Towards a Domain Theory for Termination Proofs

Stefan Kahrs*

University of Edinburgh
Laboratory for Foundations of Computer Science
King's Buildings, EH9 3JZ
email: `smk@dcs.ed.ac.uk`

Abstract

We present a general framework for termination proofs for Higher-Order Rewrite Systems. The method is tailor-made for having simple (and modular) proofs showing the termination of enriched λ-calculi.

1 Introduction

The termination problem for term rewriting systems is in general undecidable. Thus there cannot be a general method to show the termination of rewriting, or can it?

The answer to that question depends on how one interprets the word *method*. Clearly the answer is "No!" if *method* is understood to mean *algorithm*. But if not all parts of a method are required to be algorithmic, such a method might still be powerful enough to cover the entire problem. Methods of this kind are typically and almost necessarily semantical.

A semantical method for termination proofs of first-order term rewriting systems has been presented by Zantema at the CTRS'92 workshop [32]. Jaco van de Pol extended this method to higher-order term rewriting systems [27]. However, his extension has certain drawbacks; in particular it is ill-suited to show termination for many enriched λ-calculi, as we shall point out later in more detail. This restriction is extremely annoying, because enriched λ-caluli are the most interesting HRSs.

The reason for many of the technical problems which occur when one tries to establish a semantical method for HRS termination proofs is the meta-level of HRSs. HRS rewrite steps are between $=_{\beta\eta}$-equivalence classes of terms of the simply typed λ-calculus $\lambda^{\rightarrow}$. Any semantic interpretation has to assign the same values to any member of such a class. This typically means to interpret a function type $\sigma \rightarrow \tau$ as a suitable subset of functions between $[\![\sigma]\!]$ and $[\![\tau]\!]$, making it possible to interpret syntactic application and abstraction as semantic application and abstraction, respectively. The trouble is: for $\lambda^{\rightarrow}$ these objectives are incompatible. The most suitable subset restriction for functions is the restriction to monotonic functions, monotonic w.r.t. some well-ordering. This would imply that context application preserves termination. However, variable

*The research reported here was partially supported by SERC grant GR/J07303.

abstraction can introduce non-monotonic functions: constant functions like $\lambda x.c$ are not monotonic w.r.t. strict orderings.

Van de Pol's approach [27] maintains the canonical interpretation of λ-terms, i.e. interpreting $\beta\eta$-equivalent terms equally. This means that non-monotonic functions appear in the semantics and have to be dealt with. The central idea behind in van de Pol's paper is to distinguish between the semantic domains for terms and contexts. The practical problem with this idea is that (i) the semantic domain for contexts is rather involved, (ii) function symbols are contexts, and (iii) interpretations are determined by the interpretation of function symbols. In other words: the user of the method is confronted with the complicated semantic domain for contexts. To be practicable, van de Pol's method needs an extension hiding the semantic domain of contexts.

Years before higher-order rewrite systems were invented, Robin Gandy used a different semantic approach for showing termination of enriched λ-calculi [7]. Gandy achieved that all terms are interpreted by monotonic functions by sacrificing the other objective of mirroring abstraction and application semantically. In particular, a syntactic term $\lambda x.c$ is not interpreted as the constant function $(x \mapsto c)$, but instead as a function $(x \mapsto c + L(x))$, where L is some monotonic type conversion function, and where $+$ is some appropriate addition on the result type. The choices of the mentioned addition and type conversions look a bit arbitrary when one glances at Gandy's paper and he gives no criteria how to obtain these $+$ and L operations in general; however, they naturally arise from a categorical semantics, as we shall see later.

The present paper is an extension of Gandy's approach in several respects. The main aim is to establish the semantic foundation of the method, and indeed of other methods as well — the section on the semantics is abstract enough to cover a variety of approaches towards termination. A direct consequence of operating on this abstract level is a generalisation from natural numbers (used in Gandy's interpretation) to arbitrarily large ordinals. In the second part, Higher-Order Rewrite Systems are linked with the semantic foundations by providing an appropriate interpretation of λ-terms. Based on this interpretation we obtain a recipe for showing termination of particular rewrite systems.

2 Outline

The technical content we have to master is quite considerable and so it seems worthwhile to explain the underlying ideas behind the approach before we enter the technical stage.

The first main idea is: find a semantic interpretation $[\![\_]\!]$ of terms (pointwise for each symbol) such that $t \to_R u$ implies $[\![t]\!] > [\![u]\!]$ where $>$ is a terminating relation. The termination of $\to_R$ (trivially) implies the existence of such an interpretation, and the existence of such an interpretation implies the termination of $\to_R$. This means: we interpret each type as an object in the category of (partially) well-ordered sets which we shall call WO. Notice that even in an "untyped" HRS we still have — besides the object-level type o — all meta-level types $o \to o \to o$, etc.

When we are trying to find an interpretation for a concrete rewrite system, we can start looking at the rules: if the above should succeed then in particular we need $[\![l]\!] > [\![r]\!]$ for every rule $l \to r$. However: it should be sufficient to look at the rules only to establish a termination proof, i.e. all term-building operations should preserve the termination ordering. Thus, the second main idea is to interpret function symbols as morphisms in WO — these are exactly the functions that preserve the strict termination order $>$.

For a given rule $l \to r$, the motto is: find an interpretation for l which is more or less the same as $[\![r]\!]$, but "add 1" (or more) to it. This requires a slight departure from traditional semantic thinking in which both sides of a rule would be typically given the same value. Example:

$$\begin{array}{lcr} \mathtt{Add}(\mathtt{0}, x) & \to & x \\ \mathtt{Add}(\mathtt{S}(x), y) & \to & \mathtt{S}(\mathtt{Add}(x, y)) \end{array}$$

This is an addition operation for natural numbers. We can interpret the (only) type as the natural numbers with their usual ordering, $\mathtt{0}$ as 0 and $\mathtt{S}$ as $+1$ and $\mathtt{Add}$ as $\lambda x \lambda y. x + x + y + 1$ and can easily compute that both rules decrease the interpretation of a term. Considering the interpretation of $\mathtt{Add}$, we can see that we need the $+1$ for the first rule and the extra $x+$ for the second rule.

We were fortunate here that the interpreted type provided us with an arithmetic to add 1 and an extra copy of x to the result. We cannot always expect to have that much structure in our semantic domain. In particular, it would be nice if data structures such as pairs, lists, function types, etc. could be semantically interpreted as pairs, lists, function types, etc. and only the (interpretation of) defined functions operating on these data would add the extra content. This would allow us to interpret constructors as constructors and focus our attention on defined symbols[1]. We can see this more clearly with another example:

$$\begin{array}{lcr} \mathtt{Length}(\mathtt{Nil}) & \to & \mathtt{0} \\ \mathtt{Length}(\mathtt{Cons}(x, y)) & \to & \mathtt{S}(\mathtt{Length}(y)) \end{array}$$

How can we find an interpretation that shows termination for this example? If we interpret syntactic lists as semantic lists then we could give $\mathtt{Length}$ the meaning $\lambda x.\, 2 * \#x + 1$, where $\#x$ is just the length of the list x. With this interpretation, the left-hand sides have larger values than the right-hand sides. Again, the $+1$ arises from the first rule and the extra copy of $\#x$ is needed because of the second rule. However, this interpretation is not good enough, because $[\![\mathtt{Length}]\!]$ is not monotonic; the contents of the lists is ignored, in particular the possible presence of redexes in list components. We have to give the terms $\mathtt{Length}(\mathtt{Cons}(\mathtt{Add}(\mathtt{0}, \mathtt{0}), \mathtt{Nil}))$ and $\mathtt{Length}(\mathtt{Cons}(\mathtt{0}, \mathtt{Nil}))$ different interpretations because the former rewrites to the latter. Suppose we have a function $\mathtt{out} : \mathtt{list} \to \mathtt{nat}$ which is order-preserving, then we can interpret $\mathtt{Length}$ as $\lambda x.\, 2 * \#x + 1 + \mathtt{out}(x)$ and this interpretation indeed shows the termination of our example, provided $\mathtt{out}([\![\mathtt{Cons}]\!](x, y)) \geq \mathtt{out}(y)$. The proviso is true if we define the function $\mathtt{out}$ on lists by $\mathtt{out}[x_1, \ldots, x_n] = \mathtt{out}(x_1) + \cdots + \mathtt{out}(x_n)$.

The need for such $\mathtt{out}$ functions arises for all erasing rules, i.e. rules which delete a subterm, since this subterm could have contained arbitrary redexes and we have to distinguish the interpretations before and after redex contraction. In general, the codomain of $\mathtt{out}$ should be an ordinal large enough to prove termination for the given rewrite system, i.e. the natural numbers will not always suffice. For allowing erasing rules of arbitrary type τ we need order-preserving functions $\mathtt{out}_\tau : \tau \to \alpha$ for some appropriately large ordinal α. But this is only half the story, because the result type of $\mathtt{Length}$ happened to be numbers. For the general picture, consider the following rule:

$$\mathtt{Tl}(\mathtt{Cons}(x, y)) \to y$$

[1] The distinction between constructors and defined symbols applies to arbitrary TRS, see [24].

Using the outlined principle of how to deal with erasing rules, the interpretation of `Tl` should be: $\lambda y.\, \texttt{out}(\texttt{hd}(y)) + 1 + \texttt{tl}(y)$ where `hd` and `tl` are semantic "head" and "tail" functions. But this does not quite make sense yet: following our principle of interpreting lists by lists, $\texttt{tl}(y)$ is a list and we have added numbers to it! What we need is an operation $\boxplus_\tau : \alpha \times \tau \to \tau$ which allows us to add numbers (ordinal numbers in α) on top of values of arbitrary types. This function should be order-preserving have the property $n \boxplus x \geq x$. This is the third main idea, the core part of the section on semantics: providing the operations $\boxplus_\tau$ and $\texttt{out}_\tau$ for arbitrary types.

Another technical problem we have to master is the interpretation of the typed λ-calculus that serves as the meta-level for HRS rewriting. HRS rewriting operates on equivalence classes of λ-terms, the equivalence relation being β-conversion. Hence it would be convenient to give β-convertible terms the same semantic interpretation. However, this goal clashes with the earlier mentioned objective that the term-forming operations should be order-preserving in WO, because $\lambda^\to$ contains terms like $\lambda x.c$ that could not be interpreted as monotonic functions in such a setting. Instead, we consider HRS rewriting as operating on β-normal forms — this is not a restriction since each β-equivalence class in $\lambda^\to$ contains exactly one normal form. We can then give an interpretation of $\lambda^\to$ within WO (using the operations $\boxplus_\tau$ and $\texttt{out}_\tau$), i.e. each λ-abstraction will be interpreted as a monotonic function. This is the fourth main idea.

The inconvenient side of using β-normal forms as the objects of rewriting is that a rule application may introduce β-redexes which in our setting have to be contracted to get the "real" semantic values. Thus, we have to consider in general the combined effect of β-reduction and semantic interpretation of a rule instance. Very often applications of rewrite rules only create new β-redexes on the right-hand sides (if at all); for such rules it is sufficient to look solely at the semantic interpretation, because we have set up the interpretation of λ-terms in such a way that β-reduction can only decrease the value. Most functional programs can easily be translated into HRSs that have this property. But even if rule-application creates β-redexes on left-hand sides we can still make termination proofs with this method, though it requires reasoning about substitutions. In the section that briefly explains the application of the method we give an example involving such a rule, the η-rule for coproducts.

3 Preliminaries

An *Abstract Reduction System* (short: ARS) consists of a set A and a binary relation $\to$ on A. We write $\boldsymbol{A} \models P$ if the ARS $\boldsymbol{A} = (A, \to)$ has the property P. Given an ARS $\boldsymbol{A} = (A, \to_A)$, $t \in A$ is a *normal form* if $\neg\exists u \in A.\ t \to_A u$. We write $\mathrm{NF}_{\mathcal{A}}$ for the set of normal forms of $\mathcal{A}$. t *has* a normal form $u \in \mathrm{NF}_{\mathcal{A}}$ if $t \twoheadrightarrow_A u$. An ARS $\boldsymbol{A} = (A, \to)$ is *strongly normalising*, $\boldsymbol{A} \models \mathrm{SN}$, iff there is no non-empty relation R on A satisfying the equation $R \ = \ \to; R$, i.e. iff there are no infinite chains of $\to$-steps.

We do not have the room to introduce the concepts and terminology we borrow from category theory to make this paper self-contained. The reader can find the missing definitions in most standard works on category theory, e.g. in [20] or [26] or [3]. Knowing (and understanding) some amount of category is necessary to fully appreciate the section on the semantics. Even without such knowledge it is still possible to understand the syntactic part of the approach presented here, as it merely rests on one semantic result, theorem 35.

Here is a brief summary defining some of the non-elementary terminology.

Given a category $\mathcal{A}$ and an object $X \in |\mathcal{A}|$, the *comma* category $\mathcal{A} \downarrow X$ has as objects pairs (Y, f), such that $Y \in |\mathcal{A}|$ and $f \in \mathcal{A}(Y, X)$, and a morphism $m \in (\mathcal{A} \downarrow X)((Y, g), (Z, h))$ is a morphism $m \in \mathcal{A}(Y, Z)$ such that $h \circ m = g$. There is a forgetful functor $U : \mathcal{A} \downarrow X \to \mathcal{A}$ defined as $U(X, f) = X$ and $U(m) = m$. Dually, $X \downarrow \mathcal{A}$ is defined as $(\mathcal{A}^{op} \downarrow X)^{op}$.

A *monad* in a category $\mathcal{A}$ is a triple $(\boldsymbol{T}, \mu, \eta)$ where $\boldsymbol{T} : \mathcal{A} \to \mathcal{A}$ is a functor and $\mu : \boldsymbol{T}(\boldsymbol{T}(\_)) \to \boldsymbol{T}(\_)$ and $\eta : \_ \to \boldsymbol{T}(\_)$ are natural transformations satisfying $\mu \circ \eta = id$ and $\mu \circ \mu = \mu \circ \boldsymbol{T}(\mu)$. Given such a monad, the category $\mathcal{A}^{\boldsymbol{T}}$ of $\boldsymbol{T}$-algebras has as objects pairs (X, f), such that $X \in |\mathcal{A}|$, $f \in \mathcal{A}(\boldsymbol{T}(X), X)$, $f \circ \mu = f \circ \boldsymbol{T}(f)$ and $f \circ \eta = id$; a morphism $g \in \mathcal{A}^{\boldsymbol{T}}((X, f), (X', f'))$ is a morphism $g \in \mathcal{A}(X, X')$ such that $g \circ f = f' \circ \boldsymbol{T}(g)$. There is a forgetful functor $U : \mathcal{A}^{\boldsymbol{T}} \to \mathcal{A}$ defined as $U(X, f) = X$ and $U(g) = g$.

