Termination Proofs in an Abstract Setting

Stefan Kahrs
University of Kent at Canterbury,
Computing Laboratory
Canterbury CT2 7NF

Abstract

We present a general framework for termination proofs. These are termination proofs of Higher-Order Rewrite Systems, but one can equally apply the framework to programs written in (pure) functional programming languages. We borrow some ideas from domain theory in that we interpret each type with a semantic "domain" — an object in some category with sufficient structure for basic termination proofs — and each type constructor with a functor of the corresponding arity.

Termination is a property of binary relation on some set, in rewrite speak: Abstract Reduction Systems. This simple insight can be extended to termination proofs: a termination proof is a morphism between Abstract Reduction Systems where the target system is a (canonical) well-ordered set. Therefore, the category ARS is our starting point when we look for suitable notions of "termination domains".

1 Introduction and Outline

Given some set of values A and a binary relation $\rightarrow_R$ on A, we can ask the question whether the relation terminates: is there any function $a : \omega \rightarrow A$ such that for all $n \in \omega$ we have $a(n) \rightarrow_R a(n+1)$? If there is no such function then $\rightarrow_R$ is said to be terminating.

This definition of terminating relations is by no means different from the usual notions of termination for Turing-Machines, Term Rewriting Systems, or even programs in some programming language, but it strips "termination" to what it really is: a property of relations.

This is not a very deep but useful insight. The presence of termination at the level of relations is often used as an induction principle to prove other properties of relations. However, showing termination *itself* is usually confined to a more concrete setting, in which our set A has extra structure. But it is not difficult to see that at least termination proofs also exist on this abstract level.

1.1 Abstract Termination Proofs

An Abstract Reduction System (ARS) is just a pair $(A, \to_A)$ where A is a set and $\to_A$ a binary relation on A. An ARS morphism is a structure-preserving function, i.e. an ARS morphism between $\boldsymbol{A} = (A, \to_A)$ and $\boldsymbol{B} = (B, \to_B)$ is a function $f : A \to B$ such that $x \to_A y$ implies $f(x) \to_B f(y)$.

This is by no means the only useful notion of ARS morphism, see e.g. [43] which uses the stronger notion of functional bisimulation. Our choice was guided by the following observation.

Suppose we have two ARSs $\boldsymbol{A}$ and $\boldsymbol{B}$ and an ARS morphism $f : \boldsymbol{A} \to \boldsymbol{B}$. Since f preserves chains of rewrite steps we have that if $\boldsymbol{B}$ is terminating then so is $\boldsymbol{A}$. We can turn that around: given the knowledge that $\boldsymbol{B}$ is terminating, we can *prove* the termination of $\boldsymbol{A}$ by constructing an ARS morphism from $\boldsymbol{A}$ to $\boldsymbol{B}$. This observation alone does not get us very far, because we still have to establish the termination of $\boldsymbol{B}$, which in turn requires us to find some morphism g to some terminating ARS $\boldsymbol{C}$, etc. We need some elementarily terminating ARSs to halt this process. For this purpose we can choose (representatives of) ordinals as targets of termination proofs. One may claim[1] that every termination proof can be brought into this form.

To exploit the existence of termination proofs in this abstract setting we need some operations on termination proofs that allow us to construct termination proofs in a more flexible manner than just by composition. Which operations we have to our disposition is determined by abstract properties of the category ARS of Abstract Reduction Systems and their morphisms. Which operations are of interest depends on the area of application we have in mind.

1.2 Application Area

A typical application area are first-order and higher-order term rewriting systems. Some of the properties of term rewriting can tell us about the range of interpretations for which a termination proof may succeed.

For *typed* rewrite systems (or programs) we typically have the subject reduction property: if t has type τ and t reduces to u then u has type τ as well. This means that it suffices to interpret a program by a family of ARSs, one for every type. Moreover, since many typed frameworks support type constructors (such as the function space constructor) to build new types from old ones and thus have infinitely many types we would need to interpret type constructors accordingly, as endofunctors on our category. This is exactly the same as in domain theory: interpret types as objects, terms as morphisms, type constructors as functors; just the base categories change.

Notice that our type constructors need to be interpreted by *endo*-functors, since our semantic type constructors need to preserve whichever structure we require for "termination domains". We may have to put more structure into our termination domains than we need for the actual termination proofs themselves,

[1] Intuitionists may dispute this claim but we merely use it as a further endorsement on the meta-level — there is no technical reliance upon that property.

just in order to have enough information around to build the (needed) structure for the composite domains. In particular, just the information that our objects of concern in ARS are terminating is in itself insufficient to build function type objects; none of the monoidal closed structures on ARS we consider is closed on terminating objects.

Let us be a bit more concrete. Any unary function symbol F in a term rewriting system has the property that $t \to_R u$ implies $F(t) \to_R F(u)$. But this means that F, viewed as a function in the term algebra, is itself an ARS morphism. The same can be said for function symbols of arbitrary arity, though we have to be careful with the interpretation of tupling: a pair (a_1, b_1) rewrites to a pair (a_2, b_2) if $a_1 \to_R a_2$ and $b_1 = b_2$, or if $a_1 = a_2$ and $b_1 \to_R b_2$. This mirrors the definition of the single-step rewrite relation which allows the rewriting of exactly one subterm. In fact, we can define these tuples and their relation via a bifunctor $\_ \otimes \_$ on ARS, which we shall see later.

Being a bifunctor, $\_ \otimes \_$ not only operates on ARSs, the objects in the category ARS, but also on ARS morphisms. In particular, we can use it to preserve termination proofs: Let $f : \boldsymbol{A} \to \alpha$ and $g : \boldsymbol{B} \to \beta$ be termination proofs, then we get a map $f \otimes g : \boldsymbol{A} \otimes \boldsymbol{B} \to \alpha \otimes \beta$. One can always construct an ordinal γ with a termination proof for $\alpha \otimes \beta$, and pre-composition with $f \otimes g$ then gives us the required termination proof for $\boldsymbol{A} \otimes \boldsymbol{B}$. This shows that $\_ \otimes \_$ is closed on terminating objects.

Things are not quite so smooth when it comes to higher-order rewrite systems [30, 35]. For first-order TRSs, the previous considerations show that it is convenient to view a many-sorted TRS as a family of TRSs, indexed by their type. For higher-order rewrite systems we would like to develop this idea and define an ARS $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ for two given ARSs $\boldsymbol{A}$ and $\boldsymbol{B}$, playing a similar rôle as the previously mentioned $\boldsymbol{A} \otimes \boldsymbol{B}$. We have a natural choice here by defining $\boldsymbol{A} \Rightarrow \_$ to be the right adjoint of $\_ \otimes \boldsymbol{A}$. This right adjoint exists and a routine calculation shows that (i) $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ contains the ARS morphisms from $\boldsymbol{A}$ to $\boldsymbol{B}$, and (ii) two morphisms f, g are related via the relation of $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ iff $\forall x \in A. f(x) \to_B g(x)$.

Again, $\_ \Rightarrow \_$ is a functor and so we can check whether its morphism part preserves termination proofs. The ARS $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ inherits its relation from $\boldsymbol{B}$ and so we do not need a termination proof for $\boldsymbol{A}$. However, we do need that $\boldsymbol{A}$ is inhabited, because if $\boldsymbol{A}$ is the empty ARS $(\emptyset, \emptyset)$ then $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ contains as its sole element the empty map which *is related to itself*, making the relation on $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ non-terminating.

Fortunately, inhabitance proofs for ARSs also live in ARS. An ARS $\boldsymbol{A}$ is non-empty iff there is a morphism $n : \mathbf{1} \to \boldsymbol{A}$ where $\mathbf{1}$ is the ordinal 1, the object $(\{0\}, \emptyset)$[2]. Given an inhabitance proof $n : \mathbf{1} \to \boldsymbol{A}$ and a termination proof $f : \boldsymbol{B} \to \alpha$ we can apply the $\_ \Rightarrow \_$ functor and get a morphism $n \Rightarrow f$ from $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ to $\mathbf{1} \Rightarrow \alpha$; by post-composing this with the isomorhism $\iota : (\mathbf{1} \Rightarrow \alpha) \cong \alpha$ we obtain the desired termination proof.

Unfortunately, this leaves us with the new problem of having to take care

[2]This is different from the terminal object in ARS which is $(\{0\}, \{(0,0)\})$

of inhabitance proofs. This is a bit awkward though possible when we restrict the method to specific termination proofs and inhabitance proofs. We shall see later how this works. For the time being it suffices to know that we look for a specific inhabitance proof $n : \alpha \to \boldsymbol{A}$ dual to the termination proof $f : \boldsymbol{A} \to \alpha$.

What we have so far is a method to construct terminating ARSs from others which follows the term-construction principles of TRSs. This means that if we have an interpretation $[\![\_]\!]$ of a TRS's signature in such an ARS $(A, \to_A)$ we only have to show that $t \to_R u$ implies $[\![t]\!] \to_A [\![u]\!]$. Ideally, it should suffice to show this property for the rewrite rules themselves, i.e. the termination proof becomes substantially simpler if substitution and context application are monotonic w.r.t. the ARS relations.

This is indeed the way one can proceed for first-order TRSs [51], but there are some problems with higher-order systems, which are closely linked to the abstract properties of the category ARS.

1.3 Application Examples

The data we have so far gives us the range in which we can search for interpretations, in other words: it gives us the interpretation of types. In general, this is not enough to find a corresponding interpretation for the function symbols, except for simple examples, or examples which we can interpret directly in an ordinal, for instance:

$$\begin{array}{rcl} A(0,n) & \to & S(n) \\ A(S(m),0) & \to & A(m,S(0)) \\ A(S(m),S(n)) & \to & A(m,A(S(m),n)) \end{array}$$

This is the well-known Ackermann function, written as a first-order TRS. We can prove the termination of this rewrite system by interpreting the terms in the ordinal ϵ_0 using the interpretation:

$$\begin{array}{rcl} [\![0]\!] & = & 0 \\ [\![S]\!](x) & = & x+1 \\ [\![A]\!](x,y) & = & \omega^{(\omega^x)} * (y+1) + x + 1 \end{array}$$

One has to check two properties: (i) the interpretation of the symbols lies in the semantic domain, i.e. $[\![S]\!]$ and $[\![A]\!]$ are monotonic functions in $\epsilon_0 \to \epsilon_0$ and $\epsilon_0 \otimes \epsilon_0 \to \epsilon_0$, respectively; (ii) the three left-hand sides are larger under the interpretation than the corresponding right-hand sides. Both checks are routine exercises in ordinal arithmetic which I might recommend to the reader.

We cannot always find an interpretation in an ARS where the relation is the strict part of a total ordering and thus we do not always have ordinal arithmetic at our disposal. The standard example that shows the need for partial orderings is:

$$\begin{array}{rcl} F(C) & \to & F(D) \\ G(D) & \to & G(C) \end{array}$$

The only way we can interpret this terminating TRS in a totally ordered ARS is to choose $[\![C]\!] = [\![D]\!]$, but then the order cannot be irreflexive. We can construct a termination proof directly into the ordinal $\mathbf{2}$, by mapping any term to 1 that contains either $F(C)$ or $G(D)$ as a subterm, and any other term to 0. If the signature contains "idle" symbols of arity ≥ 2 then we need a larger ordinal ($\boldsymbol{\omega}$) for the termination proof: we can interpret each term by the number of occurrences of $F(C)$ or $G(D)$ as subterms.

