Termination Proofs for Higher-order Rewrite Systems

Jaco van de Pol

Department of Philosophy, Utrecht University
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
email: jaco@phil.ruu.nl

Abstract. This paper deals with termination proofs for Higher-Order Rewrite Systems (HRSs), introduced in [12]. This formalism combines the computational aspects of term rewriting and simply typed lambda calculus. The result is a proof technique for the termination of a HRS, similar to the proof technique "Termination by interpretation in a well-founded monotone algebra", described in [8, 19]. The resulting technique is as follows: Choose a higher-order algebra with operations for each function symbol in the HRS, equipped with some well-founded partial ordering. The operations must be strictly monotonic in this ordering. This choice generates a model for the HRS. If the choice can be made in such a way that for each rule the interpretation of the left hand side is greater than the interpretation of the right hand side, then the HRS is terminating. At the end of the paper some applications of this technique are given, which show that this technique is natural and can easily be applied.

1 Introduction

In the field of automated proof verification one sees a development towards higher-order concepts. In the generic theorem prover Isabelle [15], typed lambda calculus is used as the syntax for the formulae. In other systems, as Coq [14], typed lambda calculus is even used for the logic, using the Curry-Howard isomorphism which links formulae to types and proofs to terms.

This development is mirrored in the research on Term Rewriting Systems (TRS). There are different formalisms dealing with the combination of term rewriting and an abstraction mechanism. In [11] the concept of Combinatory Reduction Systems (CRS) was introduced. These systems essentially are TRSs with bound variables. In [12, 13] the formalism of Higher-order Rewrite Systems (HRS) is described, which is very similar to CRSs in essence, but rather different in presentation. A precise comparison is given in [17]. A more general setting is given in [18]. Quite other approaches can be found in [4, 9].

Two important issues concerning rewrite systems are termination and confluence. For results about local confluence of HRSs and confluence of orthogonal HRSs the reader is referred to [12] and [13] respectively. In [11, ch. II.3] the confluence of regular CRSs is proved. However, the question of termination of the higher-order frameworks seems hardly to have been explored. As far as we know,

only in [11, ch. II.6.2] a sufficient condition for termination of regular CRSs is given. With this condition, stated in terms of redexes and descendants, a termination proof for CRSs remains a syntactical matter. Other work on this line is done in [10]. We also refer to [9] where a recursion scheme for higher-order rules is given that guarantees termination.

Termination of first-order Term Rewriting is already an undecidable problem. But as the termination of TRSs is an interesting question, many semi-algorithms and characterisations of termination are proposed in the literature. A nice characterisation of termination is given in [19]. It builds on the "Increasing Interpretation Method" of [8, p. 367]. The function symbols of a TRS $\mathcal{R}$ have to be interpreted as strictly monotonic operations in some well-founded algebra. This interpretation is extended to closed terms as a usual algebraic homomorphism. Now the associated rewrite relation is terminating if every left hand side is greater (under the chosen interpretation) than the belonging right hand side, for each possible interpretation of the variables in that rule.

The strength of this characterisation is that one can concentrate on the "intuitive reason" for termination. This intuition can be translated in suitable operations on well-founded orderings, thus using semantical arguments. The real termination proof consists of testing a simple condition on the rules only instead of on all possible rewrite steps or all possible redexes. This semantical approach is more convenient than a syntactical technique.

The aim of this paper is to generalise this semantical characterisation of termination for TRSs to one for HRSs. The definition of Higher-order Rewriting we use, is close to [18], so it is an extension of [12, 13]. The main result is that such a generalisation is possible. The interpretation of terms can be extended to the interpretation of higher-order terms. The orderings and the notion of strictness can also be generalised. The techniques to achieve this are similar to those used in [5, 6]. Moreover, the result that termination proofs can be given with a well-founded monotone algebra in [19] carries over to HRSs with simple conditions on the well-founded ordering. With this technique some natural HRSs are proved to be terminating (see Section 6).

I like to thank Jan Friso Groote, who supplied some crucial ideas. I am also grateful to Jan Bergstra, Marc Bezem, Tonny Hurkens, Vincent van Oostrom, Alex Sellink and Jan Springintveld for reading previous versions of this document and for their suggestions. Finally, I thank H. Schwichtenberg and both referees for their close reading of the preliminary version, resulting in several improvements.

2 Term Rewriting in the Simply Typed Lambda Calculus

2.1 Types and Terms

In this section the sets of types and terms of simply typed lambda calculus are defined. The types are constructed from a set of base types. Let $\mathcal{B}$ be the set of *base types*. Then the set $\mathbb{T}(\mathcal{B})$ of *simple types* over these base types is defined as:

Definition 1. $\mathbb{T}(\mathcal{B})$ is the smallest set satisfying

- $\mathcal{B} \subseteq \mathbb{T}(\mathcal{B})$.
- If $\sigma, \tau \in \mathbb{T}(\mathcal{B})$ then also $\sigma \to \tau \in \mathbb{T}(\mathcal{B})$.

Terms are constructed from typed constants and typed variables. Let $\mathcal{C}_\tau$ be disjoint sets of *function symbols* (also called constants) of type τ. The union $\bigcup \mathcal{C}_\tau$ is written as $\mathcal{C}$. Similarly, $\mathcal{V}_\tau$ and $\mathcal{V}$ are sets of *typed variables*. Each $\mathcal{V}_\tau$ is supposed to be infinite. Furthermore, $\mathcal{C}$ and $\mathcal{V}$ are disjoint. To specialise particular sets of constants and variables, the notion of a *signature* is introduced:

Definition 2. A signature ($\mathcal{F}$) is a triple $(\mathcal{B}, \mathcal{C}, \mathcal{V})$, where $\mathcal{B}$ are the base types, $\mathcal{C}$ is a set of typed constants and $\mathcal{V}$ is a set of typed variables.

Given a signature $\mathcal{F}$ we can define the set of *simply typed lambda terms*:

Definition 3. Let $\mathcal{F}$ be the signature $(\mathcal{B}, \mathcal{C}, \mathcal{V})$. The sets $\Lambda^{\to}_\tau(\mathcal{F})$ (with $\tau \in \mathbb{T}(\mathcal{B})$) are defined inductively by:

- $\mathcal{V}_\tau \subseteq \Lambda^{\to}_\tau(\mathcal{F})$,
- $\mathcal{C}_\tau \subseteq \Lambda^{\to}_\tau(\mathcal{F})$,
- If $m \in \Lambda^{\to}_{\sigma\to\tau}(\mathcal{F})$ and $n \in \Lambda^{\to}_\sigma(\mathcal{F})$, then $mn \in \Lambda^{\to}_\tau(\mathcal{F})$ (application),
- If $x \in \mathcal{V}_\sigma$ and $m \in \Lambda^{\to}_\tau(\mathcal{F})$, then $\lambda x.m \in \Lambda^{\to}_{\sigma\to\tau}(\mathcal{F})$ (abstraction).

The set of simply typed lambda terms over the signature $\mathcal{F}$ is $\bigcup_{\tau\in\mathbb{T}(\mathcal{B})} \Lambda^{\to}_\tau(\mathcal{F})$ and is denoted by $\Lambda^{\to}(\mathcal{F})$.

The set of free variables of a well-typed term t is defined as usual and denoted by $FV(t)$. In the sequel we often abbreviate $\mathbb{T}(\mathcal{B})$ and $\Lambda^{\to}(\mathcal{F})$ to $\mathbb{T}$ and $\Lambda^{\to}$. Terms without free variables are called closed. A variable x is called *bound* in a term t if it occurs in a subterm of t of the form $\lambda x.s$. Terms that only differ in the renaming of bound variables (known as α-conversion) are identified. This permits us to stick to the convention that variables never occur free and bound as well in any mathematical context. See [2, p. 26] for details about the *variable convention*.

To express the complexity of a term, the notion of *type level* is defined inductively on types:

Definition 4. The type level of a term in $\Lambda^{\to}_\sigma$ is $TL(\sigma)$, where

- $TL(b) = 0$, for $b \in \mathcal{B}$,
- $TL(\sigma \to \tau) = max(TL(\sigma) + 1, TL(\tau))$.

A *substitution* is a mapping from variables to terms of the same type. More precisely:

Definition 5. A substitution θ is a finite function $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, with $t_i \in \Lambda^{\to}_\tau$ if $x_i \in \mathcal{V}_\tau$. The set $\{x_1, \ldots, x_n\}$ is called the domain of this substitution, denoted by $DOM(\theta)$.

Substitutions are extended to homomorphisms on terms in the standard way. Due to the variable convention, substitution cannot create new bindings. We denote $\theta(t)$ by t^θ or, in case $\theta = \{x \mapsto a\}$, by $t[x := a]$.

We recall the β- and η-reduction schemes for the lambda calculus, denoted by $\rightarrow_\beta$ and $\rightarrow_\eta$ respectively:

Definition 6. The relation $\rightarrow_\beta$ is the smallest compatible relation satisfying $(\lambda x.m)n \rightarrow_\beta m[x := n]$. The relation $\rightarrow_\eta$ is the smallest compatible relation satisfying $\lambda x.(fx) \rightarrow_\eta f$. The reflexive, symmetric and transitive closure of $\rightarrow_\eta$ is denoted by $=_\eta$.

Standard theory tells us that every $\Lambda^{\rightarrow}$-term has a unique *normal form* with respect to β-conversion. This β-normal form is denoted by $t\downarrow_\beta$. Normal forms are always of the form $\lambda x_1.\cdots\lambda x_n.(at_1\cdots t_m)$, where $a \in \mathcal{V} \cup \mathcal{C}$.