Let $\mathcal{A}$ and $\mathcal{B}$ be categories and $U : \mathcal{A} \to \mathcal{B}$ be a functor. A functor $F : \mathcal{B} \to \mathcal{B}$ *can be lifted along* U if there is a functor $F' : \mathcal{A} \to \mathcal{A}$ such that $F \circ U = U \circ F'$. If $\mathcal{A} = \mathcal{B}^{\boldsymbol{T}}$ we omit the "along U" and understand U to be the forgetful functor U of the monad. Similarly for $\mathcal{A} = \mathcal{B} \downarrow X$.

A *monoidal category* is a tuple $(\mathcal{A}, \otimes, I, a, l, r)$ where $\mathcal{A}$ is a category, $\otimes$ is a functor $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$, I is an object in $\mathcal{A}$, and a, l, and r are natural isomorphisms $a : A \otimes (B \otimes C) \cong (A \otimes B) \otimes C$, $l : I \otimes A \cong A$, $r : A \otimes I \cong A$ (natural in A, B and C) such that "all coherence diagrams" commute, i.e. all diagrams only involving the isomorphisms and $\otimes$. A monoidal category is *symmetric* if there is also a natural isomorphism $s : A \otimes B \cong B \otimes A$ maintaining the property that all coherence diagrams commute. A (symmetric) monoidal category is called *closed* if the functor $\_ \otimes A$ has a right adjoint $A \Rightarrow \_$ for any $A \in |\mathcal{A}|$. We write $ap : (A \Rightarrow \_) \otimes A \to \_$ for the co-unit of the adjunction and $cur(f)$ as shorthand for $(id \Rightarrow f) \circ \eta$ where $\eta : \_ \to A \Rightarrow (\_ \otimes A)$ is the unit of the adjunction.

A *monoid* in a monoidal category $(\mathcal{A}, \otimes, I, a, l, r)$ is a triple $(X, \oplus, 0)$, where $X \in |\mathcal{A}|$, $\oplus \in \mathcal{A}(X \otimes X, X)$, and $0 \in \mathcal{A}(I, X)$, such that $\oplus \circ (0 \otimes id) \circ l^{-1} = \oplus \circ (id \otimes 0) \circ r^{-1} = id$ and $\oplus \circ (\oplus \otimes id) = \oplus \circ (id \otimes \oplus) \circ a^{-1}$. The *representation monad* of a monoid is the monad $(X \otimes \_, \mu, \eta)$, where $\mu = (\oplus \otimes id) \circ a$ and $\eta = (0 \otimes id) \circ l^{-1}$. If the monoidal category $\mathcal{A}$ is just sets with some structure (as it will be throughout this paper) then a monoid in $\mathcal{A}$ is just an ordinary monoid (i.e. in the category of sets) with the additional property that $\oplus$ preserves the additional structure of $\mathcal{A}$-objects.

Throughout this paper, all categories we are dealing with have sets with structure as objects and their homomorphisms as morphisms. Monoids in such categories are just ordinary monoids with the additional restriction that $\oplus$ and 0 are structure-preserving.

4 Semantics

For proving termination in a semantical setting, we want to interpret types by (partially) well-ordered sets. To make this general enough to cater for typed rewrite systems with non-elementary types like products, coproducts, or function types, we have to develop something like a domain theory for well-orderings.

Definition 1. A *partially well-ordered set* is an ARS $\boldsymbol{A} = (A, >_A)$ such that $\boldsymbol{A} \models \mathrm{SN}$ and that $>_A$ is transitive. Convention: we shall write $\geq_A$ for the reflexive closure of $>_A$ on A. If $\geq_A$ is a total order, we call $\boldsymbol{A}$ a *well-ordered set*.

4.1 The category of well-ordered sets

Definition 2. The category WO is defined as follows:

Objects: $\boldsymbol{A} = (A, >_A) \in |\mathsf{WO}|$ if $\boldsymbol{A}$ is a partially well-ordered set.

Morphisms: a morphism $f : \boldsymbol{A} \to \boldsymbol{B}$ is a function $f : A \to B$ satisfying
$\forall a, a' \in A.\ a >_A a' \Rightarrow f(a) >_B f(a')$.

Composition and identities: as in Set, the category of sets.

We write TWO for the full subcategory of WO that only contains well-ordered sets.

Suppose we have an enriched λ-calculus with non-elementary types such as $\sigma \to \tau$, $\sigma \times \tau$, etc. To find well-ordered structures (objects in WO) for these composite types means to find the corresponding endofunctors on WO. Notice that WO has no terminal object, i.e. it is not cartesian. With a terminal object one can express constant functions (those that factor through the terminal object), but constant functions do not preserve strict orders such as any (non-empty) well-order $>$ in WO.

Definition 3. We define two summation functors on WO, categorical sum $\mathsf{WO} \sqcup \mathsf{WO} \to \mathsf{WO}$ and ordinal sum $\mathsf{WO} + \mathsf{WO} \to \mathsf{WO}$ as follows: let $\boldsymbol{A} = (A, >_A)$ and $\boldsymbol{B} = (B, >_B)$, then

- $\boldsymbol{A} \sqcup \boldsymbol{B} = (A \times \{0\} \cup B \times \{1\}, >_{A \sqcup B})$ such that $(a, 0) >_{A \sqcup B} (a', 0) \iff a >_A a'$ and $(b, 1) >_{A \sqcup B} (b', 1) \iff b >_B b'$. On morphisms, $f \sqcup g$ is $f + g$ from Set.
- $\boldsymbol{A} + \boldsymbol{B}$ is defined just as $\boldsymbol{A} \sqcup \boldsymbol{B}$, except for the order:

$$(x, n) >_{A+B} (x', n') \iff (x, n) >_{A \sqcup B} (x', n') \ \vee \ n > n'$$

The notation $\boldsymbol{A} \sqcup \boldsymbol{B}$ for the coproduct is motivated by the observation that the order type of $\alpha \sqcup \beta$ (for ordinals α and β) is just the ordinal $\alpha \cup \beta$, i.e. the maximum of the two. It is routine to check that the categorical sum is indeed a coproduct in the sense of category theory.

The ordinal sum is associative (modulo isomorphisms) but not commutative.

Definition 4. We define three multiplication functors, categorical product $\mathsf{WO} \sqcap \mathsf{WO} \to \mathsf{WO}$, ordinal product $\mathsf{WO} * \mathsf{WO} \to \mathsf{WO}$, and symmetric product $\mathsf{WO} \otimes \mathsf{WO} \to \mathsf{WO}$ as follows: let $\boldsymbol{A} = (A, >_A)$ and $\boldsymbol{B} = (B, >_B)$, then

- $\boldsymbol{A} \sqcap \boldsymbol{B} = (A \times B, >_{A \sqcap B})$ such that $(a,b) >_{A \sqcap B} (a',b') \iff a >_A a' \ \wedge \ b >_B b'$. On morphisms, $f \sqcap g$ is $f \times g$ from Set.
- $\boldsymbol{A} * \boldsymbol{B}$ is defined just as $\boldsymbol{A} \sqcap \boldsymbol{B}$, except for the order:

$$(a,b) >_{A*B} (a',b') \quad \iff \quad a >_A a' \ \vee \ (a \geq_A a' \wedge b >_B b')$$

- Similarly, $\boldsymbol{A} \otimes \boldsymbol{B}$ is defined just as $\boldsymbol{A} \sqcap \boldsymbol{B}$, except for the partial well-order $>_{A \otimes B}$:

$$(a,b) >_{A \otimes B} (a',b') \iff a >_A a' \ \wedge \ b \geq_B b' \ \vee \ a \geq_A a' \ \wedge \ b >_B b'.$$

On first view, one might think that the multiplication functors differ only in minor details, but these details are quite significant. Writing bold numbers for the objects corresponding to ordinals in WO we have for example: $\mathbf{3} \sqcap \mathbf{7} \approx \mathbf{3}$, $\mathbf{3} * \mathbf{7} \approx \mathbf{21}$, and $\mathbf{3} \otimes \mathbf{7} \approx \mathbf{9}$. Here, $\boldsymbol{A} \approx \boldsymbol{\alpha}$ is used to mean that there are morphisms $\boldsymbol{\alpha} \xrightarrow{f} \boldsymbol{A} \xrightarrow{g} \boldsymbol{\alpha}$. So we characterise an object $\boldsymbol{A} \in |\mathsf{WO}|$ by the longest chains it includes.

One can define the relation $\approx$ on any category, it is always an equivalence relation between objects. In many categories it is a rather trivial relation (e.g. two equivalence classes in Set), but in WO we have one equivalence class for every ordinal[2]; we could even require $g \circ f = id$ without changing the relation, the inverse holds in TWO where $\approx$ coincides with $\cong$.

Proposition 5. $(\mathsf{WO}, \otimes, \mathbf{1})$ *is a symmetric monoidal category, where* $\mathbf{1} = (\{0\}, \emptyset)$.

Proof. The obvious underlying isomorphisms in Set preserve the orders, i.e. they are morphisms in WO. $\square$

We also have such results for the categorical product and for the ordinal product, but each with one restriction. The categorical product is associative and commutative (modulo isomorphisms), but it lacks a neutral element, essentially because of the Burali-Forti paradox. The ordinal product is associative and has a neutral element, but it cannot be symmetric as there is no isomorphism (monotonic bijection) between $\boldsymbol{\omega} * \mathbf{2}$ and $\mathbf{2} * \boldsymbol{\omega}$.

Proposition 6. *The functor* $\_ \otimes A$ *has a right adjoint* $A \Rightarrow \_$, *i.e.* WO *is monoidal closed.*

Proof. We define the functor $\_ \Rightarrow \_ : \mathsf{WO}^{op} \times \mathsf{WO} \to \mathsf{WO}$ by
$(A, >_A) \Rightarrow (B, >_B) = (A \Rightarrow B, >_{A \Rightarrow B})$ with:

$$\begin{array}{lcl} A \Rightarrow B & = & \{f \in A \to B \mid \forall x, y \in A.\ (x >_A y \Rightarrow f(x) >_B f(y))\} \\ f >_{A \Rightarrow B} g & \iff & \forall x \in A.\ f(x) >_B g(x) \end{array}$$

We have to slightly amend this definition in case A is the empty set: $>_{\emptyset \Rightarrow B} = \emptyset$, the empty relation is SN. If A is non-empty, it has an element $a \in A$, and each infinite chain $f_1 >_{A \Rightarrow B} f_2 >_{A \Rightarrow B} \ldots$ can be mapped to an infinite chain $f_1(a) >_B f_2(a) >_B \ldots$ in B. On morphisms, we have as usual $(f \Rightarrow g)(h) = g \circ h \circ f$. Checking the adjunction properties is routine. $\square$

[2] These two properties depend on presence of the axiom of choice.

There is an analogous right adjoint to $\_ * A$, but it seems much less useful for practical considerations, because its ordering on functions is too restrictive: $f > g \iff \forall x, y \in A.\ f(x) >_B g(y)$. The other ordinal multiplication functor $A * \_$ has no right adjoint, because we would need to infer $f(a) > g(a')$ solely from $a > a'$. Concerning the categorical product, $\_ \sqcap A$ does not have a right adjoint either; currying is not monotonic, i.e. we do not have $a >_A a' \Rightarrow (b \mapsto f(a,b)) >_{B \to C} (b \mapsto f(a',b))$ if f is a morphism $f : A \sqcap B \to C$. With $A \otimes B$ we do not have this problem, as $(a,b) >_\otimes (a',b)$.

Notice also the slight technical problem that occurred with empty sets. To get a faithful model of ordinary rewriting (as opposed to: rewriting on ground terms), we need to interpret each type with a non-empty well-ordered set anyway, because we have term variables for any type. Thus, the special case disappears.

Apart from (ordinal) addition and multiplication, we can also define a corresponding (ordinal) exponentiation. This is only defined for totally well-ordered sets.

Definition 7. We define a functor $\mathcal{F} : \mathsf{TWO} \to \mathsf{TWO}$ as follows: let $\boldsymbol{A} = (A, >_A)$, $\boldsymbol{B} = (B, >_B)$, $\boldsymbol{A} \xrightarrow{f} \boldsymbol{B}$ then

- $\mathcal{F}(\boldsymbol{A}) = (\mathcal{F}(A), \succ_A)$ where $\mathcal{F}(A)$ is the set of finite subsets of A, and $M \succ_A N \iff \exists m \in (M \setminus N).\ \forall n \in (N \setminus M).\ m >_A n$.
- $\mathcal{F}(f)(M) = \{f(m) \mid m \in M\}$.

The restriction to TWO is necessary to make $\mathcal{F}$ functorial. All morphisms $f : \boldsymbol{A} \to \boldsymbol{B}$ in TWO are monic, in particular they guarantee that the witness $m \in (M \setminus N)$ for $M \succ_A N$ is mapped to a witness $f(m) \in (f(M) \setminus f(N)) = f(M \setminus N)$ for $f(M) \succ_B f(N)$, which is necessary to make $\mathcal{F}(f)$ monotonic.

$\mathcal{F}(A)$ contains only finite subsets of A, but still the operator $\mathcal{F}$ corresponds in ordinal arithmetic to exponentiation to the power of 2. A general ordinal exponentiation can also be defined:

Definition 8. The functor $\boldsymbol{Fin} : \mathsf{TWO} \times \mathsf{TWO} \to \mathsf{TWO}$ is defined as follows. Let $\boldsymbol{A} = (A, >_A)$, $\boldsymbol{B} = (B, >_B)$, $\boldsymbol{C} = (C, >_C)$ and $\boldsymbol{D} = (D, >_D)$, and let $\boldsymbol{A} \xrightarrow{f} \boldsymbol{B}$ and $\boldsymbol{C} \xrightarrow{g} \boldsymbol{D}$. Let $\perp_B$ be the smallest element of $\boldsymbol{B}$ (if it exists).

- $\boldsymbol{Fin}(\boldsymbol{A}, \boldsymbol{B}) = (A \rightsquigarrow B, \succ_{A*B})$ where
  $A \rightsquigarrow B = \{M \in \mathcal{F}(A * B) \mid (a,b) \in M \wedge (a,b') \in M \implies b = b' \ \wedge\ b >_B \perp_B\}$.
  The order $\succ_{A*B}$ is inherited from $\mathcal{F}(\boldsymbol{A} * \boldsymbol{B})$.
- $\boldsymbol{Fin}(f,g) = \mathcal{F}(f * g)$

Notice that for $(B, >_B) = \boldsymbol{B} \in |\mathsf{TWO}|$ a smallest element $\perp_B \in B$ always exists, with the only exception $\boldsymbol{B} = \boldsymbol{0} = (\emptyset, \emptyset)$ for which we have $\boldsymbol{Fin}(\boldsymbol{A}, \boldsymbol{0}) = \boldsymbol{1} = (\{\emptyset\}, \emptyset)$.