Even if the codomain of our interpretation is not totally ordered, we can still import ordinal arithmetic via the maps from and into α that we use to show inhabitance and termination.

However, this is not enough in general. Most typically, we need an operation that allows us to add "rewrite noise" to the interpretation of terms of arbitrary types. Consider the following example:

$$\pi_1(x, y) \to x$$

If x and y are interpreted in the terminating ARSs $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, then we have to find an interpretation for π_1 in $\boldsymbol{A} \otimes \boldsymbol{B} \Rightarrow \boldsymbol{A}$.

We cannot use the first projection itself, because $\otimes$ is not a categorical product: we need that $y_1 \to_B y_2$ implies $[\![\pi_1]\!](x, y_1) \to_A [\![\pi_1]\!](x, y_2)$ and we cannot have $x \to_A x$ if the relation $\to_A$ is terminating. Thus, the first projection is not even in the semantic domain. Using inhabitance and termination, we can construct a map in $\boldsymbol{A} \otimes \boldsymbol{B} \Rightarrow \boldsymbol{A}$: we apply the termination proofs for $\boldsymbol{A}$ and $\boldsymbol{B}$ to get a pair in $\alpha \otimes \alpha$, apply a special addition operator (which exists for the ordinals we consider) to get an element in α and finally apply the inhabitance proof of $\boldsymbol{A}$ to get an element in $\boldsymbol{A}$. However, with this interpretation we do not have (in general) $[\![\pi_1(x, y)]\!] \to_A [\![x]\!]$.

What we need is an addition operation on $\boldsymbol{A}$ itself, i.e. an operation $\boxplus$ in $\boldsymbol{A} \otimes \alpha \Rightarrow \boldsymbol{A}$ such that $a \boxplus m \to_A a$ for arbitrary $m \in \alpha$ and $a \in A$. The m is the mentioned rewrite noise. The point is that in a term $\pi_1(t, u)$ we might perform arbitrary rewrite steps within u before we apply our projecting rule and get rid of u or its reduct. Somehow, u does not contribute to the value of $\pi_1(t, u)$, except that we have to model u's "potential for rewriting". The latter sounds vague, but we can formalise it suitably: it is the termination proof for $\boldsymbol{B}$ applied to $[\![u]\!]$. We get:

$$[\![\pi_1]\!](x, y) = x \boxplus b(y)$$

where $b : \boldsymbol{B} \to \alpha$ is the termination proof for $\boldsymbol{B}$ and now we indeed have $[\![\pi_1(t, u)]\!] \to_A [\![t]\!]$ for arbitrary t and u. We are not able to find such an operation $\boxplus$ for all ARSs, but (i) we can find it for some, and (ii) one can show that certain functors (our semantic type constructors) preserve the existence of such functions.

There is a certain link between the presence of "erasing" rewrite rules, rules in which the right-hand side contains fewer variables than the left-hand side, and the need for such a $\boxplus$ operation for termination proofs. As we shall see, there are natural interpretations of all the combinators[3] in the I-fragment of (typed) Combinatory Logic (of CL_I, see [2]) in ARS. The only one missing from CL is the K combinator which has no such natural interpretation, but using $\boxplus$ we can construct something which is often sufficiently close to it.

1.4 Expressive Power

Approaches for showing termination of rewrite systems can largely be classified into syntactical and semantical methods, although there exist hybrid approaches [44] as well. Syntactical methods have typically the advantage of being automatisable and the disadvantage of limiting the range of rewrite systems that can be dealt with. For semantical methods it is the other way around, the potential for automisation [28] is small, but virtually every terminating rewrite system has a semantic termination proof.

In particular, this applies for first-order TRS. For every terminating TRS over the signature Σ there is a terminating ARS $(A, \rightarrow_A)$ where A is a Σ-algebra and $\rightarrow_A$ a terminating relation on A such that $t \rightarrow_R u$ implies $[\![t]\!]_\rho \rightarrow_A [\![u]\!]_\rho$ for all environments ρ. But this is trivial: we can choose $A = \mathrm{T}\Sigma$ and $\rightarrow_A = \rightarrow_R$.

Somewhat surprisingly, such a claim is not so easily established for higher-order rewrite systems. As interpretation for a particular terminating HRS we are looking for an ARS $\boldsymbol{A}$ and (i) an ordinal α plus inhabitance proof $n : \alpha \rightarrow A$ and termination proof $f : A \rightarrow \alpha$; (ii) a map $\boxplus : A \otimes \alpha \rightarrow A$ with $x \boxplus m \rightarrow_A x$. By appeal to the axiom of choice we can always find (i); surprisingly it is not even a restriction to require the same α for all types, provided we are solely interested in termination of reduction on *open terms*. However (ii) may not exist. In particular for $A = \alpha$, the map exists iff α has the form ω^γ for some γ; this is not really a problem, because there is a canonical way of lifting α, A, n, and f to larger ordinals that have this form, for instance by using multiset extensions of the orderings.

Another problem is that our modelling of function types in an HRS involves the set of *all* ARS morphisms between two ARSs. For an exact modelling of the HRS behaviour (roughly: rewriting modulo β-conversion) this set is too large, it contains functions which are not expressible as HRS terms and substitutions. The problem is similar to finding a fully abstract model for a sequential programming language.

A side remark: the problem just might go away if one adopts some (extreme) constructivist setting as foundation of mathematics, one in which there simply are only λ-definable functions and no nasty diagonalisation surprises. Just dreaming.

1.5 Related Work

The work presented here is an extended and corrected version of [19]. It is closely related to older work by Robin Gandy [10] and Roel de Vrijer [47], but

[3] People accustomed with Linear Logic may prefer this view: all structural rules except Weakening have natural interpretations.

we go much further in analysing the abstract side of termination. As a result, our approach is more general: we can treat arbitrarily large ordinals rather than sticking to the natural numbers; there is a price to pay in that several of de Vrijer's results cannot be generalised. The reason why natural numbers behave more smoothly than larger ordinals is that ordinal addition and natural sum (see below or [16]) coincide for natural numbers but may differ on larger operands. Natural sum is the more fundamental operation in our semantics.

Although technically the approach presented here is most closely related to the two mentioned papers, especially Gandy's, I did not know either of them at the time I started working on this subject. The original inspiration were Hans Zantema's semantic labelling of first-order TRSs [49, 51] and its adaptation to HRSs by Jaco van de Pol and Helmut Schwichtenberg [38, 40, 39].

2 Preliminaries

An *Abstract Reduction System* (short: ARS) consists of a set A and a binary relation $\rightarrow$ on A. Given two relations R and S on A, we write $R\,;S$ for their composite, given by $x\,R\,;S\,y \iff \exists z \in A.\, xRz \wedge zSy$. The ARS $\boldsymbol{A}$ is *transitive* if $\rightarrow_A ; \rightarrow_A \subseteq \rightarrow_A$; it is *total* iff $\forall x, y \in A.\ x = y \vee x \rightarrow_A y \vee y \rightarrow_A x$.

We write $\boldsymbol{A} \models P$ if the ARS $\boldsymbol{A} = (A, \rightarrow)$ has the property P. Given an ARS $\boldsymbol{A} = (A, \rightarrow_A)$, $t \in A$ is a *normal form* if $\neg \exists u \in A.\ t \rightarrow_A u.\boldsymbol{A}$. An element t *has* a normal form u if $t \twoheadrightarrow_A u$ and u is a normal form. An ARS $\boldsymbol{A}$ is *strongly normalising*, $\boldsymbol{A} \models \mathrm{SN}$, iff there is no non-empty relation R on A satisfying the inequation $R \subseteq\ \rightarrow ; R$, i.e. iff there are no infinite chains of $\rightarrow$-steps.

We do not have the room (or patience) to introduce the concepts and terminology we borrow from category theory in a fully self-contained manner. The reader can find the missing definitions in most standard works on category theory, e.g. in [29] or [37] or [5]. Knowing (and understanding) some amount of category theory 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 3.4.

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 as morphisms morphisms of $\mathcal{A}$ where $m \in (\mathcal{A} \downarrow X)((Y, g), (Z, h))$ iff $m \in \mathcal{A}(Y, Z)$ and $h \circ m = g$. There is a forgetful functor $U : \mathcal{A} \downarrow X \rightarrow \mathcal{A}$ defined as $U(X, f) = X$ and $U(m) = m$. Dually, $X \downarrow \mathcal{A}$ is defined as $(\mathcal{A}^{\mathrm{op}} \downarrow X)^{\mathrm{op}}$.

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

Dually, a *comonad* in a category $\mathcal{A}$ is a triple $\boldsymbol{G} = (G, \delta, \epsilon)$ where $G : \mathcal{A} \to \mathcal{A}$ is a functor and $\delta : G(\_) \to G(G(\_))$ and $\epsilon : G(\_) \to \_$ are natural transformations satisfying the dual equations to a monad. Similarly, there is a category $\mathcal{A}_{\boldsymbol{G}}$ of $\boldsymbol{G}$-coalgebras and a corresponding forgetful functor.

We shall make use of one important property of monads (see theorem 5 on page 137 in [4]):

Theorem 2.1 *Suppose the functor T of a monad (T, μ, η) has a right adjoint G. Then there is a comonad $\boldsymbol{G} = (G, \delta, \epsilon)$ with $\delta = G(\mu) \circ G(T(a)) \circ G(\eta)$ and $\epsilon = e \circ G(\eta)$ where a and e are unit and counit of the adjunction, such that $\mathcal{A}^{\boldsymbol{T}}$ and $\mathcal{A}_{\boldsymbol{G}}$ are equivalent categories. Moreover, the forgetful functor U has left and right adjoints.*

One can slightly improve on that result: under the premises, $\mathcal{A}^{\boldsymbol{T}}$ and $\mathcal{A}_{\boldsymbol{G}}$ are *isomorphic* — but this scarcely extends the power of the theorem.

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 these natural 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 *(left) representation monad* of a monoid is the monad $(X \otimes \_, \mu, \eta)$, where $\mu = (\oplus \otimes id) \circ a$ and $\eta = (0 \otimes id) \circ l^{-1}$. The *right* representation monad is the representation monad given by the dual monoidal category $(\mathcal{A}, \otimes', \ldots)$ where $A \otimes' B = B \otimes A$ with the obvious coherence isomorphisms.

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

Another elementary property of monads will be of further use:

Theorem 2.2 *Let $\mathcal{A}$ be a monoidal category, $\boldsymbol{T}$ the left representation monad of a monoid X and $\boldsymbol{S}$ the right representation monad of a monoid Y. Then the monad $\boldsymbol{S}$ can be lifted to $\mathcal{A}^{\boldsymbol{T}}$.*

Proof. For the lifting of monads, see [4], page 312ff. In our case the so-called "distributive law λ_Z" is the associativity isomorphism $a_{XZX'}$; all required equations for λ follow from the coherence properties of $\mathcal{A}$. $\square$

3 Semantics

For proving termination in a semantical setting, we want to interpret types by sets with (preferably terminating) relations. 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 abstract reduction systems.