2.2 Higher-order Rewrite Systems

There are various definitions of higher-order rewrite mechanisms in recent literature [11, 4, 12, 13, 18]. The definition in this subsection is not meant to add a new formalism to the existing ones. Most conditions on the rules are dropped, because they are not necessary in the proof. The rewrite relation is as liberal as possible. Of course, the main result applies to formalisms admitting fewer rules and fewer rewrite steps. The chosen formalism is much like the formalism in [18, ch. 4.1], but we made some choices.

A *Higher-order Rewrite System* is given by a signature and a set of rules:

Definition 7. A *Higher-order Rewrite System* $\mathcal{R}$ is a tuple $(\mathcal{F}, R)$, where $\mathcal{F}$ is a signature and R is a set of rules in this signature. A *rule* is a pair (l, r), with $l, r \in \Lambda^{\rightarrow}_{\tau}(\mathcal{F})$, both closed β-normal forms.

Rules (l, r) are denoted by $l \rightarrow r$. In this rule, l is called the left hand side and r the right hand side.

The definition of a rule is the same as in [18, p. 46], except that we don't use η-long forms. Free variables are not admitted in a rule, for reasons explained below. In the examples, we mostly leave out the λ-binders in front of the left- and right hand side of the rules for shortness, thus introducing free variables par abuse. The definition of HRSs in [13, p. 308] is a special case of Definition 7, because it has some extra conditions on the occurrences of the free variables (which should be abstracted from in the present definition) to guarantee decidability of a rewrite step. This restriction is not needed in the termination result.

Definition 8. A *context* is a term $\lambda\Box.C$, where C is in β-normal form and $\Box \in FV(C)$.

Definition 9. Let $\mathcal{R} = (\mathcal{F}, R)$ be an HRS. The rewrite relation is defined by: $s \rightarrow_{\mathcal{R}} t$ if and only if there exist a context C and a rule $l \rightarrow r \in R$, such that $(Cl)\downarrow_\beta =_\eta s$ and $(Cr)\downarrow_\beta =_\eta t$. Only s and t in β-normal form will be considered.

Definition 9 differs in three aspects from the definition of the higher-order rewrite relation in [18, p. 46]. Firstly, η-long forms are not required, but furthermore the notion of positions is circumvented. In this way the variable convention can be upheld[1]. To be able to rewrite subterms with bound variables the rules are required to be closed.

The proposed definition of rewriting is inspired by the Leibniz-equality in higher-order logic: $l =_D r$ abbreviates $\forall P : D \to Prop.P(r) \to P(l)$. An application of this formula reduces any goal $P(l)$ to a goal $P(r)$. To have a proper notion of reduction, the condition $\Box \in FV(C)$ is added. To ensure that C really depends on $\Box$, the condition that C is in β-normal form is added. Without these conditions we would never have termination:

Example 1. Let $l \to r$ be a rule of HRS $\mathcal{R}$. If $\lambda\Box.a$ were a context, we would have the rewrite step $a = ((\lambda\Box.a)l)\downarrow_\beta \to_{\mathcal{R}} ((\lambda\Box.a)r)\downarrow_\beta = a$. The same example can be given if $\lambda\Box.(\lambda y.a)\Box$ were a context.

The last difference with [18] is that more occurrences of $\Box$ in a context are admitted. This is another advantage of seeing contexts as ordinary λ-terms and is close to the pragmatics of Leibniz equality. The transitive closure of the rewrite relation is not changed if we allow this form of parallelism, so confluence and termination are maintained.

Here is an example of rewriting:

Example 2. Let $+$ be a binary function symbol. If there is a rule $\lambda X.X + X \to \lambda X.X$, then we have the following rewrite steps:

Term		Reduct	Context used
$P + P$	$\to$	P	$\lambda\Box.\Box P$
$(P + P) + (P + P)$	$\to$	$(P + P)$	$\lambda\Box.\Box(P + P)$ or $\lambda\Box.(\Box P) + (\Box P)$
$(P + P) + (P + P)$	$\to$	P	$\lambda\Box.\Box(\Box P)$
$\lambda qy.(qy) + (qy)$	$\to$	$\lambda qy.(qy)$	$\lambda\Box qy.\Box(qy)$

3 The Model of Hereditarily Monotonic Functionals

We try to apply the general idea of the proof technique "termination by interpretation" for TRSs in [8, 19] to HRSs. The outline of this technique is as follows: The function symbols are interpreted by operations of the same arity in an algebra, equipped with a well-founded partial order. This interpretation is extended to the terms of the TRS in an algebraic way. The interpretation is chosen in such a way that for all rules, the left hand side is interpreted by a greater value than the right hand side. If such an interpretation can be found, the TRS is terminating. To prove the correctness of this technique, first we have to show that the ordering on terms is closed under substitution. The other step is to show that the ordering is closed under placing terms into a context. This can be proved

[1] In the notion of positions and occurrences, a replacement can introduce new bindings. Therefore contexts cannot be seen modulo α-conversion. See [12, p. 343]

using the fact that the function symbols are interpreted by strictly monotonic functions, thus preserving the ordering. Now we have for any substitution θ and context C that $C[l^\theta] > C[r^\theta]$. This is exactly the form of a rewrite step, thus showing that a rewrite step can be translated to a decrease in the well-founded ordering. In this way termination of rewriting is guaranteed.

In our definition of higher-order rewriting, we don't need closure under substitution, as the rules consist of closed terms. We only need closure under placing terms into a context; but this doesn't help us, because substitutions can easily be coded as special contexts. Matters are more complicated than in the first-order case. The notions of interpretation, strictness and ordering have to be extended to higher-order concepts. After these definitions the same idea can be used. It will turn out that we have to use two different orderings to show termination. The condition on the rules of an HRS is stated in terms of one ordering, so we try to show that for every rule $(l \to r)$, $l >_1 r$. Now we use a second ordering to show that for any context C, $Cl >_2 Cr$. This second ordering will be well-founded, thus proving termination of the HRS.

In this chapter we will define an interpretation for the terms into the hereditarily monotonic functions. A similar idea occurs already in [6] and [5]. One difference is that in this paper two β-equivalent terms have the same interpretation. This difference stems from the fact that we don't count β-steps. They are only used in the meta-language.

3.1 Interpretation of types by monotonic function spaces

Let us start with a type interpretation $\mathcal{I}$ of the set of base types $\mathcal{B}$:

Definition 10. A type interpretation $\mathcal{I}$ is a set of non-empty strict partial orders $\{(\mathcal{M}_B, >_B) | B \in \mathcal{B}\}$.

Starting from a base type interpretation $\mathcal{I}$, we define the interpretation of all types as sets of hereditarily monotonic functionals. A similar idea occurs in [6, p. 457], which shows that terms of the typed λ-I calculus denote strictly monotonic functions. The sets of *hereditarily monotonic functions* (monotonic functions for short) are denoted by $\mathcal{M}_\rho$ and depend on an ordering, denoted by ${}_{\text{mon}}\!\geq$. This partial order is inherited from the partial order on the interpretation of the base types in the way defined below. This ordering itself depends on the notion of monotonicity, so we give a definition by simultaneous induction:

Definition 11.

- For $\rho \in \mathcal{B}$:
  - $\mathcal{M}_\rho$ is as given by $\mathcal{I}$,
  - $a \ {}_{\text{mon}}\!\geq_\rho \ b$ iff $a, b \in \mathcal{M}_\rho$ and $a >_\rho b$ or $a = b$.
- For $\rho = \sigma \to \tau$:
  - $\mathcal{M}_\rho = \{f \in \mathcal{M}_\sigma \to \mathcal{M}_\tau \mid \text{for all } x, y \in \mathcal{M}_\sigma, \text{ if } x \ {}_{\text{mon}}\!\geq_\sigma \ y \text{ then } f(x) \ {}_{\text{mon}}\!\geq_\tau f(y)\}$,
  - $f \ {}_{\text{mon}}\!\geq_\rho \ g \iff f, g \in \mathcal{M}_\rho$ and for all $x \in \mathcal{M}_\sigma$, $f(x) \ {}_{\text{mon}}\!\geq_\tau \ g(x)$.

Instead of $a \in \mathcal{M}_\rho$ for some ρ we say: a is *monotonic*. Furthermore, a set $\mathcal{M}_\rho$ is called a *domain*. Now we can define the following strict partial order:

Definition 12. The relation ${}_{\text{mon}}{>}_\rho$ on domains $\mathcal{M}_\rho$ with $\rho \in \mathbb{T}$ is defined with induction on ρ:

- If $\rho \in \mathcal{B}$ and $a, b \in \mathcal{M}_\rho$ then $a \; {}_{\text{mon}}{>}_\rho \; b$ if and only if $a >_\rho b$.
- If $\rho = \sigma \to \tau$ and $f, g \in \mathcal{M}_\rho$ then $f \; {}_{\text{mon}}{>}_\rho \; g$ if and only if for all $x \in \mathcal{M}_\sigma$, $f(x) \; {}_{\text{mon}}{>}_\tau \; g(x)$.