Because $\boldsymbol{Fin}$ operates in TWO, any morphisms f and g are injective and hence $f * g$ sends finite maps to finite maps. Similarly, f and g cannot map non-$\perp$ elements to $\perp$, because they are monotonic. That $\mathcal{F}(f * g)$ is monotonic follows easily from the fact that we inherit the order from $\mathcal{F}(\boldsymbol{A} * \boldsymbol{B})$.

A good intuition for $\boldsymbol{Fin}$ is difficult to come up with, although there exists one in the case of ordinal numbers, see the following section.

4.2 Ordinals

The functors described so far allow to construct new (partially) well-ordered sets from given ones. For termination proofs, it is useful to have an arithmetic and logic for well-orderings available. The obvious choice is ordinal arithmetic.

Definition 9. An *ordinal* is a set α such that all its elements are ordinals and

- $\forall \beta \in \alpha.\, \forall \gamma \in \alpha.\ \beta \in \gamma \vee \gamma \in \beta \vee \beta = \gamma$
- $\forall \beta \in \alpha.\, \forall \gamma \in \beta.\ \gamma \in \alpha$

These are the so-called "von Neumann ordinals" [22, 29]. It is not the aim of this paper to describe or investigate a foundation of transfinite induction or recursion, I simply assume a suitable set theory that supports it, i.e. I do not require ordinals to be the primitive notion of transfiniteness. Each ordinal α corresponds to an object $\boldsymbol{\alpha} = (\alpha, \ni)$ in TWO. Thus I understand the underlying set theory to include the Axiom of Foundation, for details see some standard book on set theory, e.g. [14, 18, 28]. In the following, I identify 0, 1, 2, etc. with their corresponding ordinal, ω is used for the ordinal corresponding to the set of natural numbers.

The functor I called "ordinal addition" is not quite ordinal addition in the usual sense, because we would need to postcompose it with a functor TYPE that maps partially well-ordered sets to their order type (the corresponding ordinal). We can do that by the following principle:

Definition 10. Given an endofunctor $F : \mathsf{TWO} \to \mathsf{TWO}$ (or a bifunctor $\_ \odot \_ : \mathsf{TWO} \times \mathsf{TWO} \to \mathsf{TWO}$), we define a corresponding function $\lfloor F \rfloor$ (or $\lfloor \odot \rfloor$) on the class of ordinals as follows: for any ordinals α and β, $\lfloor F \rfloor(\alpha)$ (or $\alpha \lfloor \odot \rfloor \beta$) is the unique ordinal γ such that there is an isomorphism $F(\boldsymbol{\alpha}) \cong \boldsymbol{\gamma}$ (or $\boldsymbol{\alpha} \odot \boldsymbol{\beta} \cong \boldsymbol{\gamma}$).

If no confusion arises, I shall write $F(\alpha)$ (or $\alpha \odot \beta$) instead of $\lfloor F \rfloor(\alpha)$ (or $\alpha \lfloor \odot \rfloor \beta$), i.e. if it is clear from the context that α and/or β are ordinals and not objects in TWO. A more traditional notation for $\lfloor \boldsymbol{Fin} \rfloor(\gamma, \beta)$ is $\beta^{(\gamma)}$, i.e. ordinal exponentiation.

As promised, there is an intuitive understanding of $\boldsymbol{Fin}(\gamma, \beta)$. Each element in the carrier set can be seen as the (unique) representation of an ordinal to base β, such that the domain of the finite map gives the exponents, and the range the coefficients. This also explains the exclusion of bottom ($= 0$) coefficients, and the use of lexicographic ordering, because "exponents matter more than coefficients".

Definition 11. Given an ordinal α, we define a map $\_\#\_ : \boldsymbol{Fin}(\alpha, \omega) \times \boldsymbol{Fin}(\alpha, \omega) \to \boldsymbol{Fin}(\alpha, \omega)$ as follows:

$$\begin{aligned} M\#N = \ & \{(a, m+n) \mid (a, m) \in M \wedge (a, n) \in N\} \cup \\ & \{(a, m) \in M \mid \neg \exists n.(a, n) \in N\} \cup \{(a, n) \in N \mid \neg \exists m.(a, m) \in M\} \end{aligned}$$

Intuitively, $\#$ does the following: it takes the representation of two ordinals to the base ω (the so-called "Cantorian normal form") and then adds the coefficients pointwise. Since addition is closed on natural numbers (and the coefficients happen to be natural numbers here), the result $M\#N$ is still in $\boldsymbol{Fin}(\alpha, \omega)$, i.e. we do not have to worry about carries. It is easy to see that $\#$ is associative, commutative, and has a neutral element (the empty map). This form of addition (the function $\lfloor \# \rfloor$ on ordinals) is the so-called "natural sum" of Hessenberg, see [9, 11, 6, 1].

Proposition 12. *For any ordinal α, $\#$ is a morphism in* TWO*:*

$$\boldsymbol{Fin}(\alpha,\omega) \otimes \boldsymbol{Fin}(\alpha,\omega) \xrightarrow{\#} \boldsymbol{Fin}(\alpha,\omega).$$

Proof. Since $\#$ is commutative, we only have to show:

$$M \succ_{\alpha*\omega} M' \Rightarrow M\#N \succ_{\alpha*\omega} M'\#N$$

There has to be a largest $a \in \alpha$ such that there is an element $p = (a, m) \in M$ but $p \notin N$. This a is also the largest element at which $M\#N$ and $M'\#N$ differ. $M\#N$ contains the pair $(a, m + n_a)$ and $M'\#N$ contains either no pair $(a, \_)$ or a pair $(a, m' + n_a)$ with $m \ni m'$, $\omega \ni n_a \supseteq 0$ and thus $m + n_a \ni m' + n_a$. Because the order in $\succ_{\alpha*\omega}$ is lexicographic, (a, m) is greater than all (a', k) for $a \ni a'$ and is therefore larger than all elements in $(M'\#N) \setminus (M\#N)$. □

Definition 13. Let α be an ordinal. It is called a *limit ordinal* iff $\forall \beta \in \alpha. \beta + 1 \in \alpha$. It is called *indecomposable* iff $\forall \beta \in \alpha.\ \beta + \alpha = \alpha$. It is called an *epsilon ordinal* iff $\forall \beta \in \alpha.\ \beta \ni 1 \Rightarrow \beta^{(\alpha)} = \alpha$.

The relevance of indecomposable[3] ordinals to this paper is that they can be understood as algebras for ordinal addition (and also natural sum), i.e. their elements are closed under ordinal addition. Indecomposable ordinals (greater than 0) are exactly those of the form $\omega^{(\gamma)}$ for some ordinal γ. This property is very closely related to the mentioned properties of $\#$. Indecomposability of α is sufficient *and* necessary for the existence of a morphism from $\alpha \otimes \alpha$ to α, for the latter case see lemma 15 in [5].

Proposition 14. *Let α be an indecomposable ordinal. Then $(\boldsymbol{\alpha}, \#, 0)$ is a monoid in* WO*, where $\#$ is natural sum and $0 : \mathbf{1} \to \boldsymbol{\alpha}$ is the function that maps the element of 1 to 0.*

Proof. Since α is indecomposable, there is an ordinal γ such that $\boldsymbol{\alpha} \cong \boldsymbol{Fin}(\boldsymbol{\gamma}, \boldsymbol{\omega})$. The rest follows easily from proposition 12 and the well-known properties of natural sum. □

Definition 15. Let α be an indecomposable ordinal. We write $\boldsymbol{M}$ for the representation monad of the monoid $(\boldsymbol{\alpha}, \#, 0)$.

4.3 Type-casting Ordinals

For concrete termination proofs we need to export the arithmetical structure on ordinals in some way to all partially well-ordered sets. The purpose of the following construction is exactly that.

Definition 16. Let $\mathcal{A}$ be a category and $X \in |\mathcal{A}|$. We define the category $\mathcal{A} \downdownarrows X$ as follows: $\mathcal{A} \downdownarrows X$ is the comma-category $X \downarrow \mathcal{A} \downarrow X$. Thus, an object in $\mathcal{A} \downdownarrows X$ is an object Z in $\mathcal{A}$, accompanied by two distinguished morphisms $X \xrightarrow{z} Z$ (encoding) and $Z \xrightarrow{\bar{z}} X$ (decoding), such that $\bar{z} \circ z = id_X$. A morphism from $X \xrightarrow{z} Z \xrightarrow{\bar{z}} X$ to $X \xrightarrow{y} Y \xrightarrow{\bar{y}} X$ in $\mathcal{A} \downdownarrows X$ is a morphism $f \in \mathcal{A}(Z, Y)$ such that the diagram in figure 1 commutes.

$$
\begin{tikzcd}
& X \arrow[dl, "z"'] \arrow[dr, "y"] & \\
Z \arrow[rr, "f"] \arrow[dr, "\bar{z}"'] & & Y \arrow[dl, "\bar{y}"] \\
& X &
\end{tikzcd}
$$

Figure 1: Morphisms in $\mathcal{A} \downdownarrows X$

To be precise, the category $X \downarrow \mathcal{A} \downarrow X$ should be written as either $(X \downarrow \mathcal{A}) \downarrow (X \xrightarrow{id} X)$ or as $(X \xrightarrow{id} X) \downarrow (\mathcal{A} \downarrow X)$, but both constructions result in identical categories.

In particular, we are interested in categories like $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ where α is an ordinal. In this case, the maps z and $\bar{z}$ convert ordinals less than α into elements of Z and back; the required equation $\bar{z} \circ z = id$ is another way of saying that decoding is the inverse of encoding. It is not required that the converse is true; in particular, an object $\boldsymbol{Z}$ may not be totally ordered although $\boldsymbol{\alpha}$ is.

If α is an indecomposable ordinal then we can take any values sitting in the carrier sets of objects $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$, convert them into ordinals, "add" them, and convert them back into any object in $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ we wish. The last step here requires the indecomposability of α. Moreover, all these conversions are monotonic; even the addition, provided we use $\#$ on $\boldsymbol{\alpha} \otimes \boldsymbol{\alpha}$.

One may think that requiring all this structure is too much to allow $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ to carry any non-trivial objects. But there is a similar situation in domain theory: each approximation D_n of the model D_∞ for the pure λ-calculus (see [10] for a detailed construction) can be mapped to the (continuous) function space $[D_n \to D_n]$ and back; these maps are structure-preserving and form a retraction, i.e. the decoding is the inverse of the encoding. Following this analogy further, one may wonder whether one should perhaps require $id \succeq x \circ \bar{x}$ (for some appropriately defined relation $\succeq$ on morphisms), in the style of "embedding-projection pairs" (see e.g. [8]), but in WO such a property is often too strong.

To interpret types in $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ we have to lift the "datatype" functors on WO to $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$. Notice that $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ is only an auxiliary construction in the sense that its morphisms (encoding homomorphisms) are too restricted as to be candidates for internalisation: a function type object in $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ should internalize arrows from WO, not from $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$. Similarly, categorical products in $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ are unsuitable for the interpretation of product types; products in $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ are pullbacks in WO, carrying pairs of values with the same decoding.

Proposition 17. *Consider the object $X \xrightarrow{id} X \xrightarrow{id} X$ in $\mathcal{A} \downdownarrows X$, which — by an abuse of notation — we simply call X. We have:*

[3] The name is taken from [28]; in [5] they are called "additive principal"; Hessenberg [9] called them "Hauptzahlen", main numbers.

$$\begin{tikzcd}
 & X \arrow[dl, "id"'] \arrow[d, "z"] \arrow[dr, "id"] & \\
X \arrow[r, "z"] \arrow[dr, "id"'] & Z \arrow[r, "\bar{z}"] \arrow[d, "\bar{z}"] & X \arrow[dl, "id"] \\
 & X &
\end{tikzcd}$$

Figure 2: Proof of proposition 17

1. *X is a null object in $\mathcal{A} \downdownarrows X$, i.e. it is initial and terminal.*

2. *For any two objects $A, B \in |\mathcal{A} \downdownarrows X|$ there is a morphism $A \xrightarrow{0} B$ uniquely defined by $0 = A \to X \to B$.*

Proof. The first point is a simple diagram chase in $\mathcal{A}$, see figure 2. The second property holds for any category with a null object [20]. □

Definition 18. We call an endofunctor $F : \mathcal{A} \to \mathcal{A}$ *retractable at* $X \in |\mathcal{A}|$ iff there is an object $X \xrightarrow{F} F(X) \xrightarrow{\bar{F}} X$ in $\mathcal{A} \downdownarrows X$.

Proposition 19. *If F is retractable at X then F can be lifted to $\mathcal{A} \downdownarrows X$. $F' : \mathcal{A} \downdownarrows X \to \mathcal{A} \downdownarrows X$ maps each object $X \xrightarrow{z} Z \xrightarrow{\bar{z}} X$ to*

$$X \xrightarrow{F} F(X) \xrightarrow{F(z)} F(Z) \xrightarrow{F(\bar{z})} F(X) \xrightarrow{\bar{F}} X$$

On morphisms, we just define $F'(f) = F(f)$.

Proof. It is easy to see that the properties of being an object or morphism in $\mathcal{A} \downdownarrows X$ are preserved, e.g. for objects: $\bar{F} \circ F(\bar{x}) \circ F(x) \circ F = \bar{F} \circ F(\bar{x} \circ x) \circ F = \bar{F} \circ F(id) \circ F = \bar{F} \circ id \circ F = \bar{F} \circ F = id$. □

We can extend the notion of retractability to bifunctors:

Definition 20. A bifunctor $\_ \odot \_ : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ or $\bar{\_} \odot \_ : \mathcal{A}^{op} \times \mathcal{A} \to \mathcal{A}$ is called retractable at X if there is an object $X \xrightarrow{\odot} X \odot X \xrightarrow{\bar{\odot}} X$ in $\mathcal{A} \downdownarrows X$.

This is equivalent to the requirement that both $\_ \odot A$ and $A \odot \_$ are retractable at X, provided the constant functor $C_A(x) = \boldsymbol{A}$, $C_A(f) = id_A$ is retractable at X, i.e. there exists coding and decoding for A making it an object in $\mathcal{A} \downdownarrows X$. We obtain an analogous result to proposition 19 for bifunctors: each functor which is retractable at X determines a bifunctor on $\mathcal{A} \downdownarrows X$; the proof is almost identical.