3.1 The category of ARSs

Definition 3.1 *The category* ARS *is defined as follows:*

Objects: $\boldsymbol{A} = (A, >_A) \in |\mathsf{ARS}|$ *iff* $\boldsymbol{A}$ *is an ARS.*

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

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

We write OS *for the full subcategory of* ARS *in which each object is total, transitive and irreflexive. We write* TWO *for the full subcategory of* OS *in which each object is additionally SN.*

For objects $\boldsymbol{A}, \boldsymbol{B} \in |\mathsf{ARS}|$ we write $\boldsymbol{A} \approx \boldsymbol{B}$, $\boldsymbol{A}$ *is weakly similar to* $\boldsymbol{B}$, iff there are morphisms $f : \boldsymbol{A} \to \boldsymbol{B}$ and $g : \boldsymbol{B} \to \boldsymbol{A}$. We call the equivalence classes of $\approx$ *weak order types*. The usual notion of order type and similarity (see e.g. [45]) also requires that f and g are each others inverses. But for considering SN the relation $\approx$ is more appropriate: if $\alpha \approx \beta$ for ordinals α and β, then $\alpha = \beta$; and $\boldsymbol{A} \models \mathrm{SN}$ iff $\boldsymbol{A} \approx \alpha$ for some ordinal α.

In many respects, ARS is very similar to the category of sets, it inherits much of the abstract structure of Set. Several operations on relations can be lifted to endofunctors on ARS, especially all closure operators on relations. Given an ARS $\boldsymbol{A} = (A, >_A)$ we define the ARS $\boldsymbol{A}^=$ to be $(A, \geq_A)$ where $\geq_A$ is the reflexive closure of $>_A$. This extends to a functor $\_^=$ by setting $f^= = f$. Notice that $\mathbf{1}^=$, the reflexive closure of the ordinal 1, is the terminal object of ARS. Similarly, we define a functor $\_^{-1}$ by $\boldsymbol{A}^{-1} = (A, <_A)$.

Proposition 3.1 ARS *has all finite limits and colimits.*

Proof. It has an initial object (the ordinal $\mathbf{0}$), a terminal object ($\mathbf{1}^=$), products and coproducts (which we shall see later), and equalisers and coequalisers. Equalisers and coequalisers can be obtained very similarly to Set, i.e. by restriction to a subset and quotienting with an equivalence, respectively. $\square$

We are ultimately interested in terminating ARSs, another subcategory of ARS. Considering that an ARS $\boldsymbol{A}$ is terminating iff there is a morphism from $\boldsymbol{A}$ to an ordinal α, it is obvious that all limits except the terminal object preserve termination.

Proposition 3.2 *Let $D : \mathcal{G} \to \mathsf{ARS}$ be a diagram. Let $(\boldsymbol{S}, \{p_a\})$ be a limit of D. If there is an object $x \in |\mathcal{G}|$ with $\boldsymbol{D(x)} \models SN$ then $\boldsymbol{S} \models SN$.*

Proof. Since $D(x)$ is terminating there must be a morphism $f : D(x) \to \alpha$ for some ordinal α. Thus we get a morphism $f \circ p_x : \boldsymbol{S} \to \alpha$ which proves the termination of $\boldsymbol{S}$. $\square$

The converse implication does not hold in general, example: take the object $\boldsymbol{\omega}^{-1}$ and the two endomorphisms id and $+1$. The object $\boldsymbol{\omega}^{-1}$ is non-terminating, but the equaliser of the two maps is the ordinal 0.

Colimits do not behave so smoothly. The initial object is terminating and coproducts preserve termination, but coequalisers generally do not. This should not be too surprising, because coequalisers are computed by quotienting, and if for instance quotienting identifies related objects then the relation cannot any longer be irreflexive. On the other hand, colimits *reflect* termination in the same trivial way as limits preserve it, i.e. if a colimit is terminating then so are all objects in its diagram.

The preservation of termination is important when we are looking for the interpretation of a typed rewrite system, where the type structure has parametric types such as function types, product types, list types, etc. Here, we would like to interpret the type constructors as functors in ARS, preferably by functors that also preserve termination proofs.

Definition 3.2 *We define two summation functors on ARS, categorical sum $\mathsf{ARS} \sqcup \mathsf{ARS} \to \mathsf{ARS}$ and ordinal sum $\mathsf{ARS} + \mathsf{ARS} \to \mathsf{ARS}$ as follows: let $\boldsymbol{A} = (A, >_A)$ and $\boldsymbol{B} = (B, >_B)$, then*

- $\boldsymbol{A} \sqcup \boldsymbol{B}$ *is the object $(A \times \{0\} \cup B \times \{1\}, >_{A \sqcup B})$ such that $(x, i) >_{A \sqcup B} (y, j)$ iff $i = j$ and then either $i = 0$ and $x >_A y$ or $i = 1$ and $x >_B y$. 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 relation: $(x, n) >_{A+B} (x', n') \iff (x, n) >_{A \sqcup B} (x', n') \;\vee\; n > n'$.*

The notation $\boldsymbol{A} \sqcup \boldsymbol{B}$ for the coproduct is motivated by the observation that the weak 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 categorical coproduct.

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

Definition 3.3 *We define three multiplication functors on ARS, categorical product $\mathsf{ARS} \sqcap \mathsf{ARS} \to \mathsf{ARS}$, ordinal product $\mathsf{ARS} * \mathsf{ARS} \to \mathsf{ARS}$, and symmetric product $\mathsf{ARS} \otimes \mathsf{ARS} \to \mathsf{ARS}$ as follows: let $\boldsymbol{A} = (A, >_A)$ and $\boldsymbol{B} = (B, >_B)$, then*

A$\sqcap$***B*** *is* $(A\times B, >_{A\sqcap B})$ *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.

A $*$ ***B*** *is defined just as* ***A*** $\sqcap$ ***B****, except for the relation:*
$(a,b) >_{A*B} (a',b') \iff a >_A a' \ \vee\ (a \geq_A a' \wedge b >_B b')$.

A $\otimes$ ***B*** *is similarly defined just as* ***A*** $\sqcap$ ***B****, except for the relation* $>_{A\otimes B}$*:*
$(a,b) >_{A\otimes B} (a',b') \iff a >_A a' \wedge b \geq_B b' \ \vee\ a \geq_A a' \wedge b >_B b'$.

On first view, one might think that the multiplication functors differ only in minor details, but these details are highly significant for their (weak) order types. Writing bold numbers for the objects corresponding to ordinals in ARS we have for example: $\mathbf{3} \sqcap \mathbf{7} \approx \mathbf{3}$, $\mathbf{3} * \mathbf{7} \approx \mathbf{21}$, and $\mathbf{3} \otimes \mathbf{7} \approx \mathbf{9}$.

The functor $\_ \otimes \_$ is not the one mentioned in the introduction, but we shall use it for the same purposes. It corresponds to a slightly modified notion of "single rewrite step", in which parallel rewriting is permitted.

Proposition 3.3 $(\mathsf{ARS}, \otimes, \mathbf{1})$ *is a symmetric monoidal category.*

Proof. The obvious underlying isomorphisms in Set preserve the relations, i.e. they are ARS morphisms. □

We shall not be using the monoidal structures of the other multiplication or summation functors.

Proposition 3.4 *The functor* $\_ \otimes A$ *has a right adjoint* $A \Rightarrow \_$*, i.e.* ARS *is monoidal closed.*

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

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

On morphisms, we have as usual $(f \Rightarrow g)(h) = g \circ h \circ f$. Checking the adjunction properties is routine. □

Again, the functor $\_ \Rightarrow \_$ slightly differs from the one used in the introduction, simply because it is right adjoint to a different tensor product.

There is an analogous right adjoint to $\_ * A$, which we can call with some justification $\_ \div A$, since e.g. $\mathsf{ARS}(\mathbf{5} * \mathbf{7}, \mathbf{35}) \cong \{0\} \cong \mathsf{ARS}(\mathbf{5}, \mathbf{35} \div \mathbf{7})$. The corresponding function space has the same underlying carrier set as the exponential of the tensor, but its ordering on functions is more restrictive: $f >_{B\div A} g \iff \forall x, y \in A.\ f(x) >_B g(y)$.

The other ordinal multiplication functor $A * \_$ apparently has no right adjoint, e.g. the monotonicity requirement for the co-unit would force us to infer $f(a) > g(a')$ solely from $a > a'$.

The categorical product $\_ \sqcap A$, also has a right adjoint $A \to \_$, i.e. ARS is cartesian closed. The underlying set of the exponential $|A \to B|$ is just the full function space between the underlying sets $|A|$ and $|B|$; the order on $A \to B$ is given by: $f > g \iff \forall x, y \in A.\, x >_A y \Rightarrow f(x) >_B g(y)$. In particular, any monotonic function is related to itself. This shows that $A \to B$ is hardly ever terminating when we need it, e.g. $A \to A$ cannot be terminating as it contains the identity. Thus, although ARS is cartesian closed, we cannot really use its CCC-ness for our purpose of termination proofs.

Instead of interpretating all of (typed) Combinatory Logic, we can try to interpret just a fragment of it in ARS. This means to find natural transformations for the corresponding combinators in such a way that the usual combinator equations are satisfied.

We can find such interpretations for the combinators I, C, and B in any symmetric monoidal closed category:

$$
\begin{aligned}
[\![I_X]\!] &= id_X \\
[\![C_{XYZ}]\!] &= s_{XY} \Rightarrow id_Z \\
[\![B_{XYZ}]\!] &= cur(\epsilon_{XY} \circ (id_{X \Rightarrow Y} \otimes \epsilon_{ZX}) \circ a)
\end{aligned}
$$

The coherence properties of the monoidal structure and its adjunction ensure that the combinator equations hold. In the same way one can show that the interpretation is extensional, i.e. $B\, f\, I = f$ holds as well.

It is not possible to extend the interpretation to the K combinator, unless $\otimes$ is a categorical product. Similarly, we need some extra structure to find an interpretation for the S combinator and thus for the λ_I-calculus. Essentially, we need the ability to *copy* arguments.

Definition 3.4 *A symmetric monoidal category has* diagonals *if there is a natural transformation $\delta_X : X \to X \otimes X$ such that (i) $\delta_X = s_{XX} \circ \delta_X$, (ii) $\delta_I = l_I^{-1}$, and (iii) $a_{XXX} \circ (id_X \otimes \delta_X) \circ \delta_X = (\delta_X \otimes id_X) \circ \delta_X$.*

As one would expect, ARS has indeed diagonals, the same as in Set ($\delta(x) = (x, x)$). The reason for choosing a monoidal structure different from the one mentioned in the introduction is this very property. We need that $x >_A y$ implies $\delta(x) = (x, x) >_{A \otimes A} (y, y)$ to make δ_X an ARS morphism and we have it, as the $\otimes$ functor "allows parallel rewriting".

In a symmetric monoidal closed category with diagonals, one can give an interpretation of the S-combinator. This is straightforward (exercise for the reader — it is easier and sufficient to define the W-combinator), though it involves quite a few more coherence maps than the definition of the other combinators requires. Bart Jacobs seems to be the first person to have investigated this.