The type subscripts in ${}_{\text{mon}}{>}_\rho$ and ${}_{\text{mon}}{\geq}_\rho$ are omitted when this is not confusing. Note that $f \; {}_{\text{mon}}{>} \; g$ means that f is pointwise greater than g; only monotonic points are in the domain of f and g. The reader is warned not to confuse $f \; {}_{\text{mon}}{\geq} \; g$ with $f \; {}_{\text{mon}}{>} \; g \vee f = g$. We have the following fact about ${}_{\text{mon}}{>}$ and ${}_{\text{mon}}{\geq}$ only in one direction:

Lemma 13. *For all $\rho \in \mathbb{T}$ and $f, g \in \mathcal{M}_\rho$, if $f \; {}_{\text{mon}}{>} \; g$ or $f = g$, then $f \; {}_{\text{mon}}{\geq} \; g$.*

Proof. (Induction on the type of f and g) Let $f \; {}_{\text{mon}}{>} \; g$, or $f = g$. If $\rho \in \mathcal{B}$ it is trivial. If $\rho = \sigma \to \tau$, take an $x \in \mathcal{M}_\sigma$. Then $f(x) \; {}_{\text{mon}}{>} \; g(x)$ (in case $f \; {}_{\text{mon}}{>} \; g$) or $f(x) = g(x)$. But then, by induction hypothesis, $f(x) \; {}_{\text{mon}}{\geq} \; g(x)$. Now by Definition 11, $f \; {}_{\text{mon}}{\geq} \; g$. □

Lemma 14. *Transitivity between ${}_{\text{mon}}{>}$ and ${}_{\text{mon}}{\geq}$:*

- *If $f \; {}_{\text{mon}}{\geq} \; g$ and $g \; {}_{\text{mon}}{\geq} \; h$ then $f \; {}_{\text{mon}}{\geq} \; h$.*
- *If $f \; {}_{\text{mon}}{\geq} \; g$ and $g \; {}_{\text{mon}}{>} \; h$ then $f \; {}_{\text{mon}}{>} \; h$.*
- *If $f \; {}_{\text{mon}}{>} \; g$ and $g \; {}_{\text{mon}}{\geq} \; h$ then $f \; {}_{\text{mon}}{>} \; h$.*
- *If $f \; {}_{\text{mon}}{>} \; g$ and $g \; {}_{\text{mon}}{>} \; h$ then $f \; {}_{\text{mon}}{>} \; h$.*

Proof. Simple induction on the type of f. □

3.2 Interpretation of terms in domains

Now we can take our next step: A term in $\Lambda^{\to}_{\tau}$ has to be interpreted as a value in the domain $\mathcal{M}_\tau$. To interpret the terms we have to specify what the interpretation of the free variables and the constants will be. The free variables are dealt with by *valuations*: mappings from variables to values. The interpretation of constants is given by a *constant interpretation*.

Definition 15. A valuation α is a family of mappings α_ρ, with $\alpha_\rho : \mathcal{V}_\rho \to \mathcal{M}_\rho$ for $\rho \in \mathbb{T}$.

With $\alpha[x := a]$ we denote the valuation that behaves like α on all variables except x, where it returns a. To compare two valuations we define:

Definition 16. $\alpha_1 \geq \alpha_2$ if and only if for each $x \in \mathcal{V}$, $\alpha_1(x) \; {}_{\text{mon}}{\geq} \; \alpha_2(x)$.

The constants have to be interpreted by functionals of the right domain. Therefore the following notion is introduced:

Definition 17. A constant interpretation $\mathcal{J}$ is a family of functions $\mathcal{J}_\sigma : \mathcal{C}_\sigma \to \mathcal{M}_\sigma$ for $\sigma \in \mathbb{T}$.

Now we are ready to define the interpretation of terms, depending on a particular choice for the constant interpretation $\mathcal{J}$.

Definition 18. The interpretation of a term under the valuation α is defined inductively on the structure of the term:

- $[\![x]\!]_\alpha = \alpha(x)$ for $x \in \mathcal{V}$.
- $[\![c]\!]_\alpha = \mathcal{J}(c)$ for $c \in \mathcal{C}$.
- $[\![mn]\!]_\alpha = [\![m]\!]_\alpha [\![n]\!]_\alpha$.
- $[\![\lambda x.m]\!]_\alpha = \boldsymbol{\lambda} a \in \mathcal{M}_\sigma . [\![m]\!]_{\alpha[x:=a]}$ for $x \in \mathcal{V}_\sigma$.

If s is a closed term, then $[\![s]\!]_\alpha$ does not depend on α, so we may suppress this subscript.[2] The following theorem states that the interpretation of each term is monotonic. The proof is by simultaneous induction of the following conjunct:

Proposition 19. *For each $s \in \Lambda^{\to}$ and valuations α, β:*

1. *$[\![s]\!]_\alpha$ is monotonic.*
2. *If $\alpha \geq \beta$ then $[\![s]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![s]\!]_\beta$.*

Proof. (induction on the structure of s)
If $s = x \in \mathcal{V}$:

1. $[\![s]\!]_\alpha = \alpha(x)$ is monotonic by Definition 15.
2. $[\![s]\!]_\alpha = \alpha(x) \ {}_{\text{mon}}\!\geq \beta(x) = [\![s]\!]_\beta$. (From Definition 16).

If $s = c \in \mathcal{C}$:

1. $[\![s]\!]_\alpha = \mathcal{J}(c)$ is monotonic by Definition 17.
2. $[\![s]\!]_\alpha = \mathcal{J}(c) = [\![s]\!]_\beta$, so they are ${}_{\text{mon}}\!\geq$-related (Lemma 13).

If $s = mn$:

1. By induction hypothesis (1) both $[\![m]\!]_\alpha$ and $[\![n]\!]_\alpha$ are monotonic. Then by Definition 11, also $[\![s]\!]_\alpha = [\![m]\!]_\alpha [\![n]\!]_\alpha$ is monotonic.
2. By induction hypothesis (2) $[\![m]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![m]\!]_\beta$. By induction hypothesis (1) $[\![n]\!]_\alpha$ is monotonic, so we have, with Definition 11, that $[\![m]\!]_\alpha [\![n]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![m]\!]_\beta [\![n]\!]_\alpha$. We also get from the induction hypotheses (1,2) that $[\![n]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![n]\!]_\beta$ and $[\![m]\!]_\beta$ is monotonic. Therefore, with Definition 11, $[\![m]\!]_\beta [\![n]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![m]\!]_\beta [\![n]\!]_\beta$. Now, using transitivity of ${}_{\text{mon}}\!\geq$ we have $[\![s]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![s]\!]_\beta$.

If $s = \lambda x.t$: (Say $x \in \mathcal{V}_\sigma$)

1. Firstly, choose $a \in \mathcal{M}_\sigma$, then $[\![\lambda x.t]\!]_\alpha(a) = [\![t]\!]_{\alpha[x:=a]}$. This is in $\mathcal{M}_\tau$ by induction hypothesis (1), so $[\![s]\!]_\alpha \in \mathcal{M}_\sigma \to \mathcal{M}_\tau$. Furthermore, if $a \ {}_{\text{mon}}\!\geq_\sigma b$, then $\alpha[x := a] \geq \alpha[x := b]$, so by induction hypothesis (2) $[\![t]\!]_{\alpha[x:=a]} \ {}_{\text{mon}}\!\geq [\![t]\!]_{\alpha[x:=b]}$. This is equivalent to $[\![s]\!]_\alpha(a) \ {}_{\text{mon}}\!\geq [\![s]\!]_\alpha(b)$, so $[\![s]\!]_\alpha$ is monotonic.

[2] The existence of at least one valuation is guaranteed because the base domains are non-empty

2. Let $a \in \mathcal{M}_\sigma$. We have $\alpha[x := a] \ {}_{\text{mon}}\geq \beta[x := a]$. So using induction hypothesis (2) we can compute: $[\![\lambda x.t]\!]_\alpha(a) = [\![t]\!]_{\alpha[x:=a]} \ {}_{\text{mon}}\geq [\![t]\!]_{\beta[x:=a]} = [\![\lambda x.t]\!]_\beta(a)$. So indeed, $[\![s]\!]_\alpha \ {}_{\text{mon}}\geq [\![s]\!]_\beta$. □

Corollary 20. *Given a valuation α, $[\![\cdot]\!]_\alpha$ is a family of functions from $\Lambda^{\rightarrow}_\sigma$ to $\mathcal{M}_\sigma$ for each $\sigma \in \mathbb{T}$.*

We have an ordering on the domains and an interpretation mapping terms into domains. Using this interpretation we can define an ordering on terms. The free variables are dealt with by valuations:

Definition 21. For terms s, t in $\Lambda^{\rightarrow}$, $s \ {}_{\text{mon}}> t$ if and only if for each valuation α, $[\![s]\!]_\alpha \ {}_{\text{mon}}> [\![t]\!]_\alpha$.