Theorem 21. *Let $\mathcal{A}$ be a monoidal closed category with binary products and coproducts.*

$$
\begin{tikzcd}
X \arrow[r, "r^{-1}"] \arrow[d, "cur(\oplus)"'] \arrow[dr, phantom, "(i)"] & X \otimes I \arrow[r, "id \otimes 0"] \arrow[d, "cur(\oplus) \otimes id"] \arrow[dr, phantom, "(ii)"] & X \otimes X \arrow[r, "\oplus"] \arrow[d, "cur(\oplus) \otimes id"] \arrow[dr, phantom, "(iii)"] & X \\
X \Rightarrow X \arrow[r, "r^{-1}"'] & (X \Rightarrow X) \otimes I \arrow[r, "id \otimes 0"'] & (X \Rightarrow X) \otimes X \arrow[r, "ap"'] & X \arrow[u, "id"']
\end{tikzcd}
$$

Figure 3: Theorem 21, case 3

1. *Categorical product $\sqcap$ and coproduct $\sqcup$ are retractable at any X:*

$$X \xrightarrow{\langle id,id\rangle} (X \sqcap X) \xrightarrow{\pi_1} X$$

is a retraction for the product; the coproduct is dual.

2. *If $(X, \oplus, 0)$ is a monoid in $\mathcal{A}$ then the tensor product $\otimes$ is retractable at X.*

$$X \cong I \otimes X \xrightarrow{0 \otimes id} (X \otimes X) \xrightarrow{\oplus} X$$

is a retraction by the coherence properties of a monoid.

3. *The exponential $\Rightarrow$ (right adjoint of $\otimes$) is retractable at monoids:*

$$X \xrightarrow{cur(\oplus)} (X \Rightarrow X) \cong (X \Rightarrow X) \otimes I \xrightarrow{id \otimes 0} (X \Rightarrow X) \otimes X \xrightarrow{ap} X$$

is a retraction; ap is the co-unit of the adjunction and $cur(\oplus)$ is the curried form of $\oplus : X \otimes X \to X$, also given by the adjunction.

Proof. Cases 1 and 2 are immediate. For case 3, see figure 3: (i) commutes by naturality of r^{-1}, one of the natural isomorphisms of the monoidal structure of $\mathcal{A}$, (ii) commutes by functorality of $\otimes$, and (iii) is the co-unit equation for ap. The upper line, $\oplus \circ (id \otimes 0) \circ r^{-1}$, is the identity because $(X, \oplus, 0)$ is a monoid. $\square$

Taking $\mathcal{A} = \mathsf{WO}$ it follows that $\otimes$ and $\Rightarrow$ are retractable at indecomposable ordinals, $\oplus$ being the natural sum. Ordinal addition, multiplication and exponentiation are typically not retractable, except in trivial cases. For instance, none of them is retractable at $\boldsymbol{\omega}$, because e.g. $\boldsymbol{\omega} + \boldsymbol{\omega}$ is totally ordered and corresponds to a strictly larger ordinal than ω, i.e. the morphism set $\mathsf{WO}(\boldsymbol{\omega} + \boldsymbol{\omega}, \boldsymbol{\omega})$ is empty.

The functoriality of $\otimes$ on $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$ means that for any object $\boldsymbol{\alpha} \xrightarrow{x} \boldsymbol{X} \xrightarrow{\bar{x}} \boldsymbol{\alpha}$ we have another object $\boldsymbol{\alpha} \xrightarrow{\langle x, x(0)\rangle} \boldsymbol{X} \otimes \boldsymbol{X} \xrightarrow{\oplus \circ (\bar{x} \otimes \bar{x})} \boldsymbol{\alpha}$. By postcomposing the retraction of that object with the map x we also get a monotonic map $\boxplus$ from $\boldsymbol{X} \otimes \boldsymbol{X}$ to $\boldsymbol{X}$. Although monotonic, this map will not have all the algebraic properties one would hope for, e.g. we do not have in general $a \boxplus x(0) = a$ and $a \boxplus x(1) >_X a$.

4.4 α-addition everywhere

Especially something like the second property is useful, because it allows to build infinite *increasing* chains from any element of an object. For termination proofs for a rewrite system, we need such a property for a type τ if the system contains a rule of type τ of the form $l \to x$ where x is a variable occurring in l; in TRS theory, such rules are called *collapsing*. We can substitute l for x in l as often as we please and obtain a rewrite sequence of arbitrary length. To show termination for such a rule, we have to find a semantic interpretation for l that preserves the interpretation of its x component.

Recall the monad $\boldsymbol{M} = (\boldsymbol{\alpha} \otimes \_, \mu, \eta)$ from definition 15. $(\boldsymbol{A}, \boxplus)$ being an $\boldsymbol{M}$-algebra means in particular that (i) $\boxplus \circ \eta = id$, which is the same as saying that $0 \boxplus x = x$, and (ii) $\boxplus \circ \mu = \boxplus \circ (id \otimes \boxplus)$, which is the same as $(m\#n) \boxplus x = m \boxplus (n \boxplus x)$. Especially the unit-property is useful: in $\boldsymbol{\alpha} \otimes \boldsymbol{A}$ there are α-chains if $\boldsymbol{A}$ is non-empty, e.g. $(0,x) < (1,x) < \ldots$. The presence of a monotonic map $\boxplus$ means that there are also α-chains in $\boldsymbol{A}$; because of the retraction property $0 \boxplus x = x$ we have α-chains starting from any element x of A: $x = 0 \boxplus x < 1 \boxplus x < \ldots$; this mirrors the behaviour of rewrite systems with collapsing rules.

For most of the theorems in this section we do not need to focus on WO and show them for arbitrary monoidal closed categories.

Proposition 22. *Let $\mathcal{A}$ be a monoidal category and $(X, \oplus, 0)$ be a monoid in $\mathcal{A}$ and let $\boldsymbol{T}$ be the representation monad of this monoid. Then $(X, \oplus) \in |\mathcal{A}^{\boldsymbol{T}}|$.*

Proof. Trivial, the coherence properties for this algebra coincide with the monad properties. □

Although the observation in proposition 22 is rather trivial, it is important for the whole method. We can interpret "atomic" types by $\boldsymbol{\alpha}$ and leave the interpretation of composite types to functors on $\mathsf{WO}^{\boldsymbol{M}}$, supporting collapsing rules of composite types.

To maintain the existence of a $\boxplus$-operation with nice algebraic properties we have to make the functors we are interested in operate on $\mathsf{WO}^{\boldsymbol{M}}$. To some extent, this is possible in an abstract setting, i.e. without referring to this particular category.

Definition 23. Let $\mathcal{A}$ be a category and $\boldsymbol{T} = (T, \mu, \eta)$ a monad on $\mathcal{A}$. A functor $F : \mathcal{A} \to \mathcal{A}$ is called $\boldsymbol{T}$*-distributive* if there is a natural transformation $\delta : T(F(U(\_))) \to F(T(U(\_)))$ such that $F(\boxplus_A) \circ \delta_A \circ \eta = id$ and $F(\boxplus_A) \circ \delta_A \circ \mu = F(\boxplus_A) \circ \delta_A \circ T(F(\boxplus_A) \circ \delta_A)$. Here, $U : \mathcal{A}^{\boldsymbol{T}} \to \mathcal{A}$ is the forgetful functor of the monad and $\boxplus_A : T(A) \to A$ is the algebra morphism on A.

Note that the use of the forgetful functor U makes δ live in $\mathcal{A}^{\boldsymbol{T}}$, i.e. for its construction we can always assume that domain and codomain are $\boldsymbol{T}$-algebras. For $\boldsymbol{T}$-distributive functors, we get a new $\boxplus'$ operation on $F(A)$ as $\boxplus' = F(\boxplus_A) \circ \delta$.

As mentioned, $\boldsymbol{T}$-distributivity implies lifting.

Proposition 24. *If F is $\boldsymbol{T}$-distributive then it can be lifted to $\mathcal{A}^{\boldsymbol{T}}$.*

Proof. See figure 4; the upper two diagrams show the preservation on objects; notice that $F(\boxplus) \circ F(\eta) = F(\boxplus \circ \eta) = F(id) = id$, by functoriality of F and because $(X, \boxplus)$ is an $\boldsymbol{T}$-algebra. Thus $F(\boxplus) \circ \delta$ gives the required $\boxplus'$ on $F(X)$. The lower diagram shows the preservation of $\mathcal{A}^{\boldsymbol{T}}$-morphisms: the left square commutes by the $\mathcal{A}^{\boldsymbol{T}}$-morphism property of f, the right square by naturality of δ. □

$$
\begin{tikzcd}
TFX \arrow[r, "\delta"] & FTX \arrow[dl, "F\boxplus"] \\
FX \arrow[u, "\eta"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
TTFX \arrow[r, "T(F\boxplus\circ\delta)"] \arrow[d, "\mu"'] & TFX \arrow[d, "F\boxplus\circ\delta"] \\
TFX \arrow[r, "F\boxplus\circ\delta"'] & FX
\end{tikzcd}
$$

$$
\begin{tikzcd}
FA \arrow[d, "Ff"'] & FTA \arrow[l, "F\boxplus"'] \arrow[d, "FTf"] & TFA \arrow[l, "\delta"'] \arrow[d, "TFf"] \\
FB & FTB \arrow[l, "F\boxplus"] & TFB \arrow[l, "\delta"]
\end{tikzcd}
$$

Figure 4: Proposition 24

To show that a bifunctor $\_ \odot \_$ can be lifted along U, it is sufficient to show that $\_ \odot U(X)$ can be lifted for any X. Thus we can now check whether the retractable functors we have so far, i.e. product, coproduct, symmetric multiplication, and arrow, are $\boldsymbol{M}$-distributive for a fixed argument.

Lemma 25. *Let $\mathcal{A}$ be a category with binary products $\_ \sqcap \_$. Let $\boldsymbol{T} = (T, \mu, \eta)$ be a monad on $\mathcal{A}$. Then $\_ \sqcap X$ is $\boldsymbol{T}$-distributive if $(X, \boxplus) \in |\mathcal{A}^{\boldsymbol{T}}|$.*

Proof. We can define δ as $\langle T(\pi_1), \boxplus \circ T(\pi_2)\rangle$. The new addition is $\boxplus' = F\boxplus \circ \delta = (\boxplus \sqcap id) \circ \langle T\pi_1, \boxplus \circ T\pi_2\rangle = \langle \boxplus \circ T\pi_1, \boxplus \circ T\pi_2\rangle$.

Unit property: $\boxplus' \circ \eta = \langle \boxplus \circ T\pi_1, \boxplus \circ T\pi_2\rangle \circ \eta = \langle \boxplus \circ T\pi_1 \circ \eta, \boxplus \circ T\pi_2 \circ \eta\rangle = \langle \boxplus \circ \eta \circ \pi_1, \boxplus \circ \eta \circ \pi_2\rangle = \langle \pi_1, \pi_2\rangle = id$.

Associativity: $\boxplus' \circ T\boxplus' = \langle \boxplus \circ T\pi_1, \boxplus \circ T\pi_2\rangle \circ T\boxplus' = \langle \boxplus \circ T\pi_1 \circ T\boxplus', \boxplus \circ T\pi_2 \circ T\boxplus'\rangle = \langle \boxplus \circ T(\pi_1 \circ \boxplus'), \boxplus \circ T(\pi_2 \circ \boxplus')\rangle = \langle \boxplus \circ T(\boxplus \circ T\pi_1), \boxplus \circ T(\boxplus \circ T\pi_2)\rangle = \langle \boxplus \circ T(\boxplus) \circ TT\pi_1, \boxplus \circ T(\boxplus) \circ TT\pi_2\rangle = \langle \boxplus \circ \mu \circ TT\pi_1, \boxplus \circ \mu \circ TT\pi_2\rangle = \langle \boxplus \circ T\pi_1 \circ \mu, \boxplus \circ T\pi_2 \circ \mu\rangle = \langle \boxplus \circ T\pi_1, \boxplus \circ T\pi_2\rangle \circ \mu = \boxplus' \circ \mu$. □

Another way of stating that the product functor can be lifted to $\mathcal{A}^{\boldsymbol{T}}$ is to say that the forgetful functor $U : \mathcal{A}^{\boldsymbol{T}} \to \mathcal{A}$ reflects products. Something stronger is true: U creates limits [20].

For the other functors we cannot show this on quite such a general level, but we can restrict our attention to monoidal (closed) catgories and representation monads.

Lemma 26. *Let $\mathcal{A}$ be a symmetric monoidal closed category with binary coproducts $\_ \sqcup \_$. Let X be a monoid in $\mathcal{A}$ and $\boldsymbol{T}$ be its representation monad. If $(Z, \boxplus) \in |\mathcal{A}^{\boldsymbol{T}}|$ then the functor $\_ \sqcup Z$ is $\boldsymbol{T}$-distributive.*

Proof. Because $\mathcal{A}$ is symmetric monoidal closed, the functor $X \otimes \_$ has a right adjoint which implies that it preserves colimits. Hence we have a natural isomorphism $\iota : X \otimes (\_ \sqcup Z) \cong (X \otimes \_) \sqcup (X \otimes Z)$ given by $\iota = ap \circ s \circ (id \otimes [cur(i_1 \circ s), cur(i_2 \circ s)])$ (where i_1 and i_2 are the coproduct injections and s is the symmetry isomorphism) and we get $\delta = (id \sqcup \boxplus) \circ \iota$. Checking the equations for δ is routine. □

Lemma 27. *Let $\mathcal{A}$ be a monoidal category, X a monoid in $\mathcal{A}$ and $\boldsymbol{T}$ the representation monad of X. Then the functor $\_ \otimes Y$ is $\boldsymbol{T}$-distributive.*

Proof. Immediate by choosing δ to be the associativity isomorphism. $\square$

Lemma 28. *Let $\mathcal{A}$ be a monoidal closed category, X a monoid in $\mathcal{A}$, $\boldsymbol{T}$ the representation monad of X, and $(Y, \boxplus) \in |\mathcal{A}^{\boldsymbol{T}}|$. Then the functor $Y \Rightarrow \_$ is $\boldsymbol{T}$-distributive.*

Proof. We can define $\delta = cur((id \otimes ap) \circ a^{-1})$ and have to check that the two upper diagrams in figure 4 commute. The new addition is $\boxplus' = (id \Rightarrow \boxplus) \circ \delta = (id \Rightarrow \boxplus) \circ cur((id \otimes ap) \circ a^{-1}) = cur(\boxplus \circ (id \otimes ap) \circ a^{-1})$. We write μ and η as abbreviations for the multiplication and unit of the monad.