Now we have interpretations for the combinators S, B, C, and I which satisfy the combinator equations:

$$
\begin{aligned}
Sxyz &= xz(yz) \\
Bxyz &= x(yz) \\
Cxyz &= xzy \\
Ix &= x
\end{aligned}
$$

In other words, we have a model of (typed) CL_I. Because the model is extensional (the morphisms are functions, syntactic application is modelled by function application), it is in fact a model of the (typed) $\lambda\eta_I$-calculus. So we can interpret any λ_I-term in that model for a given fixed interpretation of the free variables.

Concerning the interpretation of higher-order rewriting, this is good news as long as we are happy with $\lambda\eta_I$ instead of $\lambda\eta$ as our meta-theory. If $\lambda\eta_I$ does not suffice we may have to settle for something weaker for K. We shall come to that later.

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

Definition 3.5 *Consider the full subcategory* OS *of total, transitive, and irreflexive ARSs. We define a functor* $\mathcal{F} : \mathsf{OS} \to \mathsf{OS}$ *as follows: let* $\boldsymbol{A} = (A, >_A)$, $\boldsymbol{B} = (B, >_B)$, $\boldsymbol{A} \xrightarrow{f} \boldsymbol{B}$ *then*

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

$\mathcal{F}(f)(M)$ *is* $\{f(m) \mid m \in M\}$.

The restriction to ordered sets (OS) is necessary to make $\mathcal{F}$ functorial. All morphisms $f : \boldsymbol{A} \to \boldsymbol{B}$ in OS are injective functions, 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. Remark: transitivity of the relation is not required to make this construction functorial, but without it the resulting ARS would not be total, i.e. $\mathcal{F}$ would not be an endofunctor.

$\mathcal{F}(A)$ 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 exponentiation for ordered sets [41] can also be defined:

Definition 3.6 *Let* $\mathsf{OS}_\perp$ *be the full subcategory of* OS *in which each object has a least element. The functor* $\boldsymbol{Fin} : \mathsf{OS} \times \mathsf{OS}_\perp \to \mathsf{OS}_\perp$ *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 least element of* $\boldsymbol{B}$.

$\boldsymbol{Fin}(\boldsymbol{A}, \boldsymbol{B})$ *is* $(A \rightsquigarrow B, \succ_{A*B})$ *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* $\mathcal{F}(\boldsymbol{A} * \boldsymbol{B})$.

$\boldsymbol{Fin}(f, g)$ *is* $\mathcal{F}(f * g)$.

Because $\boldsymbol{Fin}$ operates on ordered sets, any morphisms f and g are injective functions and hence $f * g$ sends finite maps to finite maps. Similarly, f and g cannot map non-$\perp$ elements to $\perp$, because they are injective and 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. It is perhaps slightly surprising that $\boldsymbol{Fin}$ is covariant in its first argument — this is possible because morphisms in OS are injective functions.

3.2 Ordinals

The functors described so far allow to construct new ordered sets from given ones. Moreover, such a construction preserves strong normalisation, with the only exception that $\mathbf{0} \Rightarrow \boldsymbol{A}$ is not terminating.

For termination proofs, it is useful to have an arithmetic and logic for well-orderings available. The obvious choice is ordinal arithmetic.

Definition 3.7 *An ordinal is a set α such that all its elements are ordinals and $\forall\beta \in \alpha. \forall\gamma \in \alpha.\ \beta \in \gamma \vee \gamma \in \beta \vee \beta = \gamma$ and $\forall\beta \in \alpha. \forall\gamma \in \beta.\ \gamma \in \alpha$.*

These are the so-called "von Neumann ordinals" [34, 42]. 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. [21, 26, 41]. 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 3.8 *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 3.9 *Let α be an ordinal. We define a function $\_\#\_ : \boldsymbol{Fin}(\alpha, \omega) \times \boldsymbol{Fin}(\alpha, \omega) \to \boldsymbol{Fin}(\alpha, \omega)$ as follows:*

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

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 [13, 16, 9, 1].

Proposition 3.5 *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 $(M'\#N) \setminus (M\#N)$. $\square$

Definition 3.10 *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[4] 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 ω^γ 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 necessity see lemma 15 in [8].

[4]The name is taken from Potter's book on set theory [41]; in [8] and [7] they are called *additive principal*; Hessenberg introduced the concept in [13] and called them *Hauptzahlen*, main numbers.

Proposition 3.6 *Let α be an indecomposable ordinal. Then $(\boldsymbol{\alpha}, \#, 0)$ is a monoid in $(\mathsf{ARS}, \otimes, \ldots)$, 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 3.5 and the well-known properties of natural sum. □

Beside indecomposable ordinals there is another class of monoids on this monoidal structure, because each monoid in ARS has a dual: if $\boldsymbol{\alpha}$ is a monoid then so is $\boldsymbol{\alpha}^{-1}$. This follows from $\_^{-1}$ being a functor with the properties $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ and $\mathbf{1}^{-1} = \mathbf{1}$. Thus we could easily extend our approach to show that a given TRS is "increasing" [24].

Definition 3.11 *Let α be an indecomposable ordinal. We write $\boldsymbol{M}$ for the representation monad of the monoid $(\boldsymbol{\alpha}, \#, 0)$ and $\boldsymbol{W}$ for the corresponding comonad.*

We shall not do very much with the monad itself, we are merely interested in its algebras (which are sometimes called the "left actions" of the monoid). Each of these algebras is given by some appropriate morphism $\boxplus : \boldsymbol{\alpha} \otimes \boldsymbol{A} \to \boldsymbol{A}$ which can be used to "add rewrite noise" on top of elements of A. This is its intended application for ARSs, but behind it is a more general picture for the interpretation of the λ-calculus in monoidal closed categories.

The comonad $\boldsymbol{W}$ is determined by theorem 2.1. Instantiating it to our situation, we get as natural transformations of the comonad $\epsilon : W(\_) \to \_$ given as

$$W(\boldsymbol{X}) = (\boldsymbol{\alpha} \Rightarrow \boldsymbol{X}) \;\cong\; (\boldsymbol{\alpha} \Rightarrow \boldsymbol{X}) \otimes \mathbf{1} \xrightarrow{id \otimes 0} (\boldsymbol{\alpha} \Rightarrow \boldsymbol{X}) \otimes \boldsymbol{\alpha} \xrightarrow{ap} \boldsymbol{X}$$

and $\delta : W(\_) \to W(W(\_))$ given as

$$W(\boldsymbol{X}) = (\boldsymbol{\alpha} \Rightarrow \boldsymbol{X}) \xrightarrow{\# \Rightarrow id} ((\boldsymbol{\alpha} \otimes \boldsymbol{\alpha}) \Rightarrow \boldsymbol{X}) \;\cong\; (\boldsymbol{\alpha} \Rightarrow (\boldsymbol{\alpha} \Rightarrow \boldsymbol{X})).$$

As an exercise in categorical diagram chasing, the reader is encouraged to check the comonad equations for δ and ϵ.

Proposition 3.7 *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 $(X, \oplus) \in |\mathcal{A}^{\boldsymbol{T}}|$. Moreover, if T has a right adjoint G then $(X, G(\oplus) \circ \eta) \in |\mathcal{A}_{\boldsymbol{G}}|$ for the corresponding comonad $\boldsymbol{G}$.*

Proof. Trivial, the coherence properties for this algebra coincide with the monad properties; similarly for the coalgebra and the comonad. □

We shall see soon how we can exploit the fact that $(\boldsymbol{\alpha}, \#)$ and $(\boldsymbol{\alpha}, cur(\#))$ are algebras and coalgebras for $\boldsymbol{M}$ and $\boldsymbol{W}$, respectively.

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

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

3.3 Type-casting Ordinals

To establish termination of a rewrite system we have to interpret it in an ARS whose relation is terminating. We can do this separately for each type, but we have to make sure that the type constructors are interpreted by functors that preserve the termination behaviour.

As indicated in the introduction, there is a problem with function types and empty ARSs, because the ARS $\mathbf{0} \Rightarrow \boldsymbol{A}$ is isomorphic to $\mathbf{1}^{=}$ and thus non-terminating, although $\mathbf{0}$ is trivially terminating ($\boldsymbol{A}$ is arbitrary). Restricting the attention to inhabited ARSs does not help, because the $\Rightarrow$ functor does not preserve inhabitance either.

Instead of having an arbitrary inhabitance proof of $\boldsymbol{A}$, i.e. a morphism $f : \mathbf{1} \to \boldsymbol{A}$, we are looking for something much more specific: for each terminating ARS $\boldsymbol{A}$ there is a smallest ordinal $\boldsymbol{\alpha}$ with a termination proof $p : \boldsymbol{A} \to \boldsymbol{\alpha}$. This ordinal is also the largest ordinal with a copy in $\boldsymbol{A}$, i.e. a morphism $f : \boldsymbol{\alpha} \to \boldsymbol{A}$. We can even require that $p \circ f = id_\alpha$.

Definition 3.12 *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.*

The above definition is slightly inaccurate in that the category $X \downarrow \mathcal{A} \downarrow X$ should be written as either $(X \downarrow \mathcal{A}) \downarrow (X \xrightarrow{\mathrm{id}} X)$ or as $(X \xrightarrow{\mathrm{id}} X) \downarrow (\mathcal{A} \downarrow X)$; however, both constructions result in identical categories.

In particular, we are interested in categories like $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ 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 $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$, convert them into ordinals, "add" them, and convert them back into any object in $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ 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 $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ 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 [14] 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. [12]), but in ARS such a property is often too strong. It should be noted though that one can achieve and maintain such a property in $\mathsf{ARS} \downdownarrows \boldsymbol{\omega}$, provided we are only interested in function types. This situation has been thoroughly studied by de Vrijer [47].

To interpret types in $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ we have to lift the "datatype" functors on ARS to $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$. Notice that $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ 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 $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ should internalize arrows from ARS, not from $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$. Similarly, categorical products in $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ are unsuitable for the interpretation of product types; products in $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ are pullbacks in ARS, carrying pairs of values with the same decoding.

Proposition 3.8 *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:*

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

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

Proof. The first point is a simple diagram chase in $\mathcal{A}$. The second property holds for any category with a null object [29]. □

Definition 3.13 *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 3.9 *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 3.14 *A bifunctor* $\_ \odot \_ : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ *or* $\_ \odot \_ : \mathcal{A}^{\mathrm{op}} \times \mathcal{A} \to \mathcal{A}$ *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 3.9 for bifunctors: each functor which is retractable at X determines a bifunctor on $\mathcal{A} \downdownarrows X$; the proof is almost identical.

Theorem 3.1 *Let* $\mathcal{A}$ *be category and* $\boldsymbol{T} = (T, \mu, \eta)$ *a monad on* $\mathcal{A}$*. Then* T *is retractable at any* $\boldsymbol{T}$*-algebra.*

Proof. Rectractability boils down to the unit property of $\boldsymbol{T}$-algebras. □

Dually, comonad functors are retractable at the coalgebras of the comonad.