The next lemma shows that free and bound variables are treated similarly with respect to the ordering ${}_{\text{mon}}>$:

Lemma 22. *If for two $\Lambda^{\rightarrow}_\tau$-terms $s \ {}_{\text{mon}}> t$ then $\lambda x.s \ {}_{\text{mon}}> \lambda x.t$.*

Proof. Let $s \ {}_{\text{mon}}> t$, say $x \in \mathcal{V}_\sigma$. Choose $k \in \mathcal{M}_\sigma$ arbitrarily. Let α be any valuation. Then

$$[\![\lambda x.s]\!]_\alpha(k) = [\![s]\!]_{\alpha[x:=k]} \ {}_{\text{mon}}>_\tau [\![t]\!]_{\alpha[x:=k]} = [\![\lambda x.t]\!]_\alpha(k) \ .$$

The equalities hold by Definition 18, the inequality by Definition 21. The calculation holds for any k, so $[\![\lambda x.s]\!]_\alpha \ {}_{\text{mon}}>_{\sigma\to\tau} [\![\lambda x.t]\!]_\alpha$. This holds for any α, so $\lambda x.s \ {}_{\text{mon}}> \lambda x.t$ □

3.3 Hereditarily monotonic functionals serve as a model of $\boldsymbol{\lambda}^{\rightarrow}_{\beta,\eta}$

We show that the monotonic functionals are a model of $\lambda^{\rightarrow}_{\beta,\eta}$. Lemma 24 establishes a link between substitutions and valuations. As a corollary we get that the ${}_{\text{mon}}>$-ordering is closed under substitution. Furthermore, the interpretation in the model of two β,η-convertible terms is the same. Together with Lemma 19 this proves that $\mathcal{M}$ forms a $\lambda^{\rightarrow}$-model. No new technique is used here, the proofs resemble those in [6, 5, 19].

Definition 23. Let θ be a substitution and α a valuation. Then the composition $\alpha \circ \theta$ is a new valuation, mapping $x \mapsto [\![x^\theta]\!]_\alpha$.

The mapping $\alpha \circ \theta$ is indeed a valuation, because $[\![x^\theta]\!]_\alpha$ is monotonic by Proposition 19. We have the following connection between valuations and substitutions:

Lemma 24 (Substitution Lemma). *Let θ be a substitution and α a valuation. Then for each term $s \in \Lambda^{\rightarrow}$ we have $[\![s^\theta]\!]_\alpha = [\![s]\!]_{\alpha\circ\theta}$.*

Proof. (induction on the structure of s)

- $s = x \in \mathcal{V}$: by definition of $\alpha \circ \theta$.

– $s = c \in \mathcal{C}$: $[\![c^\theta]\!]_\alpha = [\![c]\!]_\alpha = \mathcal{J}(c) = [\![c]\!]_{\alpha\circ\theta}$.
– $s = mn$: Using the induction hypothesis for m and n, we have

$$[\![s^\theta]\!]_\alpha = [\![m^\theta]\!]_\alpha [\![n^\theta]\!]_\alpha = [\![m]\!]_{\alpha\circ\theta}[\![n]\!]_{\alpha\circ\theta} = [\![s]\!]_{\alpha\circ\theta} \ .$$

– $s = \lambda x.m$: Say $x \in \mathcal{V}_\sigma$. Let $a \in \mathcal{M}_\sigma$.

Claim. *In this situation we have* $(\alpha[x := a]) \circ \theta = (\alpha \circ \theta)[x := a]$.

Proof. By the variable convention, $x \notin DOM(\theta)$, so $x^\theta = x$, and for all $y \in DOM(\theta)$, $x \notin FV(y^\theta)$. Under this convention, $((\alpha[x := a]) \circ \theta)(x) = [\![x^\theta]\!]_{\alpha[x:=a]} = a = ((\alpha \circ \theta)[x := a])(x)$. And for $y \neq x$: $((\alpha[x := a]) \circ \theta)(y) = [\![y^\theta]\!]_{\alpha[x:=a]} = [\![y^\theta]\!]_\alpha = (\alpha \circ \theta)(y) = ((\alpha \circ \theta)[x := a])(y)$. □

Now this case can be proved by the following calculation:

$$\begin{aligned}
[\![s^\theta]\!]_\alpha &= \boldsymbol{\lambda} a \in \mathcal{M}_\sigma.([\![m^\theta]\!]_{\alpha[x:=a]}) \\
&= \boldsymbol{\lambda} a \in \mathcal{M}_\sigma.([\![m]\!]_{(\alpha[x:=a])\circ\theta}) \text{ (by i.h.)} \\
&= \boldsymbol{\lambda} a \in \mathcal{M}_\sigma.([\![m]\!]_{(\alpha\circ\theta)[x:=a]}) \text{ (by claim)} \\
&= [\![s]\!]_{\alpha\circ\theta} \ .
\end{aligned}$$

□

From the Substitution Lemma we have two corollaries, one with respect to the ordering $_{\text{mon}}>$ and one (Proposition 26) with respect to β-equality.

Corollary 25. *If for terms* s, t *in* $\Lambda^{\rightarrow}$, $s \ _{\text{mon}}> t$ *then for each substitution* θ, $s^\theta \ _{\text{mon}}> t^\theta$.

Proof. Let $s \ _{\text{mon}}> t$. Then by Lemma 24, $[\![s^\theta]\!]_\alpha = [\![s]\!]_{\alpha\circ\theta} \ _{\text{mon}}> [\![t]\!]_{\alpha\circ\theta} = [\![t^\theta]\!]_\alpha$ for any valuation α and substitution θ, so indeed $s^\theta \ _{\text{mon}}> t^\theta$. □

Proposition 26. *If* $s \to_\beta t$ *for two terms in* $\Lambda^{\rightarrow}$, *then* $[\![s]\!]_\alpha = [\![t]\!]_\alpha$ *for each valuation* α.

Proof. (induction to the depth of the β-redex in s.)

Let $s \to_\beta t$. The only interesting case is that the β-step takes place on top-level: Let $s = (\lambda x.s_1)s_2$, then $t = s_1[x := s_2]$. We have the following calculation:

$$[\![(\lambda x.s_1)s_2]\!]_\alpha = [\![s_1]\!]_{\alpha[x:=[\![s_2]\!]_\alpha]} = [\![s_1[x := s_2]]\!]_\alpha \ .$$

The first equality holds by Definition 18, the second because of Lemma 24, applied on α and the substitution $\{x := s_2\}$. □

A similar result holds for η-reduction:

Proposition 27. *If* $s \to_\eta t$ *for two terms in* $\Lambda^{\rightarrow}$, *then* $[\![s]\!]_\alpha = [\![t]\!]_\alpha$ *for each valuation* α.

Proof. This is obvious, because the domains have extensional equality. □

4 An ordering which is closed under context

4.1 On strictness

In section 3 we saw that the ordering $_{\text{mon}}>$ on terms is closed under substitution. We would like that this ordering is also closed under placing a term into a context. The first objection to this is the interpretation of the constants. We have to ensure that this interpretation is order preserving. The proof in [19] also uses the condition that the constants have to be interpreted by *strictly monotonic* operations. Therefore we define the following notion:

Definition 28. The predicate "f is strictly $_{\text{mon}}>$-monotonic" with $f \in \mathcal{M}_\rho$ is defined with induction on ρ:

- For $\rho \in \mathcal{B}$: $f \in \mathcal{M}_\rho$ is always strictly $_{\text{mon}}>$-monotonic.
- For $\rho = \sigma \to \tau$: $f \in \mathcal{M}_\rho$ is strictly $_{\text{mon}}>$-monotonic if and only if for all $x \in \mathcal{M}_\sigma$, $f(x)$ is strictly $_{\text{mon}}>$-monotonic, and for all $x, y \in \mathcal{M}_\sigma$, if $x \;_{\text{mon}}>_\sigma y$ then $f(x) \;_{\text{mon}}>_\tau f(y)$.

Unfortunately this notion of strict monotonicity is not strong enough. This is shown in the following example:

Example 3. In this example there is one base type, o, which is interpreted by the natural numbers. Type $o \to o$ is written as 1, type $1 \to o$ is written as 2. Consider the following functional: $S := \boldsymbol{\lambda} G \in \mathcal{M}_2.\boldsymbol{\lambda} g \in \mathcal{M}_1.G(g) + g(0)$. This is strictly $_{\text{mon}}>$-monotonic: Let $G \in \mathcal{M}_2$ and take $f \;_{\text{mon}}>_1 g$, Then $S(G)(f) = G(f) + f(0) \;_{\text{mon}}> G(f) + g(0) \;_{\text{mon}}\geq G(g) + g(0) = S(G)(g)$. The last inequality holds, because G is monotonic. This shows that $S(G)$ is strictly $_{\text{mon}}>$-monotonic. Now, take $G \;_{\text{mon}}>_2 F$ and $g \in \mathcal{M}_1$, then $S(G)(g) = G(g) + g(0) \;_{\text{mon}}> F(g) + g(0) = S(F)(g)$. This shows that $S(G) \;_{\text{mon}}> S(F)$. So S is strictly $_{\text{mon}}>$-monotonic.

But in fact, this is undesired. Consider $G := \boldsymbol{\lambda} f.fA$ and $g := \boldsymbol{\lambda} x.5$. Then $S(G)(g) = 10$, so the A leaves no traces.

This example can be used to "prove" that the following non-terminating HRS is terminating:

Example 4. The definitions of Example 3 are used. Take the signature with one base type o and constants $\mathcal{C} = \{c, d : o; @ : 2 \to 1 \to o\}$. The set of variables $\mathcal{V}$ includes $\{x, y : o; f, g : 1\}$. In this signature we define the HRS with one rule:

$$@(\lambda f.fx)(\lambda y.c) \to @(\lambda g.g(@(\lambda f.fx)(\lambda y.c)))(\lambda y.d).$$

First of all, this HRS is not terminating, because the left hand side is a subterm of the right hand side. Now we choose as interpretation:

$$\begin{array}{ll} \mathcal{I}(o) = \mathbb{N} & \mathcal{J}(c) = 2 \\ \mathcal{J}(d) = 1 & \mathcal{J}(@) = S \ . \end{array}$$

Under this interpretation we can compute the left hand side and the right hand side. It will turn out that the left hand side equals 4 and the right hand side equals 2. This clearly shows that S cannot be called "strict".