We have: $a^{-1} \circ (\eta \otimes id) = a^{-1} \circ ((0 \otimes id) \otimes id) \circ (l^{-1} \otimes id) = (0 \otimes id) \circ a^{-1} \circ (l^{-1} \otimes id) = (0 \otimes id) \circ l^{-1} = \eta$. This gives us the unit property: $\boxplus' \circ \eta = cur(\boxplus \circ (id \otimes ap) \circ a^{-1}) \circ \eta = cur(\boxplus \circ (id \otimes ap) \circ a^{-1} \circ (\eta \otimes id)) = cur(\boxplus \circ (id \otimes ap) \circ \eta) = cur(\boxplus \circ \eta \circ ap) = cur(ap) = id$.

For associativity we have $\boxplus' \circ \mu = cur(\boxplus \circ (id \otimes ap) \circ a^{-1}) \circ \mu = cur(\boxplus \circ (id \otimes ap) \circ a^{-1} \circ (\mu \otimes id))$ and on the other side of the equation $\boxplus' \circ (id \otimes \boxplus') = cur(\boxplus \circ (id \otimes ap) \circ a^{-1}) \circ (id \otimes \boxplus') = cur(\boxplus \circ (id \otimes ap) \circ a^{-1} \circ ((id \otimes \boxplus') \otimes id))$.

We can now show that both sides are equal (leaving out the "cur"), starting from the left: $\boxplus \circ (id \otimes ap) \circ a^{-1} \circ (\mu \otimes id) = \boxplus \circ (id \otimes ap) \circ a^{-1} \circ (((\oplus \otimes id) \circ a) \otimes id) = \boxplus \circ (id \otimes ap) \circ a^{-1} \circ ((\oplus \otimes id) \otimes id) \circ (a \otimes id) = \boxplus \circ (id \otimes ap) \circ (\oplus \otimes id) \circ a^{-1} \circ (a \otimes id) = \boxplus \circ (\oplus \otimes id) \circ (id \otimes ap) \circ a \circ (id \otimes a^{-1})) \circ a^{-1} = \boxplus \circ (\oplus \otimes id) \circ a \circ (id \otimes (id \otimes ap)) \circ (id \otimes a^{-1})) \circ a^{-1} = \boxplus \circ (id \otimes \boxplus) \circ (id \otimes (id \otimes ap)) \circ (id \otimes a^{-1})) \circ a^{-1} = \boxplus \circ (id \otimes (\boxplus \circ (id \otimes ap) \circ a^{-1})) \circ a^{-1} = \boxplus \circ (id \otimes (ap \circ cur(\boxplus \circ (id \otimes ap) \circ a^{-1}) \otimes id)) \circ a^{-1} = \boxplus \circ (id \otimes (ap \circ \boxplus' \otimes id)) \circ a^{-1} = \boxplus \circ (id \otimes ap) \circ (id \otimes (\boxplus' \otimes id)) \circ a^{-1} = \boxplus \circ (id \otimes ap) \circ a^{-1} \circ ((id \otimes \boxplus') \otimes id)$ $\square$

Theorem 29. *Let $\mathcal{A}$ be a symmetric monoidal closed category, X a monoid in $\mathcal{A}$, and $\boldsymbol{T}$ the representation monad of X. Then the functor $\_ \odot \_$ can be lifted to $\mathcal{A}^{\boldsymbol{T}}$, where $\odot$ ranges over $\sqcup$, $\sqcap$, $\otimes$, $\Rightarrow$.*

Proof. Follows immediately from proposition 24 and lemmas 25, 26, 27, and 28: we simply define $X \Rightarrow Y$ on $\mathcal{A}^{\boldsymbol{T}}$ by applying the lifting of $(U(X) \Rightarrow \_)$ (given by lemma 28) to Y, etc. $\square$

4.5 Combining Algebras and Conversion

Now it would be nice if we could combine the construction of $\mathsf{WO}^{\boldsymbol{M}}$ and $\mathsf{WO} \downdownarrows \boldsymbol{\alpha}$, i.e. lift the functors to $\mathsf{WO}^{\boldsymbol{M}} \downdownarrows (\boldsymbol{\alpha}, \#)$. We can again look at this construction in a more abstract setting.

Lemma 30. *Let $\mathcal{A}$ be a category and $\boldsymbol{T} = (T, \mu, \eta)$ be a monad on $\mathcal{A}$. An endofunctor $F : \mathcal{A} \to \mathcal{A}$ can be lifted to $\mathcal{A}^{\boldsymbol{T}} \downdownarrows (X, \oplus)$ if*

1. *F is $\boldsymbol{T}$-distributive,*
2. *F is retractable at X,*
3. *the diagram in figure 5 commutes.*

$$
\begin{tikzcd}
 & T(X,\oplus) \arrow[r, "\oplus"] \arrow[dl, "TF"'] & (X,\oplus) \arrow[d, "F"] \\
TF(X,\oplus) \arrow[r, "\delta"] \arrow[dr, "T\bar{F}"'] & FT(X,\oplus) \arrow[r, "F\oplus"] & F(X,\oplus) \arrow[d, "\bar{F}"] \\
 & T(X,\oplus) \arrow[r, "\oplus"] & (X,\oplus)
\end{tikzcd}
$$

Figure 5: Condition (iii) for lemma 30

Proof. The first two points imply that F lifts to $\mathcal{A}^{\boldsymbol{T}}$ and $\mathcal{A} \Downarrow X$. The diagram ensures that the retraction maps are morphisms in $\mathcal{A}^{\boldsymbol{T}}$. $\square$

Lemma 31. *Let $\mathcal{A}$ be a category with binary products $\_ \sqcap \_$ and let $\boldsymbol{T} = (T, \mu, \eta)$ be a monad on $\mathcal{A}$. Then the lifted $\_ \sqcap \_$ (on $\mathcal{A}^{\boldsymbol{T}}$) is retractable at any object.*

Proof. We use the lifting of lemma 25 and the retraction of theorem 21. Checking the diagram in figure 5 is easy; upper half (encoding): $\langle \oplus \circ T\pi_1, \oplus \circ T\pi_2 \rangle \circ T\langle id, id \rangle = \langle \oplus \circ T\pi_1 \circ T\langle id, id \rangle, \oplus \circ T\pi_2 \circ T\langle id, id \rangle \rangle = \langle \oplus \circ T(\pi_1 \circ \langle id, id \rangle), \oplus \circ T(\pi_2 \circ \langle id, id \rangle) \rangle = \langle \oplus \circ Tid, \oplus \circ Tid \rangle = \langle \oplus, \oplus \rangle = \langle id, id \rangle \circ \oplus$. Lower half (decoding): $\pi_1 \circ \langle \oplus \circ T\pi_1, \oplus \circ T\pi_2 \rangle = \oplus \circ T\pi_1$. $\square$

For coproducts and tensor it is even simpler:

Lemma 32. *Premises as in lemma 26. The lifted functor $\_ \sqcup \_$ is retractable at any object.*

Proof. As pointed out in the proof of lemma 26, the lifted $\_ \sqcup \_$ is still a categorical coproduct, i.e. theorem 21 applies. $\square$

Lemma 33. *Let $\mathcal{A}$ be a monoidal category, $(X, \oplus, 0)$ a monoid in $\mathcal{A}$, and $\boldsymbol{T}$ be the representation monad of this monoid. Then the lifted $\_ \otimes \_$ is retractable at $(X, \oplus)$.*

Proof. It is easy to see that $\mathcal{A}^{\boldsymbol{T}}$ is itself a monoidal category, inheriting the tensor product etc. from $\mathcal{A}$. $((X, \oplus), \oplus, 0)$ is a monoid in $\mathcal{A}^{\boldsymbol{T}}$ and thus theorem 21 applies. $\square$

The same method does not work for $\_ \Rightarrow \_$ because the lifted version will (in general) not be the right adjoint of the tensor product in $\mathcal{A}^{\boldsymbol{T}}$ if it has one at all. For example in $\mathsf{WO}^{\boldsymbol{M}}$, $\_ \otimes (X, \oplus)$ has a right adjoint, but it differs from $X \Rightarrow \_$ which internalises too many morphisms in the resulting objects making it impossible to lift the unit and co-unit of the adjunction accordingly. So we have to prove the commutation of diagram 5 directly.

Lemma 34. *Premises as in lemma 28. Then the lifted $\_ \Rightarrow \_$ is retractable at X.*

Proof. Checking the equations in diagram 5 is tedious but routine (very similar to the proof of lemma 28). $\square$

Putting these results together, we have the following recipe for interpreting types by partially well-ordered sets:

Theorem 35. *Let α be an indecomposable ordinal. For any object $\boldsymbol{A} \in \mathsf{WO}$, which we can build from applying the functors $\sqcap$, $\sqcup$, $\otimes$, and $\Rightarrow$ in arbitrary order to $\boldsymbol{\alpha}$, we have:*

1. *There are morphisms $\boldsymbol{\alpha} \xrightarrow{a} \boldsymbol{A} \xrightarrow{\bar{a}} \boldsymbol{\alpha}$ such that $\bar{a} \circ a = id_\alpha$.*
2. *For any other object $\boldsymbol{B}$ built this way, we have a morphism $\boldsymbol{A} \xrightarrow{0} \boldsymbol{B}$.*
3. *We have a morphism $\boldsymbol{\alpha}\otimes\boldsymbol{A} \xrightarrow{\boxplus} \boldsymbol{A}$ such that $0\boxplus x = x$ and $m\boxplus(n\boxplus x) = (m\oplus n)\boxplus x$.*
4. *The conversions $\boldsymbol{A} \xrightarrow{0} \boldsymbol{B}$ preserve addition, i.e. $0(n \boxplus x) = n \boxplus 0(x)$.*

Proof. This is just a summary of some of the results above. $\square$

Moreover, we can extend this result to other endofunctors on WO, provided they are retractable at $\boldsymbol{\alpha}$, they are $\boldsymbol{M}$-distributive and their retraction maps are $\mathsf{WO}^{\boldsymbol{M}}$-homomorphisms.

5 Syntax

The previous section presented a semantic domain for the interpretation of rewrite systems that supports termination proofs. We still have to provide a syntactic framework powerful enough to formulate the rewrite systems we are interested in and connect it with the semantics. On the other hand, this syntactic framework has to be restricted enough to have a "nice" interpretation in the category WO (or rather $\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}$), it should somehow support termination proofs.

First-order term rewriting systems have such a nice interpretation (see [32, 34]), but they have limited expressive power. Especially, they do not support rewriting modulo substitutions. One can internalise substitutions to TRSs, see [19, 21], but this makes the termination proof of the substitution calculus part of the application of the method, rather than keeping it in the framework. In other words: for any particular system, one has to re-formulate the substitution rules (to support substitution for the full signature) and re-do the termination proof. Apart from this mere inconvenience, internalised substitutions may destroy the termination of a rewrite relation and they do not support *per se* the application of substitutions in left-hand sides of rewrite rules.

More expressive than ordinary TRSs are the Higher-Order Rewrite Systems *à la* Nipkow [23], a generalisation of Klop's Combinatory Reduction Systems [16, 17] — for their relationship see [25]. As we are not concerned here with implementability issues, we can even consider Wolfram's generalisation of HRSs (see chapter 4 in [31]). HRSs are based on simply typed λ-calculus $\lambda^{\rightarrow}$. Its terms can be seen as either equivalence classes of λ-terms, the equivalence relation being $=_{\beta\eta}$, or as long β-normal forms, i.e. as canonical representatives of those classes; see [12] for a direct definition of HRSs without quotienting.

The problem with HRSs is the lack of "nice" interpretations of $\lambda^{\rightarrow}$ in WO, simply because WO is not a CCC. Such an interpretation should assign the same values to $\beta\eta$-convertible terms, because HRSs rewrite modulo $\beta\eta$-conversion. Moreover, it should also interpret function types as sets of monotonic functions; this is necessary to lift termination of rewriting to its congruence closure, i.e. to rewriting on subterms. These objectives are conflicting for $\lambda^{\rightarrow}$: mapping convertible λ-terms to the same values means to interpret syntactic λ-abstraction by semantic λ-abstraction; but $\lambda^{\rightarrow}$ contains constant functions (like $\lambda x.c$) the semantic equivalent of which do not preserve strict orderings.

The approach of van de Pol [27] tries to solve this problem by weakening the second objective and allowing certain non-monotonic functions in function types. The technical complications arising from this decision are considerable, and the application conditions van de Pol requires for an interpretation to show the termination of a particular HRS are quite strong. It is far from obvious how they could be satisfied for enriched λ-calculi with internalised function types.

5.1 Term, Types, and Their Interpretations

Instead of allowing non-monotonic functions in the semantic domain, we sacrifice the other mentioned objective and allow β-equivalent terms to have different semantic interpretations.

To interpret types and terms in WO (or $\mathsf{WO}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}$), we shall give some functions from sets of types or terms into $|\mathsf{WO}|$ or $\bigcup|\mathsf{WO}|$, respectively. The notation $\bigcup|\mathsf{WO}|$ is shorthand for $\bigcup\{s \mid (s,>) \in |\mathsf{WO}|\}$; similarly for $\mathsf{WO}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}$, we suppress the application of the forgetful functor $U : \mathsf{WO}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha} \rightarrow \mathsf{Set}$. This is slightly problematic, as neither $|\mathsf{WO}|$ nor $\bigcup|\mathsf{WO}|$ are *sets*, they are proper classes. Since the domain of the mentioned functions is always a set, their graph is a set as well and so we shall not further worry about foundational issues.

Definition 36. Given a set of *base types* $\mathcal{B}$ we define the set of *types over* $\mathcal{B}$, $\mathit{Typ}(\mathcal{B})$, as the smallest set of words over the alphabet $\{\rightarrow, (,)\} \uplus \mathcal{B}$ satisfying:

1. $\sigma \in \mathcal{B} \Rightarrow \sigma \in \mathit{Typ}(\mathcal{B})$
2. $\alpha, \beta \in \mathit{Typ}(\mathcal{B}) \Rightarrow (\alpha \rightarrow \beta) \in \mathit{Typ}(\mathcal{B})$

$\mathit{Typ}(\mathcal{B})$ comprises the types of $\lambda^{\rightarrow}$. As usual, we drop many parentheses and take $\rightarrow$ to be right-associative.

Having only one type constructor for non-base types reflects the meta-level we are dealing with, i.e. $\lambda^{\rightarrow}$. This does not prevent us from giving base types an internal structure reflecting the type structure we want on the object-level.

Definition 37. Let $\mathcal{B}$ be a set of base types, let $b : \mathcal{B} \rightarrow |\mathsf{WO}|$ a function mapping base types to objects in WO. We define a map $[\![\_]\!]_b : \mathit{Typ}(\mathcal{B}) \rightarrow |\mathsf{WO}|$ as follows:

$$\begin{array}{rcl} [\![\sigma \rightarrow \tau]\!]_b &=& [\![\sigma]\!]_b \Rightarrow [\![\tau]\!]_b \\ [\![\tau]\!]_b &=& b(\tau),\ \text{if}\ \tau \in \mathcal{B} \end{array}$$

Here, "$\Rightarrow$" is the functor from proposition 6.