Theorem 3.2 *Let* $\mathcal{A}$ *be a monoidal closed category with binary products and coproducts. Categorical product* $\sqcap$ *and coproduct* $\sqcup$ *are retractable at any* X*. The tensor and exponential are retractable at monoids.*

Proof. Products: the map $X \xrightarrow{\langle \mathrm{id},\mathrm{id} \rangle} (X \sqcap X) \xrightarrow{\pi_1} X$ is a retraction for the product; coproducts are dual.

Suppose $\boldsymbol{X} = (X, \oplus, 0)$ is a monoid. Then we get the retractablity of $X \otimes \_$ from theorem 3.1 and proposition 3.7 using the representation monad of $\boldsymbol{X}$. For the exponential it is similar using theorem 2.1. □

Remark: for the tensor and exponential there is another retraction based on the *right* representation monad. For technical reasons (it allows later to employ theorem 2.2) it is more convenient to use that retraction.

The product/coproduct part of theorem 3.2 can easily be extended to arbitrary limits and colimits of diagrams which are in the image of the forgetful functor $U : \mathcal{A} \downdownarrows X \to \mathcal{A}$. In other words: the limits/colimits of these diagrams are in the image of U themselves; but this does not mean that these limits/colimits are reflected by U.

Taking $\mathcal{A} = \mathsf{ARS}$ it follows from theorem 3.2 that $\otimes$ and $\Rightarrow$ are retractable at indecomposable ordinals, $\oplus$ being the natural sum. Concretely, this means that a function f from $\boldsymbol{\alpha} \to \boldsymbol{\alpha}$ is translated (decoded) into a number simply by applying it to 0, and that a number $n \in \alpha$ is translated (encoded) into the function $x \mapsto n \# x$. In particular, for $f \in \boldsymbol{\omega} \Rightarrow \boldsymbol{\omega}$ (with encoding $e(n) = \lambda m.m + n$ and decoding $\bar{e}(g) = g(0)$) we have that $f \geq e(\bar{e}(f)) = \lambda x.f(0) + x$, because for natural numbers this easily follows from $\forall x.\ f(x+y) \geq f(x)+y$. The property fails to hold for indecomposable ordinals larger than ω, because ordinal sum and natural sum no longer coincide.

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 $\mathsf{ARS}(\boldsymbol{\omega}+\boldsymbol{\omega},\boldsymbol{\omega})$ is empty.

The retractability of $\otimes$ on $\mathsf{ARS} \downdownarrows \boldsymbol{\alpha}$ means the following: for any object

$$\boldsymbol{\alpha} \xrightarrow{x} \boldsymbol{X} \xrightarrow{\bar{x}} \boldsymbol{\alpha}$$

we have another object

$$\boldsymbol{\alpha} \xrightarrow{(x\otimes 0)\circ r^{-1}} \boldsymbol{X}\otimes\boldsymbol{X} \xrightarrow{\oplus\circ(\bar{x}\otimes\bar{x})} \boldsymbol{\alpha}.$$

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 $a \boxplus x(1) >_X a$.

3.4 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 3.11. $(\boldsymbol{A},\boxplus)$ 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.

Recall the rather trivial observation of proposition 3.7, that monoids are algebras of their representation monad. Although 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 $\mathsf{ARS}^{\boldsymbol{M}}$, 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 $\mathsf{ARS}^{\boldsymbol{M}}$.

To show that a bifunctor $\_\odot\_$ can be lifted along the forgetful functor 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, can be lifted for a fixed argument.

Lemma 3.1 *Let $\mathcal{A}$ be a category with binary products $\_ \sqcap \_$. Let $\boldsymbol{T} = (T, \mu, \eta)$ be a monad on $\mathcal{A}$. Then $\_ \sqcap \_$ can be lifted to to $\mathcal{A}^{\boldsymbol{T}}$ (and it remains a product).*

Proof. The forgetful functor $U : \mathcal{A}^{\boldsymbol{T}} \to \mathcal{A}$ creates limits (see theorem 1 in [4], page 117). This implies that: (i) $\mathcal{A}^{\boldsymbol{T}}$ has binary products; (ii) $U(A) \sqcap U(B) \cong U(A \times B)$, where $A \times B$ is some product of A and B in $\mathcal{A}^{\boldsymbol{T}}$. The rest follows easily, bearing in mind that objects isomorphic to $\boldsymbol{T}$-algebras are themselves $\boldsymbol{T}$-algebras and that isomorphisms between $\boldsymbol{T}$-algebras in $\mathcal{A}$ are also isomorphisms in $\mathcal{A}^{\boldsymbol{T}}$. □

In fact, lemma 3.1 easily generalises to arbitrary limits. In programming languages, this typically applies to coinductive datatypes such as streams.

For coproducts, we need one additional premise to obtain the analogous result: the monad functor has to have a right adjoint which allows us to apply theorem 2.1. In case of ARS and our monad $\boldsymbol{M}$, the monad functor is $\boldsymbol{\alpha} \otimes \_$ (for some fixed $\boldsymbol{\alpha}$), for which we already know that it has a right adjoint since ARS is symmetric and closed.

Lemma 3.2 *Let $\mathcal{A}$ be a category with binary coproducts $\_ \sqcup \_$. Let $\boldsymbol{T} = (T, \mu, \eta)$ be a monad on $\mathcal{A}$ such that T has a right adjoint. Then $\_ \sqcup \_$ can be lifted to $\mathcal{A}^{\boldsymbol{T}}$ (and it remains a coproduct).*

Proof. We can apply theorem 2.1. The forgetful functor $U_G : \mathcal{A}_{\boldsymbol{G}} \to \mathcal{A}$ creates colimits (dual case of the already mentioned theorem 1 in [4], page 117) and hence so does $U : \mathcal{A}^{\boldsymbol{T}} \to \mathcal{A}$. From here on, the argument is similar to the proof of lemma 3.1. □

Although the proofs of the last two lemmas are fairly complicated when spelt out, the addition operations we obtain from them for products and coproducts in ARS are fairly simple. The addition operation for products is: $n \boxplus (x, y) = (n \boxplus_X x, n \boxplus_Y y)$ and for coproducts: $n \boxplus (0, x) = (0, n \boxplus_X x)$ and $n \boxplus (1, y) = (1, n \boxplus_Y y)$.

Again, lemma 3.2 generalises easily to arbitrary colimits. For programming languages the result is particularly useful, because colimits correspond to inductive datatypes such as binary trees: an inductive datatype is the colimit of its finite approximations. The (generalised) lemma then says that if all these approximations are $\boldsymbol{T}$-algebras then so is their colimit.

Example: consider the ML datatype of binary trees.

```
datatype 'a tree =
    Leaf of 'a | Bin of ('a tree) * ('a tree)
```

We can model any type `t tree` semantically as a colimit of the sequence: $t \to t \sqcup (t \otimes t) \to t \sqcup ((t \sqcup (t \otimes t)) \otimes (t \sqcup (t \otimes t))) \to \ldots$ where t is the object corresponding to `t`. Suppose t is a $\boldsymbol{T}$-algebra. Since $\sqcup$ and $\otimes$ (see the following lemma) both preserve $\boldsymbol{T}$-algebras all elements of the sequence are $\boldsymbol{T}$-algebras, and furthermore all morphisms are morphisms in $\mathcal{A}^{\boldsymbol{T}}$ (they are built from the coproduct injections). Hence, by the generalised version of lemma 3.2, the colimit of this diagram is a $\boldsymbol{T}$-algebra.

Lemma 3.3 *Let $\mathcal{A}$ be a monoidal category, X a monoid in $\mathcal{A}$ and $\boldsymbol{T}$ the representation monad of X. Then the functor $\_ \otimes Y$ can be lifted to $\mathcal{A}^{\boldsymbol{T}}$.*

Proof. Immediate by using the associativity isomorphism; the algebra map is: $T(Z \otimes Y) = X \otimes (Z \otimes Y) \cong (X \otimes Z) \otimes Y = T(Z) \otimes Y \xrightarrow{\boxplus \otimes id} Z \otimes Y$. □

We do not even need that Y is a $\boldsymbol{T}$-algebra. If Y is a $\boldsymbol{T}$-algebra then the result follows also from theorem 2.2.

Note that there is a difference between addition on $A \sqcap B$ and $A \otimes B$ in ARS, although both are "pairs with structure". We have: $n \boxplus_{A \sqcap B} (a, b) = (n \boxplus_A a, n \boxplus_B b)$ but for the tensor it is: $n \boxplus_{A \otimes B} (a, b) = (n \boxplus_A a, b)$.

Lemma 3.4 *Let $\mathcal{A}$ be a symmetric monoidal closed category, X a monoid in $\mathcal{A}$, $\boldsymbol{T}$ the representation monad of X, and $Y \in |\mathcal{A}|$. Then the functor $Y \Rightarrow \_$ can be lifted to $\mathcal{A}^{\boldsymbol{T}}$.*

Proof. Consider the comonad $\boldsymbol{G}$ of theorem 2.1. Similarly to the proof of lemma 3.3 we can lift the functor $Y \Rightarrow \_$ to $\mathcal{A}_{\boldsymbol{G}}$, to coalgebra map is:

$$(Y \Rightarrow Z) \xrightarrow{id \Rightarrow cur(\boxplus)} (Y \Rightarrow (X \Rightarrow Z)) \cong (X \Rightarrow (Y \Rightarrow Z)) = G(Y \Rightarrow Z)$$

The result follows by the equivalence of $\mathcal{A}^{\boldsymbol{T}}$ and $\mathcal{A}_{\boldsymbol{G}}$. □

Theorem 3.3 *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 $\sqcap$, $\sqcup$, $\otimes$, $\Rightarrow$.*

Proof. This is just a collection of the results of lemmas 3.1, 3.2, 3.3, and 3.4. Of course, $\boldsymbol{T} = X \otimes \_$ has a right adjoint since $\mathcal{A}$ is biclosed, i.e. the premises of lemma 3.2 are satisfied. □

3.5 Combining Algebras and Conversion

Now it would be nice if we could combine the construction of $\mathsf{ARS}^{\boldsymbol{M}}$ and $\mathsf{ARS} \Downarrow \boldsymbol{\alpha}$, i.e. lift the functors to $\mathsf{ARS}^{\boldsymbol{M}} \Downarrow (\boldsymbol{\alpha}, \#)$. In application examples, such a property is not always necessary. However, the algebra maps and the encoding/decoding maps sit in the meta-theory for our approach and it is useful if we know how they interact with each other.

