstraction and application. Ideally, a term $\mathtt{map} \cdot x \cdot (\mathtt{cons} \cdot y \cdot z)$ could be translated to its functional counterpart $\mathtt{map}(x, \mathtt{cons}(y, z))$. But initially this is impossible: a system with the rule $x \cdot 0 \Rightarrow f \cdot 0$ admits a self-loop $f \cdot 0 \Rightarrow f \cdot 0$, whereas the corresponding functional rule, $x \cdot 0 \Rightarrow f(0)$, is terminating.

Another difficulty is the form of the left-hand side of a rule. Methods like the dependency pair approach crucially rely on rules having a form $f(\ldots) \Rightarrow r$ or, in the recent dependency pair analysis for AFSs in [8], $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m \Rightarrow r$. Consequently, systems with rules like $\lambda x.F \cdot x \Rightarrow F$ cannot be handled.

Termination techniques are often defined only on a restricted subset of AFSs. Since most common examples are expressed in a well-behaved manner, this does not seem too high a price. However, a transformational approach, where for instance a term $f(s, t)$ is replaced by $t \cdot s$, is likely to create rules which do not follow the usual assumptions. Instead, we will see how any AFS can be transformed so that all rules have a form $l = f(\boldsymbol{s}) \cdot \boldsymbol{t} \Rightarrow r$ with l and r both β-normal and l not containing leading free variables. We tackle standard assumptions (monomorphism and η-normality) which can be made about terms, and show that, after the first transformations, functional and applicative syntax are interchangeable. We aim to keep the complexity of the transformations minimal: a finite system remains finite after transforming, a monomorphic system remains monomorphic.

4 Polymorphism

In a first step towards simplifying the system, let us investigate polymorphism. To start, observe that polymorphism is only needed to define rules; for termination, at least, we never need to consider polymorphic terms.

Theorem 1. *If a system is non-terminating, there is an infinite reduction on monomorphic terms.*

Proof. Given an infinite reduction $s_0 \Rightarrow_{\mathcal{R}} s_1 \Rightarrow_{\mathcal{R}} \ldots$, let θ be a type substitution which replaces all type variables in s_0 by a type constructor b of arity 0. Since $\Rightarrow_{\mathcal{R}}$ does not create type variables and is closed under type substitution, $s_0\theta \Rightarrow_{\mathcal{R}} s_1\theta \Rightarrow_{\mathcal{R}} \ldots$ is an infinite monomorphic reduction.

Polymorphism has its purpose in defining rules: any set of rules corresponds with a monomorphic set, but instantiating type variables leads to infinitely many rules. Finiteness is a high price to pay, since both humans and computers have trouble with the infinite. Nevertheless, from a perspective of reasoning we might as well use monomorphic rules, as long as we remember how they were generated.

Let a *rule scheme* be a pair $l \Rightarrow r$ of equal-typed (polymorphic) terms, such that all free variables and type variables occurring in r also occur in l. Given a set R of rule schemes, let $\mathcal{R}_R = \{l\theta \Rightarrow r\theta | l \Rightarrow r \in R$ and θ a type substitution mapping all type variables in l to monomorphic types$\}$. The following is evident:

Theorem 2. *For a given set of rule schemes R, the set $\mathcal{R}_R$ is a set of monomorphic rules and $\Rightarrow_R$ is terminating if and only if $\Rightarrow_{\mathcal{R}_R}$ is.*

Henceforth, *rules* are understood to be monomorphic. *Rule schemes* may not be. $\mathcal{R}$ indicates a set of rules, R a set of rule schemes.

A pleasant consequence of using monomorphic rules is that type substitution is no longer needed to define the rewrite relation $\Rightarrow_{\mathcal{R}}$; $s \Rightarrow_{\mathcal{R}} t$ if either $s \Rightarrow_\beta t$ or $s = C[l\gamma]$ and $t = C[r\gamma]$ for some substitution γ, context C and rule $l \Rightarrow r$.

In the following sections we will define transformations on a set of rule schemes. Note that a set $\mathcal{R}$ of rules is always generated from a set of rule schemes, since even for monomorphic systems $R := \mathcal{R}$ is a suitable set.

5 Leading Variables

The presence of leading variables in the left-hand side of a rule $l \Rightarrow r$ (that is, subterms $x \cdot s$ where x is free in l) hinders techniques like dependency pairs[2] and makes it impossible to swap freely between functional and applicative notation (see also Section 7). We can avoid this problem by making application a function symbol: replace $s \cdot t$ everywhere by $@_{(\sigma\to\tau\times\sigma)\longrightarrow\tau}(s,t)$ and add $@_{(\alpha\to\varepsilon\times\alpha)\longrightarrow\tau}(x,y) \Rightarrow x \cdot y$ to R. The resulting system is terminating if and only if the original was. However, as discussed in Section 3, this transformation is not very good. A mostly applicative system would become almost impossible to handle with conventional techniques. In addition, the new rule scheme uses type variables, while the original system might be monomorphic. Thus, we will use a more complicated transformation that leads to an easier system.

Sketch of the Transformation We sketch the idea for transforming a monomorphic system. Polymorphism brings in additional complications, but the rough idea is the same. First we instantiate headmost variables with functional terms: for any rule $l = C[x \cdot s] \Rightarrow r$ all possible rules $l[x := f(\boldsymbol{y}) \cdot \boldsymbol{z}] \Rightarrow r[x := f(\boldsymbol{y}) \cdot \boldsymbol{z}]$ are added. Now when a rule with leading variables is used, we can assume these variables are not instantiated with a functional term. Second, we introduce a symbol @ and replace occurrences $s \cdot t$ in any rule by $@(s,t)$ if s is not functional, and its type corresponds with the type of a leading variable in any left-hand side. We add rules $@_\sigma(x,y) \Rightarrow x \cdot y$ only for those $@_\sigma$ occurring in the changed rules. With this transformation, the applicative `map` rule $\mathtt{map}_{(\mathtt{nat}\to\mathtt{nat})\to\mathtt{list(nat)}\to\mathtt{list(nat)}}$ $F \cdot (\mathtt{cons} \cdot x \cdot y) \Rightarrow \mathtt{cons} \cdot (F \cdot x) \cdot (\mathtt{map} \cdot F \cdot y)$ either stays unchanged (if there are no rules with a leading variable of type $\mathtt{nat} \to \mathtt{nat}$ in the left-hand side) or becomes $\mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot x \cdot y) \Rightarrow \mathtt{cons} \cdot @(F,x) \cdot (\mathtt{map} \cdot F \cdot y)$ (if there are).

[2] Leading variables in the right-hand side *also* complicate dependency pairs, but are harder to avoid; existing methods use various techniques to work around this issue.

5.1 Output Arity

Polymorphism complicates this transformation. Even with finite $\mathcal{F}$ there may be infinitely many terms of the form $f(\boldsymbol{x}) \cdot \boldsymbol{y}$. So assign to every $f_\sigma \in \mathcal{F}$ an integer $oa(f) \geq 0$; terms of the form $f(\boldsymbol{s}) \cdot t_1 \cdots t_m$ with $m \leq oa(f)$ are "protected". The choice for oa is not fixed, but has one restriction: $oa(f) > 0$ for only finitely many symbols f. Typically, if $\sigma = (\boldsymbol{\tau}) \longrightarrow \rho_1 \to \ldots \to \rho_m \to \iota$ we choose $oa(f) = m$. We may also choose the highest number m such that $f(\boldsymbol{s}) \cdot t_1 \cdots t_m$ occurs in any rule scheme. Alternatively, in a (mostly) functional system we could choose $oa(f) = 0$ for all f; Transformations 1-2 have no effect then. We say a term is *limited functional* if it has the form $f(\boldsymbol{s}) \cdot t_1 \cdots t_m$ with $m < oa(f)$ (note that $f(\boldsymbol{s})$ is *not* limited functional if $oa(f) = 0$!).

Example 5. We follow the second guideline, so $oa(f)$ is the highest number m such that $f(\boldsymbol{s}) \cdot t_1 \cdots t_m$ occurs in a rule scheme. In the system $R_{\mathtt{map}}$ from Example 4 this gives output arity 0 for all symbols except $\mathtt{op}$, which gets output arity 1.

To start, we adjust rules for the chosen output arity. For any rule $f(\boldsymbol{s}) \cdot t_1 \cdots t_n \Rightarrow r$ with $n < oa(f)$, we add a new rule $f(\boldsymbol{s}) \cdot t_1 \cdots t_n \cdot x \Rightarrow r \cdot x$. This is done because an application of the form $f(\boldsymbol{s}) \cdot \boldsymbol{t} \cdot u$ will be "protected" while $r \cdot u$ may not be.

Transformation 1 *(respecting output arity)* Given a set of rule schemes R, for every $l \Rightarrow r \in R$ with l limited functional add a rule scheme $l \cdot x \Rightarrow r \cdot x$ if this is well-typed (if $l : \alpha$ first apply the type substitution $[\alpha := \alpha \to \varepsilon]$). Repeat for all newly added rule schemes. This process terminates because $oa(f)$ is bounded, and the result, R^{res}, is finite if R is. $\Rightarrow_{R^{res}}$ and $\Rightarrow_R$ define the same relation.

Example 6. Since none of the left-hand sides of $R_{\mathtt{map}}$ are limited functional, Transformation 1 has no effect. If we had chosen, for example, $oa(\mathtt{pow}) = 2$ (this is allowed, even if there is no point in doing so), then we would have had to add four additional rules:

$$\begin{array}{llll} \mathtt{pow}_\sigma(F, 0) \cdot y & \Rightarrow (\lambda x.x) \cdot y & \mathtt{pow}_\sigma(F, \mathtt{s}(x)) \cdot y & \Rightarrow \mathtt{op}(F, \mathtt{pow}(F, x)) \cdot y \\ \mathtt{pow}_\tau(F, 0) \cdot y \cdot z & \Rightarrow (\lambda x.x) \cdot y \cdot z & \mathtt{pow}_\tau(F, \mathtt{s}(x)) \cdot y \cdot z & \Rightarrow \mathtt{op}(F, \mathtt{pow}(F, x)) \cdot y \cdot z \end{array}$$

Here, σ is the type declaration $(\alpha \to \alpha \times \mathtt{nat}) \longrightarrow \alpha \to \alpha$ and τ is $((\alpha \to \varepsilon) \to (\alpha \to \varepsilon) \times \mathtt{nat}) \longrightarrow (\alpha \to \varepsilon) \to \alpha \to \varepsilon$.

5.2 Filling in Head Variables

With this preparation, we can proceed to a larger transformation. Let $HV(s)$ be the set of those $x_\sigma \in FVar(s)$ where x_σ occurs at the head of an application in s (so $s = C[x_\sigma \cdot t]$ for some C, t). We will replace any rule $l = C[x_\sigma \cdot t] \Rightarrow r$ by a set of rules where a limited functional term $f(\boldsymbol{y}) \cdot \boldsymbol{z}$ is substituted for x_σ.

Transformation 2 *(filling in head variables)* For every rule scheme $l \Rightarrow r$ in R^{res}, every $x_\sigma \in HV(l)$, every function symbol $f_\tau \in \mathcal{F}$ and $n < oa(f)$ such that $(\ldots) \longrightarrow \alpha_1 \to \ldots \to \alpha_n \to \sigma$ unifies with τ, let θ, χ be their most general unifiers and add a rule scheme $l\theta\delta \Rightarrow r\theta\delta$, where $\delta = [x_{\sigma\theta} := f_{\tau\chi}(\boldsymbol{y}) \cdot z_1 \cdots z_n]$ (with $y_1, \ldots, y_k, z_1, \ldots, z_n$ fresh variables). Repeat this for the newly added rule schemes. If R^{res} is finite, this process terminates and the result, R^{fill}, is also finite. Otherwise define R^{fill} as the limit of the procedure.

Example 7. Following on Example 6, the only rule with a leading variable in the left-hand side is the η-reduction rule $\lambda x_{\alpha_1}.F_{\alpha_1\to\alpha_2}\ x_{\alpha_1} \Rightarrow F_{\alpha_1\to\alpha_2}$. Consequently, Transformation 2 completes after one step, with a single new rule; R^{fill} contains:

$$
\begin{array}{llll}
\mathtt{map}(F, \mathtt{nil}) & \Rightarrow \mathtt{nil} & \mathtt{op}(F, G) \cdot x & \Rightarrow F \cdot (G \cdot x) \\
\mathtt{map}(F, \mathtt{cons}(x, y)) & \Rightarrow \mathtt{cons}(F \cdot x, \mathtt{map}(F, y)) & \lambda x.F \cdot x & \Rightarrow F \\
\mathtt{pow}(F, 0) & \Rightarrow \lambda x.x & \lambda x.\mathtt{op}(F, G) \cdot x & \Rightarrow \mathtt{op}(F, G) \\
\mathtt{pow}(F, \mathtt{s}(x)) & \Rightarrow \mathtt{op}(F, \mathtt{pow}(F, x)) & &
\end{array}
$$

It is not hard to see that R^{fill} and R^{res} generate the same relation. Moreover:

Lemma 2. *If $s \Rightarrow_{R^{fill}} t$ with a topmost step, then there are $l \Rightarrow r \in R^{fill}$, type substitution θ and substitution γ such that $s = l\theta\gamma$, $t = r\theta\gamma$ and $\gamma(x)$ is not limited functional for any $x \in HV(l)$.*

Proof. By definition of topmost step, there exist l, r, θ, γ such that $s = l\theta\gamma$ and $t = r\theta\gamma$; using induction on the size of $\{x_\sigma | x_\sigma \in HV(l) | \gamma(x_{\sigma\theta})$ is limited functional$\}$ and the definition of R^{fill} the Lemma follows without much effort.

5.3 Preparing Polymorphic Types

As suggested in the sketch of the transformation, we will introduce new symbols $@_\sigma$ only for those σ where it is necessary. Formally, let S be a set of functional types such that its closure under type substitution, S^c, contains all types σ where $x_\sigma \in HV(l)$ for some variable x and $l \Rightarrow r \in R^{fill}$. Transformation 4 will replace subterms $u \cdot v$ by $@(u, v)$ if $u : \tau \leq \sigma$ and u is not limited functional. There is one remaining problem: a subterm $u \cdot v$ where the type of u unifies with a type in S but is not an instance. We deal with this problem by adding some rule schemes.

Transformation 3 *(S-normalising the rules)* For every rule scheme $l \Rightarrow r \in R^{fill}$, add a rule scheme $l\theta \Rightarrow r\theta$ if either l or r has a subterm $s \cdot t$ with $s : \sigma$ not limited functional, and σ unifies with a type $\tau \in S$ such that $\tau \not\geq \sigma$. Here, θ and some χ are the most general unifiers of σ and τ. Repeat this for the newly added rules. If S and R^{fill} are both finite, this procedure terminates and the result, R^{normS}, is finite. Otherwise we define R^{normS} as the limit of the procedure.

Example 8. Consider our main Example; R^{fill} is given in Example 7. We must choose $S = \{\alpha \to \varepsilon\}$ due to the rule $\lambda x.F_{\alpha\to\varepsilon} \cdot x \Rightarrow F$. But $\alpha \to \varepsilon \geq$ any functional type, so Transformation 3 has no effect.

Again it is evident that the rewrite relations generated by R^{normS} and R^{fill} are the same. Moreover, we can derive the following (technical) result:

Lemma 3. *For $l \Rightarrow r \in R^{fill}$, type substitution θ, there are $l' \Rightarrow r' \in R^{normS}$ and type substitution χ such that $l\theta = l'\chi$, $r\theta = r'\chi$ and for $u \cdot v$ occurring in l' or r' with $u : \sigma$ not limited functional, either both $\sigma, \sigma\chi \in S^c$ or both $\sigma, \sigma\chi \notin S^c$.*

5.4 Explicit Application

Finally, then, we move on to the main substitution. For every type $\sigma \to \tau \in S$, introduce a new symbol $@_{(\sigma\to\tau\times\sigma)\longrightarrow\tau}$ and for all terms s define $\mathtt{exp}(s)$ as follows:

$$\begin{array}{ll}
\mathtt{exp}(f(s_1,\ldots,s_n)) & = f(\mathtt{exp}(s_1),\ldots,\mathtt{exp}(s_n)) \\
\mathtt{exp}(x) & = x \ (x \text{ a variable}) \\
\mathtt{exp}(\lambda x.s) & = \lambda x.\mathtt{exp}(s) \\
\mathtt{exp}(s \cdot t) & = \begin{cases} \mathtt{exp}(s) \cdot \mathtt{exp}(t) \text{ if } s \text{ limited functional or type}(s) \notin S^c \\ @(\mathtt{exp}(s), \mathtt{exp}(t)) \ \text{ otherwise} \end{cases}
\end{array}$$

That is, subterms $s \cdot t$ are replaced by $@_\sigma(s,t)$, provided the split does not occur in a "protected" functional term, and s has a "dangerous" type.