Analogously, we derive from a function $b : \mathcal{B} \to |\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$ a function for all types $[\![\_]\!]_b : \mathit{Typ}(\mathcal{B}) \to |\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$, provided α is an indecomposable ordinal, because we can lift $\_ \Rightarrow \_$ at monoids, see theorem 35. In other words, the interpretation $[\![\sigma]\!]_b$ of a type σ in $\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}$ is an object $\boldsymbol{\alpha} \xrightarrow{\sigma} ((S, >_\sigma), \boxplus_\sigma) \xrightarrow{\bar{\sigma}} \boldsymbol{\alpha}$ in $\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}$. We shall write $\bar{\sigma}$, $\boxplus_\sigma$, and $>_\sigma$ to refer to the corresponding components of $[\![\sigma]\!]_b$.

Definition 38. An HRS-signature is a tuple $(\mathcal{B}, \mathcal{S}, \mathcal{C})$, where $\mathcal{B}$ is a set, $\mathcal{S}$ a set of *symbols*, and $\mathcal{C} : \mathcal{S} \to \mathit{Typ}(\mathcal{B})$ is a function assigning types to symbols.

For typed signatures, one usually allows overloaded symbols, i.e. $\mathcal{C}$ would be a function from $\mathit{Typ}(\mathcal{B})$ to $2^{\mathcal{S}}$. This is mostly considered an insignificant difference, although there are known examples for which this distinction matters: Zantema's type removal theorem [33] and virtually all preservation properties of Currying [4, 15, 13] rely on the property that symbols have unique types or arities. This difference is also important when we want to apply a semantic interpretation to a term t or even a derived typing judgement $\Gamma \vdash t : \tau$. The judgement may have essentially different derivations, i.e. derivations assigning different types to symbols. For a termination proof by semantic interpretation, we would need to show that $s \to_R t$ implies $[\![s]\!] > [\![t]\!]$ *for every resolution of overloading* which is strange as it also affects subterms of the context of the redex, i.e. subterms that (in an intuitive sense) do not take part in the rewrite step. One may think that the real thing would be an interpretation of the entire derivation of the typing judgement together with a rewrite relation on these derivation trees. But this is questionable: by appropriately massaging the rules of the type system we can have $[\![d_1]\!] > [\![d_2]\!]$ for two derivations d_1 and d_2 with identical conclusions. For instance, we could present the theory of $\lambda^{\to}$ with proof rules that have a distinctive λ_I flavour: with a weakening rule like

$$\frac{\Gamma \vdash t : \tau \qquad \Delta \vdash s : \sigma}{\Gamma \cup \Delta \vdash t : \tau}$$

we have a weakening witness s and although s does not occur in the derived judgement the interpretation of the proof tree may well take s into account.

Independently from particular signatures, we assume the existence of a countably infinite set of *variables*, called $\mathcal{V}$.

Definition 39. Given a signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, a *preterm over* Σ is a λ-term with variables taken from $\mathcal{V}$ and constants taken from $\mathcal{S}$. We require abstractions to be in Church-style [2], i.e. an abstraction has the form $(\lambda x : \sigma.t)$, where x is a variable, $\sigma \in \mathit{Typ}(\mathcal{B})$, and t is a preterm over Σ. We write $\Lambda\Sigma$ for the set of all preterms. Given a preterm t, we write $t{\downarrow}$ for the β-normal form of t if it exists.

Preterms are just untyped λ-terms with type annotations for abstractions. They may or may not be well-typed in some type system. The reason for requiring type annotations in abstractions is similar to the reason for banning symbol overloading: in Curry-style $\lambda^{\to}$, the term $(\lambda x.c)(\lambda y.y)$ type-checks with type $\mathcal{C}(c)$ by choosing $y : \sigma$ and $x : \sigma \to \sigma$ for any σ. As we shall see later, the choice of σ can be significant for the semantic interpretation of this term.

Definition 40. Given a signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, a *context* (over Σ) is a finite set $\Gamma \subset \mathcal{V} \times \mathit{Typ}(\mathcal{B})$ such that $(x, \tau) \in \Gamma \wedge (x, \sigma) \in \Gamma \Rightarrow \sigma = \tau$. Convention: we write $x : \tau$ instead of (x, τ) for elements of a context. We write $x \notin \Gamma$ as shorthand for $\neg\exists\sigma.(x, \sigma) \in \Gamma$.

$$\frac{x \notin \Gamma}{\Gamma \cup \{x:\sigma\} \vdash x:\sigma} \qquad \frac{}{\Gamma \vdash c : \mathcal{C}(c)}$$

$$\frac{\Gamma \vdash t : \sigma \to \tau \quad \Gamma \vdash u : \sigma}{\Gamma \vdash (t\, u) : \tau} \qquad \frac{\Gamma \cup \{x:\sigma\} \vdash t : \tau \quad x \notin \Gamma}{\Gamma \vdash (\lambda x : \sigma . t) : \sigma \to \tau}$$

Figure 6: The type system $\lambda^{\to}$

One can view a context as a finite map from variables to types. The elements of a context are in no particular order; we do not need such an order for $\lambda^{\to}$.

Definition 41. Let $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$ be a signature. A *judgement* (over Σ) is a triple (Γ, t, τ), written $\Gamma \vdash t : \tau$, where Γ is a context (over Σ), t a preterm (over Σ), and $\tau \in \mathit{Typ}(\mathcal{B})$. Given a judgement $J = \Gamma \vdash t : \tau$, we write $J{\downarrow}$ for the judgement $\Gamma \vdash t{\downarrow} : \tau$ (which exists if $t{\downarrow}$ exists).

Definition 42. For any signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, the *type theory* $\lambda^{\to}$ is the smallest set of judgements over Σ that is closed under the rules in figure 6.

Proposition 43. *Let $J \in \lambda^{\to}$. Then $J{\downarrow}$ exists and $J{\downarrow} \in \lambda^{\to}$.*

Proof. This is a direct consequence of strong normalisation and subject-reduction for $\lambda^{\to}$. □

Proposition 44. *Let $\Gamma \vdash t : \tau$ and $\Gamma \vdash t : \tau'$ be derivable judgements in $\lambda^{\to}$. Then $\tau = \tau'$.*

This is the uniqueness of types property for $\lambda^{\to}$-Church, see Proposition 3.2.12 in [2].

Definition 45. Let $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$ be an HRS-signature. Let $b : \mathcal{B} \to |\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$ be a function.

A *symbol interpretation* (of Σ w.r.t. b) is a function $\varrho : \mathcal{S} \to \bigcup |\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$ such that $\varrho(s) \in [\![\mathcal{C}(s)]\!]_b$.

A *variable interpretation* is a function $v : \mathcal{V} \to \bigcup |\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$. If v is a variable interpretation and $x \in \mathcal{V}$ then we write $v[x \mapsto y]$ for a variable interpretation v' with $v'(x) = y$ and $v'(x') = v(x')$ if $x \neq x'$. A variable interpretation v is *consistent* w.r.t. Γ and b if $\forall (x, \tau) \in \Gamma.\ v(x) \in [\![\tau]\!]_b$.

An *interpretation* is a pair (ϱ, v) of a symbol interpretation ϱ and a variable interpretation v. An interpretation (ϱ, v) is called *consistent* (w.r.t. to b and Γ) if v is.

In the following, we shall usually suppress the interpretation of base types b, i.e. we assume a fixed b unless otherwise stated. Similarly, we say "consistent w.r.t. Γ" as shorthand for "consistent w.r.t. Γ and b".

Definition 46. Let Σ be an HRS-signature. Let $\rho = (\varrho, \upsilon)$ be an interpretation of Σ. We define a partial function $[\![\_]\!]_\rho : \lambda^{\rightarrow} \rightharpoonup \bigcup |\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$ with domain $\{\Gamma \vdash t : \tau \mid \rho \text{ consistent with } \Gamma\}$ by the following equations:

$$
\begin{array}{rcl}
[\![\Gamma \vdash c : \tau]\!]_\rho & = & \varrho(c) \\
[\![\Gamma \vdash x : \tau]\!]_\rho & = & \upsilon(x) \\
[\![\Gamma \vdash (f\ a) : \tau]\!]_\rho & = & [\![\Gamma \vdash f : \sigma \rightarrow \tau]\!]_\rho([\![\Gamma \vdash a : \sigma]\!]_\rho) \\
 & & \text{where } (\Gamma \vdash a : \sigma) \in \lambda^{\rightarrow} \\
[\![\Gamma \vdash (\lambda x : \sigma.\, t) : \tau]\!]_\rho & = & (z \mapsto \overline{\sigma}(z) \boxplus_\tau [\![\Gamma \cup \{x : \sigma\} \vdash t : \tau]\!]_{\rho[x \mapsto z]}) \\
 & & \text{where } z \in [\![\sigma]\!]_b
\end{array}
$$

To see that this definition is well-formed observe the following properties of semantic interpretation of types and judgements.

Theorem 47.

1. $[\![\Gamma \vdash t : \tau]\!]_\rho \in [\![\tau]\!]_b$
2. *Let $x \in FV(t)$, $z, z' \in [\![\sigma]\!]_b$ and $z >_\sigma z'$. If $\Gamma \vdash t : \tau$ and if $\rho[x \mapsto z]$ is consistent w.r.t. Γ then*
$$[\![\Gamma \vdash t : \tau]\!]_{\rho[x \mapsto z]} >_\tau [\![\Gamma \vdash t : \tau]\!]_{\rho[x \mapsto z']}$$

Proof. By induction over the term structure. For variables and constants the result follows immediately from the assumptions about ρ.

Applications: for some σ, $\Gamma \vdash a : \sigma$ is a derivable judgement, because the domain of $[\![\_]\!]_\rho$ only contains derivable judgements and it does contain $\Gamma \vdash (f\ a) : \tau$. Moreover, proposition 44 claims that σ is unique.

Abstractions: the side-condition $z \in [\![\sigma]\!]_b$ ensures that the new interpretation $\rho[x \mapsto z]$ is consistent w.r.t. the larger context $\Gamma \cup \{x : s\}$. This allows us to apply the induction hypothesis to $\Gamma \cup \{x : \sigma\} \vdash t : \tau$. It remains to show that the constructed function is of the right form, e.g. monotonic. It is clear that it has the right domain and codomain, because $\bar{\sigma}$ maps σ-elements to α and $\boxplus_\tau$ is a function from $\alpha \times \tau$ to τ. For checking monotonicity we use the abbreviation $V(z)$ for $[\![\Gamma \cup \{x : \sigma\} \vdash t : \tau]\!]_{\rho[x \mapsto z]}$. The variable x either occurs free in t or not. If it does, then for $z >_\sigma z'$ we get by induction hypothesis $V(z) >_\tau V(z')$, if it does not $V(z) = V(z')$; thus in both cases $V(z) \geq_\tau V(z')$. The function $\bar{\sigma}$ is monotonic, hence $\bar{\sigma}(z) \ni \bar{\sigma}(z')$ and we get $(\bar{\sigma}(z), V(z)) >_{\boldsymbol{\alpha} \otimes \tau} (\bar{\sigma}(z'), V(z'))$. Since $\boxplus_\tau : \boldsymbol{\alpha} \otimes \tau \rightarrow \tau$ is monotonic we get the required result. □

The chosen interpretation for λ-abstraction may look a bit peculiar, because (as advertised) it does not have the property that β-convertible terms have equal interpretations. Therefore, β-reduction is only of limited use for the meta-level of rewriting.

Definition 48. Let Σ be a signature. A *presubstitution* (over Σ) is a function $\theta : \mathcal{V} \rightarrow \Lambda\Sigma$. Given $t \in \Lambda\Sigma$, we write t^θ for the preterm we get by replacing all free variables in t by their image under θ, avoiding name capture by α-conversion. A *substitution* is a triple (θ, Γ, Δ), written $\theta : \Gamma \rightarrow \Delta$, if θ is a presubstitution and Γ and Δ are contexts (over Σ) such that
$$\forall (x : \tau) \in \Gamma.\ \Delta \vdash \theta(x) : \tau.$$

Lemma 49. *Let $\Gamma \vdash t : \tau$ and $\theta : \Gamma \to \Delta$. Then $\Delta \vdash t^\theta : \tau$.*

This is the standard substitution lemma for $\lambda^\to$, generalised to substitutions that replace all free variables at once. It motivates the following definition:

Definition 50. Let $J = \Gamma \vdash t : \tau$ and $\vartheta = \theta : \Gamma \to \Delta$. We write J^ϑ for the judgement $\Delta \vdash t^\theta : \tau$.

Definition 51. Let $\rho = (\varrho, \upsilon)$ be an interpretation consistent w.r.t. Δ and let $\vartheta = \theta : \Gamma \to \Delta$ be a substitution. We define another interpretation $\rho \circ \vartheta$ as $(\varrho, \upsilon \circ \vartheta)$ where the variable interpretation $\upsilon \circ \vartheta$ is given by:

$$(\upsilon \circ \vartheta)(x) = \begin{cases} [\![\Delta \vdash \theta(x) : \tau]\!]_\rho, & \text{if } (x, \tau) \in \Gamma \\ \upsilon(x), & \text{otherwise} \end{cases}$$

It is easy to see that $\rho \circ \vartheta$ is consistent w.r.t. Δ.

Proposition 52. *Let $J = (\Gamma \vdash t : \tau) \in \lambda^\to$, $\vartheta = \theta : \Gamma \to \Delta$, and ρ be an interpretation consistent w.r.t. Δ. Then $[\![J^\vartheta]\!]_\rho = [\![J]\!]_{\rho \circ \vartheta}$.*

Proof. Straightforward induction over the term structure, similar to lemma 3.15 in [27]. □

Proposition 52 is a typical argument often used in semantic interpretations of the λ-calculus; it does quite happily work with rather non-standard interpretations of λ-abstractions as in our case. However, it does require a standard interpretation of weakening, which in our definition of $\lambda^\to$ is implicit in the rules for variables and constants.

Based on proposition 52 it is easy to show that β-reduction can only make the interpretation smaller, see the following proposition.

Proposition 53. *Let $(\Gamma \vdash t : \tau) \in \lambda^\to$ and $t \to_\beta t'$. Let ρ be an interpretation consistent with Γ. Then $[\![\Gamma \vdash t : \tau]\!]_\rho \geq_\tau [\![\Gamma \vdash t' : \tau]\!]_\rho$.*

Proof. By induction over the term structure of t.