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

Figure 2: Condition (iii) for lemma 3.5

Lemma 3.5 *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 F can be lifted to $\mathcal{A}^{\boldsymbol{T}}$ (with algebra map $\oplus'_X$), if F is retractable at X (with encoding F and decoding $\bar{F}$), and if the diagram in figure 2 commutes.*

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

Considering the functors we have studied throughout, i.e. product, coproduct, tensor product, and the exponential of the tensor product, the desired property follows easily from the results we have already:

Lemma 3.6 *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. According to lemma 3.1 the lifted $\_ \sqcap \_$ is still a product, so theorem 3.2 applies. □

Lemma 3.7 *Premises as in lemma 3.2. The lifted functor $\_ \sqcup \_$ is retractable at any object.*

Proof. Again, lemma 3.2 shows that the lifted $\_ \sqcup \_$ is still a categorical coproduct, i.e. theorem 3.2 applies. □

Lemma 3.8 *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. This follows from theorem 2.2 and theorem 3.1. □

Lemma 3.9 *Let $\mathcal{A}$ be a symmetric monoidal closed category, $(X, \oplus, 0)$ a monoid in $\mathcal{A}$, and $\boldsymbol{T}$ be the representation monad of this monoid. Then the lifted $\_ \Rightarrow \_$ is retractable at $(X, \oplus)$.*

Proof. Using theorem 2.1 we know that there are left and right representation comonads $\boldsymbol{G}$ and $\boldsymbol{G}'$. From the dual of theorem 2.2 we know that $\boldsymbol{G'}$ can be lifted to $\mathcal{A}_{\boldsymbol{G}}$ and hence, because $\mathcal{A}_{\boldsymbol{G}} \approx \mathcal{A}^{\boldsymbol{T}}$, also to $\mathcal{A}^{\boldsymbol{T}}$. The result then follows from the dual of theorem 3.1. □

Theorem 3.4 *Let α be an indecomposable ordinal. For any object $\boldsymbol{A} \in \mathsf{ARS}$, which we can build from applying the functors $\sqcap$, $\sqcup$, $\otimes$, and $\Rightarrow$ in arbitrary order to $\boldsymbol{\alpha}$, we have:*

There are morphisms $\boldsymbol{\alpha} \xrightarrow{a} \boldsymbol{A} \xrightarrow{\bar{a}} \boldsymbol{\alpha}$ such that $\bar{a} \circ a = id_{\alpha}$.

For any other object $\boldsymbol{B}$ built this way, we have a morphism $\boldsymbol{A} \xrightarrow{\bullet} \boldsymbol{B}$.

We have a morphism $\boldsymbol{\alpha} \otimes \boldsymbol{A} \xrightarrow{\boxplus} \boldsymbol{A}$ such that $0 \boxplus x = x$ and $m \boxplus (n \boxplus x) = (m \# n) \boxplus x$.

The conversions $\boldsymbol{A} \xrightarrow{\bullet} \boldsymbol{B}$ preserve addition, i.e. $\bullet(n \boxplus x) = n \boxplus \bullet(x)$.

Proof. This is mainly a summary of some of the results above. □

Moreover, we can extend this result to other endofunctors on ARS, provided they are retractable at $\boldsymbol{\alpha}$, they can be lifted to $\mathsf{ARS}^{\boldsymbol{M}}$ and their retraction maps are $\mathsf{ARS}^{\boldsymbol{M}}$-morphisms. We also have these ingredients available for limits and colimits.

4 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 ARS (or rather $\mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}$), it should somehow support termination proofs.

We can indeed aim for something more general, corresponding to the generality provided by our semantic results. This would mean to interpret terms in a category $\mathcal{A}^{\boldsymbol{T}} \downdownarrows X$ where $\mathcal{A}$ is symmetric monoidal closed (and has diagonals) with a monoid $(X, \oplus, 0)$ the representation monad of which is $\boldsymbol{T}$. I do not know whether this has other applications than termination proofs, but the abstract result allows to make certain modifications of our base category if the need arises. For instance, we could extend the objects of ARS to carry more data; in particular, adding a reflexive relation gives rise to a tensor product with a more fine-grained "order". However, we are not going to maintain the abstract flavour of the last section in order to make the approach easier to grasp for people who do not indulge in category theory.

Coming back to termination: There is a nice way to show the termination of first-order term rewriting systems by semantic interpretation, see [49, 51]. Many so-called rewrite systems like the λ-calculus or various process calculi do not fit into the formal framework of first-order TRS, because they require the use of substitution not only in the meta-theory, but in the rewrite rules themselves. One can regard these systems as rewriting modulo the equational theory of substitutions.

It is possible to internalise substitutions to TRSs, see [27, 33], but this makes the termination proof of the substitution calculus part of the application of the method, rather than keeping it in the framework. 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.

The last paragraph may sound a bit contradictory. If the internalisation of substitutions destroys termination then our rewrite system (modulo substitutions) is not terminating and so we cannot prove termination in the first place, can we?

The answer to this objections is "Yes, but...". There are two reasons why this is not the end of the story. If the relation R we are interested in is not terminating then we could instead study some other relation S with the property $S \subseteq R \subseteq S^*$ since then $R^* = S^*$. And S may well be terminating and give rise to an appropriate notion of normal form. The second reason is that certain equational theories are very likely to destroy termination and substitution calculi belong into that category. If the theory E can prove $E \vdash C[x, y] = x$ for any binary context $C[\,]$ then $\rightarrow_R/E$ is terminating only if $\rightarrow_R$ is the empty relation. Otherwise, $a \rightarrow_R b$ implies $x = C[x, a] \rightarrow_R C[x, b] = x$: the relation $\rightarrow_R/E$ is reflexive.

There is a general way of finding an S of the mentioned form that circumvents this problem: when we define the context-closure of rewrite steps, we restrict ourselves to non-erasing contexts: we need for $C[\,]$ that $x \in \mathrm{FV}_E(C[x])$ which means $\forall t.\, E \vdash C[x] = t \Rightarrow x \in \mathrm{Vars}(t)$ (see [20, 31]).

This trick is used when it comes to define the one-step rewrite relation of various forms of higher-order rewrite systems, although the authors hardly ever acknowledge that something has been sifted out and they most certainly do not address the problem in this general form.

In this paper, we are particularly interested in Higher-Order Rewrite Systems *à la* Nipkow [35], a generalisation of Klop's Combinatory Reduction Systems [23, 25] — for their relationship see [36]. As we are not concerned here with implementability issues, we can even consider Wolfram's generalisation of HRSs (see chapter 4 in [48] or [30]). 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 [17] for a direct definition of HRS terms without quotienting.

The problem with showing termination for HRSs is the lack of "nice" interpretations of $\lambda^{\rightarrow}$ in ARS, simply because termination is not preserved by the CCC-structure. 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 irreflexivity. To put it differently: $\beta\eta$-conversion happens to be one of those problematic equational theories mentioned earlier.

The approach of van de Pol [38, 40] tries to solve this problem by weakening the second objective and allowing certain non-monotonic functions in function types. However, this affects the preservation of (non-) termination by context application. In section 6, this is explained in some more detail.

4.1 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 ARS (or $\mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}$), we shall give some functions from sets of types or terms into $|\mathsf{ARS}|$ or $\bigcup|\mathsf{ARS}|$, respectively. The notation $\bigcup|\mathsf{ARS}|$ is shorthand for $\bigcup\{s \mid (s,>) \in |\mathsf{ARS}|\}$; similarly for $\mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}$, we suppress the application of the forgetful functor $U : \mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha} \to \mathsf{Set}$. This is slightly problematic, as neither $|\mathsf{ARS}|$ nor $\bigcup|\mathsf{ARS}|$ 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 4.1 *Given a set of base types $\mathcal{B}$ we define the set of types over $\mathcal{B}$, $Typ(\mathcal{B})$, as the smallest set of words over the alphabet $\{\to, (,)\} \uplus \mathcal{B}$ satisfying:*

$$\sigma \in \mathcal{B} \Rightarrow \sigma \in Typ(\mathcal{B})$$
$$\alpha, \beta \in Typ(\mathcal{B}) \Rightarrow (\alpha \to \beta) \in Typ(\mathcal{B})$$

$Typ(\mathcal{B})$ comprises the types of $\lambda^{\rightarrow}$. As usual, we drop many parentheses and take $\to$ 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 4.2 *Let $\mathcal{B}$ be a set of base types, let $b : \mathcal{B} \to |\mathsf{ARS}|$ a function mapping base types to objects in ARS. We define a map $[\![\_]\!]_b : Typ(\mathcal{B}) \to |\mathsf{ARS}|$ as follows:*

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

Here, "$\Rightarrow$" is the functor from proposition 3.4.

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

Definition 4.3 *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 [50] and virtually all preservation properties of Currying [6, 22, 18] 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 4.4 *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 [3], 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. The type annotation is pure syntax, i.e. a preterm may or may not be well-typed

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

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

Figure 3: The type system $\lambda^{\rightarrow}$

in some type system. The reason for requiring type annotations in abstractions is similar to the reason for banning symbol overloading: in Curry-style $\lambda^{\rightarrow}$, the term $(\lambda x.c)(\lambda y.y)$ type-checks with type $\mathcal{C}(c)$ by choosing $y : \sigma$ and $x : \sigma \rightarrow \sigma$ for any σ. As we shall see later, the choice of σ can be significant for the semantic interpretation of this term.

Definition 4.5 *Given a signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, a context (over Σ) is a finite set $\Gamma \subset \mathcal{V} \times Typ(\mathcal{B})$ 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 $\neg\exists\sigma.(x, \sigma) \in \Gamma$.*

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 $\lambda^{\rightarrow}$.

Definition 4.6 *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 $\tau \in Typ(\mathcal{B})$. 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 4.7 *For any signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, the* type theory $\lambda^{\rightarrow}$ *is the smallest set of judgements over Σ that is closed under the rules in figure 3.*

Proposition 4.1 *Let $J \in \lambda^{\rightarrow}$. Then $J\!\downarrow$ exists and $J\!\downarrow \in \lambda^{\rightarrow}$.*

This is a direct consequence of strong normalisation and subject-reduction for $\lambda^{\rightarrow}$, see [3].

Proposition 4.2 *Let $\Gamma \vdash t : \tau$ and $\Gamma \vdash t : \tau'$ be derivable judgements in $\lambda^{\rightarrow}$. Then $\tau = \tau'$.*

This is the uniqueness of types property for $\lambda^{\rightarrow}$-Church, see Proposition 3.2.12 in [3].

Definition 4.8 *Let $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$ be an HRS-signature.*
Let $b : \mathcal{B} \rightarrow |\mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}|$ be a function. A symbol interpretation *(of Σ w.r.t. b) is a function $\varrho : \mathcal{S} \rightarrow \bigcup |\mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}|$ such that $\varrho(s) \in [\![\mathcal{C}(s)]\!]_b$.*

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

An interpretation is a pair (ϱ, υ) of a symbol interpretation ϱ and a variable interpretation υ. An interpretation (ϱ, υ) is called consistent (w.r.t. to b and Γ) if υ 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 4.9 *Let Σ be an HRS-signature. Let $\rho = (\varrho, \upsilon)$ be an interpretation of Σ. We define a partial function $[\![\_]\!]_\rho : \lambda^{\to} \rightharpoonup \bigcup |\mathsf{ARS}^{\boldsymbol{M}} \Downarrow \boldsymbol{\alpha}|$ with domain $\{\Gamma \vdash t : \tau \mid \rho$ consistent with $\Gamma\}$ by the following equations:*

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

To justify the correctness of this definition we observe the following properties of semantic interpretations of types and judgements.

Theorem 4.1 *The following are true:*

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 4.2 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 $\boxplus_\tau : \boldsymbol{\alpha} \otimes \tau \to \tau$ is monotonic we get the required result. $\square$

The theorem and its proof have very much the flavour of logical relations and their basic lemma, see section 3 in [32]. However, there are quite a few technical differences, in particular the non-standard interpretation of λ-abstraction and the insistence that x actually occurs. 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.