Transformation 4 *(Embedding head symbols)* Let $R^{\mathrm{noapp}} = \{\mathtt{exp}(l) \Rightarrow \mathtt{exp}(r) | l \Rightarrow r \in R^{normS}\} \cup \{@_{(\sigma\to\tau\times\sigma)\longrightarrow\tau}(x,y) \Rightarrow x \cdot y | \sigma \to \tau \in S\}$.

Transformations 1–4 preserve monomorphism and finiteness, yet R^{noapp} will not have leading (free) variables. We pose the main theorem of Section 5.

Theorem 3. *The rewrite relation $\Rightarrow_{R^{\mathrm{noapp}}}$ generated by R^{noapp} is terminating if and only if $\Rightarrow_R$ is.*

Proof (Sketch). For one direction, if $s \Rightarrow_{R^{\mathrm{noapp}}} t$ then also $s' \Rightarrow^{=}_{R^{normS}} t'$, where s', t' are s, t with occurrences of $@(u,v)$ replaced by $u \cdot v$. Equality only occurs if s has less @ symbols than t, so any infinite $\Rightarrow_{R^{\mathrm{noapp}}}$ reduction leads to an infinite $\Rightarrow_{R^{normS}}$ reduction, and R^{normS} defines the same relation as $\mathcal{R}$. For the other direction, $s \Rightarrow_{R^{res}} t$ implies $\mathtt{exp}(s) \Rightarrow^{+}_{R^{\mathrm{noapp}}} \mathtt{exp}(t)$ by induction on the size of s. For the induction step the only difficult case is when $s = u \cdot v \Rightarrow_{R^{res}} u' \cdot v$ with u limited functional while u' is not, but using Transformation 1 we can assume this is a topmost step. For the base case, if $s \Rightarrow_{R^{res}} t$ by a topmost rule step, we note that using Lemmas 2 and 3, $s = l\theta\gamma$ and $t = r\theta\gamma$ with $l \Rightarrow r \in R^{normS}$, $\gamma(x)$ not limited functional if $x \in HV(l\theta)$ and for any subterm $u \cdot v$ of l with $u : \tau$, either both τ and $\tau\theta \in S^c$ or neither. With these facts it is easy to show (using induction on the definition of $\mathtt{exp}$) that $\mathtt{exp}(l\theta\gamma) = \mathtt{exp}(l)\theta\gamma^{\mathtt{exp}}$ and $\mathtt{exp}(r)\theta\gamma^{\mathtt{exp}} \Rightarrow^{*}_{R^{\mathrm{noapp}}} \mathtt{exp}(r\theta\gamma)$. If $s \Rightarrow_\beta t$ we use that $\mathtt{exp}(u)[x := \mathtt{exp}(v)] \Rightarrow^{*}_{R^{\mathrm{noapp}}} \mathtt{exp}(u[x := v])$. Thus, any $\Rightarrow_{\mathcal{R}}$ reduction leads to a $\Rightarrow_{R^{\mathrm{noapp}}}$ reduction of at least equal length.

Example 9. Considering our example with $S = \{\alpha \to \varepsilon\}$, R^{noapp} consists of:

$$\begin{array}{llll}
\mathtt{map}(F, \mathtt{nil}) & \Rightarrow \mathtt{nil} & \mathtt{op}(F,G) \cdot x & \Rightarrow @(F, @(G,x)) \\
\mathtt{map}(F, \mathtt{cons}(x,y)) & \Rightarrow \mathtt{cons}(@(F,x), \mathtt{map}(F,y)) & \lambda x.@(F,x) & \Rightarrow F \\
\mathtt{pow}(F, 0) & \Rightarrow \lambda x.x & \lambda x.\mathtt{op}(F,G) \cdot x & \Rightarrow \mathtt{op}(F,G) \\
\mathtt{pow}(F, \mathtt{s}(x)) & \Rightarrow \mathtt{op}(F, \mathtt{pow}(F,x)) & @(F,x) & \Rightarrow F \cdot x
\end{array}$$

6 Abstractions in left-hand sides and β-redexes

The next step is to get rid of rule schemes $\lambda x.l \Rightarrow r$, where an abstraction is reduced directly; rules like this will form a definite blockade to working with η-expanded terms and they make it hard to define dependency pairs. The solution is very similar to the one employed in Section 5: we identify the types of all rule schemes of this form, and replace abstractions $\lambda x.s$ of such a type σ by $\Lambda_\sigma(\lambda x.s)$, where Λ_σ is a new function symbol. As a side bonus, we will get rid of any remaining β-redexes in the rule schemes (note that the transformations of Section 5 may already have removed such redexes).

Formally, let Q be a set of types such that its closure under type substitution, Q^c, contains all types σ such that $\lambda x.l : \sigma \Rightarrow r \in R$, or $(\lambda x.s) \cdot t$ occurs in any rule scheme. We could choose the set of all such types, or for instance $\{\alpha \to \varepsilon\}$. As before we need to prepare polymorphic rule schemes for a type match.

Transformation 5 *(Q-normalising the rules)* For every rule scheme $l \Rightarrow r \in R$, add a rule scheme $l\theta \Rightarrow r\theta$ if either l or r has a subterm $\lambda x.s : \sigma$, and σ unifies with a type $\tau \in Q$ such that $\tau \not\geq \sigma$. Here, θ and some χ are the most general unifiers of σ and τ. Repeat this for the newly added rules. If Q and R are both finite, this procedure terminates and the result, R^{normQ}, is finite. Otherwise define R^{normQ} as the limit of the procedure.

We can derive a Lemma very similar to Lemma 3, but it would not bring much news. Let us instead pass straight to the main transformation:

$$\begin{array}{ll}
\mathtt{expL}(f(s_1,\ldots,s_n)) = f(\mathtt{expL}(s_1),\ldots,\mathtt{expL}(s_n)) \\
\mathtt{expL}(s \cdot t) = \mathtt{expL}(s) \cdot \mathtt{expL}(t) \\
\mathtt{expL}(x) = x \ (x \text{ a variable}) \\
\mathtt{expL}(\lambda x.s) = \begin{cases} \Lambda_{(\sigma)\longrightarrow\sigma}(\lambda x.\mathtt{expL}(s)) & \text{if } \lambda x.s : \sigma \in Q^c \\ \lambda x.\mathtt{expL}(s) & \text{otherwise} \end{cases}
\end{array}$$

Transformation 6 *(Marking Abstractions)* $R^\Lambda := \{\mathtt{expL}(l) \Rightarrow \mathtt{expL}(r) | l \Rightarrow r \in R^{normQ}\} \cup \{\Lambda_{(\sigma)\longrightarrow\sigma}(x) \Rightarrow x | \sigma \in S\}$

It is evident that R^Λ has no rule schemes of the form $\lambda x.l \Rightarrow r$ and is β-normal. Moreover, its termination is equivalent to termination of the original system.

Theorem 4. *$\Rightarrow_{R^\Lambda}$ is terminating if and only if $\Rightarrow_R$ is.*

Proof. It is not too hard to derive that $s \Rightarrow_R t$ implies $\mathtt{expL}(s) \Rightarrow^+_{R^\Lambda} \mathtt{expL}(t)$, using that $\mathtt{expL}(C[u]) = \mathtt{expL}(C)[\mathtt{expL}(u)]$ and a separate induction for the top step (using Transformation 5 to choose the right rule and type substitution). Defining s', t' as s, t with occurrences of any Λ_σ erased, it is also obvious that $s \Rightarrow_{R^\Lambda} t$ implies $s' \Rightarrow^=_R t'$, with equality only if the former was Λ-erasing.

Example 10. Continuing the transformation of $R_{\mathtt{map}}$, we choose $Q = \{\alpha \to \varepsilon\}$ (we have no other choice, because of the rule $\lambda x.@(F_{\alpha\to\varepsilon}, x) \Rightarrow F_{\alpha\to\varepsilon}$). Transformation 5 has no effect, and Transformation 6 introduces Λ around all abstractions:

$\mathtt{map}(F, \mathtt{nil})$	$\Rightarrow \mathtt{nil}$	$\Lambda(\lambda x.@(F, x))$	$\Rightarrow F$
$\mathtt{map}(F, \mathtt{cons}(x, y))$	$\Rightarrow \mathtt{cons}(@(F, x), \mathtt{map}(F, y))$	$\Lambda(\lambda x.\mathtt{op}(F, G) \cdot x)$	$\Rightarrow \mathtt{op}(F, G)$
$\mathtt{pow}(F, 0)$	$\Rightarrow \Lambda(\lambda x.x)$	$\Lambda_{\alpha\to\varepsilon}(F)$	$\Rightarrow F$
$\mathtt{pow}(F, \mathtt{s}(x))$	$\Rightarrow \mathtt{op}(F, \mathtt{pow}(F, x))$	$@(F, x)$	$\Rightarrow F \cdot x$
$\mathtt{op}(F, G) \cdot x$	$\Rightarrow @(F, @(G, x))$		

Summing Up Combining Sections 5 and 6, we can transform a set of rule schemes, without affecting termination, to satisfy the following properties:

1. both sides of rule schemes are β-normal
2. left-hand sides l have no subterms $x \cdot s$ with x a free variable
3. left-hand sides have the form $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$ with $m \geq n$

Property (3) holds by elimination: after transforming, the left-hand side of a rule scheme is neither an abstraction, nor an application headed by an abstraction or variable. If it is a variable, $x \Rightarrow r \in R$, the AFS is non-terminating and (since termination is all we are interested in) we might replace R by the set $\{a \Rightarrow a\}$.

Henceforth, rule schemes are assumed to satisfy the requirements listed above.

7 Currying

Let us turn our eyes to the status of application. As mentioned in Section 3, an applicative AFS cannot be handled with most existing termination techniques, nor can we naively turn it into a functional system. The issues are partial application (an applicative `map` system has terms like $\mathtt{map} \cdot \mathtt{s}$ which have no functional counterpart) and leading free variables (a terminating rule $g \cdot (x \cdot 0) \Rightarrow g \cdot f(0)$ has an applicative counterpart $g \cdot (x \cdot 0) \Rightarrow g \cdot (f \cdot 0)$ which is not terminating). However, we have excluded rules with leading free variables in the left-hand side. The issue of partial application can be dealt with using η-expansion.

There are two directions we might take. Usually, we would like to *uncurry* an applicative system, transforming a term $f \cdot s \cdot t$ into $f(s, t)$. Such a form is more convenient in for instance path orderings, or to define argument filterings. On the other hand, we will have to deal with application anyway, since it is part

of the term syntax; to simplify the formalism it might be a good move to *curry* terms, making the system entirely applicative.

Transformation 7 *(Currying)* Let R be a set of rules schemes over a set of function symbols $\mathcal{F}$. We define the following mapping on type declarations: $\mathtt{flat}((\sigma_1 \times \ldots \sigma_n) \longrightarrow \tau) = \sigma_1 \to \ldots \to \sigma_n \to \tau$. Next we define the mapping $\mathtt{flat}$ from functional terms over $\mathcal{F}$ to applicative terms over the 'flattened version' of $\mathcal{F}$, notation $\mathcal{F}^{\mathtt{flat}}$, as follows:

$$\begin{array}{ll} \mathtt{flat}(f_\sigma(s_1,\ldots,s_n)) & = f_{\mathtt{flat}(\sigma)} \cdot \mathtt{flat}(s_1) \cdots \mathtt{flat}(s_n) \\ \mathtt{flat}(\lambda x.s) & = \lambda x.\mathtt{flat}(s) \\ \mathtt{flat}(s \cdot t) & = \mathtt{flat}(s) \cdot \mathtt{flat}(t) \\ \mathtt{flat}(x) & = x \quad (x \text{ a variable}) \end{array}$$

The flattened version $R^{\mathtt{flat}}$ of the set of rule schemes R consists of the rule scheme $\mathtt{flat}(l) \Rightarrow \mathtt{flat}(r)$ for every rule scheme $l \Rightarrow r$ in R.

Theorem 5. *$\Rightarrow_R$ is well-founded on terms over $\mathcal{F}$ if and only if $\Rightarrow_{R^{\mathtt{flat}}}$ is well-founded on terms over $\mathcal{F}^{\mathtt{flat}}$.*

Proof. It is easy to see that $s \Rightarrow_R t$ implies $\mathtt{flat}(s) \Rightarrow_{R^{\mathtt{flat}}} \mathtt{flat}(t)$ (with induction on the size of s, and a separate induction for topmost steps to see that flattening is preserved under substitution); this provides one direction. For the other, let $\mathtt{flat}^{-1}$ be the "inverse" transformation of $\mathtt{flat}$, which maps occurrences of $f \cdot s_1 \cdots s_k$ with $f_{(\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau} \in \mathcal{F}$ to $\lambda x_{k+1} \ldots x_n.f(\mathtt{flat}^{-1}(s_1),\ldots,\mathtt{flat}^{-1}(s_k), x_{k+1},\ldots,x_n)$ if $k < n$ or to $f(\mathtt{flat}^{-1}(s_1),\ldots,\mathtt{flat}^{-1}(s_n)) \cdot \mathtt{flat}^{-1}(s_{n+1}) \cdots \mathtt{flat}^{-1}(s_k)$ otherwise. It is not hard to see that $\mathtt{flat}^{-1}(s)[\boldsymbol{x} := \mathtt{flat}^{-1}(\boldsymbol{t})] \Rightarrow_\beta^* \mathtt{flat}^{-1}(s[\boldsymbol{x} := \boldsymbol{t}])$, and this $\Rightarrow_\beta^*$ is an equality if $HV(s) = \emptyset$. Therefore, and because $\mathtt{flat}^{-1}(R^{\mathtt{flat}})$ is exactly R, $\mathtt{flat}^{-1}(s) \Rightarrow_R^+ \mathtt{flat}^{-1}(t)$ holds if $s \Rightarrow_{R^{\mathtt{flat}}} t$.

Note the *if and only if* in Theorem 5. Because of this equivalence the theorem works in two ways. We can turn a functional system applicative, but also turn an applicative system functional, simply by taking the inverse of Transformation 7. For an applicative system, there are usually many sets of corresponding functional rules, all of which are equivalent for the purpose of termination.

Example 11. Our running example can be transformed into the applicative AFS:

$$\begin{array}{llll} \mathtt{pow} \cdot F \cdot 0 & \Rightarrow \Lambda \cdot (\lambda x.x) & \Lambda \cdot (\lambda x.@ \cdot F \cdot x) & \Rightarrow F \\ \mathtt{pow} \cdot F \cdot (\mathtt{s} \cdot x) & \Rightarrow \mathtt{op} \cdot F \cdot (\mathtt{pow} \cdot F \cdot x) & \Lambda \cdot (\lambda x.\mathtt{op} \cdot F \cdot G \cdot x) & \Rightarrow \mathtt{op} \cdot F \cdot G \\ \mathtt{op} \cdot F \cdot G \cdot x & \Rightarrow @ \cdot F \cdot (@ \cdot G \cdot x) & @ \cdot F \cdot x & \Rightarrow F \cdot x \\ \mathtt{map} \cdot F \cdot \mathtt{nil} & \Rightarrow \mathtt{nil} & \Lambda \cdot F & \Rightarrow F \\ \mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot x \cdot y) & \Rightarrow \mathtt{cons} \cdot (@ \cdot F \cdot x) \cdot (\mathtt{map} \cdot F \cdot y) & & \end{array}$$

Related Work In first-order rewriting, the question whether properties such as confluence and termination are preserved under currying or uncurrying is studied in [5,6,2]. In [6] a currying transformation from (functional) term rewriting systems (TRSs) into applicative term rewriting systems (ATRSs) is defined; a TRS is terminating if and only if its curried form is. In [2], an uncurrying transformation from ATRSs to TRSs is defined that can deal with partial application and leading variables, as long as they do not occur in the left-hand side of rewrite rules. This transformation is sound and complete with respect to termination.

However, these results do not apply to AFSs, both due to the presence of typing and because AFSs use a mixture of functional and applicative notation. We may for instance have terms of the form $f(x)\cdot y$, and currying might introduce new interactions via application.

8 η-expansion

Finally, we consider η-expansion. It would often be convenient if we could assume that every term of some functional type $\sigma \to \tau$ has the form $\lambda x_\sigma.s$, which only reduces if its subterm s does. This is the case if we work modulo η, equating $s : \sigma \to \tau$, with s not an abstraction, to $\lambda x_\sigma.(s \cdot x_\sigma)$. As is well-known, simply working modulo η in the presence of β-reduction causes problems. Instead, we will limit reasoning to η-long terms.