Base case: suppose t is a β-redex $(\lambda x.u)\ a$ and t' its contractum $u[a/x]$; we write $\hat{a}$ as shorthand for $[\![\Gamma \vdash a : \sigma]\!]_\rho$. Using proposition 52 and the algebraic properties of $\boxplus$ we get $[\![\Gamma \vdash t : \tau]\!]_\rho = [\![\Gamma \vdash (\lambda x.u)\ a : \tau]\!]_\rho = \bar{\sigma}(\hat{a}) \boxplus_\tau [\![\Gamma \cup \{x : \sigma\} \vdash u : \tau]\!]_{\rho[x \mapsto \hat{a}]} = \bar{\sigma}(\hat{a}) \boxplus_\tau [\![\Gamma \vdash t' : \tau]\!]_\rho \geq_\tau 0 \boxplus_\tau [\![\Gamma \vdash t' : \tau]\!]_\rho = [\![\Gamma \vdash t' : \tau]\!]_\rho$.

The induction steps are trivial, using the first part of theorem 47. □

5.2 Rules and Their Interpretations

There are various notions of HRSs around. The following is a slight variation of the definitions of van de Pol or Wolfram [27, 31].

Definition 54. Given a signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, a *HRS-rule* is a tuple (Γ, l, r, τ) such that the following conditions hold:

- Γ is a context over Σ.

– $\tau \in \mathcal{B}$

– $\Gamma \vdash l : \tau$ and $\Gamma \vdash r : \tau$ are in $\lambda^{\rightarrow}$.

Notation: we write rules as $\Gamma \vdash l \rightarrow r : \tau$.

The difference to both van de Pol's and Wolfram's definition is that they make the context Γ redundant by requiring the rules to be closed. The condition that τ is a base type does not restrict the expressive power as we can always η-expand rules by adding fresh variables to the context. The reason for using a context rather than λ-abstractions is the interpretation we have chosen for abstractions which does not quite coincide with substitution.

Definition 55. An HRS is a pair (Σ, R) where Σ is a signature and R a set of rules over Σ.

An HRS is associated with an ARS. The elements of this ARS are (derivable) judgements in β-normal form and the relation is given by the following notion of rule application.

Definition 56. Let (Σ, R) be an HRS. For a given a judgement $(\Gamma \vdash C : \sigma) \in \lambda^{\rightarrow}$ a *rule application* is pair $(\vartheta_l, \vartheta_r)$ of substitutions, $\vartheta_l = \theta_l : \Gamma \rightarrow \Delta$ and $\vartheta_r = \theta_r : \Gamma \rightarrow \Delta$, such that for all $(x : \sigma) \in \Gamma$ either $(*)$ $\theta_l(x) = \theta_r(x)$, or $(**)$ there is a rule $(E \vdash l \rightarrow r : \tau) \in R$ with $E = \{y_1 : \sigma_1, \ldots, y_n : \sigma_n\}$, $\sigma = \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \tau$, and $\theta_l(x) = \lambda y_1 : \sigma_1. \cdots \lambda y_n : \sigma_n.\, l$ and $\theta_r(x) = \lambda y_1 : \sigma_1. \cdots \lambda y_n : \sigma_n.\, r$. (Notation: we shall abbreviate this as $\theta_l(x) = \boxed{l}$ and $\theta_r(x) = \boxed{r}$.) A rule application is called *proper* if for at least one $(x : \sigma) \in \Gamma$ we have $x \in \mathrm{FV}(C{\downarrow})$ and $(**)$.

We define a relation $\rightarrow_R$ on β-normal forms of judgements in $\lambda^{\rightarrow}$ as follows: given a judgement $J = (\Gamma \vdash C : \sigma) \in \lambda^{\rightarrow}$ and a proper rule application $(\vartheta_l, \vartheta_r)$ then $J^{\vartheta_l}{\downarrow} \rightarrow_R J^{\vartheta_r}{\downarrow}$.

One can easily check that the above notion gives a more or less canonical definition of HRS reduction; it is slightly more general than the definitions in the literature [31, 27] as it supports reduction with more than one rule at a time, but for the purposes of termination proofs this difference is insignificant.

The reason for requiring properness is the following lemma:

Lemma 57. *Let J be judgement in $\lambda^{\rightarrow}$ and $(\vartheta_l, \vartheta_r)$ be a rule application which is not proper. Then $J^{\vartheta_l}{\downarrow} = J^{\vartheta_r}{\downarrow}$.*

Proof. For any substitution ϑ, $J^{\vartheta}{\downarrow} = (J{\downarrow}^{\vartheta}){\downarrow}$. For $(\vartheta_l, \vartheta_r)$ to be not proper means that they only differ on variables which are not free in $J{\downarrow}$. □

Therefore we need properness to give $\rightarrow_R$ a chance to be terminating. Any approach attempting to reason about termination of HRS reduction has to make similar restrictions in the definition of its reduction relation $\rightarrow_R$.

It is often convenient to assume that a rule application only instantiates one rule at one particular position in a term. We can define this as follows:

Definition 58. A rule application $(\vartheta_l, \vartheta_r)$ for a judgement J is called *linear* if $\vartheta_l(y) = \vartheta_r(y)$ for all but one y from the context of J, and if $\vartheta_l(x) = \boxed{l}$ and $\vartheta_r(x) = \boxed{r}$ then x occurs at most once in $J\!\downarrow$.

For termination proofs it is sufficient to consider only linear rule applications as the following lemma shows.

Lemma 59. *Let $J \to_R J'$. Then there are judgements $J_1, \ldots, J_n \in \lambda^{\to}$ such that $J = J_1 \to_R J_2 \to_R \cdots \to_R J_n = J'$ where each $J_i \to_R J_{i+1}$ by a linear rule application.*

Proof. Let $J = C^{\vartheta_l}$ and $J' = C^{\vartheta_r}$ where $(\vartheta_l, \vartheta_r)$ is a proper rule application for the judgement $C \in \lambda^{\to}$. W.l.o.g. we can assume that any free variable in C occurs not more than once in $C\!\downarrow$. Let $\{x_1, \ldots, x_k\}$ be the set of all free variables in C. We define rule applications $(\theta_{li}, \theta_{ri}), 1 \leq i < k$ for C as follows:

$$\begin{aligned} \theta_{li}(x_j) &= \begin{cases} \theta_l(x_j) \text{ if } i \leq j \\ \theta_r(x_j) \text{ if } i > j \end{cases} \\ \theta_{ri}(x_j) &= \begin{cases} \theta_l(x_j) \text{ if } i < j \\ \theta_r(x_j) \text{ if } i \geq j \end{cases} \end{aligned}$$

Clearly all rule applications are linear and moreover $\vartheta_{li+1} = \vartheta_{ri}$ and $\vartheta_l = \vartheta_{l1}$ and $\vartheta_r = \vartheta_{rk}$. Using lemma 57 we can simply discard all non-proper rule applications and get the desired result. $\square$

Lemma 59 is perhaps slightly surprising as it shows that even the nested application of HRS-rules does not increase the expressive power of many-step reduction.

Definition 60. Let ϱ be a symbol interpretation. A rule $\Gamma \vdash l \to r : \tau$ is called *ϱ-decreasing* if for all substitutions $\theta : \Gamma \to \Delta$ and all variable interpretations υ that are consistent with Δ it is true that $[\![\Delta \vdash l^{\theta}\!\downarrow : \tau]\!]_{(\varrho,\upsilon)} >_{\tau} [\![\Delta \vdash r^{\theta}\!\downarrow : \tau]\!]_{(\varrho,\upsilon)}$.

Theorem 61. *Let (Σ, R) be a HRS. If ϱ is a symbol interpretation such that all rules in R are ϱ-decreasing then $(\boldsymbol{\lambda^{\to}}, \boldsymbol{\to_R}) \models \mathrm{SN}$.*

Proof. We simply prove that $J \to_R J'$ implies $[\![J]\!]_{\rho} >_{\tau} [\![J']\!]_{\rho}$ where $J = B \vdash t : \tau$ and $\rho = (\varrho, \upsilon)$ for some variable interpretation υ consistent with B. By lemma 59 it is sufficient to consider the case in which $J \to_R J'$ by a linear rule application.

Suppose we get $J \to_R J'$ by applying the proper rule application $(\vartheta_l, \vartheta_r)$ to the judgement $C = E \vdash c : \sigma$. We can assume w.l.o.g. that $c = c\!\downarrow$. We show $[\![E \vdash c : \sigma]\!]_{\rho\circ\vartheta_l} >_{\sigma} [\![E \vdash c : \sigma]\!]_{\rho\circ\vartheta_r}$ by induction on the term structure of c.

If c is a symbol then we have a contradiction because the rule application cannot be proper.

If c is a variable then by properness $\theta_l(c) = \boxed{l}$ and $\theta_r(c) = \boxed{r}$. By assumption, the rule $\{y_1 : \sigma_1, \ldots, y_n : \sigma_n\} \vdash l \to r : \tau$ is ϱ-decreasing and from monotonicity of $n \boxplus \_$ for fixed n we conclude $[\![\theta_l(c)]\!]_{\rho} >_{\sigma} [\![\theta_r(c)]\!]_{\rho}$.

If c is a λ-abstraction $\lambda x : \tau.t$ then we define a new rule application $(\vartheta'_l, \vartheta'_r)$ for the judgement $E \cup \{x : \tau\} \vdash t : \sigma'$ which is as the old one except $\vartheta'_l(x) = x = \vartheta'_r(x)$. Obviously this rule application is still proper. By induction hypothesis we have $[\![E \cup \{x :$

$\tau\} \vdash t : \sigma']\!]_{\rho\circ\vartheta'_l} >_{\sigma'} [\![E \cup \{x : \tau\} \vdash t : \sigma']\!]_{\rho\circ\vartheta'_r}$. Using again monotonicity of $n \boxplus \_$ we get the result for c.

If c is an application $t\, t_1 \cdots t_n$ $(n \geq 1)$ such that t is not an application, then t is either a constant or a variable since we assumed c to be in β-normal form. If t is either a constant or a variable that is not instantiated to a rule by the rule application, then $t^{\theta_l} = t^{\theta_r}$ and the rule application is still proper for at least one of the judgements $E \vdash t_i : \sigma_i$. We can apply either the induction hypothesis or lemma 57 to the t_i. By monotonicity of $[\![t^{\theta_l}]\!]_\rho$ the result follows. If t is instantiated to a rule $\Gamma \vdash l \to r : \tau$ with $\Gamma = \{y_1 : \sigma_1, \ldots, y_k : \sigma_k\}$ then we have (if $k = n$) $c^{\theta_l}\!\downarrow = (t\, t_1 \cdots t_n)^{\theta_l}\!\downarrow = l^{\theta'_l}\!\downarrow$ where $\theta'_l : \Gamma \to \Delta$ is a substitution with $\theta'_l(y_i) = t_i^{\theta_l}\!\downarrow$. The same argument applies to $c^{\theta_r}\!\downarrow$, giving us another substitution $\theta'_r : \Gamma \to \Delta$. Since we assumed the rule application to be linear, it is not proper for the judgements $J_i = E \vdash t_i : \sigma_i$. Thus $t_i^{\theta_l} = t_i^{\theta_r}$ by lemma 57 and we have $\theta'_l = \theta'_r$. We conclude $[\![l^{\theta'_l}\!\downarrow]\!] >_\tau [\![r^{\theta'_l}\!\downarrow]\!] = [\![r^{\theta'_r}\!\downarrow]\!]$ because the rule is ϱ-decreasing. If $k \neq n$, i.e. if the number (k) of arguments of the rule differs from the number (n) of arguments of t then $n < k$ because τ is a base type. In this case, the abstractions $\lambda y_{n+1} : \sigma_{n+1}. \cdots \lambda y_k : \sigma_k$ remain and we have to show that $[\![\lambda y_{n+1}. \cdots \lambda y_k.\, l^{\theta'_l}\!\downarrow]\!]_\rho >_{\sigma_{n+1}\to\cdots\to\sigma_k\to\tau} [\![\lambda y_{n+1}. \cdots \lambda y_k.\, r^{\theta'_l}\!\downarrow]\!]_\rho$. Again we can use the ϱ-decreasing argument and get the result from the monotonicity of $\boxplus$. $\square$

6 Applying the method

We can apply theorem 61 to show that a given HRS is terminating. For this we need the following ingredients:

- an interpretation b for any base type
- a symbol interpretation ϱ consistent w.r.t. b and
- a proof that each rule is ϱ-decreasing.

To see how this works let us consider an example typical for the kind of rewrite systems we have in mind.

Suppose we want to define an enriched λ-calculus $\lambda^{\to,\times,+}$ with products and coproducts over some set of elementary types, see for example [30]. The first problem we have is that this is not quite an HRS, because we have product and coproduct types that can carry function types. The solution is to consider all types of this calculus to be base types and build $\lambda^{\to}$ on top of it. We get a function b from base types to objects in $\mathsf{WO}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}$ by mapping each elementary type to $\boldsymbol{\alpha}$ (in the example, any indecomposable limit ordinal will do) and each type constructor to some functor; in this case we can take the corresponding functors from theorem 35.

We now have two arrow type constructors: $\sigma \Rightarrow \tau$ on object level, i.e. σ and τ must be base ($\equiv$ object level) types too, and $\sigma \to \tau$ where σ and τ may be non-basic. Two arrow types also require two applications and two λ-abstractions — for the object-level these can be represented by symbol applications. We also need different symbols on different base types, e.g. one object-level application symbol for every combination of argument and result type. Notationally I will suppress this difference, but we have to keep in mind that each symbol carries its type information. In particular, we have infinitely many symbols and rules.

$$\begin{aligned}
\texttt{AP}\;(\texttt{LAM}\;f)\;a &\to f\;a && (1)\\
\texttt{LAM}\;(\texttt{AP}\;f) &\to f && (2)\\
\texttt{FST}\;(\texttt{PAIR}\;x\;y) &\to x && (3)\\
\texttt{SND}\;(\texttt{PAIR}\;x\;y) &\to y && (4)\\
\texttt{PAIR}\;(\texttt{FST}\;p)\;(\texttt{SND}\;p) &\to p && (5)\\
\texttt{CASE}\;(\texttt{INL}\;x)\;f\;g &\to f\;x && (6)\\
\texttt{CASE}\;(\texttt{INR}\;x)\;f\;g &\to g\;x && (7)\\
\texttt{CASE}\;c\;(\lambda x:\sigma.\,f\;(\texttt{INL}\;x))\;(\lambda y:\tau.\,f\;(\texttt{INR}\;y)) &\to f\;c && (8)
\end{aligned}$$

Figure 7: Rules for $\lambda^{\to,\times,+}$

The rewrite rules for β- and η-reduction of this calculus are shown in figure 7. Each rule is only a schema for infinitely many rules of the same shape for each combination of base types, so we have to look for a corresponding schematic interpretation of the schematic symbols. The easiest case are the products as they are purely first-order:

$$\begin{aligned}
\varrho(\texttt{PAIR}_{\sigma,\tau})\;x\;y &= ((1\#\bar{\tau}(y))\boxplus_\sigma x, (1\#\bar{\sigma}(x))\boxplus_\tau y)\\
\varrho(\texttt{FST}_{\sigma,\tau})\;(x,y) &= x\\
\varrho(\texttt{SND}_{\sigma,\tau})\;(x,y) &= y
\end{aligned}$$

The three rules involving products (3–5) are clearly ϱ-decreasing, because we get $(1\#\bar{\tau}(y))\boxplus_\sigma x >_\sigma x$, $(1\#\bar{\sigma}(x))\boxplus_\tau y >_\tau y$, and $((1\#\bar{\tau}(y))\boxplus_\sigma x, (1\#\bar{\sigma}(x))\boxplus_\sigma y) >_{\sigma\sqcap\tau} (x,y)$, respectively. The only problem was to define $\varrho(\texttt{PAIR})$ in such a way that it is a monotonic function in $\sigma \Rightarrow \tau \Rightarrow (\sigma \sqcap \tau)$. Alternatively, we could have interpreted the object-level product type by the tensor product, making the interpretation of PAIR simple, but then the projections would be slightly more complicated. In that case, $\texttt{FST}_{\sigma,\tau}$ could be interpreted as $\lambda(x,y).\bar{\tau}(y)\boxplus_\sigma x$, etc.