Another objection against strict $_{\text{mon}}>$-monotonicity of contexts, is the interpretation of the variables:

Example 5. Let $l \,_{\text{mon}}> r$ for some terms l and r. Then it is not the case that $xl \,_{\text{mon}}> xr$: Take a valuation α with $\alpha(x) = \boldsymbol{\lambda} a.c$, for some constant c, then $[\![xl]\!]_\alpha = [\![xr]\!]_\alpha$.

This example shows that although $l \,_{\text{mon}}> r$, we cannot expect that $xl \,_{\text{mon}}> xr$ for any value for x. So this order is not closed under placing terms into a context. Fortunately, we can weaken our desires. We don't need this order for all values for x, because the free variables of the context (which is in β-normal form) cannot be instantiated during a rewrite step. Furthermore, if we have the rewrite step $xl \to_{\mathcal{R}} xr$, then it is not the case, that we have the same step for all substitutions for x: Some substitutions lead to a non legal context. So we have the freedom to restrict the condition and to look at some particular value of x. The idea, due to Jan Friso Groote, is to look at precisely those x that preserve the order, that is for the *strictly monotonic* x. This leads to a new ordering, $_{\text{str}}>$, which (intuitively) runs: $f \,_{\text{str}}> g$ if and only if for all strictly monotonic x, $f(x) \,_{\text{str}}> g(x)$. This new ordering is used to compare the terms of the rewrite sequence.

Below a simultaneous definition of a new notion of *strictness* and a new ordering $_{\text{str}}>$ is given, in such a way that strictness is stronger than strictly $_{\text{mon}}>$-monotonic and $_{\text{str}}>$ is weaker than $_{\text{mon}}>$. Here we diverge from [6, 5].

Definition 29. The relation $_{\text{str}}>_\rho$ and the set $\mathcal{S}_\rho$ of hereditarily strict functionals are defined with induction on $\rho \in \mathbb{T}$:

- For $\rho \in \mathcal{B}$:
  - $a \,_{\text{str}}>_\rho b$ if and only if $a, b \in \mathcal{M}_\rho$ and $a >_\rho b$ (according to a fixed type interpretation $\mathcal{I}$).
  - $\mathcal{S}_\rho = \mathcal{M}_\rho$.
- For $\rho = \sigma \to \tau$: Let $f, g \in \mathcal{M}_\rho$, then
  - $f \,_{\text{str}}>_\rho g$ if and only if $f \,_{\text{mon}}\geq_\rho g$[3] and for all $x \in \mathcal{S}_\sigma$, $f(x) \,_{\text{str}}>_\tau g(x)$.
  - $f \in \mathcal{S}_\rho$ if and only if
    - $f \in \mathcal{M}_\rho$ and
    - for all $x \in \mathcal{M}_\sigma$, $f(x) \in \mathcal{S}_\tau$ and
    - for all $x, y \in \mathcal{M}_\sigma$, if $x \,_{\text{str}}>_\sigma y$ then $f(x) \,_{\text{mon}}>_\tau f(y)$.

If $f \in \mathcal{S}_\rho$ for some $\rho \in \mathbb{T}$, we say that f is *strict*. We will omit the type subscripts for $_{\text{str}}>_\rho$. Now we can prove the following relation between the different partial orders:

Lemma 30. *Let $\rho \in \mathbb{T}$ and $f, g \in \mathcal{M}_\rho$. If $f \,_{\text{mon}}>_\rho g$ then $f \,_{\text{str}}>_\rho g$.*

Proof. (induction on ρ):

If $\rho \in \mathcal{B}$ the two orderings coincide. Let $f \,_{\text{mon}}>_\rho g$ for $\rho = \sigma \to \tau$. By Lemma 13, $f \,_{\text{mon}}\geq g$. Take an arbitrary strict $x \in \mathcal{S}_\sigma$. Then also $x \in \mathcal{M}_\sigma$,

[3] This addition is due to H. Schwichtenberg, and appears to be necessary in applications with arbitrary high order.

so $f(x)\ {}_{\text{mon}}{>}_\tau\ g(x)$. By induction hypothesis, $f(x)\ {}_{\text{str}}{>}_\tau\ g(x)$, giving $f\ {}_{\text{str}}{>}\ g$ (Definition 29). □

Corollary 31. *Let f be strict, then both f is strictly ${}_{\text{mon}}{>}$-monotonic and f is strictly ${}_{\text{str}}{>}$-monotonic.*

Proof. If $x\ {}_{\text{mon}}{>}\ y$ then $x\ {}_{\text{str}}{>}\ y$ (Lemma 30) and $f(x)\ {}_{\text{mon}}{>}\ f(y)$, by strictness of f. If $x\ {}_{\text{str}}{>}\ y$ then by strictness of f: $f(x)\ {}_{\text{mon}}{>}\ f(y)$ and by Lemma 30 $f(x)\ {}_{\text{str}}{>}\ f(y)$. □

Thus strict is indeed stronger than strictly ${}_{\text{mon}}{>}$-monotonic, and ${}_{\text{str}}{>}$ is indeed weaker than ${}_{\text{mon}}{>}$. The following example shows that the problem from Example 3 is solved:

Example 6. The functional $S = \boldsymbol{\lambda} G \in \mathcal{M}_2.\boldsymbol{\lambda} g \in \mathcal{M}_1.G(g)+g(0)$ from Example 3 is not strict: Take

$$G = \boldsymbol{\lambda} g. \begin{cases} 5 & \text{if } g(0) = g(1) = 0 \\ 10 & \text{else,} \end{cases}$$

and $F = \boldsymbol{\lambda} g.5$. Then F and G are both monotonic, clearly $G\ {}_{\text{str}}{>}\ F$, but the relation $S(G)\ {}_{\text{mon}}{>}\ S(F)$ doesn't hold, because $S(G)(\boldsymbol{\lambda} x.0) = S(F)(\boldsymbol{\lambda} x.0)$.

The type level of functional S is 3. In the next two lemmas this complexity is shown to be essential.

Lemma 32. *Let $f, g \in \mathcal{M}_\rho$. If $TL(\rho) \leq 1$ then $f\ {}_{\text{str}}{>}_\rho\ g$ implies $f\ {}_{\text{mon}}{>}_\rho\ g$.*

Proof. (induction on ρ).

- $\rho \in \mathcal{B}$: By Definition 29.
- $\rho = \sigma \to \tau$: $TL(\rho)$ must be 1, so $\sigma \in \mathcal{B}$ and $TL(\tau) \leq 1$. Let $f\ {}_{\text{str}}{>}\ g$ and $x \in \mathcal{M}_\sigma$, then x is strict (Definition 29 for base type), so $f(x)\ {}_{\text{str}}{>}_\tau\ g(x)$. By induction hypothesis $f(x)\ {}_{\text{mon}}{>}\ g(x)$, so indeed $f\ {}_{\text{mon}}{>}\ g$. □

Lemma 33. *Let $f \in \mathcal{M}_\rho$. If $TL(\rho) \leq 2$ then f strictly ${}_{\text{mon}}{>}$-monotonic implies f is strict.*

Proof. (induction on ρ).

- If $\rho \in \mathcal{B}$ then f is strict by Definition 29.
- If $\rho = \sigma \to \tau$ then $TL(\sigma) \leq 1$ and $TL(\tau) \leq 2$. Let f be strictly ${}_{\text{mon}}{>}$-monotonic, then for $x, y \in \mathcal{M}_\sigma$ we have $f(x)$ is strictly ${}_{\text{mon}}{>}$-monotonic and by induction hypothesis $f(x)$ is strict. Furthermore, if $x\ {}_{\text{str}}{>}_\sigma\ y$ then $x\ {}_{\text{mon}}{>}_\sigma\ y$ (Lemma 32), so $f(x)\ {}_{\text{mon}}{>}\ f(y)$. These two facts yield that f is strict (Definition 29). □

The notion of strictness is extended to the interpretation of constants and variables:

Definition 34. A constant interpretation $\mathcal{J}$ is strict, if $\mathcal{J}(c)$ is strict for all $c \in \mathcal{C}$.

Definition 35. A valuation α is *strict* if for each $x \in \mathcal{V}$, $\alpha(x)$ is strict.

We can lift $_{\mathrm{str}}>$ from the domain level into the term level in the same way as we lifted $_{\mathrm{mon}}>$. To treat bound and free variables similarly, we now use strict valuations only:

Definition 36. For terms s, t of $\Lambda^{\rightarrow}$, $s \;_{\mathrm{str}}> t$ if for each *strict* valuation α, $[\![s]\!]_\alpha \;_{\mathrm{str}}> [\![t]\!]_\alpha$.

Lemma 37. *For all $s, t \in \Lambda^{\rightarrow}$, if $s \;_{\mathrm{mon}}> t$ then $s \;_{\mathrm{str}}> t$.*

Proof. If $s \;_{\mathrm{mon}}> t$ for $s, t \in \Lambda^{\rightarrow}_\tau$ then for all valuations α, $[\![s]\!]_\alpha \;_{\mathrm{mon}}>_\tau [\![t]\!]_\alpha$. This implies (using Lemma 30) that $[\![s]\!]_\alpha \;_{\mathrm{str}}>_\tau [\![t]\!]_\alpha$. This holds for all α, so certainly for strict α this relation holds. Therefore $s \;_{\mathrm{str}}> t$. □

We end this section with an example that the ordering $_{\mathrm{str}}>$ cannot be used for the rules, so the ordering $_{\mathrm{mon}}>$ is still necessary.