Theorem 6. *Let $\mathcal{R}$ be a set of rules in restricted η-long form, that is, $l = l{\uparrow^\eta_{FVar(l)}}$ and $r = r{\uparrow^\eta_{FVar(r)}}$ for every rewrite rule $l \Rightarrow r$ in $\mathcal{R}$. Then the set of η-long terms is closed under rewriting. Moreover, the rewrite relation $\Rightarrow_\mathcal{R}$ is terminating on terms iff it is terminating on η-long terms.*

Proof. Evidently, if $\Rightarrow_\mathcal{R}$ is terminating then it is terminating on all η-long terms. For the less obvious direction, we see that $s \Rightarrow_\mathcal{R} t$ implies $s{\uparrow^\eta} \Rightarrow^+_\mathcal{R} t{\uparrow^\eta}$. Hence any infinite reduction can be transformed to an infinite reduction on η-long terms. Writing $\gamma^\uparrow := \{x \mapsto \gamma(x){\uparrow^\eta} | x \in \mathrm{dom}(\gamma)\}$ a simple inductive reasoning shows that $s{\uparrow^\eta_V}\gamma^\uparrow \Rightarrow^*_\beta s\gamma{\uparrow^\eta}$ if $V = \mathrm{dom}(\gamma)$, and this is an equality if $HV(s) = \emptyset$. Thus, if $s \Rightarrow_\mathcal{R} t$ by a topmost reduction, then also $s{\uparrow^\eta} = l\gamma{\uparrow^\eta} = l{\uparrow^\eta_{FVar(l)}}\gamma^\uparrow = l\gamma^\uparrow \Rightarrow_\mathcal{R} r\gamma^\uparrow = r{\uparrow^\eta_{FVar(r)}}\gamma^\uparrow \Rightarrow_\beta r\gamma{\uparrow^\eta} = t{\uparrow^\eta}$. This forms the base case for an induction on s, which proves $s{\uparrow^\eta} \Rightarrow^+_\mathcal{R} t{\uparrow^\eta}$ whenever $s \Rightarrow_\mathcal{R} t$.

The requirement that the rules should be η-long is essential. Consider for example the system with a single rule $f_{o\to o} \cdot x_o \Rightarrow g_{(o\to o)\to o} \cdot f_{o\to o}$. The relation generated by this rule is terminating, but evidently the set of η-long terms is not closed under rewriting. The η-long variation of this rule, $f_{o\to o} \cdot x_o \Rightarrow g_{(o\to o)\to o} \cdot (\lambda y_o.f_{o\to o} \cdot y_o)$, is not terminating, as the left-hand side can be embedded in the right-hand side. This example is contrived, but it shows that we cannot be careless with η-expansion. However, when developing methods to prove termination of a system the most essential part of any transformation is to preserve *non-termination*. At the price of completeness, we can use Transformation 8:

Transformation 8 *(η-expanding rules)* Let $\mathcal{R}$ be a set of rules. Define $\mathcal{R}^{\uparrow}$ to be the set consisting of the rules $(l \cdot x^1_{\sigma_1} \cdots x^n_{\sigma_n}){\uparrow}^{\eta}_{V} \Rightarrow (r \cdot x^1_{\sigma_1} \cdots x^n_{\sigma_n}){\uparrow}^{\eta}_{V}$, for every rule $l \Rightarrow r$ in $\mathcal{R}$, with $l : \sigma_1 \to \ldots \sigma_n \to \iota$, all $x^i_{\sigma_i}$s fresh variables, and $V = FVar(l) \cup \{x^1_{\sigma_1}, \ldots, x^n_{\sigma_n}\}$.

The proof of the following theorem is a straightforward adaptation of the proof of Theorem 6.

Theorem 7. *If the rewrite relation generated by $\mathcal{R}^{\uparrow}$ is terminating on η-long terms, then the relation generated by $\mathcal{R}$ is terminating on the set of terms.*

Note that Theorems 6 and 7 concern *rules*, not rule schemes. The η-expansion of a non-terminating set of rule schemes may be terminating, as demonstrated by the system with $R = \{f_{\alpha\to\alpha} \cdot g_\alpha \Rightarrow h_\alpha,\ h_{\mathtt{nat}\to\mathtt{nat}} \cdot 0_{\mathtt{nat}} \Rightarrow f_{(\mathtt{nat}\to\mathtt{nat})\to\mathtt{nat}\to\mathtt{nat}} \cdot g_{\mathtt{nat}\to\mathtt{nat}} \cdot 0_{\mathtt{nat}}\}$. Thus, η-expansion is mainly useful on monomorphic systems, or for termination methods which, given rule schemes R, prove termination of $\mathcal{R}^{\uparrow}_{R}$.

9 Conclusions

We have seen various techniques to transform AFSs, essentially making it possible to pose restrictions on terms and rule schemes without losing generality. Considering existing results, this has various applications:

Applicative terms As mentioned before, most applicative systems cannot be dealt with directly. Consider for example the system with symbols $\mathtt{split}_{\mathtt{nat}\to\mathtt{tuple}}$ and $\mathtt{pair}_{\alpha\to\varepsilon\to\mathtt{tuple}}$ which has the following rule:

$$\mathtt{split} \cdot (x_{\mathtt{nat}\to\mathtt{nat}} \cdot y_{\mathtt{nat}}) \Rightarrow \mathtt{pair} \cdot x_{\mathtt{nat}\to\mathtt{nat}} \cdot y_{\mathtt{nat}}$$

Even the computability path ordering [1], which is the latest definition in a strengthening line of path orderings, cannot deal with this rule. However, using Transformations 1–4 we introduce $@_{(\mathtt{nat}\to\mathtt{nat}\times\mathtt{nat})\longrightarrow\mathtt{nat}}$ and this system becomes:

$$\mathtt{split} \cdot @(x, y) \Rightarrow \mathtt{pair} \cdot x \cdot y \qquad @(x, y) \Rightarrow x \cdot y$$

This system has the same curried form as:

$$\mathtt{split}(@(x, y)) \Rightarrow \mathtt{pair}(x, y) \qquad @(x, y) \Rightarrow x \cdot y$$

Consequently, termination of one implies termination of the other by Theorem 5. But the latter is immediate with HORPO [4], using a precedence $@ >_{\mathcal{F}} \mathtt{pair}$.

CPO The latest recursive path ordering, CPO, is defined only for monomorphic systems where all symbols have a data type as output type. It cannot, for instance, deal with a system with rules:

$$\begin{aligned} \mathtt{emap}(F, \mathtt{nil}) &\Rightarrow \mathtt{nil} \\ \mathtt{emap}(F, \mathtt{cons}(x, y)) &\Rightarrow \mathtt{cons}(F \cdot x, \mathtt{emap}(\mathtt{twice}(F), y)) \\ \mathtt{twice}(F) \cdot x &\Rightarrow F \cdot (F \cdot x) \end{aligned}$$

Here, $\mathtt{twice}$ has type declaration $(\mathtt{nat} \to \mathtt{nat}) \longrightarrow \mathtt{nat} \to \mathtt{nat}$. By Theorem 6 we can η-expand these rules, the result of which has the same curried form as:

$$\begin{aligned} \mathtt{emap}(F, \mathtt{nil}) &\Rightarrow \mathtt{nil} \\ \mathtt{emap}(F, \mathtt{cons}(x, y)) &\Rightarrow \mathtt{cons}(F \cdot x, \mathtt{emap}(\lambda z.\mathtt{twice}(F, z), y)) \\ \mathtt{twice}(F, x) &\Rightarrow F \cdot (F \cdot x) \end{aligned}$$

Thus, if this system can be proved terminating with CPO (which it can, if a reverse lexicographical ordering is used), the original system is terminating. CPO can be applied on any monomorphic system in this way, although the transformation may lose termination due to the η-expansion.

Dependency Pairs Since rules can be assumed to have a form $f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$, the dependency pair method for AFSs in [8] is now applicable without restrictions other than monomorphism; a termination tool which has Transformations 1–6 built into the input module could build around a dependency pair framework without losing out.

Summary and Future Work In this paper we discussed transformations which simplify Algebraic Functional Systems significantly. We saw that polymorphism only has a function in defining rule schemes, that rule schemes can be assumed to be β-normal and that there is no need for leading free variables in the left-hand side of rules. We know that applicative and functional notation can be interchanged, and rule schemes can be assumed to have a form $f \cdot l_1 \cdots l_n \Rightarrow r$ with f a function symbol. Moreover, when we are interested only in proving termination, we may η-expand the rules and restrict attention to η-long terms.

A monomorphic version of the transformations given here was implemented in WANDA v1.0 [7], which participated in the Termination Competition 2010 [13].

In the future, we intend to look further into other formalisms, and give conditions and techniques to transfer results across.

Acknowledgement. We are indepted to the anonymous referees for their remarks which helped to improve the paper.

References

1. 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.
2. Nao Hirokawa, Aart Middeldorp, and Harald Zankl. Uncurrying for termination. In *LPAR 2008*, volume 5330 of *LNAI*, pages 667–681, Doha, 2008. Springer-Verlag.
3. J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *LICS 1991*, pages 350–361, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.
4. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *LICS 1999*, pages 402–411, Trento, Italy, July 1999.
5. S. Kahrs. Confluence of curried term-rewriting systems. *Journal of Symbolic Computation*, 19:601–623, 1995.

6. R. Kennaway, J.W. Klop, M.R. Sleep, and F.J de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21(1):15–39, 1996.
7. C. Kop. Wanda. http://www.few.vu.nl/ kop/code.html.
8. C. Kop and F. van Raamsdonk. Higher order dependency pairs for algebraic functional systems. In *Proceedings of RTA 2011*, June 2011. To Appear.
9. 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.
10. T. Nipkow. Higher-order critical pairs. In *LICS 1991*, pages 342–349, Amsterdam, The Netherlands, July 1991.
11. J.C. van de Pol. *Termination of Higher-order Rerwite Systems*. PhD thesis, University of Utrecht, 1996.
12. 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.
13. Wiki. Termination portal. http://www.termination-portal.org/.

In this appendix I will present complete proofs of lemmas, theorems and loose statements in the text.

A definition that we will regularly use is the following: $s \Rightarrow_{R,top} t$ if there are $l \Rightarrow r \in R$, type substitution θ and substitution γ such that $s = l\theta\gamma$ and $t = r\theta\gamma$.

If $s \Rightarrow_{R_1,top} t$ implies $s \Rightarrow^+_{R_2} t$, then $s \Rightarrow_{R_1} t$ implies $s \Rightarrow^+_{R_2} t$, by definition of the rewrite relation.

Preliminaries

Lemma 4 (welldefinedness of type substitution). *If $s : \sigma$ then $s\theta : \sigma\theta$, so $s\theta$ is a term.*

Proof. A trivial induction on the form of s.

Lemma 5 (interchanging substitutions and type substitutions). *Given a type substitution θ and substitution γ, let γ^θ be the substitution mapping $x_{\sigma\theta}$ to $\gamma(x_\sigma)\theta$ for $x_\sigma \in \mathrm{dom}(\gamma)$. Then $(s\gamma)\theta = s\theta\gamma^\theta$.*

Proof. Note first that the substitution γ^θ is welldefined: if $\gamma(x_\sigma) : \sigma$ then $\gamma^\theta(x_{\sigma\theta}) = \gamma(x_\sigma)\theta : \sigma\theta$ by Lemma 4. To prove the lemma, we might perform induction on s, which is utterly trivial. The only case where anything happens is the case where s is a variable:
If $s = x_\sigma$ with $x \in \mathrm{dom}(\gamma)$ then $s\gamma\theta = \gamma(x_\sigma)\theta = \gamma^\theta(x_{\sigma\theta}) = s\theta\gamma^\theta$.
If $s = x_\sigma$ with $x \in \mathcal{V} \setminus \mathrm{dom}(\gamma)$ then $s\gamma\theta = x_\sigma\theta = x_{\sigma\theta} = x_{\sigma\theta}\gamma^\theta = s\theta\gamma^\theta$.

The other cass all proceed with the induction hypothesis. I copied them out below for completeness.

If $s = \lambda x_\sigma.t$ we can assume x does not occur in $\mathrm{dom}(\gamma)$; $(\lambda x_\sigma.t)\gamma\theta = (\lambda x_\sigma.t\gamma)\theta = \lambda x_{\sigma\theta}.t\gamma\theta =$ (IH) $\lambda x_{\sigma\theta}.t\theta\gamma^\theta = (\lambda x_{\sigma\theta}.t\theta)\gamma^\theta = (\lambda x_\sigma.t)\theta\gamma^\theta$.
If $s = t \cdot u$ then $s\gamma\theta = (t\gamma\theta) \cdot (u\gamma\theta) =$ (IH) $(t\theta\gamma^\theta) \cdot (u\theta\gamma^\theta) = s\theta\gamma^\theta$.
Finally, if $s = f_{(\sigma_1 \times \ldots \times \sigma_m) \to \tau}(s_1, \ldots, s_n)$, then $s\gamma\theta = f_{(\sigma_1 \times \ldots \times \sigma_m) \to \tau}(s_1\gamma, \ldots, s_m\gamma)\theta = f_{(\sigma_1\theta \times \ldots \times \sigma_m\theta) \to \tau\theta}(s_1\gamma\theta, \ldots, s_m\gamma\theta) =$ (IH) $f_{(\sigma_1\theta \times \ldots \times \sigma_m\theta) \to \tau\theta}(s_1\theta\gamma^\theta, \ldots, s_m\theta\gamma^\theta) = s\theta\gamma^\theta$.

*Proof (**Proof of Lemma 1**).* Let us say a type substitution θ unifies σ, τ if $\sigma\theta = \tau\theta$. This is the usual definition of unifying. Noting that our types are essentially first-order terms (if we consider the $\to$ as a binary function symbol), the following is a well-known result:
If there is a type substitution θ which unifies σ, τ, then there is a type substitution χ which unifies σ, τ such that for any type substitution θ' which unifies σ and τ there is some type substitution d such that $\theta'_{\restriction FTVar(\sigma) \cup FTVar(\tau)} = d \circ \chi$.

Now, given two types σ and τ, let b be a type substitution which renames all type variables in τ to fresh type variables; obviously b is invertible. If θ_1, θ_2 unify σ and τ, then $\theta_1 \cup (\theta_2 \circ b^{-1})$ is a single type substitution which unifies σ and τb. But then there is a minimal type substitution χ. Choosing $\chi_1 := \chi_{\restriction FTVar(\sigma)}$ and $\chi_2 := \chi_{\restriction Ran(b)} \circ b$ we have the required minimal unifiers.

Section 4: Polymorphism

*Proof (**Proof of Theorem 1**).* Let $s_0 \Rightarrow_{\mathcal{R}} s_1 \Rightarrow_{\mathcal{R}} \ldots$ be an infinite sequence, and $\alpha_1, \ldots, \alpha_n$ be the typevariables occuring in s_0. Let b be a type constructor of arity 0 (note that we have required such a type constructor exists) and let $\theta(\alpha_i) = b$ for all i. Then for all i the term $s_i\theta$ is monomorphic, since s_i can contain at most those type variables occurring in s_{i-1} (due to the requirement that $FTVar(r) \subseteq FTVar(l)$ for $l \Rightarrow r \in \mathcal{R}$, and because $\Rightarrow_\beta$ also does not create type variables). Also, a trivial induction on the size of the reduced term shows that $\Rightarrow_{\mathcal{R}}$ is preserved under type substitution, so $s_0\theta \Rightarrow_{\mathcal{R}} s_1\theta \Rightarrow_{\mathcal{R}} \ldots$ is an infinite reduction on monomorphic terms.

*Proof (**Proof of Theorem 2**).* Note that the definition of rule scheme corresponds with our original definition of *rule*, and that if $l \Rightarrow r$ is a rule scheme then also $l\theta \Rightarrow r\theta$ is a rule scheme for any type substitution θ.

Now, R contains rules $l\theta \Rightarrow r\theta$ if $l \Rightarrow r \in R$ and θ is a type substitution mapping all $\alpha \in FTVar(l)$ to monomorphic types. Since $FTVar(r) \subseteq FTVar(l)$ it is evident that neither $l\theta$ nor $r\theta$ contain any type variables. Thus, R is indeed a set of monomorphic rules.