Essentially, what happens is the following: given a λ-term t we replace all its subterms which are λ-abstractions $\lambda y{:}\tau.\, u$ (of result type σ) by the λ_I-term $\lambda y{:}\tau.\, K^*_\sigma\ (O_\tau\ y)\ u$ and then translate the new term in a standard way [2] to CL_I. As we have indicated in section 3.1, ARS is an extensional model of CL_I; the additional constants are interpreted as $[\![O_\tau]\!] = \bar{\tau}$ and $[\![K^*_\sigma]\!] = \boxplus_\sigma$.

This interpretation does not follow the structure of the type system as it is presented. We could provide an interpretation that is more closely connected to the rules, but it would require more applications of $\boxplus_\sigma$ and $\bar{\tau}$, because the rules for variables and constants have implicit weakening. Similarly we could present the type system in a way that more directly reflects the semantics, but such a presentation would not appear very natural.

Another question one might ask: when translating a λ-term $\lambda y{:}\tau.\, u$ in the outlined way we create $\lambda y{:}\tau.\, K^*_\sigma\ (O_\tau\ y)\ u$ regardless whether y is free in u or not. For generating a λ_I-term we could leave out the $K^*_\sigma\ (O_\tau\ y)$ bit if y is already free in u. Moreover, this would constitute a well-defined interpretation in ARS. So why did not we do it that way?

In fact, this alternative interpretation (call it $[\![\ ]\!]^\bullet$) is perfectly reasonable, and de Vrijer [47] uses a very similar[5] interpretation. The reason for not using it is the lack of one very useful meta-theoretical property. We come to that very soon.

Definition 4.10 *Let Σ be a signature. A presubstitution (over Σ) is a function $\theta : \mathcal{V} \to \Lambda\Sigma$. 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 $\theta : \Gamma \to \Delta$, 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 4.1 *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 4.11 *Let $J = \Gamma \vdash t : \tau$ and $\vartheta = \theta : \Gamma \to \Delta$. We write J^ϑ for the judgement $\Delta \vdash t^\theta : \tau$.*

[5] The only difference is that de Vrijer additionally adds 1 to the argument. The reason for this is that de Vrijer avoids a meta-level and is solely interested in β-reduction and its termination behaviour.

Definition 4.12 *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:*

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

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

Proposition 4.3 *Let $J = (\Gamma \vdash t : \tau) \in \lambda^{\rightarrow}$, $\vartheta = \theta : \Gamma \to \Delta$, 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 [38]. ☐

Proposition 4.3 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 $\lambda^{\rightarrow}$ is implicit in the rules for variables and constants.

Based on proposition 4.3 it is easy to show that β-reduction can only make the interpretation smaller, see the following proposition.

Proposition 4.4 *Let $(\Gamma \vdash t : \tau) \in \lambda^{\rightarrow}$ 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 4.3 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 4.1. ☐

Proposition 4.4 is the one that fails for the alternative interpretation $[\![\ ]\!]^\bullet$, while all the other properties we have studied so far would happily carry over. The reduction lemma 4.8 in [47] can be read as claiming this property for $[\![\ ]\!]^\bullet$ as well, but de Vrijer only reasons about an interpretation *into the natural numbers*. The property fails for larger indecomposable ordinals. It is intriguing to see why it fails:

For all morphisms f between ordinals in ARS we have $f(m+n) \geq f(m)+n$. This property does not generalise to natural sum.

Proposition 4.5 *There is a morphism $f \in \mathsf{ARS}(\omega^3, \omega^3)$ such that for some $m, n \in \omega^3$ we have $f(m \# n) < f(m) \# n$.*

Proof. Take f to be the function that maps k to $\omega^2 \# k$ if ω^1 occurs in the Cantor normal form of k, and to $\omega^2 \# k + 1$ otherwise. Let $m = 0$ and $n = \omega$ then we have: $f(0 \# \omega) = f(\omega) = \omega^2 + \omega < \omega^2 + \omega + 1 = f(0) \# \omega$. ☐

The reason why proposition 4.5 is crucial becomes clear from the proof of the following proposition:

Proposition 4.6 *There are terms t and u, $t \to_\beta u$, and an interpretation ρ such that $\neg[\![t]\!]^\bullet_\rho \geq_\sigma [\![u]\!]^\bullet_\rho$.*

Proof. Take $t = \lambda n.f\,((\lambda x.m)\,n) \to_\beta \lambda n.f\,m = u$ and apply proposition 4.5. □

Proposition 4.6 is not necessarily a disaster for $[\![\;]\!]^\bullet$, but it makes termination proofs based on the interpretation $[\![\;]\!]^\bullet$ more awkward than necessary for a large class of very typical rewrite systems. Also, the mentioned counterexample relies on using ordinals larger than ω. In some applications, a restriction to ω is necessary anyway to obtain some additional properties, see [11].

4.2 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 [38, 48].

Definition 4.13 *Given an HRS signature $\Sigma = (\mathcal{B}, \mathcal{S}, \mathcal{C})$, an 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 $\lambda^{\to}$.
Notation: we write rules as $\Gamma \vdash l \to r : \tau$.

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 for binding the variables of a rule is the interpretation we have chosen for abstractions which does not quite coincide with substitution.

We provide a link by the following notational convention:

Definition 4.14 *Given a judgement $\Gamma \vdash t : \sigma$ and a linear order on the variables in Γ we write $\boxed{t}$ for the preterm $\lambda y_1 : \sigma_1. \cdots \lambda y_n : \sigma_n.\, t$ where $\{y_1 : \sigma_1, \ldots, y_n : \sigma_n\} = \Gamma$ and the variables y_i are listed in order.*

We never care about the order of the variables in Γ but assume a fixed order, i.e. $\boxed{t}$ and $\boxed{u}$ list the variables in the same order if judged in the same context. Both the context and its associated order are really parameters of the $\boxed{t}$ notation, although we suppress them in our notation.

Proposition 1 *Let $\Gamma \vdash t : \tau$. Then $\emptyset \vdash \boxed{t} : \tau'$ for some τ'.*

The above property is fairly obvious. We could even replace the empty context $\emptyset$ by an arbitrary context Δ; strictly speaking, this requires (in general) α-conversions.

Definition 4.15 *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. For our purposes it is convenient to associate an HRS with *two* ARSs — one on terms (judgements), the other on substitutions.

Definition 4.16 *Let* (Σ, R) *be an HRS. We define a relation* $\to_R$ *on substitutions as follows:* $\vartheta_l \to_R \vartheta_r$ *iff*

- $\vartheta_l = \theta_l : \Gamma \to \Delta$ *and* $\vartheta_r = \theta_r : \Gamma \to \Delta$ *for some contexts* Γ *and* Δ *and pre-substitutions* θ_l, θ_r. *(All w.r.t. signature* Σ.*)*
- *For all* $(x : \sigma) \in \Gamma$ *either* $\theta_l(x) = \theta_r(x)$, *or there is a rule* $(E \vdash l \to r : \tau) \in R$ *and* $\theta_l(x) = \boxed{l}$ *and* $\theta_r(x) = \boxed{r}$. *In the latter case we say that the variable* x *is strict (w.r.t. the other data), in the former that is weak.*

In other words, $\to_R$ on relations is the pointwise extension of (the reflexive closure of) R. Note that a variable x can be both weak and strict in $\vartheta_l \to_R \vartheta_r$ but this only occurs if left-hand and right-hand sides of a rule are identical.

Definition 4.17 *Let* (Σ, R) *be an HRS. For a given a judgement* $(\Gamma \vdash C : \sigma) \in \lambda^{\to}$ *a rule application is pair* $(\vartheta_l, \vartheta_r)$ *of substitutions such that* $\vartheta_l \to_R \vartheta_r$. *A rule application is called proper if for at least one strict variable* $(x : \sigma) \in \Gamma$ *we have* $x \in FV(C{\downarrow})$.

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

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 [48, 38] 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 in the definition of $\to_R$ is the following lemma:

Lemma 4.2 *Let* J *be judgement in* $\lambda^{\to}$ *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 $\to_R$ a chance to be irreflexive. Any approach attempting to reason about termination of HRS reduction has to make similar restrictions in the definition of its reduction relation $\to_R$. The same is true for rewriting modulo an equivalence whenever the equivalence fails to have the property that equivalent terms contain the same free variables, as e.g. in the theory of groups.

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 4.18 *A rule application $(\vartheta_l, \vartheta_r)$ for a judgement J is called affine if at most one variable x is not weak in $\vartheta_l \to_R \vartheta_r$; moreover, x occurs at most once in $J\!\downarrow$.*

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

Lemma 4.3 *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 an affine 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 $(\theta_{li}, \theta_{ri}), 1 \le i < k$ for C as follows:

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

Clearly all rule applications are affine and moreover $\vartheta_{li+1} = \vartheta_{ri}$ and $\vartheta_l = \vartheta_{l1}$ and $\vartheta_r = \vartheta_{rk}$. Using lemma 4.2 we can simply discard all non-proper rule applications and get the desired result. □

Lemma 4.3 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 4.19 *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 v that are consistent with Δ it is true that $[\![\Delta \vdash l^{\theta}\!\downarrow : \tau]\!]_{(\varrho,v)} >_{\tau} [\![\Delta \vdash r^{\theta}\!\downarrow : \tau]\!]_{(\varrho,v)}$.*

Theorem 4.2 *Let (Σ, R) be a HRS. If ϱ is a symbol interpretation such that all rules in R are ϱ-decreasing then $(\boldsymbol{\lambda}^{\to}, \boldsymbol{\to_R}) \models SN$.*

Proof. We simply prove that $J \to_R J'$ implies $[\![J]\!]_{\rho} >_{\tau} [\![J']\!]_{\rho}$ where $J = B \vdash t : \tau$ and $\rho = (\varrho, v)$ for some variable interpretation v consistent with B. By lemma 4.3 it is sufficient to consider the case in which $J \to_R J'$ by an affine 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 assume w.l.o.g. that $c = c\!\downarrow$. By induction on the term structure of c we show $[\![E \vdash c : \sigma]\!]_{\rho \circ \vartheta_l} >_{\sigma} [\![E \vdash c : \sigma]\!]_{\rho \circ \vartheta_r}$.

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 4.2 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 affine, 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 4.2 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 $\boxplus$. $\square$

One can adapt the definition of ϱ-decreasing to the alternative interpretation $[\![\ ]\!]^\bullet$ and get the corresponding theorem. It is more awkward to apply though, because the interpretations of J and $J\!\downarrow$ might be unrelated.

5 Applying the method

We can apply theorem 4.2 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.

5.1 Enriched λ-Calculus

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 [46]. 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

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

Figure 4: Rules for $\lambda^{\to,\times,+}$

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 $\mathsf{ARS}^{\boldsymbol{M}} \downdownarrows \boldsymbol{\alpha}$ 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 3.4.

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.

The rewrite rules for β- and η-reduction of this calculus are shown in figure 4. 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 (-ally ρ-decreasing) interpretation of the schematic symbols.