Example 7. Let for constants l and r of base type, the only rule of an HRS be $\lambda x.xl \to \lambda x.xr$. An interpretation can be chosen, such that $l \;_{\mathrm{mon}}> r$. Then $\lambda x.xl \;_{\mathrm{str}}> \lambda x.xr$, but yet the HRS is non-terminating: Choose as context $C = \lambda\Box.\Box(\lambda y.c)$. Then $(C(xl))\downarrow_\beta \to (C(xr))\downarrow_\beta$, which is equivalent to $c \to c$. So the ordering $_{\mathrm{str}}>$ is too weak for the rules.

4.2 The ordering $_{\mathrm{str}}>$ is well-founded

The definition of strictness (Definition 29) is rather complex and it is not a priori clear that strict functionals exist. Some extra conditions are needed to guarantee that strict functionals exist in every domain. These conditions are collected in the following notion:

Definition 38. A type interpretation $\mathcal{I} = \{(\mathcal{M}_B, >_B) | B \in \mathcal{B}\}$ is called a *well-founded type interpretation* if the following conditions on the interpretation of base types are satisfied:

- For each $b \in \mathcal{B}$, $\mathcal{M}_b$ is non-empty.
- For each $b \in \mathcal{B}$, $>_b$ is well-founded.
- For each $b, c \in \mathcal{B}$, there exists a strict function in $\mathcal{M}_{b \to c \to c}$.

We suppose that for each combination (b, c) of base types one strict function is chosen and called $+_{b,c}$ (written infix when possible). This suggestive definition is justified by Proposition 40 which states that under a well-founded type interpretation, the domains are well-founded. But first, we need a lemma, saying that strict inhabitants exist for all domains, if we have a well-founded type interpretation:

Lemma 39. *If the underlying type interpretation is well-founded, then for every $\rho \in \mathbb{T}$ there is at least one inhabitant of $\mathcal{S}_\rho$, which we will call S_ρ.*

Proof. (with induction on the structure of ρ)

If ρ is of base type, we can choose any $S_\rho \in \mathcal{M}_\rho$, because the domains $\mathcal{M}_b$ are non-empty for base types b, and elements of $\mathcal{M}_b$ are always strict.

Now let $\rho = \sigma \to \tau$. Then σ and τ can be decomposed in $\sigma_1 \to \cdots \sigma_m \to c$ and $\tau_1 \to \cdots \tau_n \to d$, with $c, d \in \mathcal{B}$. By induction hypothesis there exist strict inhabitants S_{σ_j} and S_τ. Because the type interpretation is well-founded, we have the strict element $+_{c,d}$. Using this material we can define[4]:

$$S_\rho = \boldsymbol{\lambda} y \in \mathcal{M}_\sigma . \boldsymbol{\lambda} x_1 \in \mathcal{M}_{\tau_1} \cdots \boldsymbol{\lambda} x_n \in \mathcal{M}_{\tau_n} . (y S_{\sigma_1} \cdots S_{\sigma_m}) +_{c,d} (S_\tau x_1 \cdots x_n) \ .$$

Clearly, this functional is in the appropriate domain and it is also strict in y and $x_1, \ldots, x_n$:

$$\begin{aligned} & y_1 \ {}_{\mathrm{str}}{>}_\sigma \ y_2 \\ \Rightarrow\ & y_1 S_{\sigma_1} \cdots S_{\sigma_m} \ {}_{\mathrm{str}}{>}_c \ y_2 S_{\sigma_1} \cdots S_{\sigma_m} \\ \Rightarrow\ & +_{c,d}(y_1 S_{\sigma_1} \cdots S_{\sigma_m}) \ {}_{\mathrm{mon}}{>}_{d \to d} \ +_{c,d}(y_2 S_{\sigma_1} \cdots S_{\sigma_m}) \\ \Rightarrow\ & S_\rho y_1 \ {}_{\mathrm{mon}}{>}_\tau \ S_\rho y_2 \ . \end{aligned}$$

A similar reasoning applies when some x_i is varied: Let $x_1, \ldots, x_n, x' \in \mathcal{M}$. Let $x_i \ {}_{\mathrm{str}}{>} \ x'$. Because $S_\tau x_1 \cdots x_{i-1}$ is strict, $S_\tau x_1 \cdots x_n \ {}_{\mathrm{mon}}{>} \ S_\tau x_1 \cdots x' \cdots x_n$, so for monotonic y, $S_\rho y x_1 \cdots x_i \ {}_{\mathrm{mon}}{>} \ S_\rho y x_1 \cdots x_{i-1} x'$. □

Proposition 40. *If the underlying type interpretation is well-founded, then for every $\rho \in \mathbb{T}$, the partial order $(\mathcal{M}_\rho, \ {}_{\mathrm{str}}{>}_\rho)$ is well-founded.*

Proof. (induction on the structure of ρ)

For $\rho \in \mathcal{B}$ well-foundedness is given by the well-foundedness of $>_\rho$. Let $\rho = \sigma \to \tau$. Assume that there exist $\{f_i | i \in \mathbb{N}\}$ with $f_i \ {}_{\mathrm{str}}{>}_{\sigma \to \tau} \ f_{i+1}$. But now, using $S_\sigma \in \mathcal{S}_\sigma$ we can construct the descending chain $\{f_i(S_\sigma) | i \in \mathbb{N}\}$, with $f_i(S_\sigma) \ {}_{\mathrm{str}}{>}_\tau \ f_{i+1}(S_\sigma)$, which contradicts the induction hypothesis. □

Proposition 41. *The relation $\ {}_{\mathrm{str}}{>}$ is well-founded on terms of $\Lambda^{\to}$.*

Proof. Let $\{s_i | i \in \mathbb{N}\}$ of some type τ be given such that $s_i \ {}_{\mathrm{str}}{>} \ s_{i+1}$. By Definition 36 this means that for every strict valuation α, $[\![s_i]\!]_\alpha \ {}_{\mathrm{str}}{>}_\tau \ [\![s_{i+1}]\!]_\alpha$. Let α be the valuation $\{x \mapsto S_\sigma | x \in \mathcal{V}_\sigma, \sigma \in \mathbb{T}\}$ (see Lemma 39), then α is such a strict valuation, giving rise to a descending $\ {}_{\mathrm{str}}{>}$ chain in $\mathcal{M}_\tau$, which contradicts Proposition 40. □

4.3 Contexts preserve some order

There is no difference between bound and free variables with respect to the ordering $\ {}_{\mathrm{str}}{>}$. For $\ {}_{\mathrm{mon}}{>}$ this is stated in Lemma 22.

Lemma 42. *If for two $\Lambda^{\to}$-terms $s \ {}_{\mathrm{str}}{>} \ t$ then $\lambda x.s \ {}_{\mathrm{str}}{>} \ \lambda x.t$.*

[4] See [6, p. 461] and [5, p. 83] for comparable functionals

Proof. Analogous to the proof of Lemma 22. Instead of arbitrary k and α, *strict* k and α have to be chosen. □

Now we are ready to prove that contexts preserve some order:

Proposition 43. *If $s \ {}_{\text{mon}}\!> t$ and $\mathcal{J}$ is a strict constant interpretation, then for all contexts C, $Cs \ {}_{\text{str}}\!> Ct$.*

Proof. Let $s \ {}_{\text{mon}}\!> t$ and let C be a context, then C is of the form $\lambda\Box.C'$, with $\Box \in FV(C')$. Because C' is in β-normal form, it is of the form $\lambda x_1 \cdots \lambda x_n.as_1 \cdots s_m$, with $a \in \mathcal{V} \cup \mathcal{C}$. (Note that a may be $\Box$.) So $Cs \to_\beta C'[\Box := s]$ and $Ct \to_\beta C'[\Box := t]$.

Now we prove with induction on the structure of the β-normal form of C': $\Box \in FV(C') \Rightarrow C'[\Box := s] \ {}_{\text{str}}\!> C'[\Box := t]$. Induction hypothesis (*) is: $\Box \in FV(s_i) \Rightarrow s_i[\Box := s] \ {}_{\text{str}}\!> s_i[\Box := t]$. Let α be an arbitrary strict valuation, then the following claim can be proved by induction on k:

Claim. *For $0 \leq k \leq m$, either $[\![(as_1 \cdots s_k)[\Box := s]]\!]_\alpha \ {}_{\text{mon}}\!> [\![(as_1 \cdots s_k)[\Box := t]]\!]_\alpha$, or $\Box \notin FV(as_1 \cdots s_k)$ and $[\![as_1 \cdots s_k]\!]_\alpha$ is strict.*

Before proving this claim we proceed with the main proof. We have that $\Box \in FV(C')$, so the claim yields $[\![(as_1 \cdots s_m)[\Box := s]]\!]_\alpha \ {}_{\text{mon}}\!> [\![(as_1 \cdots s_m)[\Box := t]]\!]_\alpha$. By Lemma 30 we may change ${}_{\text{mon}}\!>$ into ${}_{\text{str}}\!>$ and by Lemma 42 we may put the λ-binders in front, which yields $[\![C'[\Box := s]]\!]_\alpha \ {}_{\text{str}}\!> [\![C'[\Box := t]]\!]_\alpha$. After applying Proposition 26 on both sides we get $[\![Cs]\!]_\alpha \ {}_{\text{str}}\!> [\![Ct]\!]_\alpha$. The valuation α was an arbitrary strict one, so using Definition 29 we conclude that $Cs \ {}_{\text{str}}\!> Ct$. □

We still have to prove the claim:

Proof. (induction on k).