By Theorem 1, we only have to consider monomorphic terms for termination. So we must see: for monomorphic s, t: $s \Rightarrow_R t$ if and only if $s \Rightarrow_{\mathcal{R}_R} t$. Since $\Rightarrow_\beta$ is by definition included in both relations, it suffices to show that $s \Rightarrow_{R,top} t$ if and only if $s \Rightarrow_{\mathcal{R}_R,top} t$ (if this is the case, evidently the monotonic closures of both relations are also equal). Thus, suppose $s = l\theta\gamma$ and $t = r\theta\gamma$ with $l \Rightarrow r \in R$. Since s is monomorphic, it must be true that all $\mathrm{dom}(\theta)$ contains all type variables in l and maps to monomorphic types. Since additional variables in the domain have no effect, we can safely assume $\mathrm{dom}(\theta) = FTVar(l)$. But then $l' := l\theta \Rightarrow r\theta =: r'$ is a rule of $\mathcal{R}_R$, so $s = l'\gamma$ and $t = r'\gamma$. On the other hand, if $s = l\theta\gamma$ and $t = r\theta\gamma$ with $l \Rightarrow r \in \mathcal{R}_R$, note that l and r are monomorphic and thus θ has no effect on them; additionally, we can write $l = l'\chi$ and $r = r'\chi$ for type substitution χ and $l' \Rightarrow r' \in R$. Thus, $s = l'\chi\gamma$ and $t = r'\chi\gamma$ as required.

The proof above also demonstrates the claim made below Theorem 2, that type substitution is not needed to define the rewrite relation $\Rightarrow_{\mathcal{R}}$.

Section 5: Head Variables

In this appendix I will give more formal and algorithmic definitions of the transformations provided in the paper. This makes it easier (at least notationally) to perform induction on them. However, the result of the transformations is exactly the same.

Note that many transformations repeatedly add rule schemes. We consider rule schemes modulo renaming of free variables, so a rule scheme $f(x) \Rightarrow x$ is the same as $f(y) \Rightarrow y$. This equality is essential for finiteness of the transformations, and to avoid adding masses of unnecessary rule schemes.

.1 Output Arity

Formal Version of Transformation 1
Let $R_0 := R$ and, for each $n \in \mathbb{N}$: $R_{n+1} = R_n \cup \{l\theta \cdot x \Rightarrow r\theta \cdot x | l \Rightarrow r \in R_n,\ l$ limited functional, $x \in \mathcal{V}$ fresh and θ, χ most general unifiers of $type(l)$ and $\alpha \to \varepsilon\}$. Let $R^{res} = \bigcup_{n \in \mathbb{N}} R_n$.

Lemma 6 (Stated in the Transformation). *There is always N such that all $oa(f) \leq N$.*

Proof. Consider the set $\{oa(f) | f \in \mathcal{F}\}$. Since there are only finitely many symbols with $oa(f) > 0$, this set is finite, and therefore has a maximum N.

Lemma 7 (Stated in the Transformation). *If $\{oa(f) | f \in \mathcal{F}\}$ is bounded by N then $R^{res} = R_N$, is finite if $\mathcal{R}$ is.*

Proof. In step k, only rule schemes of the form $l \cdot x_1 \cdots x_k \Rightarrow r \cdot x_1 \cdots x_k$ with $l \Rightarrow r \in R$ are newly added, if we ignore typing; we see this with induction on k: when defining R_1 (step 1) this is evident, and in step $k+1$, if $l' \cdot y \Rightarrow r' \cdot y$ is added with $l' \Rightarrow r' \in R_k$, we can safely assume that $l' \Rightarrow r'$ was added in step k (because if $l' \Rightarrow r' \in R_{k-1}$ then already $l' \cdot y \Rightarrow r' \cdot y \in R_k$). But then by the induction hypothesis $l' \Rightarrow r'$ has the form $l \cdot x_1 \cdots x_k \Rightarrow r \cdot x_1 \cdots x_k$, so $l' \cdot y \Rightarrow r' \cdot y$ has the required form $l \cdot x_1 \cdots x_{k+1} \Rightarrow r \cdot x_1 \cdots x_{k+1}$.

Consequently, in step $N+1$ only rule schemes of the form $l \cdot x_1 \cdots x_{N+1} \Rightarrow r \cdot x_1 \cdots x_{N+1}$ can be added. But then $l \cdot x_1 \cdots x_N$ must be limited functional. Which is impossible, since any limited functional term has the form $f(\boldsymbol{s}) \cdot t_1 \cdots t_m$ with $m < oa(f) \leq N$. We see: in step $N+1$ and beyond no new rule schemes are added. Moreover, in every step at most as many new rule schemes are added as were already there; if R_0 is finite, then any R_k is, including R_N.

Lemma 8 (Stated in the Transformation). *R^{res} and R define the same relation.*

Proof. Since $R = R_0 \subseteq R_1 \subseteq \ldots \subseteq R_N = R^{res}$, it is obvious that $\Rightarrow_R$ is included in $\Rightarrow_{R^{res}}$. For the other direction, note that any rule scheme in R^{res} has a form $l\theta \cdot \boldsymbol{x}$ with $l \Rightarrow r \in R$: this holds with θ the substitution that assigns all type variables in $FTVar(l)$ to themselves and $|\boldsymbol{x}| = 0$ if the rule scheme was present in R_0, and if it was added in step $k+1$ then by induction it has a form $(l\theta \cdot x^1_{\sigma_1} \cdots x^k_{\sigma_k})\chi \cdot x^{k+1}_{\sigma_{k+1}} \Rightarrow (r\theta \cdot x^1_{\sigma_1} \cdots x^k_{\sigma_k})\chi \cdot x^{k+1}_{\sigma_{k+1}}$. This is the same as $l\theta\chi \cdot x^1_{\sigma_1\chi} \cdots x^k_{\sigma_k\chi} \cdot x^{k+1}_{\sigma_{k+1}} \Rightarrow r\theta\chi \cdot x^1_{\sigma_1\chi} \cdots x^k_{\sigma_k\chi} \cdot x^{k+1}_{\sigma_{k+1}}$. Since $l\theta\chi$ and $r\theta\chi$ are exactly the same as $l(\chi \circ \theta)$ and $r(\chi \circ \theta)$ (as long as $\mathrm{dom}(\theta) = FTVar(l)$, which we guaranteed in the base case and satisfy in every induction step), the statement holds.

Thus, if $s \Rightarrow_{R^{res}, top} t$ then there exist type substitutions θ, χ, variables $\boldsymbol{x}$ and substitution γ such that $s = (l\theta \cdot \boldsymbol{x})\chi\gamma = (l\chi \circ \theta \cdot \boldsymbol{y})\gamma$ and $t = (r\theta \cdot \boldsymbol{x})\chi\gamma = (r\chi \circ \theta \cdot \boldsymbol{y})\gamma$. Since $l \Rightarrow r \in R$ and by monotonicity of the rewrite relation, $s \Rightarrow_{\mathcal{R}} t$ follows.

R^{res} has a nice property that we will need later, in the proof of Theorem 3.

Lemma 9. *If $s \Rightarrow_{R^{res},head} t$ and s is limited functional, then $s \cdot u \Rightarrow_{R^{res},top} t \cdot u$.*

Here, $s \Rightarrow_{R^{res},head} t$ denotes that s has the form $l\theta\gamma \cdot \boldsymbol{v}$ and t has the form $r\theta\gamma \cdot \boldsymbol{v}$, for some rule $l \Rightarrow r \in R^{res}$, type substitution θ and substitution γ. A topmost step is also a headmost step.

Proof. If $s \Rightarrow_{R^{res},head} t$ and s is limited functional, there are $l \Rightarrow r \in R^{res}$ with $l = f(\boldsymbol{v}) \cdot w_1 \cdots w_k$, type substitution θ, substitution γ and terms $w_{k+1} \cdots w_m$ such that $s = l\theta\gamma \cdot w_{k+1} \cdots w_m$ and $t = r\theta\gamma \cdot w_{k+1} \cdots w_m$. Choose $l_1, r_1, \theta_1, k_1, \gamma_1 := l, r, \theta, k, \gamma$. Now suppose we have l_i, r_i, p_i, k_i such that l_i is limited functional, $l_i \Rightarrow r_i \in R^{res}$ and $l_i\theta_i\gamma_i \cdot w_{k_i+1} \cdots w_m = s,\ r_i\theta_i\gamma_i \cdot w_{k_i+1} \cdots w_m = t$. Note that the output type of $l_i\theta_i$ is composed (since $s \cdot u$ is well-defined): $l_i\theta_i : \sigma \to \tau$. Hence there are most general unifiers a, b which unify the output type of l_i with $\alpha \to \varepsilon$, and d such that $l_i a d = l_i\theta_i$ and $d(b(\alpha)) = \sigma, d(b(\varepsilon)) = \tau$. Then for some fresh variable x there is a rule scheme $l' := l_i a \cdot x_{b(\alpha)} \Rightarrow r_i a \cdot x_{b(\alpha)} =: r' \in R^{res}$. If $k_i < m$, choose $l_{i+1}, r_{i+1}, \theta_{i+1}, \gamma_{i+1}, k_{i+1} := l', r', d, \gamma_i \cup [x_{d(b(\alpha))} := w_{k_i+1}], k_i+1$. Each of the inductive requirements is satisfied. If $k_i = m$, let $\delta := \gamma_i \cup [x_{d(b(\alpha))} := u]$. Then $s \cdot u = l'd\delta \Rightarrow_{R^{res},top} r'd\delta = t \cdot u$ as required.

k_i increases in every step and is bounded by m, so after finitely many steps we are done.

.2 Filling in Head Variables

Formal Version of Transformation 2
Let $R_0 = R^{res}$ and, for each $n \in \mathbb{N}$: $R_{n+1} = R_n \cup \{l\theta\delta \Rightarrow r\theta\delta | l \Rightarrow r \in R_n,\ x_\sigma \in HV(l),\ f_\tau \in \mathcal{F},\ 0 \le k < oa(f) \mid \psi(x_\sigma, f_\tau, k, \theta, \delta)\}$. Here, $\psi(x_\sigma, f_\tau, k, \theta, \delta)$ indicates that:

- $\tau = (\ldots) \longrightarrow \rho$
- $\alpha_1 \to \ldots \to \alpha_k \to \sigma$ and ρ can be unified, with most general unifiers θ, χ
- $\delta = [x_{\sigma\theta} := f_{\tau\chi}(\boldsymbol{y}) \cdot \boldsymbol{z}]$ for fresh variables $\boldsymbol{y}, z_1, \ldots, z_k$.

Define $R^{fill} = \bigcup_{n \in \mathbb{N}} R_n$.

Lemma 10 (Stated in the Transformation). *If R^{res} is finite, then there is N such that $R^{fill} = R_N$ and moreover R^{fill} is finite.*

Proof. If R^{res} is finite, the set $\{\text{size}(HV(l)) | l \Rightarrow r \in R^{res}\}$ is bounded by N. With induction on k we see: the rule schemes which occur in R_k but not in R_{k-1} have at most $N - k$ head variables in the left-hand side. For $k = 0$ this is correct (defining $R_{-1} = \emptyset$) by the choice of N. For larger k, the elements of $R_k \setminus R_{k-1}$ are those rule schemes created in step k, which have the form $l\theta\gamma \Rightarrow r\theta\gamma$ with $l \Rightarrow r \in R_{k-1}$. If $l \Rightarrow r \in R_{k-2}$ this rule scheme would have been added in R_{k-1};

therefore it must have been added in step $k-1$ and thus l has at most $N-k+1$ head variables. Now, type substitution doesn't affect number of head variables, and the substitution δ removes one. Thus, $l\theta\gamma$ has at most $(N-k+1)-1 = N-k$ head variables.

We see that nothing new can be added after step N (as a term cannot have a negative number of head variables),so $R^{fill} = R_N$. With induction on k, each R_k is finite: R_0 by assumption, and then in every step there are only finitely many $l \Rightarrow r \in R_k$, finitely many $x_\sigma \in HV(l)$ and finitely many f such that $0 < oa(f)$. Thus, $R^{fill} = R_N$ is finite.

Lemma 11 (Stated in the text). *$s \Rightarrow_{R^{res}} t$ iff $s \Rightarrow_{R^{fill}} t$.*

Proof. Since $R^{res} = R_0 \subseteq \bigcup_{n\in\mathbb{N}} R_n$ one direction is evident. For the other, note that any $\Rightarrow_{R^{fill}}$ step is a $\Rightarrow_{R_m}$ step for some m (even if R^{fill} is infinite). It suffices if $s \Rightarrow_{R_{n+1},top} t$ implies $s \Rightarrow_{R_n,top} t$ for all n.

So, suppose $s = l\theta\gamma$ and $t = r\theta\gamma$ with $l \Rightarrow r \in R_{n+1}$, some type substitution θ and substitution γ. Since $l \Rightarrow r \in R_{n+1}$ either $l \Rightarrow r \in R_n$, in which case we are immediately done, or there are χ, δ and $l' \Rightarrow r' \in R_n$ such that $l = l'\chi\delta$ and $r = r'\chi\delta$. By Lemma 5 $l\theta\gamma = l'\chi\delta\theta\gamma = l'\chi\theta\delta^\theta\gamma$. As we can safely assume $\mathrm{dom}(\chi) = FTVar(l)$ (extending χ with entries $[\alpha := \alpha]$ if necessary) this term is equal to $l'\theta \circ \chi\delta^\theta\gamma$. Similarly, $r\theta\gamma = r'\theta \circ \chi\delta^\theta\gamma$, and thus $s \Rightarrow_{R_n,top} t$. □

*Proof (**Proof of Lemma 2**).* If $s \Rightarrow_{R^{fill},top} t$ there are $l_0 \Rightarrow r_0 \in R^{fill}$, type substitution θ_0 and substitution γ_0 such that $s = l_0\theta_0\gamma_0$ and $t = r_0\theta_0\gamma_0$. We can safely assume that $\mathrm{dom}(\theta_0)$ contains all type variables in $FTVar(l_0)$ and $\mathrm{dom}(\gamma_0)$ contains all variables in $FVar(l_0\theta_0)$. Let A_0 be the set of those $x_\sigma \in HV(l_0)$ such that $\gamma_0(x_{\sigma\theta_0})$ is not limited functional.

Now, given $l_i \Rightarrow r_i \in R^{fill}$ such that $s = l_i\theta_i\gamma_i$ and $t = r_i\theta_i\gamma_i$ and a set A_i which contains those $x_\sigma \in HV(l_i)$ such that $\gamma_i(x_{\sigma\theta_i})$ is not limited functional. If $A_i = \emptyset$, we are done (choosing $l, r, \theta, \gamma = l_i, r_i, \theta_i, \gamma_i$). Otherwise, we define $l_{i+1}, r_{i+1}, \theta_{i+1}, \gamma_{i+1}, A_{i+1}$ with the same properties, but with A_{i+1} having a smaller size. Since A_0 is finite, this process will terminate and find the required values.

Choose any $x_\sigma \in A_i$ and consider $\gamma_i(x_{\sigma\theta_i}) = f_{\tau\chi}(\boldsymbol{u}) \cdot v_1 \cdots v_k$ with $f_\tau \in \mathcal{F}$ and $k < oa(f)$. Let $\tau = (e_1 \times \ldots \times e_l) \longrightarrow \tau'$. Evidently $\tau'\chi$ has the form $(\ldots) \longrightarrow \tau_1 \to \ldots \to \tau_k \to \rho$. The substitution $\theta' := \theta_0 \cup [\alpha_1 := \tau_1, \ldots, \alpha_k := \tau_k]$ and χ therefore unify $\alpha_1 \to \ldots \to \alpha_k \to \sigma$ and τ'. Let a, b the most general unifiers of these types, extend b with entries $\alpha := \alpha$ for type variables α which occur in some e_j but not in τ', and let d be a type substitution such that $\theta' = d \circ a$ and $\chi = d \circ b$ (which exists by definition of most general unifier). Let δ be the substitution $[x_{a(\sigma)} := f_{\tau b}(\boldsymbol{y}) \cdot z_1 \cdots z_k]$ for fresh variables $\boldsymbol{y}, \boldsymbol{z}$ of the right types. Then $l_i a\delta \Rightarrow r_i a\delta$ is a rule scheme of R^{fill} (if $l_i \Rightarrow r_i \in R_m$ then $la\delta \Rightarrow ra\delta \in R_{m+1}$). Let γ' be the substitution $[p := \gamma(p) | p \in \mathrm{dom}(\gamma) | p \neq x_{\sigma\theta_i}] \cup [y^1_{e_1\chi} := u_1, \ldots, y^l_{e_l\chi}, z^1_{\tau_1} := v_1, \ldots, z^k_{\tau_k} := v_k]$. Then $\delta^d\gamma'$ and γ_i are equal on $\mathrm{dom}(\gamma_i)$. Thus we see: $l_i\theta_i\gamma_i = l_i ad\gamma_i = l_i ad\delta^d\gamma' =$ (Lemma 5) $(l_i a\delta)d\gamma'$ and equally so, $r_i\theta_i\gamma_i = (r_i a\delta)d\gamma'$. We define $l_{i+1}, r_{i+1}, \theta_{i+1}, \gamma_{i+1}, A_{i+1} := l_i a\delta, r_i a\delta, d, \gamma', \{y_{\tau a} | y_\tau \in A_i\}$, which clearly meets the requirements.