This was not too difficult but also very typical: to get larger values on the left-hand sides the symbols have to make some "$1\boxplus \_$" noise and to deal with erasing rules they have to garbage-collect the erased term using the addition operator. For the product rules we could ignore the effect of β-normalisation: without loss of generality we can assume that θ maps variables to normal forms, and in that case substitution application does not create β-redexes for the three product rules making β-normalisation the identity.

For function types we do essentially the same thing, but now some meta-level β-reduction can take place:

$$\begin{aligned}
\varrho(\texttt{LAM}_{\sigma,\tau})\;f &= 1\boxplus_{\sigma\Rightarrow\tau} f\\
\varrho(\texttt{AP}_{\sigma,\tau})\;f\;a &= f(a)
\end{aligned}$$

For the η-rule (2) we have the same argument as in the product case, because again there are no β-redexes created by substitution application. The β-rule (1) is only a little bit trickier: $[\![(\texttt{AP}\;(\texttt{LAM}\;f)\;a)^\theta\!\downarrow]\!] = [\![\texttt{AP}\;(\texttt{LAM}\;f^\theta)\;a^\theta]\!] = [\![\texttt{LAM}\;f^\theta]\!]([\![a^\theta]\!]) = (1\boxplus_{\sigma\Rightarrow\tau}[\![f^\theta]\!])([\![a^\theta]\!]) >_\tau [\![f^\theta]\!]([\![a^\theta]\!]) = [\![(f^\theta\;\;a^\theta)]\!] \geq_\tau [\![(f^\theta\;\;a^\theta)\!\downarrow]\!]$. The last step used proposition 53. This shows that we can ignore β-normalisation as long as it only affects the right-hand sides of rules.

The only rewrite rule in our example in which substitution application can create a β-redex on the left-hand side is rule 8, the η-rule for coproducts. Finding the interpretation for the coproduct type is similarly simple, but again we have to be careful to make $\varrho(\texttt{CASE})$ monotonic by collecting the garbage:

$$\begin{array}{rcl}
\varrho(\texttt{INL}_{\sigma,\tau})\ x &=& i_1(x) \\
\varrho(\texttt{INR}_{\sigma,\tau})\ y &=& i_2(x) \\
\varrho(\texttt{CASE}_{\sigma,\tau,\nu})\ (i_1(x))\ f\ g &=& 1 \boxplus_\nu \overline{\tau \Rightarrow \nu}(g) \boxplus_\nu f(x) \\
\varrho(\texttt{CASE}_{\sigma,\tau,\nu})\ (i_2(y))\ f\ g &=& 1 \boxplus_\nu \overline{\sigma \Rightarrow \nu}(f) \boxplus_\nu g(y)
\end{array}$$

Showing that rules 6 and 7 are ϱ-decreasing is straightforward (as for the function type), but the last rule is a bit more problematic as we have meta-level β-reduction on both sides of the rule. Here, we cannot rely any longer on proposition 53 but have to consider the exact interpretation of λ-terms. The variable f is second-order, i.e. we reach the β-normal form of $(f^\theta\ c^\theta)$ in a single β-step. For an arbitrary substitution $\theta(f) = \lambda x : \sigma + \tau.\ t$ we get as interpretation of the left-hand side of rule 8: if $[\![\theta(c)]\!] = i_1(c')$ then $1 \boxplus_\nu \overline{\tau \Rightarrow \nu}(g') \boxplus_\nu \overline{\sigma}(c') \boxplus_\nu [\![t[\texttt{INL}\ y/x]]\!]_{y \mapsto c'} \geq_\nu 1 \boxplus_\nu [\![t[\texttt{INL}\ y/x]]\!]_{y \mapsto c'} >_\nu [\![t[\texttt{INL}\ y/x]]\!]_{y \mapsto c'} = [\![t[c^\theta/x]]\!] = [\![(f\ c)^\theta{\downarrow}]\!]$. Here we used proposition 52 to compose an interpretation with a substitution. The case of the right injection is dual.

Gandy's paper [7] also considers the same example but he cannot directly show with his method the termination of the η-rule for the coproduct. Van de Pol [27] can deal with coproducts but not with internalised function types on the object-level.

7 Comparison to other work

Two other approaches for termination proofs for enriched λ-calculi by semantical methods are closely related to the work presented here.

Robin Gandy presented in [7] a similar solution for termination proofs of enriched λ-calculi. As in the present paper, Gandy also provides type conversion operations and addition functions for all types and he uses the same idea of interpreting arbitrary λ-terms via the λI-calculus. I only heard of Gandy's paper *after* my paper was already half-finished; in other words, I rediscovered his method independently. The problem with Gandy's paper is that the method appears superficially just as a bag of tricks, it is not clear what the underlying semantical framework is and what else might be in this bag. By pushing the proofs to a more abstract level (arbitrary symmetric monoidal closed categories) this should be much clearer now, e.g. the generalisation from natural numbers (as used by Gandy) to arbitrarily large ordinals is a direct consequence from making the move from natural numbers plus addition to categorical monoids.

As HRSs had not been invented yet in 1980, the difficulty of providing the syntactical framework for the method can also be felt. For example, Gandy could not treat the η-rule for coproducts (see section 6) with his method, but this failure was merely due to the missing foundations of the method, i.e. it was not obvious though possible. Our paper is based on the same fundamental ideas as Gandy's, but we go much deeper into the analysis of the semantics.

This paper was inspired by Jaco van de Pol's talk at HOA'93 [27] and the difficulties I encountered when trying to apply his method. During one week it happened twice that collegues popped into my office asking me, as I am considered the local rewrite guru, about the termination of some "rewrite system" which in both cases turned out to be some enriched λ-calculus, i.e. an HRS. So I gave both a copy of van de Pol's paper, remarking "just apply the recipe", but soon realised that they were rather unhappy with this advice.

Van de Pol interprets β-equivalent terms equally. This has the advantage that the method remains solely semantical, while I combine semantic interpretation with substitution. The price he pays for this elegance is a rather ugly semantic domain in which functions are split into three classes of monotonic, strictly monotonic, and "strict" functions. For a categorical treatment these "strict" functions are problematic as e.g. the identity function on higher types is not "strict". The "strict" functions are used to interpret contexts. When applying van de Pol's method, the user has to provide a strict interpretation, i.e. interpret all function symbols by strict functions since every function symbol can form a context.

Van de Pol's approach is not very appealing for everyday termination proofs. The semantic set-up is not only rather involved, which is seemingly unavoidable, but it also does not hide the complication of the set-up from the user. Each type is interpreted by two sets ($S \subseteq M$), one partial order and two terminating transitive relations satisfying a number of coherence conditions. The maps are interpreted in two stages: the maps of the first stage only have to preserve the partial order, the maps of the second stage (the mentioned "strict" functions) must preserve all three orders with the additional condition that a map $f : (S \subseteq M) \to (S' \subseteq M')$ is a function $f : M \to S'$. It is not quite clear to me yet if and how his ideas can be given the categorical treatment, making it flexible enough for the odd modification without the need of reproving all results. On the other hand, it is surely possible to export some of the results in this paper to van de Pol's approach extending the range of rewrite systems it can be applied to.

While Gandy indicates how his method could be extended to other parametric types, van de Pol restricts his attention (in [27]; this may have developed since then as he has a joint paper with Schwichtenberg at TLCA'95) to atomic types. Both papers solely dealt with natural numbers as the domain for atomic types. This is tempting but problematic as addition has certain properties on ω (which one likes to exploit) that do not generalise to higher ordinals. Most prominently, the function $\#$ is the smallest strictly monotonic function in $\omega \otimes \omega \to \omega$, but it loses this property on larger ordinals.

8 Conclusion and further work

We have presented another semantic approach to termination proofs for higher-order term rewriting systems. It consists of an extended foundation for termination proofs in general and a method that links the semantic foundation with the syntax of Higher-Order Rewrite Systems. The application of the method is fairly simple and one does not have to understand why the semantic foundation works to apply it to an example. In particular, we have a number of criteria that allow us to deal with parametric types: any functor that is "retractable" at certain ordinals and "$\boldsymbol{M}$-distributive" is a candidate for the interpretation of parametric types.

The essence of the semantic set-up is to provide order-preserving functions $\boxplus_\tau$: $\boldsymbol{\alpha} \otimes [\![\tau]\!] \to [\![\tau]\!]$ and $\overline{\tau} : [\![\tau]\!] \to \boldsymbol{\alpha}$ for any type τ, where $\boldsymbol{\alpha}$ is some arbitrarily large ordinal, such that $n \boxplus x \geq x$ for any $x \in [\![\tau]\!]$, $n \in \boldsymbol{\alpha}$. These are not all properties of $\boxplus$ and $\overline{\tau}$, but the most important ones, which on their own suffice to tackle termination proofs for many rewrite systems.

The method as presented here does not apply to (term-) dependent types or (type-) dependent terms, e.g. to show termination of the calculus of constructions. The main difficulty is that functors only correpond to parametric types, they do not model other dependencies of types. Thus one has to use a more sophisticated semantic construct than functors to mirror the dependency of the syntax.

Acknowledgements

This paper evolved over some period of time. People who helped developing it by providing feedback, advice, references, email, etc. include (in alphabetical order): Randall Dougherty, Marcello Fiore, Philippa Gardner, James McKinna, Jaco van de Pol, John Power, Alex Simpson and Judith Underwood. I would also like to thank the RTA'95 referees for comments and corrections.

The figures were drawn with Paul Taylor's diagram package.

References

[1] Heinz Bachmann. *Transfinite Zahlen.* Springer, 1955.

[2] Hendrik P. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science, Vol.2*, pages 117–309. Oxford Science Publications, 1992.

[3] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[4] Val Breazu-Tannen and Jean Gallier. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, pages 137–150, 1989.

[5] M. C. F. Ferreira and H. Zantema. Total termination of term rewriting. In *Rewriting Techniques and Applications*, pages 213–227, 1993. LNCS 690.

[6] Abraham A. Fraenkel. *Abstract Set Theory.* North-Holland, 1966.

[7] R.O. Gandy. Proofs of strong normalization. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.

[8] C.A. Gunter and D.S. Scott. Semantic domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, chapter 12, pages 633–674. Elsevier, 1990.

[9] Gerhard Hessenberg. Grundbegriffe der Mengenlehre. *Abhandlungen der Fries'schen Schule*, pages 479–706, 1906.

[10] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-Calculus.* Cambridge University Press, 1986.

[11] Ernst Jacobsthal. Über den Aufbau der transfiniten Arithmetik. *Mathematische Annalen*, 67:130–144, 1909.

[12] Stefan Kahrs. Context rewriting. In M. Rusinowitch and J.L. Rémy, editors, *Conditional Term Rewriting Systems*, pages 21–35, 1992. LNCS 656.

[13] Stefan Kahrs. Confluence of curried term-rewriting systems. (submitted), 1994.

[14] E. Kamke. *Theory of Sets.* Dover Publications, Inc., 1950.

[15] Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Comparing curried and uncurried rewriting. In Henk Barendregt, Marc Bezem, and Jan Willem Klop, editors, *Dirk van Dalen Festschrift*, pages 57–78. Department of Philosophy, Utrecht University, 1993.

[16] Jan Willem Klop. *Combinatory Reduction Systems.* PhD thesis, Centrum voor Wiskunde en Informatica, 1980.

[17] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[18] Kenneth Kunen. *Set Theory.* North-Holland, 1980.

[19] Pierre Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitution. In *21st ACM Symposium on Principles of Programming Languages*, pages 60–69, 1994.

[20] Saunders MacLane. *Categories for the Working Mathematician.* Springer, 1971.

[21] Rob Nederpelt and Fairouz Kamareddine. On stepwise explicit substitution. *International Journal on Foundations of Computer Science*, 4(3):197–240, 1993.

[22] John von Neumann. Zur Einführung der transfiniten Zahlen. *Acta litterarum ac scientarum*, 1:199–208, 1923.

[23] Tobias Nipkow. Higher order critical pairs. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 342–349, 1991.

[24] Enno Ohlebusch. On the modularity of confluence of constructor-sharing term rewriting systems. In *Proceedings of CAAP'94*, pages 261–275. Springer, LNCS 787, 1994.

[25] Vincent van Oostrom and Femke van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In *Higher-Order Algebra, Logic, and Term Rewriting*, pages 276–304, 1993. LNCS 816.

[26] Axel Poigné. Basic category theory. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 1*, pages 413–640. Oxford University Press, 1992.

[27] Jaco van de Pol. Termination proofs for higher-order rewrite systems. In *Higher-Order Algebra, Logic, and Term Rewriting*, pages 305–325, 1993. LNCS 816.

[28] Michael D. Potter. *Sets: An Introduction.* Oxford University Press, 1990.

[29] Raphael M. Robinson. The theory of classes, a modification of von Neumann's system. *The Journal of Symbolic Logic*, 2:29–36, 1937.

[30] Simon Thompson. *Type Theory and Functional Programming.* Addison-Wesley, 1991.

[31] D.A. Wolfram. *The Clausal Theory of Types.* Cambridge University Press, 1993.

[32] Hans Zantema. Termination of term rewriting by interpretation. In *CTRS*, pages 155–167, 1992. LNCS 656.

[33] Hans Zantema. Type removal in term rewriting. In *CTRS*, pages 148–154, 1992. LNCS 656.

[34] Hans Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):3–21, 1994.