If $k = 0$:

- If $a = \Box$, then $[\![a[\Box := s]]\!]_\alpha = [\![s]\!]_\alpha \ {}_{\text{mon}}\!> [\![t]\!]_\alpha = [\![a[\Box := t]]\!]_\alpha$.
- If $a \in \mathcal{V} - \{\Box\}$, then $\Box \notin FV(a)$, and $[\![a]\!]_\alpha = \alpha(a)$ is strict, because α is a strict valuation.
- If $a \in \mathcal{C}$, then $\Box \notin FV(a)$, and $[\![a]\!]_\alpha = \mathcal{J}(a)$ is strict, because $\mathcal{J}$ is a strict constant interpretation.

If $k = j + 1$: We have the induction hypothesis, that either $[\![(as_1 \cdots s_j)[\Box := s]]\!]_\alpha \ {}_{\text{mon}}\!> [\![(as_1 \cdots s_j)[\Box := t]]\!]_\alpha$, or $\Box \notin FV(as_1 \cdots s_j)$ and $[\![as_1 \cdots s_j]\!]_\alpha$ is strict (**).

Firstly, $[\![s_{j+1}[\Box := s]]\!]_\alpha \ {}_{\text{mon}}\!\geq [\![s_{j+1}[\Box := t]]\!]_\alpha$ (***), as the following calculation shows:

$$\begin{array}{rll} & [\![s_{j+1}[\Box := s]]\!]_\alpha & \\ = & [\![(\lambda\Box.s_{j+1})s]\!]_\alpha & \text{by Proposition 26,} \\ = & [\![\lambda\Box.s_{j+1}]\!]_\alpha [\![s]\!]_\alpha & \\ {}_{\text{mon}}\!\geq & [\![\lambda\Box.s_{j+1}]\!]_\alpha [\![t]\!]_\alpha & \text{by Proposition 19,} \\ = & [\![(\lambda\Box.s_{j+1})t]\!]_\alpha & \\ = & [\![s_{j+1}[\Box := t]]\!]_\alpha \ . & \end{array}$$

From induction hypothesis (**) two cases can be distinguished:

If $(as_1 \cdots s_j)[\Box := s] \ {}_{\text{mon}}\!> (as_1 \cdots s_j)[\Box := t]$, then

$$\begin{array}{rl} & [\![(as_1 \cdots s_{j+1})[\Box := s]]\!]_\alpha \\ = & [\![(as_1 \cdots s_j)[\Box := s]]\!]_\alpha [\![s_{j+1}[\Box := s]]\!]_\alpha \\ {}_{\text{mon}}\!> & [\![(as_1 \cdots s_j)[\Box := t]]\!]_\alpha [\![s_{j+1}[\Box := s]]\!]_\alpha \\ {}_{\text{mon}}\!\geq & [\![(as_1 \cdots s_j)[\Box := t]]\!]_\alpha [\![s_{j+1}[\Box := t]]\!]_\alpha \text{ using (***)}, \\ = & [\![(as_1 \cdots s_{j+1})[\Box := t]]\!]_\alpha \ . \end{array}$$

In case $\Box \notin FV(as_1 \cdots s_j)$ and $[\![as_1 \cdots s_j]\!]_\alpha$ is strict, we distinguish two subcases:
$\Box \notin FV(s_{j+1})$: Then $\Box \notin FV(as_1 \cdots s_{j+1})$ and by Definition 29, we have strictness of $[\![as_1 \cdots s_j]\!]_\alpha [\![s_{j+1}]\!]_\alpha = [\![as_1 \cdots s_{j+1}]\!]_\alpha$.
$\Box \in FV(s_{j+1})$:

$$\begin{array}{rl} & [\![(as_1 \cdots s_{j+1})[\Box := s]]\!]_\alpha \\ = & [\![as_1 \cdots s_j]\!]_\alpha [\![s_{j+1}[\Box := s]]\!]_\alpha \\ {}_{\text{mon}}\!> & [\![as_1 \cdots s_j]\!]_\alpha [\![s_{j+1}[\Box := t]]\!]_\alpha \text{ using IH(*) and Definition 29}, \\ = & [\![(as_1 \cdots s_{j+1})[\Box := t]]\!]_\alpha \ . \end{array}$$

□

5 A Proof Technique for Termination

The notions of Definition 34 and 38 are collected and extended in such a way that we get the conditions for termination:

Definition 44. Let $\mathcal{R}$ be an HRS, with signature $\mathcal{F} = (\mathcal{B}, \mathcal{C}, \mathcal{V})$ and R a set of rules. We say that $\mathcal{R}$ has a *Termination Model* if there is a well-founded type interpretation $\mathcal{I}$ for $\mathcal{B}$ and a strict constant interpretation $\mathcal{J}$ for $\mathcal{C}$, such that for each rule $l \to r \in R$, $l \ {}_{\text{mon}}\!> r$.

Now we are able to state the relation between a rewrite step and the ordering of the domains:

Proposition 45. *Let an HRS $\mathcal{R}$ have a Termination Model. If $s \to_{\mathcal{R}} t$ then $s \ {}_{\text{str}}\!> t$.*

Proof. Let $s \to_{\mathcal{R}} t$. Then $s =_\eta (Cl)\downarrow_\beta$ and $t =_\eta (Cr)\downarrow_\beta$, for some context C and rule $(l \to r) \in R$. Because $\mathcal{R}$ has a Termination Model, $l \ {}_{\text{mon}}\!> r$, so by Proposition 43, $Cl \ {}_{\text{str}}\!> Cr$. Applying Proposition 26 and 27 yields: $s \ {}_{\text{str}}\!> t$. □

The main theorem of this paper uses the well-foundedness of the domains:

Theorem 46. *If an HRS $\mathcal{R}$ has a Termination Model, then $\mathcal{R}$ is terminating.*

Proof. If $\mathcal{R}$ is non-terminating, then there exists a sequence $(s_i)_{i \in \mathbb{N}}$, with $s_i \to_{\mathcal{R}} s_{i+1}$. By Proposition 45 we get an infinite descending chain $(s_i)_{i \in \mathbb{N}}$, with $s_i \ {}_{\text{str}}\!> s_{i+1}$. This is impossible because of the well-foundedness of ${}_{\text{str}}\!>$ (Proposition 41). Thus $\mathcal{R}$ must be terminating. □

The following is a recollection of the conditions that occur in the notion of a Termination Model. Theorem 46 suggests the following proof technique for the termination of an HRS:

- Find convenient partial orderings $\mathcal{I}(B)$ for each base type B, satisfying:
  - $\mathcal{I}(B)$ is non-empty.
  - $\mathcal{I}(B)$ is well-founded.
  - There exist strict functions with type $\mathcal{I}(a) \to \mathcal{I}(b) \to \mathcal{I}(b)$ for all combinations (a, b) of base types. (By Lemma 33 strict $_{\text{mon}}>$-monotonicity suffices).
- Find a convenient strict interpretation $\mathcal{J}(c)$ for each constant symbol $c \in \mathcal{C}$.
- Prove that for each rule $(l \to r)$ in the system the interpretation of the left hand side is greater than the interpretation of the right hand side, in symbols $[\![l]\!] \,_{\text{mon}}> [\![r]\!]$.

In the next section the applicability of this proof method is shown.

6 Applications

6.1 Process Algebra

The first application comes from Process Algebra, or better an extension of it: μCRL [7]. We only concentrate on the fragment of Process Algebra with choice ($+$), sequential composition ($\cdot$) and deadlock (δ) and the data dependent choice (Σ) from μCRL. The Process Algebra part can be formulated in a first order Term Rewriting System (see for instance [1]). The rules for the Sum-operator require higher-order rewrite rules to deal with the bound variables. This reformulation of μCRL can be found in [16, p. 33].

There are two base types: $\{\mathsf{Proc}, \mathsf{Data}\}$. Furthermore, here is a list of function symbols with their types:

$$\begin{array}{ll} + : \mathsf{Proc} \to \mathsf{Proc} \to \mathsf{Proc} & \delta : \mathsf{Proc} \\ \cdot : \mathsf{Proc} \to \mathsf{Proc} \to \mathsf{Proc} & \Sigma : (\mathsf{Data} \to \mathsf{Proc}) \to \mathsf{Proc} \end{array}$$

$\{X, Y, Z, P, Q, D\}$ are used as free variables. Now we have the following set of rules, with the binary function symbols written infix:

$$\begin{array}{ll} \text{A3:} & X + X \to X \\ \text{A4:} & (X + Y)\cdot Z \to (X\cdot Z) + (Y\cdot Z) \\ \text{A5:} & (X\cdot Y)\cdot Z \to X\cdot(Y\cdot Z) \\ \text{A6:} & X + \delta \to X \\ \text{A7:} & \delta\cdot X \to \delta \\ \text{Sum1:} & \Sigma(\lambda d : \mathsf{Data}.X) \to X \\ \text{Sum3:} & (\Sigma P) + (P D) \to (\Sigma P) \\ \text{Sum4:} & \Sigma(\lambda d : \mathsf{Data}.((Pd) + (Qd))) \to (\Sigma P) + (\Sigma Q) \\ \text{Sum5:} & (\Sigma P)\cdot X \to \Sigma(\lambda d : \mathsf{Data}.((Pd)\cdot X)) \end{array}$$