.3 Preparing Polymorphic Types

Formal Version of Transformation 3
Let $R_0 := R^{fill}$ and for $n \in \mathbb{N}$, $R_{n+1} := R_n \cup \{l\theta \Rightarrow r\theta | l \Rightarrow r \in R_n | \phi(l,\theta) \text{ or } \phi(r,\theta)\}$. Here $\phi(s,\theta)$ if s has a subterm $t \cdot u$ such that (1) t is not limited functional, (2) $t : \sigma$ and σ unifies with some type τ in S, but not $\tau \geq \sigma$, (3) θ, χ are most general unifiers of σ and τ for some type substitution χ. Define $R^{normS} = \bigcup_{n \in \mathbb{N}} R_n$.

Lemma 12 (Stated in the Transformation). *If S and R^{fill} are both finite there is N such that $R^{normS} = R_N$.*

Proof. Suppose S and $R^{fill} = R_0$ are both finite. Define $R_{-1} := \emptyset$. Let $A_i = \{\langle l \Rightarrow r, B\rangle \mid l \Rightarrow r \in R_i,\ A \subseteq S \mid B \text{ is the set of all pairs } \langle \sigma, \tau\rangle \text{ with } \sigma \in S \text{ such that } l \text{ or } r \text{ has some subterm } t \cdot u \text{ such that } t : \tau \text{ not limited functional and } \tau \text{ unifies with } \sigma \text{ but not } \sigma \geq \tau\}$. Choose N the maximum size of any B occurring in A_0. Since A_0 is finite (because R_0 is) and any B is also finite (because a rule can only have finitely many subterms, and S is finite), N is well-defined. We will prove inductively: if $\langle l \Rightarrow r, B\rangle \in R_n \setminus R_{n-1}$ then $|B| \leq N - n$. If this is the case, evidently $A_N = A_{N+1} = \ldots$

The base case, R_0, is evident. Given $\langle l \Rightarrow r, B\rangle \in R_{n+1} \setminus R_n$, note that $l \Rightarrow r$ has the form $l'\theta \Rightarrow r'\theta$ with $l' \Rightarrow r' \in R_n$; let B' be the corresponding set such that $\langle l' \Rightarrow r',\ B'\rangle \in R_n$. Since $\langle l \Rightarrow r, B\rangle \notin R_n$ we know that $\langle l' \Rightarrow r',\ B'\rangle \notin R_{n-1}$, so by the induction hypothesis $|B'| \leq N - n$. Consider all subterms $t \cdot u$ of $l \Rightarrow r$ and types $\sigma \in S$ such that $t : \tau$ not limited functional and τ, σ unify but not $\sigma \geq \tau$. For all such subterms there is a subterm $t' \cdot u'$ of $l' \Rightarrow r'$ such that $t' : \tau'$ with $\tau'\theta = \tau$; evidently t' is also limited functional. If $\sigma \geq \tau'$ then also $\sigma \geq \tau$, so this cannot hold; $\langle \tau', \sigma\rangle \in B'$. Thus, every element in B corresponds with a unique element in B'. But there is at least one element $\langle \tau, \sigma\rangle$ in B' such that $\sigma \geq \tau\theta$. Thus, B has strictly less elements than B', that is to say, $|B| \leq |B'| - 1 \leq N - n - 1 = N - (n+1)$ as required.

Lemma 13 (Stated in the text). *R and R^{normS} generate the same rewrite relation.*

Proof. Obvious, since $\mathcal{R}_R = \mathcal{R}_{R^{normS}}$.

*Proof (**Proof of Lemma 3**).* Note that the statement "either both $\sigma, \sigma\theta \in S^c$ or both $\sigma, \sigma\theta \notin S^c$" is implied by the statement "for all $\tau \in S$ either $\tau \geq \sigma$ or not $\tau \geq \sigma\theta$" (because $\tau \geq \sigma$ implies $\tau \geq \sigma\theta$). It is the latter statement we will prove.

We are given $l_0 \Rightarrow r_0 \in R_0$ and type substitution θ_0. Suppose for all subterms $u \cdot v$ of l_n and r_n either u is limited functional, or $u : \sigma$ such that for all $\tau \in S$ either $\tau \geq \sigma$ or not $\tau \geq \sigma\theta_n$; then we are done choosing $\chi := \theta_n$, since all $R_n \subseteq R^{normS}$. Otherwise, choose any $\tau \in S$ which breaks the required assertion. Writing out the definition of $\geq$ there is ξ such that $\tau\xi = \sigma\theta_i$. Determine most

general unifiers a, b and type substitution d such that $\xi = d \circ a$ and $\theta_n = d \circ b$; R_{n+1} contains a rule $l_{n+1} \Rightarrow r_{n+1} := l_n b \Rightarrow r_n b$. The procedure continues with $l_{n+1} \Rightarrow r_{n+1}$ and $\theta_{n+1} := d$.

Note that in every step, b is irrevertible: if b was just a variable renaming, then we would have $\tau a b^{-1} = \sigma b b^{-1} = \sigma$, contradicting $\tau \not\geq \sigma\theta_n$. Therefore either l_{n+1} is larger than l_n or it has less type variables. Consequently, since the size of all l_k is bounded by the size of $l\theta$, this process must terminate (and thus find suitable χ) eventually.

.4 Explicit Application

Lemma 14 (Required for Theorem 3). *Let term s, type substitution θ and substitution γ be such that $\gamma(x)$ is not limited functional for any $x \in HV(s\theta)$ and for any subterm $t \cdot u$ of s and $\sigma \in S$ either t is limited functional, or both* $\mathrm{type}(t)$ *and* $\mathrm{type}(t)\theta \in S^c$ *or neither.*
Then $\mathtt{exp}(s\theta\gamma) = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$*, where* $\gamma^{\mathtt{exp}}$ *is the substitution* $[x := \mathtt{exp}(\gamma(x)) \mid x \in \mathrm{dom}(\gamma)]$.

Proof. Induction on the form of s. If $s = x_\sigma$ with $x \in \mathcal{V}$, then if $x_{\sigma\theta} \notin \mathrm{dom}(\gamma)$ both $\mathtt{exp}(s\theta\gamma)$ and $\mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$ are just $x_{\sigma\theta}$. However, if $x_{\sigma\theta} \in \mathrm{dom}(\gamma)$ then $\mathtt{exp}(s\theta\gamma) = \mathtt{exp}(\gamma(x_{\sigma\theta})) = \gamma^{\mathtt{exp}}(x_{\sigma\theta}) = s\theta\gamma^{\mathtt{exp}} = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$. If $s = f_\sigma(s_1, \ldots, s_n)$ then $\mathtt{exp}(s\theta\gamma) = f_{\sigma\theta}(\mathtt{exp}(s_1\theta\gamma), \ldots, \mathtt{exp}(s_n\theta\gamma)) =$ (by the induction hypothesis) $f_{\sigma\theta}(\mathtt{exp}(s_1)\theta\gamma^{\mathtt{exp}}, \ldots, s_n\theta\gamma^{\mathtt{exp}}) = f_\sigma(\mathtt{exp}(s_1), \ldots, \mathtt{exp}(s_n))\theta\gamma^{\mathtt{exp}} = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$. If $s = \lambda x.t$ then $\mathtt{exp}(s\theta\gamma) = \lambda x.\mathtt{exp}(t\theta\gamma) =$ (IH) $\lambda x.\mathtt{exp}(t)\theta\gamma^{\mathtt{exp}} = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$.

Finally, suppose $s = t \cdot u$ with $t : \sigma$. If t is limited functional, then so is $t\theta\gamma$ and therefore $\mathtt{exp}(s\theta\gamma) = \mathtt{exp}(t\theta\gamma) \cdot \mathtt{exp}(u\theta\gamma) =$ (IH) $\mathtt{exp}(t)\theta\gamma^{\mathtt{exp}} \cdot \mathtt{exp}(u)\theta\gamma^{\mathtt{exp}} = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$. If t is not limited functional, but $\sigma \notin S^c$, then by the assumption in the lemma also $\sigma\theta \notin S^c$. Therefore $\mathtt{exp}(s\theta\gamma) = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$ as before. Alternatively, if t is not limited functional and $\sigma \in S^c$ also $\sigma\theta \in S^c$. Since t is not limited functional neither is $t\theta\gamma$: this could only hold if $t = x_\rho \cdot v_1 \cdots v_m$ with $x \in \mathcal{V}$ and $\gamma(x_{\rho\theta})$ limited functional, but in this case $x_{\rho\theta} \in HV(s\theta)$, which contradicts our assumption. So in this situation $\mathtt{exp}(s\theta\gamma) = \mathtt{exp}(t\theta\gamma \cdot u\theta\gamma) = @_{\sigma\theta}(\mathtt{exp}(t\theta\gamma), \mathtt{exp}(u\theta\gamma)) =$ (induction hypothesis) $@_{\sigma\theta}(\mathtt{exp}(t)\theta\gamma^{\mathtt{exp}}, \mathtt{exp}(u)\theta\gamma^{\mathtt{exp}}) = @_\sigma(\mathtt{exp}(t), \mathtt{exp}(u))\theta\gamma^{\mathtt{exp}} = \mathtt{exp}(s)\theta\gamma^{\mathtt{exp}}$.

Lemma 15 (Required for Theorem 3). *Let term s, type substitution θ and substitution γ be such that for any subterm $t \cdot u$ of s and $\sigma \in S$ either both* $\mathrm{type}(t)$ *and* $\mathrm{type}(t)\theta \in S^c$ *or neither.*
Then $\mathtt{exp}(s)\theta\gamma^{\mathtt{exp}} \Rightarrow^*_{R^{\mathrm{noapp}}} \mathtt{exp}(s\theta\gamma)$.

Proof. The proof is much the same as the proof of Lemma 14, using $\Rightarrow^*_{R^{\mathrm{noapp}}}$ instead of $=$ (note that $\Rightarrow^*_{R^{\mathrm{noapp}}}$ is reflexive). The only different case is when $s = t \cdot u$ with $t : \tau$, $\tau \in S^c, \tau\theta \in S^c$, $t = x_\rho \cdot v_1 \cdots v_n$ and $\gamma(x)$ is limited functional. It might still be that $t\theta\gamma$ is not limited functional, in which case we proceed as before, but if $t\theta\gamma$ is limited functional while t is not, we have $\mathtt{exp}(s)\theta\gamma^{\mathtt{exp}} = @_\tau(\mathtt{exp}(t), \mathtt{exp}(u))\theta\gamma^{\mathtt{exp}} = @_{\tau\theta}(\mathtt{exp}(t)\theta\gamma^{\mathtt{exp}}, \mathtt{exp}(u)\theta\gamma^{\mathtt{exp}})$, which

by the induction hypothesis $\Rightarrow^*_{R^{\text{noapp}}} @_{\tau\theta}(\mathtt{exp}(t\theta\gamma), \mathtt{exp}(u\theta\gamma)) \Rightarrow_{R^{\text{noapp}}} \mathtt{exp}(t\theta\gamma) \cdot \mathtt{exp}(u\theta\gamma)$ (using the new rule $@_\rho(x, y) \Rightarrow x \cdot y$), which equals $\mathtt{exp}(s\theta\gamma)$.

Lemma 16 (Part of Theorem 3). *If $s \Rightarrow_{R^{res}} t$, then $\mathtt{exp}(s) \Rightarrow^+_{R^{\text{noapp}}} \mathtt{exp}(t)$.*

Proof. By induction on the size of s.

If $s \Rightarrow_{R^{res}} t$ by a topmost step rule step, the combination of Lemmas 2 and 3 provides us with $l \Rightarrow r \in R^{normS}$, type substitution θ and substitution γ such that $s = l\theta\gamma,\ t = r\theta\gamma,\ \gamma(x)$ is not limited functional for any $x \in HV(l\theta)$ and for any subterm $u \cdot v$ occurring in either l or r: either both type(u) and type$(u)\theta \in S^c$ or neither. Using Lemmas 14 and 15 then $\mathtt{exp}(s) = \mathtt{exp}(l)\theta\gamma^{\mathtt{exp}} \Rightarrow_{R^{\text{noapp}}} \mathtt{exp}(r)\theta\gamma^{\mathtt{exp}} \Rightarrow^*_{R^{\text{noapp}}} \mathtt{exp}(r\theta\gamma) = \mathtt{exp}(t)$.

If $s \Rightarrow_{R^{res}} t$ by a topmost β step, then $s = (\lambda x.u) \cdot v$ and $t = u[x := v]$. Let $\lambda x.u : \sigma$. if $\sigma \notin S^c$ then $\mathtt{exp}(s) = (\lambda x.\mathtt{exp}(u)) \cdot \mathtt{exp}(v) \Rightarrow_\beta \mathtt{exp}(u)[x := \mathtt{exp}(v)]$ which, by Lemma 15 $\Rightarrow^*_{R^{\text{noapp}}} \mathtt{exp}(u[x := v]) = \mathtt{exp}(t)$. If, however, $\sigma \in S^c$ then $\mathtt{exp}(s) = @_\sigma(\lambda x.\mathtt{exp}(u), \mathtt{exp}(v)) \Rightarrow_{R^{\text{noapp}}} (\lambda x.\mathtt{exp}(u)) \cdot \mathtt{exp}(v) \Rightarrow_\beta \mathtt{exp}(u)[x := \mathtt{exp}(v)] \Rightarrow^*_{R^{\text{noapp}}} \mathtt{exp}(t)$.

If $s = \lambda x.u_0$ or $s = f(u_1, \ldots, u_n)$ and one of the u_i is reduced, we use the induction hypothesis. If $s = u \cdot v$ and either type$(u) \notin S$ or the reduction takes place in v, we just use the induction hypothesis as well (whether u is limited functional or not). Alternatively, let $s = u \cdot v \Rightarrow_{R^{res}} u' \cdot v$ with type$(u) = \sigma \in S^c$. If both u and u' are limited functional, or they are both not, the induction hypothesis immediately gives what we need. If u is not limited functional but u' is, we have $\mathtt{exp}(s) = @_\sigma(\mathtt{exp}(u), \mathtt{exp}(v)) \Rightarrow^+_{R^{\text{noapp}}} @_\sigma(\mathtt{exp}(u'), \mathtt{exp}(v))$ by the induction hypothesis, $\Rightarrow_{R^{\text{noapp}}} \mathtt{exp}(u') \cdot \mathtt{exp}(v) = \mathtt{exp}(t)$. Finally, if u is limited functional but u' is not, u must have been reduced with a headmost reduction. Lemma 9 states that also $s \Rightarrow_{R^{res},top} t$, which brings us back to the base case, which has already been demonstrated.

Lemma 17 (Part of Theorem 3). *Let $\mathtt{exp}^{-1}(s)$ be s with all subterms of the form $@_\sigma(u, v)$ replaced by $u \cdot v$. Then $s \Rightarrow_{R^{\text{noapp}}} t$ implies $\mathtt{exp}^{-1}(s) \Rightarrow^=_{\mathcal{R}} \mathtt{exp}^{-1}(t)$, with equality only possible if s contains more $@_\sigma$ symbols than t.*

Proof. Let $\gamma^{\mathtt{exp}^{-1}}$ be the substitution assigning $\mathtt{exp}^{-1}(\gamma(x))$ for $x \in \text{dom}(\gamma)$. First we prove inductively: $\mathtt{exp}^{-1}(s)\gamma^{\mathtt{exp}^{-1}} = \mathtt{exp}^{-1}(s\gamma)$. This is evident in the base case where s is a variable and immediate with the induction hypothesis in each of the cases $s = f(s_1, \ldots, s_n),\ s = \lambda x.u$ and even $s = u \cdot v$. The most interesting case is when $s = @_\sigma(u, v)$, but even then $\mathtt{exp}^{-1}(s)\gamma^{\mathtt{exp}^{-1}} = (u \cdot v)\gamma^{\mathtt{exp}^{-1}} = u\gamma^{\mathtt{exp}^{-1}} \cdot v\gamma^{\mathtt{exp}^{-1}} = \mathtt{exp}^{-1}(s\gamma)$. It is similarly evident that $\mathtt{exp}^{-1}(s\theta) = \mathtt{exp}^{-1}(s)\theta$.

Having this, the lemma is quite easy, by induction on the size of s: each inductive case (where the reduction happens in a subterm) is trivial (for example, if $@_\sigma(u, v) \Rightarrow_{R^{\text{noapp}}} @_\sigma(u', v)$ then $\mathtt{exp}^{-1}(u) \cdot \mathtt{exp}^{-1}(v) \Rightarrow^=_{R^{\text{noapp}}} \mathtt{exp}^{-1}(u') \cdot \mathtt{exp}^{-1}(v)$ by induction), and there are three base cases: if $s = @_\sigma(u, v) \Rightarrow_{R^{\text{noapp}}} u \cdot v = t$ then $\mathtt{exp}^{-1}(s) = \mathtt{exp}^{-1}(t)$ but t has an $@_\sigma$ symbol less; if $s = (\lambda x.u) \cdot v \Rightarrow_\beta u[x := v] = t$ then $\mathtt{exp}^{-1}(s) = (\lambda x.\mathtt{exp}^{-1}(u)) \cdot \mathtt{exp}^{-1}(v) \Rightarrow_\beta \mathtt{exp}^{-1}(u)[x := \mathtt{exp}^{-1}(v)] = \mathtt{exp}^{-1}(u[x := v]) = \mathtt{exp}^{-1}(t)$ as we saw above;

if $s = l\theta\gamma$ and $t = r\theta\gamma$ with $l \Rightarrow r \in R^{\text{noapp}}$, we use the observations to note that $\texttt{exp}^{-1}(s) = \texttt{exp}^{-1}(l)\theta\gamma^{\texttt{exp}^{-1}}$ and $\texttt{exp}^{-1}(t) = \texttt{exp}^{-1}(r)\theta\gamma^{\texttt{exp}^{-1}}$. Since $\texttt{exp}^{-1}(l) \Rightarrow \texttt{exp}^{-1}(r)$ is a rule scheme in R^{normS} for all $l \Rightarrow r \in R^{\text{noapp}}$, we are done.

*Proof (**Proof of Theorem 3**).* If $\Rightarrow_{R^{normS}}$ is non-terminating, so there is an infinite reduction $s_0 \Rightarrow_{R^{normS}} s_1 \Rightarrow_{R^{normS}} \ldots$, then note that by Lemma 16 there is also an infinite reduction $\texttt{exp}(s_0) \Rightarrow^{+}_{R^{\text{noapp}}} \texttt{exp}(s_1) \Rightarrow^{+}_{R^{\text{noapp}}} \ldots$ On the other hand, if $\Rightarrow_{R^{\text{noapp}}}$ is non-terminating, so there is an infinite reduction $s_0 \Rightarrow_{R^{\text{noapp}}} s_1 \Rightarrow_{R^{\text{noapp}}} \ldots$ then by Lemma 17 there is also an infinite reduction $\texttt{exp}^{-1}(s_0) \Rightarrow^{=}_{R^{normS}} \texttt{exp}^{-1}(s_1) \Rightarrow^{=}_{R^{normS}} \ldots$, with infinitely many non-equality steps (since there cannot be an infinite tail $\texttt{exp}^{-1}(s_n) = \texttt{exp}^{-1}(s_{n+1}) = \ldots$, as s_n has only finite size).

Section 6: Abstractions in left-hand sides and β-redexes

We assume the left-hand sides of rule schemes in R do not have any subterms of the form $x \cdot s$ with x a (typed) variable.

Formal Version of Transformation 5
Let $R_0 := R$ and for $n \in \mathbb{N}$, $R_{n+1} := R_n \cup \{l\theta \Rightarrow r\theta | l \Rightarrow r \in R_n | \phi(l,\theta) \text{ or } \phi(r,\theta)\}$. Here $\phi(s,\theta)$ if s has a subterm $\lambda x.t : \sigma$ and σ unifies with some type $\tau \in S$, with minimal unifiers θ, χ. Define $R^{normQ} = \bigcup_{n\in\mathbb{N}} R_n$.

Lemma 18 (Stated in the Transformation). *If Q and R are both finite there is N such that $R^{normQ} = R_N$.*

Proof. This is essentially a copy of the proof of Lemma 12. The only difference is the kind of subterms we consider, but this is not essential for the proof style.

Lemma 19 (Stated in the text). *R and R^{normQ} generate the same rewrite relation.*

Proof. Obvious, since $\mathcal{R}_R = \mathcal{R}_{R^{normQ}}$.

Lemma 20 (Required for Theorem 4). *For $l \Rightarrow r \in R$ and type substitution θ, there are $l' \Rightarrow r' \in R^{normQ}$ and type substitution χ such that $l\theta = l'\chi$, $r\theta = r'\chi$ and for $u \cdot v$ occurring in l' or r' with $u : \sigma$ not limited functional, either both $\sigma, \sigma\chi \in Q^c$ or both $\sigma, \sigma\chi \notin Q^c$.*

Proof. This is essentially a copy of Lemma 20. The only difference is the kind of subterms we consider, but this is not essential for the proof style.

Lemma 21 (Required for Theorem 4). *Let s be a term and θ a type substitution such that for all abstractions $\lambda x.t : \sigma$ occurring anywhere in s either both $\sigma, \sigma\theta \in Q^c$ or both $\sigma, \sigma\theta \notin Q^c$. Let $\gamma^{\texttt{expL}}$ be the substitution $[x := \texttt{expL}(\gamma(x)) \mid x \in \text{dom}(\gamma)]$. Then $\texttt{expL}(s\theta\gamma) = \texttt{expL}(s)\theta\gamma^{\texttt{expL}}$.*

Proof. By induction on s. The cases $s = f(s_1, \ldots, s_n)$ and $s = t \cdot u$ are straightforward with the induction hypothesis. If $s = x_\sigma \in \mathrm{dom}(\gamma)$ then $\mathtt{expL}(s\theta\gamma) = \mathtt{exp}(\gamma(x_{\sigma\theta})) = \gamma^{\mathtt{exp}}(x_\sigma\theta) = \mathtt{expL}(s)\theta\gamma^{\mathtt{exp}}$. If $s = x_\sigma \notin \mathrm{dom}(\gamma)$ with $x \in \mathcal{V}$ then $\mathtt{expL}(s\theta\gamma) = \mathtt{expL}(x_{\sigma\theta}) = x_{\sigma\theta} = s\theta = s\theta\gamma^{\mathtt{expL}}$.

If $s = \lambda x_\tau.t : \sigma = \tau \to \rho$, suppose $\sigma \in Q^c$. Then by assumption also $\sigma\theta \in Q^c$, and therefore $\mathtt{expL}(s\theta\gamma) = \Lambda_{\sigma\theta}(\lambda x_{\tau\theta}.\mathtt{expL}(t\theta\gamma)) =$ (induction hypothesis) $\Lambda_{\sigma\theta}(\lambda x_{\tau\theta}.\mathtt{expL}(t)\theta\gamma^{\mathtt{expL}}) = \Lambda_\sigma(\lambda x_\tau.\mathtt{expL}(t))\theta\gamma^{\mathtt{expL}} = \mathtt{expL}(s)\theta\gamma^{\mathtt{expL}}$.

Now suppose $\sigma \notin Q^c$. Then by assumption also $\sigma\theta \notin Q^c$ and therefore $\mathtt{expL}(s\theta\gamma) = \lambda x_{\tau\theta}.\mathtt{expL}(t\theta\gamma) =$ (IH) $\lambda x_{\tau\theta}.\mathtt{expL}(t)\theta\gamma^{\mathtt{expL}} = (\lambda x_\tau.\mathtt{expL}(t))\theta\gamma^{\mathtt{expL}} = \mathtt{expL}(s)\theta\gamma^{\mathtt{expL}}$.

Lemma 22 (Part of Theorem 4). *If $s \Rightarrow_R t$ then also $\mathtt{expL}(s) \Rightarrow^+_{R^\Lambda} \mathtt{expL}(t)$.*

Proof. By induction on the size of s; all inductive cases (where the reduction is done in a subterm) are trivial, even both cases when s is an abstraction (note that $\Rightarrow_R$ does not change the type of a term). For the base case, assume $s \Rightarrow_R t$ by a reduction at the top of the term, either with a β-step or using a rule. In the first case, $s = (\lambda x.u) \cdot v$ and $t = u[x := v]$, let $\lambda x.u : \sigma$. We either have $\mathtt{expL}(s) = \Lambda_\sigma(\lambda x.\mathtt{expL}(u)) \cdot \mathtt{expL}(v) \Rightarrow_{R^\Lambda} (\lambda x.\mathtt{expL}(u)) \cdot \mathtt{expL}(v)$ (if $\sigma \in Q^c$) or $\mathtt{exp}(s) = (\lambda x.\mathtt{expL}(u)) \cdot \mathtt{expL}(v)$ (if not). Either way, $\mathtt{expL}(s) \Rightarrow^=_{R^\Lambda} (\lambda x.\mathtt{expL}(u)) \cdot \mathtt{expL}(v) \Rightarrow_\beta \mathtt{expL}(u)[x := \mathtt{expL}(v)] =$ (Lemma 21) $\mathtt{expL}(u[x := v]) = \mathtt{expL}(t)$. In the second case, $s = l\theta\gamma$ and $t = r\theta\gamma$ with $l \Rightarrow r \in R$. Using Lemma 20 we can find $l' \Rightarrow r' \in R^{normQ}$ and type substitution θ' such that also $s = l'\theta'\gamma$ and $t = r'\theta'\gamma$ and additionally for all subterms $\lambda x.u : \sigma$ of l' or r' we have either both $\sigma, \sigma\theta \in Q^c$ or both $\sigma, \sigma\theta \notin Q^c$. Now by Lemma 21 $\mathtt{expL}(s) = \mathtt{expL}(l'\theta'\gamma) = \mathtt{expL}(l')\theta'\gamma^{\mathtt{expL}} \Rightarrow_{R^\Lambda} \mathtt{expL}(r')\theta'\gamma^{\mathtt{expL}} = \mathtt{expL}(r'\theta'\gamma) = \mathtt{expL}(t)$.

Lemma 23. *Let $\mathtt{expL}^{-1}$ be the function that replaces all occurrences of $\Lambda(s)$ in some term by just s. Then $s \Rightarrow_{R^\Lambda} t$ implies $\mathtt{expL}^{-1}(s) \Rightarrow^=_{R^{normQ}} \mathtt{expL}^{-1}(t)$, with equality only possible if s has more Λ symbols than t.*

Proof. We first see that $\mathtt{expL}^{-1}(s\theta\gamma) = \mathtt{expL}^{-1}(s)\theta\gamma^{\mathtt{expL}^{-1}}$, where $\gamma^{\mathtt{expL}^{-1}}$ is define in the usual way. This is a straightforward induction on the size of s; all cases (s an application, functional term, variable in $\mathrm{dom}(\gamma)$, variable not in $\mathrm{dom}(\gamma)$, abstraction with type in Q^c or abstraction with type not in Q^c) are trivial. Similarly we see that always $\mathtt{expL}^{-1}(\mathtt{expL}(s)) = s$.

Now suppose $s \Rightarrow_{R^\Lambda} t$; we prove $\mathtt{expL}^{-1}(s) \Rightarrow_{R^{normQ}} \mathtt{expL}^{-1}(t)$ with induction on the size of s. The inductionstep (when the reduction was done in a subterm) is trivial in all cases (distinguishing when s is a functional term with root symbol in $\mathcal{F}$, functional term $\Lambda(s)$, application with the reduction on the left or right, or abstraction), so consider the three base steps. If $s = \mathtt{expL}(l)\theta\gamma$ and $t = \mathtt{expL}(r)\theta\gamma$ with $l \Rightarrow r \in R^{normQ}$, then as we have seen in the observations above, $\mathtt{expL}^{-1}(s) = l\theta\gamma^{\mathtt{expL}^{-1}} \Rightarrow_{R^{normQ}} \mathtt{expL}^{-1}(t) = l\theta\gamma^{\mathtt{expL}^{-1}}$. If $s = \Lambda_\sigma(t) \Rightarrow_{R^\Lambda} t$, then $\mathtt{expL}^{-1}(s) = \mathtt{expL}^{-1}(t)$, but s has a Λ symbol more than t. If $s = (\lambda x.u) \cdot v$ and $t = u[x := v]$ then $\mathtt{expL}^{-1}(s) = (\lambda x.\mathtt{expL}^{-1}(u)) \cdot \mathtt{expL}^{-1}(v) \Rightarrow_\beta \mathtt{expL}^{-1}(u)[x := \mathtt{expL}^{-1}(v)] = \mathtt{expL}^{-1}(u[x := v])$.

*Proof (**Proof of Theorem 4**).* If $\Rightarrow_R$ is non-terminating, there is an infinite reduction $s_0 \Rightarrow_R s_1 \Rightarrow_R \ldots$, and by Lemma 22 this implies an infinite reduction $\mathtt{expL}(s_0) \Rightarrow^+_{R^\lambda} \mathtt{expL}(s_1) \Rightarrow^+_{R^\lambda} \ldots$. If $\Rightarrow_{R^\lambda}$ is non-terminating, there is an infinite reduction $s_0 \Rightarrow_{R^\lambda} s_1 \Rightarrow_{R^\lambda} \ldots$, which by Lemma 22 implies an infinite reduction $\mathtt{expL}^{-1}(s_0) \Rightarrow^=_{R^{normQ}} \mathtt{expL}^{-1}(s_1) \Rightarrow^=_{R^{normQ}}$ with infinitely many strict steps. This proves this direction because by Lemma 19 $\Rightarrow_R$ and $\Rightarrow_{R^{normQ}}$ are the same relation.

Section 7: Currying

Lemma 24 (Required for Theorem 5). *For term s with symbols in $\mathcal{F}$, type substitution θ and substitution γ, let $\gamma^{\mathtt{flat}} = [x := \mathtt{flat}(\gamma(x)) \mid x \in \mathrm{dom}(\gamma)]$. Then $\mathtt{flat}(s\theta\gamma) = \mathtt{flat}(s)\theta\gamma^{\mathtt{flat}}$.*

Proof. By induction on the size of s; it is safe to assume $\mathrm{dom}(\gamma)$ contains all $x_\sigma \in FVar(s\theta)$. Thus, if $s = x_\sigma$ then $\mathtt{flat}(s\theta\gamma) = \mathtt{flat}(\gamma(x_{\sigma\theta})) = \gamma^{\mathtt{flat}}(x_{\sigma\theta}) = s\theta\gamma^{\mathtt{flat}} = \mathtt{flat}(s)\theta\gamma^{\mathtt{flat}}$. If $s = f_\sigma(s_1, \ldots, s_n)$ then write $\sigma = (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$ and $\sigma' = \sigma_1 \to \ldots \to \sigma_n \to \tau$. It should be noted that $(\sigma\theta)' = (\sigma')\theta$. We have: $\mathtt{flat}(s\theta\gamma) = \mathtt{flat}(f_{\sigma\theta}(s_1\theta\gamma, \ldots, s_1\theta\gamma)) = f'_{\sigma\theta} \cdot \mathtt{flat}(s_1\theta\gamma) \cdots \mathtt{flat}(s_1\theta\gamma) =$ (IH) $f_{\sigma'\theta} \cdot \mathtt{flat}(s_1)\theta\gamma^{\mathtt{flat}} \cdots \mathtt{flat}(s_n)\theta\gamma^{\mathtt{flat}} = (f_{\sigma'} \cdot \mathtt{flat}(s_1) \cdots \mathtt{flat}(s_n))\theta\gamma^{\mathtt{flat}} = \mathtt{flat}(s)\theta\gamma^{\mathtt{flat}}$. The other cases, where s is an application or abstraction, are both trivial with the induction hypothesis.