To prove termination of this system we interpret both base types Data and Proc by $\mathbb{N}_{\geq 1}$, with the usual ordering. This is a domain, because it is well-founded, non-empty and there exists a binary strictly monotonic function, ordinary "+" for instance. The function symbols are interpreted in the following way:

$$[\![+]\!] = \boldsymbol{\lambda} a.\boldsymbol{\lambda} b.a + b + 1 \qquad [\![\delta]\!] = 1$$
$$[\![\cdot]\!] = \boldsymbol{\lambda} a.\boldsymbol{\lambda} b.a \times b + a \qquad [\![\Sigma\,]\!] = \boldsymbol{\lambda} f.3 \times f(1) + 1$$

This is an extension of the interpretation in [1] for the Process Algebra part of the system. The type level of all these function symbols is bounded by 2, so it is sufficient to check if the interpretations for them are strictly $_{\text{mon}}>$-monotonic (Lemma 33). The first three functions are clearly strictly monotonic. The last is also strictly $_{\text{mon}}>$-monotonic: Take $f \;_{\text{mon}}> g$, then $f(1) \;_{\text{mon}}> g(1)$. But then also $3 \times f(1) + 1 \;_{\text{mon}}> 3 \times g(1) + 1$. Now we compute the values of the left hand sides and right hand sides.

interpretation of the left hand side	interpretation of the right hand side
$2 \times [\![X]\!] + 1$	$[\![X]\!]$
$([\![X]\!] + [\![Y]\!] + 1) \times [\![Z]\!] + [\![X]\!] + [\![Y]\!] + 1$	$[\![X]\!] \times [\![Z]\!] + [\![X]\!] + [\![Y]\!] \times [\![Z]\!] + [\![Y]\!] + 1$
$[\![X]\!] \times ([\![Y]\!] \times [\![Z]\!] + [\![Z]\!] + [\![Y]\!] + 1)$	$[\![X]\!] \times ([\![Y]\!] \times [\![Z]\!] + [\![Y]\!] + 1)$
$[\![X]\!] + 2$	$[\![X]\!]$
$[\![X]\!] + 1$	1
$3 \times [\![X]\!] + 1$	$[\![X]\!]$
$3 \times [\![P]\!](1) + [\![P\,D]\!] + 2$	$3 \times [\![P]\!](1) + 1$
$3 \times ([\![P]\!](1) + [\![Q]\!](1])) + 4$	$3 \times ([\![P]\!](1) + [\![Q]\!](1)) + 3$
$3 \times [\![P]\!](1) \times [\![X]\!] + [\![X]\!] + 3 \times [\![P]\!](1) + 1$	$3 \times [\![P]\!](1) \times [\![X]\!] + 3 \times [\![P]\!](1) + 1$

The reader can verify that the interpretation of the left hand side is greater than the interpretation on the right hand side on each line in the table. So this system of Process Algebra- and Sum rules is terminating.

6.2 Quantifier reasoning

In [12] some HRSs concerning first order predicate logic are presented as an example. One of them is called *mini scoping*, pushing quantifiers inwards. The base types are $\{\mathsf{Term}, \mathsf{Form}\}$. The function symbols are $\{\vee, \wedge : \mathsf{Form} \to \mathsf{Form} \to \mathsf{Form};\ \forall : (\mathsf{Term} \to \mathsf{Form}) \to \mathsf{Form}\}$, the type level is at most 2.

The rules (with free variables $\{P, P', Q, Q'\}$) are:

$$\forall(\lambda x.P) \to P$$
$$\forall(\lambda x.((P'x) \wedge (Q'x))) \to (\forall P') \wedge (\forall Q')$$
$$\forall(\lambda x.((P'x) \vee Q)) \to (\forall P') \vee Q$$
$$\forall(\lambda x.(P \vee (Q'x))) \to P \vee (\forall Q')$$

Again we take as interpretation for both base types the positive natural numbers $\mathbb{N}_{\geq 1}$. The interpretation of the function symbols is as follows:

$$[\![\wedge]\!] = [\![\vee]\!] = \boldsymbol{\lambda} a.\boldsymbol{\lambda} b.a + b + 1$$
$$[\![\forall]\!] = \boldsymbol{\lambda} f.2 \times f(1)$$

It is easily verified that under this interpretation the left hand side of each rule is greater than the interpretation of the right hand side.

6.3 Surjective Disjoint Union

We give one more example. The signature is given by the only base type Term, the function symbols:

$$\begin{aligned} case &: \mathsf{Term} \to (\mathsf{Term} \to \mathsf{Term}) \to (\mathsf{Term} \to \mathsf{Term}) \to \mathsf{Term} \\ inl, inr &: \mathsf{Term} \to \mathsf{Term} \end{aligned}$$

All function symbols have level ≤ 2. The free variables are $\{X, F, G, U\}$. The rules are:

$$\begin{aligned} case(inl(X), F, G) &\to F(X) \\ case(inr(X), F, G) &\to G(X) \\ case(U, \lambda x.F(inl(x)), \lambda x.F(inr(x))) &\to F(U) \end{aligned}$$

Note that this example does not fit in the framework of [12] (see page 347). Termination for this example is less trivial, because there is a real application in the interpretation of the function symbols. Furthermore it is not the case that the number of "case" occurrences decreases in every step: If X contains a "case" occurrence, then F can generate many copies of it in the right hand side of the first rule.

But the interpretation in a Termination Model is easy: Take $\mathcal{I}(\mathsf{Term}) = \mathbb{N}_{\geq 1}$. Furthermore, interpret:

$$\begin{aligned} [\![case]\!] &= \boldsymbol{\lambda} a.\boldsymbol{\lambda} f.\boldsymbol{\lambda} g.f(a) + g(a) + a \\ [\![inl]\!] = [\![inr]\!] &= \boldsymbol{\lambda} a.a + 1 \end{aligned}$$

These functions are strict: we only need to take into account monotonic functions for f and g. Take $a \ {}_{\text{mon}}{>}\ b$. By monotonicity of f we have $f(a) \ {}_{\text{mon}}{\geq}\ f(b)$ and the same holds for g. But then $f(a) + g(a) + a \ {}_{\text{mon}}{>}\ f(b) + g(b) + b$. Furthermore, the interpretations of the left- and right hand sides can be computed:

Left hand side	Right hand side
$[\![F]\!]([\![X]\!]+1) + [\![G]\!]([\![X]\!]+1) + [\![X]\!] + 1$	$[\![F]\!]([\![X]\!])$
$[\![F]\!]([\![X]\!]+1) + [\![G]\!]([\![X]\!]+1) + [\![X]\!] + 1$	$[\![G]\!]([\![X]\!])$
$2 \times [\![F]\!]([\![U]\!]+1) + [\![U]\!]$	$[\![F]\!]([\![U]\!])$

The left hand sides are all greater than the right hand sides, because we may restrict to monotonic functionals for F and G. So the system of "surjective disjoint union" is terminating.

References

1. G.J. Akkerman and J.C.M. Baeten. Term rewriting analysis in process algebra. Technical Report CS-R9130, Centre for Mathematics and Computer Science, June 1991.
2. H.P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North-Holland, Amsterdam, second, revised edition, 1984.
3. M. Bezem and J.F. Groote, editors. *Proceedings of the 1^{st} International Conference on Typed Lambda Calculi and Applications, TLCA '93,* Utrecht, The Netherlands, volume 664 of *Lecture Notes in Computer Science.* Springer-Verlag, 1993.

4. V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings* 3^{th} *Annual Symposium on Logic in Computer Science,* Edinburgh, pages 82–90, July 1988.
5. R. de Vrijer. *Surjective Pairing and Strong Normalization: Two Themes in Lambda Calculus.* PhD thesis, University of Amsterdam, 1987.
6. R.O. Gandy. Proofs of strong normalization. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.
7. J.F. Groote and A. Ponse. The syntax and semantics of μCRL. Report CS-R9076, CWI, Amsterdam, 1990.
8. G. Huet and D.C. Oppen. Equations and rewrite rules: A survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, 1980.
9. J.P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, The Netherlands, pages 350–361, 1991.
10. Z. Khasidashvili. Perpetual reductions and strong normalization in orthogonal term rewriting systems. Technical Report CS-R9345, CWI, Amsterdam, July 1993.
11. J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts.* Mathematisch Centrum, Amsterdam, 1980.
12. T. Nipkow. Higher-order critical pairs. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, The Netherlands, pages 342–349, 1991.
13. T. Nipkow. Orthogonal higher-order rewrite systems are confluent. In Bezem and Groote [3], pages 306–317.
14. C. Paulin-Mohring. Inductive definitions in the system Coq. Rules and properties. In Bezem and Groote [3], pages 328–345.
15. L. C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press Limited, London, 1990. In: The APIC-series 31.
16. M.P.A. Sellink. Verifying process algebra proofs in type theory. Technical Report 87, Logic Group Preprint Series, Utrecht University, March 1993.
17. V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. Technical Report IR-333, Vrije Universiteit Amsterdam, August 1993. Appears in this Volume.
18. D.A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge tracts in theoretical computer science.* Cambridge University Press, Cambridge, 1993.
19. H. Zantema. Termination of term rewriting by interpretation. In M. Rusinowitch and J.L. Rémy, editors, *Conditional Term Rewriting Systems, proceedings third international workshop CTRS-92*, volume 656 of *Lecture Notes in Computer Science*, pages 155–167. Springer-Verlag, 1993. Full version appeared as report RUU-CS-92-14, Utrecht University.

This article was processed using the LaTeX macro package with LLNCS style