Lemma 25 (Part of Theorem 5). *If $s \Rightarrow_R t$ then $\mathtt{flat}(s) \Rightarrow_{R^{\mathtt{flat}}} \mathtt{flat}(t)$.*

Proof. By induction on the form of s; the inductive cases (when the reduction takes place in a subterm) are all easy, we demonstrate only the case $s = f_\sigma(s_1, \ldots, s_i, \ldots, s_n) \Rightarrow_R f_\sigma(s_1, \ldots, s'_i, \ldots, s_n)$ because $s_i \Rightarrow_R s_n$. In this case $\mathtt{flat}(s) = f_{\sigma'} \cdot \mathtt{flat}(s_1) \cdots \mathtt{flat}(s_n)$, which by the induction hypothesis $\Rightarrow_{R^{\mathtt{flat}}} f_{\sigma'} \cdot \mathtt{flat}(s_1) \cdots \mathtt{flat}(s'_i) \cdots \mathtt{flat}(s_n) = \mathtt{flat}(f_\sigma(s_1, \ldots, s'_i, \ldots, s_n))$. As for the two base cases, if $s = (\lambda x.u) \cdot v \Rightarrow_\beta u[x := v]$ then $\mathtt{flat}(s) = (\lambda x.\mathtt{flat}(u)) \cdot \mathtt{flat}(v) \Rightarrow_\beta \mathtt{flat}(u)[x := \mathtt{flat}(v)]$ which by Lemma 24 equals $\mathtt{flat}(u[x := v]) = \mathtt{flat}(t)$. If $s = l\theta\gamma \Rightarrow_R r\theta\gamma$ then by Lemma 24 $\mathtt{flat}(s) = \mathtt{flat}(l)\theta\gamma^{\mathtt{flat}} \Rightarrow_{R^{\mathtt{flat}}} \mathtt{flat}(r)\theta\gamma^{\mathtt{flat}} = \mathtt{flat}(t)$.

Definition 1 (Inverse of `flat`).

$$
\begin{aligned}
\mathtt{flat}^{-1}(x_\sigma \cdot s_1 \cdots s_n) &= x_\sigma \cdot \mathtt{flat}^{-1}(s_1) \cdots \mathtt{flat}^{-1}(s_n) && (x \in \mathcal{V}) \\
\mathtt{flat}^{-1}((\lambda x_\sigma.s_0) \cdot s_1 \cdots s_n) &= (\lambda x_\sigma.\mathtt{flat}^{-1}(s_0)) \cdot \mathtt{flat}^{-1}(s_1) \cdots \mathtt{flat}^{-1}(s_n) && (n \geq 0) \\
\mathtt{flat}^{-1}(f_{\sigma'} \cdot s_1 \cdots s_k) &= \lambda x_{k+1} \ldots x_n.f_\sigma(\mathtt{flat}^{-1}(s_1), \ldots, \mathtt{flat}^{-1}(s_k), x_{k+1}, \ldots, x_n) && (k < n = ar(f)) \\
\mathtt{flat}^{-1}(f_{\sigma'} \cdot s_1 \cdots s_k) &= f_\sigma(\mathtt{flat}^{-1}(s_1), \ldots, \mathtt{flat}^{-1}(s_n)) \cdot \mathtt{flat}^{-1}(s_{n+1}) \cdots \mathtt{flat}^{-1}(s_k) && (k \geq n = ar(f))
\end{aligned}
$$

Here, $ar(f)$ is the unique number n such that $f_{(\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau} \in \mathcal{F}$.

Lemma 26 (Required for Theorem 5). *If s is a term without leading free variables, then $\mathtt{flat}^{-1}(s\theta\gamma) = \mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}}$, where $\gamma^{\mathtt{flat}^{-1}}$ is the substitution $[x := \mathtt{flat}^{-1}(\gamma(x)) \mid x \in \mathrm{dom}(\gamma)]$.*

Proof. We prove the lemma for s, γ such that all $\mathrm{dom}(\gamma)$ does not contain any leading free variables in s. This is evidently satisfied if s has no leading free variables.

Consider first the case $s = x_\sigma \cdot s_1 \cdots s_n$ with x a variable. By assumption either $n = 0$ or $x_{\sigma\theta} \notin \mathrm{dom}(\gamma)$. We assume the first. Then $\mathtt{flat}^{-1}(s\theta\gamma) = \mathtt{flat}^{-1}(x_{\sigma\theta} \cdot s_1\theta\gamma \cdots s_1\theta\gamma) = x_{\sigma\theta} \cdot \mathtt{flat}^{-1}(s_1\theta\gamma) \cdots \mathtt{flat}^{-1}(s_n\theta\gamma)$, which by the induction hypothesis equals $x_{\sigma\theta} \cdot \mathtt{flat}^{-1}(s_1)\theta\gamma^{\mathtt{flat}^{-1}} \cdots \mathtt{flat}^{-1}(s_1)\theta\gamma^{\mathtt{flat}^{-1}} = (x_\sigma \cdot \mathtt{flat}^{-1}(s_1) \cdots \mathtt{flat}^{-1}(s_n))\theta\gamma^{\mathtt{flat}^{-1}} = \mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}}$. Alternatively, if $n = 0$ (so $s = x_\sigma$ and $x_{\sigma\theta} \in \mathrm{dom}(\gamma)$), then $\mathtt{flat}^{-1}(s\theta\gamma) = \mathtt{flat}^{-1}(\gamma(x_{\sigma\theta})) = \gamma^{\mathtt{flat}^{-1}}(x_{\sigma\theta}) = x_\sigma\theta\gamma^{\mathtt{flat}^{-1}} = \mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}}$.

The other cases are very simple with the induction hypothesis. For completeness, and because they are used again in Lemma 27, I will present them here regardless.

Now suppose $s = (\lambda x_\sigma.s_0)\cdot s_1 \cdots s_n$. Then $\mathtt{flat}^{-1}(s\theta\gamma) = \mathtt{flat}^{-1}((\lambda x_\sigma.s_0\theta\gamma)\cdot s_1\theta\gamma \cdots s_n\theta\gamma) = (\lambda x_{\sigma\theta}.\mathtt{flat}^{-1}(s_0\theta\gamma))\cdot\mathtt{flat}^{-1}(s_1\theta\gamma) \cdots \mathtt{flat}^{-1}(s_n\theta\gamma)$. Note that, although x may occur as a leading variable in s, it does not occur in the domain of γ. Therefore we can apply the induction hypothesis to s_0 as to all other s_i, and find that $= (\lambda x_{\sigma\theta}.\mathtt{flat}^{-1}(s_0)\theta\gamma^{\mathtt{flat}^{-1}})\cdot\mathtt{flat}^{-1}(s_1)\theta\gamma^{\mathtt{flat}^{-1}} \cdots \mathtt{flat}^{-1}(s_n)\theta\gamma^{\mathtt{flat}^{-1}}$ which equals $((\lambda x_\sigma.\mathtt{flat}^{-1}(s_0))\cdot\mathtt{flat}^{-1}(s_1) \cdots \mathtt{flat}^{-1}(s_n))\theta\gamma^{\mathtt{flat}^{-1}} = \mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}}$.

Now let $\sigma = (\sigma_1 \times \ldots \times \sigma_n) \longrightarrow \tau$ and $\sigma' = \sigma_1 \to \ldots \to \sigma_n \to \tau$. Suppose $s = f_{\sigma'} \cdot s_1 \cdots s_k$ with $k < n$. Then $\mathtt{flat}^{-1}(s\theta\gamma) = \mathtt{flat}^{-1}(f_{\sigma'\theta} \cdot s_1\theta\gamma \cdots s_k\theta\gamma) = \lambda x^{k+1}_{\sigma_{k+1}\theta} \ldots x^n_{\sigma_n\theta}.f_{\sigma\theta}(\mathtt{flat}^{-1}(s_1\theta\gamma), \ldots, \mathtt{flat}^{-1}(s_k\theta\gamma), x^{k+1}, \ldots, x^n)$.
With the induction hypothesis on each of the s_i this becomes:
$\lambda x^{k+1}_{\sigma_{k+1}\theta} \ldots x^n_{\sigma_n\theta}.f_{\sigma\theta}(\mathtt{flat}^{-1}(s_1)\theta\gamma^{\mathtt{flat}^{-1}}, \ldots, \mathtt{flat}^{-1}(s_k)\theta\gamma^{\mathtt{flat}^{-1}}, x^{k+1}, \ldots, x^n) =$
$(\lambda x^{k+1}_{\sigma_{k+1}} \ldots x^n_{\sigma_n}.f_\sigma(\mathtt{flat}^{-1}(s_1), \ldots, \mathtt{flat}^{-1}(s_k), x^{k+1}, \ldots, x^n))\theta\gamma^{\mathtt{flat}^{-1}} =$
$\mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}}$.

Alternatively, if $s = f_{\sigma'} \cdot s_1 \cdots s_k$ but $k \geq n$ we obtain the induction step in an equally simple way: $\mathtt{flat}^{-1}(s\theta\gamma) = \mathtt{flat}^{-1}(f_{\sigma'\theta} \cdot s_1\theta\gamma \cdots s_k\theta\gamma) =$
$f_{\sigma\theta}(\mathtt{flat}^{-1}(s_1\theta\gamma), \ldots, \mathtt{flat}^{-1}(s_1\theta\gamma)) \cdot \mathtt{flat}^{-1}(s_{k+1}\theta\gamma) \cdots \mathtt{flat}^{-1}(s_n\theta\gamma) =$ (IH)
$f_{\sigma\theta}(\mathtt{flat}^{-1}(s_1)\theta\gamma^{\mathtt{flat}^{-1}}, \ldots, \mathtt{flat}^{-1}(s_k)\theta\gamma^{\mathtt{flat}^{-1}})\cdot\mathtt{flat}^{-1}(s_{k+1})\theta\gamma^{\mathtt{flat}^{-1}} \cdots \mathtt{flat}^{-1}(s_n)\theta =$
$(f_\sigma(\mathtt{flat}^{-1}(s_1), \ldots, \mathtt{flat}^{-1}(s_k)) \cdot \mathtt{flat}^{-1}(s_{k+1}) \cdots \mathtt{flat}^{-1}(s_n))\theta\gamma^{\mathtt{flat}^{-1}} =$
$\mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}}$ as required.

Lemma 27 (Required for Theorem 5). *For term s, type substitution θ and substitution γ, $\mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}} \Rightarrow^*_\beta \mathtt{flat}^{-1}(s\theta\gamma)$.*

Proof. We use the same inductive reasoning as in Lemma 26, with $\Rightarrow^*_\beta$ instead of $=$ in the inductive steps (note that $\Rightarrow^*_\beta$ is reflexive so the base steps are included). What remains is the case $s = x_\sigma \cdot s_1 \cdots s_m$ with $m > 0$ and $x_{\sigma\theta} \in \mathrm{dom}(\gamma)$. In this case, $\mathtt{flat}^{-1}(s)\theta\gamma^{\mathtt{flat}^{-1}} = \mathtt{flat}^{-1}(\gamma(x_{\sigma\theta}))\cdot\mathtt{flat}^{-1}(s_1)\theta\gamma^{\mathtt{flat}^{-1}} \cdots \mathtt{flat}^{-1}(s_m)\theta \Rightarrow^*_\beta$ (IH) $\mathtt{flat}^{-1}(\gamma(x_{\sigma\theta})) \cdot \mathtt{flat}^{-1}(s_1\theta\gamma) \cdots \mathtt{flat}^{-1}(s_m\theta\gamma)$. Now, if $\gamma(x_{\sigma\theta})$ is headed

by an abstraction or variable, this is exactly $\mathtt{flat}^{-1}(s\theta\gamma)$. If $\gamma(x_{\sigma\theta})$ has the form $f_{\tau'} \cdot t_1 \cdots t_k$ with $k \geq ar(f)$, then this is also the case. The only exception is when $\gamma(x_{\sigma\theta}) = f_{\tau'} \cdot t_1 \cdots t_k$ with $k < n := ar(f)$. In this case, the term we have expands to $(\lambda x_{k+1} \ldots x_n . f_\tau(\mathtt{flat}^{-1}(t_1), \ldots, \mathtt{flat}^{-1}(t_k), x_{k+1}, \ldots, x_n)) \cdot \mathtt{flat}^{-1}(s_1\theta\gamma) \cdots \mathtt{flat}^{-1}(s_m\theta\gamma)$. If $m \leq n - k$ this term β-reduces to:
$\lambda x_{k+m+1} \ldots x_n . f_\tau(\mathtt{flat}^{-1}(t_1), \ldots, \mathtt{flat}^{-1}(t_k), \mathtt{flat}^{-1}(s_1\theta\gamma), \ldots, \mathtt{flat}^{-1}(s_m\theta\gamma), x_{k+m+1}, \ldots, x_n) = \mathtt{flat}^{-1}(f_{\tau'} \cdot t_1 \cdots t_k \cdot s_1\theta\gamma \cdots s_m\theta\gamma) = \mathtt{flat}^{-1}(s\theta\gamma)$.
On the other hand, if $m > n - k$ this term β-reduces to:
$f_\tau(t_1, \ldots, t_k, \mathtt{flat}^{-1}(s_1\theta\gamma), \ldots, \mathtt{flat}^{-1}(s_{n-k}\theta\gamma)) \cdot \mathtt{flat}^{-1}(s_{n-k+1}\theta\gamma) \cdots \mathtt{flat}^{-1}(s_n\theta\gamma) = \mathtt{flat}^{-1}(f_{\sigma'} \cdot t_1 \cdots t_k \cdot s_1\theta\gamma \cdots s_m\theta\gamma) = \mathtt{flat}^{-1}(s\theta\gamma)$.

Lemma 28 (Part of Theorem 5). *If $s \Rightarrow_{R^{\mathtt{flat}}} t$ then $\mathtt{flat}^{-1}(s) \Rightarrow_R^+ \mathtt{flat}^{-1}(t)$.*

Proof. By induction on the form of s. The induction step (when the reduction takes place in a subterm) is easy in all cases (whether s is headed by a variable, headed by an abstraction, or has the form $f \cdot s_1 \cdots s_k$ with $k \leq ar(f)$ or $k > ar(f)$). Consider the two base cases. If $s = (\lambda x.u) \cdot v$ and $t = u[x := v]$ then $\mathtt{flat}^{-1}(s) = (\lambda x.\mathtt{flat}^{-1}(u)) \cdot \mathtt{flat}^{-1}(v) \Rightarrow_\beta \mathtt{flat}^{-1}(u)[x := \mathtt{flat}^{-1}(v)] \Rightarrow_\beta^* \mathtt{flat}^{-1}(u[x := v]) = \mathtt{flat}^{-1}(t)$ by Lemma 27. If $s = \mathtt{flat}(l)\theta\gamma$ and $t = \mathtt{flat}(r)\theta\gamma$ with $l \Rightarrow r \in R$ then $\mathtt{flat}^{-1}(s) = \mathtt{flat}^{-1}(\mathtt{flat}(l)\theta\gamma^{\mathtt{flat}^{-1}}$ by Lemma 26. It is evident that $\mathtt{flat}^{-1}(\mathtt{flat}(u)) = u$ for all u, so this term equals $l\theta\gamma^{\mathtt{flat}^{-1}} \Rightarrow_R r\theta\gamma^{\mathtt{flat}^{-1}}$, which by Lemma 27 $\Rightarrow_\beta^* \mathtt{flat}^{-1}(r\theta\gamma) = \mathtt{flat}^{-1}(t)$.

*Proof (**Proof of Theorem 5**).* Evident with the combination of Lemmas 25 and 28.

Section 8: η-expansion

We first discuss several lemmas about η-long forms. Although η-expansion is very common in the literature, the definition of *restricted* η-expansion used in this paper is not, and therefore some attention to the details is in order.

Lemma 29. *If $l = l\uparrow^\eta_{FVar(l)}$ and $r = r\uparrow^\eta_{FVar(r)}$ for all $l \Rightarrow r \in \mathcal{R}$, then $\mathcal{R}^\uparrow = \mathcal{R}$.*

Proof. Note that all rules in $\mathcal{R}$ are monomorphic, and have base type, the first by convention, the second because by assumption l cannot be an abstraction or variable. Thus, $\mathcal{R}^\uparrow$ only contains base-type rules, where $[\![l]\!] = l\uparrow^\eta_{FVar(l)} = l$ and $[\![r]\!] = r\uparrow^\eta_{FVar(r)} = r$.

Lemma 30. *If $s = s\uparrow^\eta_{\mathrm{dom}(\gamma)}$ and all $\gamma(x)$ are η-long, then $s\gamma$ is η-long.*

Proof. By induction on s, all cases trivial (note that a term $(\lambda x.s_0) \cdot s_1 \cdots s_n$ is η-long if all s_i are).

Lemma 31. *Suppose $l = l\uparrow^\eta_V$ and γ is a substitution on domain V, and s is an η-long term. If $s = l\gamma$ and $HV(l) \cap V = \emptyset$, then all $\gamma(x)$ are η-long for $x \in FVar(l) \cap V$.*

Proof. By induction on the size of l. If l is a (typed) variable not in V, then there are no variables in $FVar(l) \cap V$ so the lemma automatically holds. If l is a (typed) variable in V, note that $FVar(l) \cap V = \{l\}$ and $\gamma(l) = s$ is η-long by assumption. If $l = x \cdot l_1 \cdots l_n$, note that by assumption $x \notin \mathrm{dom}(\gamma)$ and therefore $s = x \cdot l_1\gamma \cdots l_n\gamma$; by induction all $\gamma(x)$ occurring in any $FVar(l_i) \cap V$ are η-long, and since all $x \in FVar(l)$ occur in some i this suffices. The other inductive cases, $l = f(l_1, \ldots, l_n) \cdot l_{n+1} \cdots l_m$ and $l = (\lambda x.l_0) \cdot l_1 \cdots l_n$, are both similarly trivial with induction.

Lemma 32 (Part of Theorem 6). *If $\mathcal{R}$ is a set of rules in η-long form and $s \Rightarrow_{\mathcal{R}} t$ with s an η-long term, then t is also η-long.*

Proof. Induction on the size of s. If $s = f(s_1, \ldots, s_n) \cdot s_{n+1} \cdots s_m$ and $t = f(s'_1, \ldots, s'_n) \cdot s'_{n+1} \cdots s'_m$, with each $s_i \Rightarrow^{=}_{\mathcal{R}} s'_i$. Since by the induction hypothesis all s'_i are η-long, so is t. If $s = x \cdot s_1 \cdots s_n$ of base type and $t = x \cdot s'_1 \cdots s'_n$, then by the induction hypothesis all s' are η-long and therefore so is t. If $s = (\lambda x.s_0) \cdot s_1 \cdots s_n$ and $t = (\lambda x.s'_0) \cdot s'_1 \cdots s'_n$ then again by induction all s_i are η-long and therefore so is t.

As for the two base cases, if $s = (\lambda x.u) \cdot v \Rightarrow_\beta u[x := v] = t$, then note that u and v are both η-long, and therefore by Lemma 30 $u[x := v]$ is as well. If $s = l\gamma \Rightarrow_{\mathcal{R}} r\gamma$ with $l \Rightarrow r \in \mathcal{R}$, then by Lemma 31 all $\gamma(x)$ are η-long (and $\mathrm{dom}(\gamma) = FVar(l)$, otherwise). Therefore we can use Lemma 30 to see that $t = r\gamma$ is η-long as well.

Lemma 33. $(s{\uparrow^\eta}) \cdot (t{\uparrow^\eta}) \Rightarrow^*_\beta (s \cdot t){\uparrow^\eta}$.

Proof. This is a property of η-expansion and independent from our definition of restricted expansion. We will prove it below for completeness.

First note that for all variables: $x{\uparrow^\eta}[x := x{\uparrow^\eta}] \Rightarrow^*_\beta x{\uparrow^\eta}$, as we can see by induction on the type of x – evident if x has base type, and if $x{\uparrow^\eta} = \lambda y_1 \ldots y_n.x \cdot y_1{\uparrow^\eta} \cdots y_n{\uparrow^\eta}$ then $x{\uparrow^\eta}[x := x{\uparrow^\eta}] = \lambda \boldsymbol{y}.(\lambda \boldsymbol{y}.x \cdot \boldsymbol{y}{\uparrow^\eta}) \cdot \boldsymbol{y}{\uparrow^\eta} \Rightarrow^*_\beta \lambda \boldsymbol{y}.x \cdot (y_1{\uparrow^\eta}[y_1 := y_1{\uparrow^\eta}]) \cdots (y_k{\uparrow^\eta}[y_k := y_k{\uparrow^\eta}])$, which by the induction hypothesis $\Rightarrow^*_\beta \lambda \boldsymbol{y}.x \cdot y_1{\uparrow^\eta} \cdots y_k{\uparrow^\eta} = x{\uparrow^\eta}$ because the type of y_i is a subtype of the type of x.

Now we prove the lemma by induction on the type of s.
If s is an abstraction then $(s{\uparrow^\eta}) \cdot (t{\uparrow^\eta}) = (s \cdot t){\uparrow^\eta}$.
Otherwise $(s{\uparrow^\eta}) \cdot (t{\uparrow^\eta}) = (\lambda x y_1 \ldots y_n.s' \cdot x{\uparrow^\eta} \cdot y_1{\uparrow^\eta} \cdots y_n{\uparrow^\eta}) \cdot t{\uparrow^\eta}$ for some term s' while $(s \cdot t){\uparrow^\eta} = \lambda y_1 \ldots y_n.s' \cdot t{\uparrow^\eta} \cdot y_1{\uparrow^\eta} \cdots y_n{\uparrow^\eta}$ (as we can see by checking each of the forms s might have). Then $(s{\uparrow^\eta}) \cdot (t{\uparrow^\eta}) \Rightarrow_\beta \lambda y_1 \ldots y_n.s' \cdot (x{\uparrow^\eta}[x := t{\uparrow^\eta}]) \cdot \boldsymbol{y}{\uparrow^\eta}$. We are done if $x{\uparrow^\eta}[x := t{\uparrow^\eta}] \Rightarrow^*_\beta t{\uparrow^\eta}$, which is certainly the case if x has base type. Otherwise, write $x{\uparrow^\eta} = \lambda z_1 \ldots z_k.x \cdot z_1{\uparrow^\eta} \cdots z_k{\uparrow^\eta}$ and $t{\uparrow^\eta} = \lambda z_1 \ldots z_k.t' \cdot z_1{\uparrow^\eta} \cdots z_k{\uparrow^\eta}$. Then $x{\uparrow^\eta}[x := t{\uparrow^\eta}] = \lambda \boldsymbol{z}.(\lambda \boldsymbol{z}.t' \cdot \boldsymbol{z}{\uparrow^\eta})\boldsymbol{z}{\uparrow^\eta}$, which β-reduces to $\lambda \boldsymbol{z}.t' \cdot (z_1{\uparrow^\eta}[z_1 := z_1{\uparrow^\eta}]) \cdots (z_k{\uparrow^\eta}[z_k := z_k{\uparrow^\eta}])$. We have seen above that this $\Rightarrow^*_\beta \lambda \boldsymbol{z}.t' \cdot \boldsymbol{z}{\uparrow^\eta}$ as required.

Lemma 34. *Let s be a term and γ a substitution. Let $V \subseteq \mathrm{dom}(\gamma)$. Then $s{\uparrow^\eta_V}\gamma^\uparrow \Rightarrow^*_\beta (s\gamma){\uparrow^\eta}$, and this is an equality if $HV(s) \cap \mathrm{dom}(\gamma) = \emptyset$ and $V = \mathrm{dom}(\gamma)$.*

Proof. By induction on the form of s.

If $s = (\lambda x.u)\cdot v\cdot w_1 \cdots w_k$ then $s{\uparrow^\eta_V}\gamma^\uparrow = \lambda \boldsymbol{y}.(\lambda x.u{\uparrow^\eta_V}\gamma^\uparrow)\cdot v{\uparrow^\eta_V}\gamma^\uparrow\cdot w_1{\uparrow^\eta_V}\gamma^\uparrow\cdots w_k{\uparrow^\eta_V}\gamma^\uparrow\cdot \boldsymbol{y}{\uparrow^\eta}$. Note that if $HV(s)\cap \mathrm{dom}(\gamma) = \emptyset$ then also $HV(u)\cap\mathrm{dom}(\gamma)=\emptyset$, since using α-conversion we can assume x is fresh. The same holds for v and each w_i. Therefore this term either $\Rightarrow^*_\beta$ or, if the requirements are satisfied, equals $\lambda\boldsymbol{y}.(\lambda x.(u\gamma){\uparrow^\eta})\cdot(v\gamma){\uparrow^\eta}\cdot(w_1\gamma){\uparrow^\eta}\cdots(w_k\gamma){\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta} = (s\gamma){\uparrow^\eta}$.

We use the induction hypothesis in the same way if either $s = x_\sigma\cdot s_1\cdots s_m$ with $x_\sigma\notin\mathrm{dom}(\gamma)$ or $s = f(s_1,\ldots,s_n)\cdot s_{n+1}\cdots s_n$.

If $s = x_\sigma\in\mathrm{dom}(\gamma)$, then $s{\uparrow^\eta_V}\gamma^\uparrow = x_\sigma\gamma^\uparrow = \gamma(x_\sigma){\uparrow^\eta} = (s\gamma){\uparrow^\eta}$ if either $x_\sigma\in V$ or σ is a data type. Otherwise $x_\sigma{\uparrow^\eta_V} = \lambda y_1\ldots y_k.x\cdot y_1{\uparrow^\eta}\cdots y_k{\uparrow^\eta}$ and we can write $\gamma(x_\sigma){\uparrow^\eta} = \lambda y_1\ldots y_k.ty_1{\uparrow^\eta}\cdots y_k{\uparrow^\eta}$ for some term t. But then $x_\sigma{\uparrow^\eta_V}\gamma^\uparrow\Rightarrow^*_\beta \lambda\boldsymbol{y}.t\cdot y_1{\uparrow^\eta}[y_1 := y_1{\uparrow^\eta}]\cdots y_k{\uparrow^\eta}[y_k := y_k{\uparrow^\eta}]$. We have seen in the proof of Lemma 33 that this term $\Rightarrow^*_\beta\lambda\boldsymbol{y}.t\cdot\boldsymbol{y}{\uparrow^\eta} = s\gamma{\uparrow^\eta}$ as required. Note that the requirements for equality are not satisfied because $V\neq\mathrm{dom}(\gamma)$.

Finally, if $s = x_\sigma\cdot s_1\cdots s_n$ with $n>0$ and $x_\sigma\in\mathrm{dom}(\gamma)$, the requirements for equality are not satisfied. However, $s{\uparrow^\eta_V}\gamma^\uparrow = \lambda\boldsymbol{y}.\gamma(x_\sigma){\uparrow^\eta}\cdot s_1{\uparrow^\eta_V}\gamma^\uparrow\cdots s_n{\uparrow^\eta_V}\gamma^\uparrow\cdot\boldsymbol{y}{\uparrow^\eta}\Rightarrow^*_\beta$ (by the induction hypothesis) $\lambda\boldsymbol{y}.\gamma(x_\sigma){\uparrow^\eta}\cdot s_1\gamma{\uparrow^\eta}\cdots s_n\gamma{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta}\Rightarrow^*_\beta$ (by Lemma 33) $\lambda\boldsymbol{y}.\gamma(x_\sigma)\cdot s_1\gamma{\uparrow^\eta}\cdots s_n\gamma{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta} = s\gamma{\uparrow^\eta}$.

Lemma 35 (Support for both theorems in Section 8). *If $s\Rightarrow_{\mathcal{R}} t$ then $s{\uparrow^\eta}\Rightarrow^+_{\mathcal{R}^\uparrow} t{\uparrow^\eta}$.*

Proof. By induction on the form of s.

If $s = \lambda x.u\Rightarrow_{\mathcal{R}}\lambda x.u' = t$, then by the induction hypothesis $s{\uparrow^\eta} = \lambda x.u{\uparrow^\eta}\Rightarrow^+_{\mathcal{R}^\uparrow} u'{\uparrow^\eta} = t{\uparrow^\eta}$.

If $s = (\lambda x.u_0)\cdot u_1\cdots u_n$ and $t = (\lambda x.u'_0)\cdot u'_1\cdots u'_n$ with each $u_i\Rightarrow^=_{\mathcal{R}} u'_i$, exactly one strict, then by the induction hypothesis each $u_i{\uparrow^\eta}\Rightarrow^*_{\mathcal{R}^\uparrow}u'_i{\uparrow^\eta}$, exactly one strict. Thus $s{\uparrow^\eta} = \lambda\boldsymbol{y}.(\lambda x.u_0{\uparrow^\eta})\cdot u_1{\uparrow^\eta}\cdots u_n{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta}\Rightarrow^+_{\mathcal{R}^\uparrow}\lambda\boldsymbol{y}.(\lambda x.u'_0{\uparrow^\eta})\cdot u'_1{\uparrow^\eta}\cdots u'_n{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta} = t{\uparrow^\eta}$.

In the same way we can immediately conclude with the induction hypothesis if $s = f(s_1,\ldots,s_n)\cdot s_{n+1}\cdots s_m$ or if $s = x_\sigma\cdot s_1\cdots s_n$ and the reduction takes place in one of the s_i.

There are two further possibilities: either $s = (\lambda x.u)\cdot v\cdot w_1\cdots w_n$ and $t = u[x:=v]\cdot w_1\cdots w_n$ or $s = l\gamma\cdot w_1\cdots w_n$ and $t = r\gamma\cdot w_1\cdots w_n$ for some $l\Rightarrow r\in\mathcal{R}$ and substitution γ. In the first case, $s{\uparrow^\eta} = \lambda\boldsymbol{y}.(\lambda x.u{\uparrow^\eta})\cdot v{\uparrow^\eta}\cdot w_1{\uparrow^\eta}\cdots w_n{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta}\Rightarrow_\beta\lambda\boldsymbol{y}.(u{\uparrow^\eta}[x:=v{\uparrow^\eta}])\cdot w_1{\uparrow^\eta}\cdots w_n{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta}$. By Lemma 34 this term β-reduces to $\lambda\boldsymbol{y}.(u[x:=v]{\uparrow^\eta})\cdot w_1{\uparrow^\eta}\cdots w_n{\uparrow^\eta}\boldsymbol{y}{\uparrow^\eta}$, which by Lemma 33 β-reduces further to $\lambda\boldsymbol{y}.(u[x:=v]\cdot w_1\cdots w_n\cdot\boldsymbol{y}){\uparrow^\eta} = s\gamma{\uparrow^\eta}$.

In the second case, write $l = f(l_1,\ldots,l_k)\cdot l_{k+1}\cdots l_m$.
Then $s{\uparrow^\eta} = \lambda\boldsymbol{y}.f(l_1\gamma{\uparrow^\eta},\ldots,l_k\gamma{\uparrow^\eta})\cdot l_{k+1}\gamma{\uparrow^\eta}\cdots l_m\gamma{\uparrow^\eta}\cdot w_1\gamma{\uparrow^\eta}\cdots w_n\gamma{\uparrow^\eta}\cdot\boldsymbol{y}{\uparrow^\eta}$. Write $l' = l\cdot x_1\cdots x_n\cdot z_1\cdots z_m{\uparrow^\eta_V}$ with $V = FVar(l)\cup\{\boldsymbol{x},\boldsymbol{z}\}$. Also write $r' = r\cdot\boldsymbol{x}\cdot\boldsymbol{z}{\uparrow^\eta_V}$. Then $l'\Rightarrow r'\in\mathcal{R}^\uparrow$. Let $\delta = \gamma\cup[x_1:=w_1,\ldots,x_n:=w_n,z_1:=y_1,\ldots,z_m:=y_m]$. Note that we can safely assume that $\mathrm{dom}(\gamma) = FVar(l)$ and that $HV(l) = \emptyset$. Therefore we can use Lemma 34 to conclude that $s{\uparrow^\eta} = \lambda\boldsymbol{y}.(l\cdot\boldsymbol{x}\cdot\boldsymbol{z})\delta{\uparrow^\eta} = \lambda\boldsymbol{y}.l'\delta^\uparrow\Rightarrow_{\mathcal{R}^\uparrow}\lambda\boldsymbol{y}.r'\delta^\uparrow\Rightarrow^*_\beta\lambda\boldsymbol{y}.(r\cdot\boldsymbol{x}\cdot\boldsymbol{z})\delta{\uparrow^\eta} = t{\uparrow^\eta}$.

*Proof (**Proof of Theorem 6**).* Let $\mathcal{R}$ be a set of rules in restricted η-long form. Then $\Rightarrow_{\mathcal{R}}$ maps η-long terms to η-long terms by Lemma 32 and evidently $\Rightarrow_{\mathcal{R}}$ is non-terminating if it is non-terminating on η-long terms. If $\Rightarrow_{\mathcal{R}}$ is non-terminating, so there is an infinite reduction $s_1 \Rightarrow_{\mathcal{R}} s_2 \Rightarrow_{\mathcal{R}} \ldots$ then by Lemma 35 also $s_1{\uparrow^\eta} \Rightarrow_{\mathcal{R}^\uparrow} s_2{\uparrow^\eta} \Rightarrow_{\mathcal{R}^\uparrow} \ldots$, where all $\Rightarrow_{\mathcal{R}^\uparrow}$ steps are also $\Rightarrow_{\mathcal{R}}$ steps by Lemma 29.

Proof. Let $\mathcal{R}$ be a set of rules and $\mathcal{R}^\uparrow$ the restricted η-long counterpart. Then by Lemma 35 any infinite $\Rightarrow_{\mathcal{R}}$ reduction can be replaced by an infinite $\Rightarrow_{\mathcal{R}^\uparrow}$ reduction on η-lnog terms.