The easiest to find case are the products as their rules are purely first-order. We have already sketched this in the introduction and we only have to make a slight adaptation due to the nature of our $\boxplus$ operation.

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

The three rules involving products (3–5) are clearly ϱ-decreasing: we can ignore the normalisation and get

$$
[\![\texttt{FST}\ (\texttt{PAIR}\ x\ y)]\!]_\rho = (1\#\bar{\tau}(\varrho(y))) \boxplus_\sigma \varrho(x) >_\sigma \varrho(x) = [\![x]\!]_\rho
$$

The second projection is dual. For the pair constructor we have:

$$\llbracket \texttt{PAIR (FST } p\texttt{) (SND } p\texttt{)} \rrbracket_\rho =$$
$$((1\#\bar{\tau}(\pi_2(\varrho(p)))) \boxplus_\sigma \pi_1(\varrho(p)), (1\#\bar{\sigma}(\pi_1(\varrho(p)))) \boxplus_\sigma \pi_2(\varrho(p))) >_{\sigma\sqcap\tau}$$
$$(\pi_1(\varrho(p)), \pi_2(\varrho(p))) = \varrho(p) = \llbracket p \rrbracket_\rho$$

The only problem was to define $\varrho(\texttt{PAIR})$ in such a way that it is a monotonic function in $\sigma \Rightarrow \tau \Rightarrow (\sigma \sqcap \tau)$. This problem is the same (up to isomorphism) as finding a monotonic function in $(\sigma \oplus \tau) \Rightarrow (\sigma \sqcap \tau)$, so we convert between two different forms of products. Now, if $x >_\sigma x'$ then $\texttt{PAIR}\ x\ y >_{\sigma\sqcap\tau} \texttt{PAIR}\ x'\ y$: in the left component we have it by monotonicity of $k \boxplus_\sigma \_$, in the right component by monotonicity of all $\_ \boxplus_\tau k$, $k\#\_$, and $\bar{\sigma}$. This is trivial, because we just build the interpretation by applying endofunctors to ARS morphisms getting other ARS morphisms — all ARS morphisms are monotonic.

Alternatively, we could have interpreted the object-level product type by the tensor product, making the interpretation of PAIR simple (just add the $1 \boxplus \_$ rewrite noise in both components), but then the projections would be slightly more complicated, as they would project from the tensor product. 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 were able to 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 β-reduction has to be taken into account since the rules for function types use second-order variables.

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

We can show that the η-rule (2) is ϱ-decreasing by the same argument as in the product case, because again there are no β-redexes created by substitution application, and the "$1 \boxplus \_$" in the definition of $\varrho(\texttt{LAM})$ ensures that the left-hand sides has a larger value.

The β-rule (1) is only a little bit trickier:

$$\llbracket (\texttt{AP (LAM } f\texttt{)}\ a)^\theta{\downarrow} \rrbracket = \llbracket \texttt{AP (LAM } f^\theta\texttt{)}\ a^\theta \rrbracket = \llbracket \texttt{LAM}\ f^\theta \rrbracket(\llbracket a^\theta \rrbracket) =$$
$$(1 \boxplus_{\sigma\Rightarrow\tau} \llbracket f^\theta \rrbracket)(\llbracket a^\theta \rrbracket) >_\tau \llbracket f^\theta \rrbracket(\llbracket a^\theta \rrbracket) = \llbracket (f^\theta\ a^\theta) \rrbracket \geq_\tau \llbracket (f^\theta\ a^\theta){\downarrow} \rrbracket = \llbracket (f\ a)^\theta{\downarrow} \rrbracket.$$

The last step used proposition 4.4. This example shows that we can ignore β-normalisation as long as it only affects the right-hand sides of rules.

For the mentioned reasons (mainly proposition 4.6), the β-rule cannot be shown (and is not) ϱ-decreasing for the alternative semantic function $\llbracket\ \rrbracket^\bullet$. But one can still show termination of this rule w.r.t. $\llbracket\ \rrbracket^\bullet$ by using a different interpretation ϱ' with $\varrho'(\texttt{AP}_{\sigma,\tau})\ f\ a = \bar{\sigma}(a) \boxplus_\tau f(a)$. This picture is very typical for interpretations that are designed to be decreasing w.r.t. $[\![\ ]\!]^{\bullet}$, i.e. we have to make the interpretation more complicated to compensate for the absence of the analogue of proposition 4.4.

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. This means that we cannot any longer rely on proposition 4.4 for disregarding β-normalisations. Finding the interpretation for the coproduct type is similarly simple, though we have to be careful again to make $\varrho(\texttt{CASE})$ monotonic by collecting the garbage:

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

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 to deal with β-normalisation on both sides of the rule. Instead of relying on proposition 4.4 we have to consider how exactly the interpretation of λ-terms is affected by β-reduction. We can safely assume that f is going to be substituted by a λ-abstraction, since this will be eliminated anyway by β-normalisation on both sides of the rule. We can also assume w.l.o.g. that x is not free in $\theta(f)$.

So, for a substitution θ with $\theta(f) = \lambda x : \sigma + \tau.\ t$ we get as interpretation of the left-hand side of rule 8: if $[\![\theta(c)]\!] = i_1(c')$ then

$$\begin{array}{l}
1 \boxplus_\nu \overline{\tau \Rightarrow \nu}(g') \boxplus_\nu \overline{\sigma}(c') \boxplus_\nu [\![t[\texttt{INL}\ y/x]]\!]_{y\mapsto c'} \geq_\nu \\
1 \boxplus_\nu [\![t[\texttt{INL}\ y/x]]\!]_{y\mapsto c'} >_\nu [\![t[\texttt{INL}\ y/x]]\!]_{y\mapsto c'} = [\![t[\theta(c)/x]]\!] = [\![(f\ c)^{\theta}\!\downarrow]\!].
\end{array}$$

Here we used proposition 4.3 to compose an interpretation with a substitution. The case of the right injection is dual.

5.2 Recursive Functional Programs

The application presented in the last chapter has a rather incestuous flavour, as we used the λ-calculus to show something about ... the λ-calculus. Moreover, the termination property of the given rewrite system may not have impressed the reader, as it was non-recursive in nature, i.e. no (defined) function symbol occurred on the right-hand sides.

The last argument is not really relevant: the absence of function symbols on right-hand sides does not guarantee termination. After erasing the type-indices from the function symbols in the example (forcing e.g. `LAM` to have type $(\tau \to \tau) \to \tau$) we obtain an untyped version of this λ-calculus which is non-terminating.

But still the problem remains of how to approach a termination proof for a "recursive" rewrite system. To illustrate the standard idea a very simple example will do.

$$\begin{aligned} \mathtt{MAP}\ f\ \mathtt{NIL} &\rightarrow \mathtt{NIL} & (9)\\ \mathtt{MAP}\ f\ (\mathtt{CONS}\ x\ y) &\rightarrow \mathtt{CONS}\ (f\ x)\ (\mathtt{MAP}\ f\ y) & (10) \end{aligned}$$

Figure 5: The Functional MAP

Figure 5 shows the usual MAP functional for lists. Lists are an inductive datatype, i.e. we can characterise it semantically as the colimit of its finite approximations. Following the arguments explained in the semantics section, we can again assume the presence of our decoding and addition functions for lists — provided we have them for the component type. This effectively means to interpret syntactic lists by semantic lists, but on the semantics we have our additional functions to play with.

In such a scenario, it is usually advisable to interpret the constructors by themselves, i.e. $\varrho(\mathtt{NIL}) = []$ and $\varrho(\mathtt{CONS})\ x\ y = x :: y$. How do we find now the interpretation of MAP? The general idea is to search for a semantic function that does essentially the same as the syntactic one, except we have to make it add the "rewrite noise" and the "garbage" on top of the result. Rewrite noise for the MAP functional is the length of the list, i.e. the recursive call in the second rule works with a shorter list. Garbage occurs in the first rule: f is garbage since it is not used on the right-hand side. These observations are enough to find us our interpretation:

$$\varrho(\mathtt{MAP}_{\sigma,\tau})\ f\ [x_1, \ldots, x_n] \quad = \quad \overline{\sigma \Rightarrow \tau}(f) \boxplus (n+1) \boxplus [f(x_1), \ldots, f(x_n)]$$

For this interpretation one can easily show that both rules are ϱ-decreasing and therefore the HRS is terminating.

There are a couple of intrinsic problems with this approach and other similar approaches. One is hidden in the "..." of the above example: when we define a semantic interpretation we have the obligation to show that our interpretation is well-formed — but this might be just as difficult as showing the termination of the rewrite system in the first place. To avoid the informal dots in the definition of $\varrho(\mathtt{MAP})$ we would have to use recursion, and to make sure that the recursive definition of $\varrho(\mathtt{MAP})$ is well-formed we would have to show its termination. This is often a vicious circle, though it should be remarked that it reduces an SN-problem into a WN-problem: if our interpretation is defined just like the rewrite system then it is well-defined if every meta-term (term on the meta-level on which we talk about semantics) has a meta-normal form.

Of course, it does not have to be like that. In the introduction we encountered a non-recursive interpretation that showed the termination of the Ackermann function, but in general such non-recursive interpretation are much more difficult to find.

Another problem is: do we actually have the right semantic domain? Does our proof prove what we want it to prove? In a functional language, we could argue that MAP f is not terminating if f is not for some of its arguments. The reason this did not appear as a problem in the proof is that (i) we simply assumed the arguments of `MAP` to be terminating functions by picking the "right" semantic domain, and (ii) all functions we can build from λ-terms and the signature $\{\mathtt{MAP}, \mathtt{CONS}, \mathtt{NIL}\}$ happen to be terminating. Also, one could argue that lazy languages support *infinite lists* and what about the termination behaviour of `MAP` when applied to infinite lists? Again, the problem did not show up in the proof because our semantic domain had finite lists only and we could not even express infinite lists with terms over our signature.

6 Comparison to other work

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

Robin Gandy presented in [10] 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.

Gandy's paper is not very easily digested — neither is mine, but for a different reason; his method appears superficially as a bag of tricks. In particular, 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.

In [47] Roel de Vrijer has a different aim: the paper is a thorough study of the termination behaviour of the simply typed λ-calculus (without any extensions). In spite of the different aim the techniques are based on Gandy's ideas and they are also closely related to some of the techniques presented here. De Vrijer defines two interpretations of λ-terms; the first is similar to the interpretation we have chosen and easily establishes the termination of β-reduction, the other corresponds to the mentioned alternative interpretation $[\![\;]\!]^\bullet$. He shows that the second interpretation gives a *sharp* upper bound for the length of β-reductions. The sharpness results crucially rely on using natural numbers as the semantic domain for base types as we have indicated above, see proposition 4.5 etc. This means that this property of $[\![\;]\!]^\bullet$ is lost when λ-calculus is added on top of a relation that is terminating, but not ω-terminating.

This paper was originally inspired by Jaco van de Pol's talk at HOA'93 [38] and the difficulties I encountered when trying to apply his method. During one week it happened twice that colleagues popped into my office asking me, as I was considered the local rewrite guru, about the termination of some "rewrite system" which in both cases turned out to be some enriched λ-calculus or process 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 — the problem was that the paper[6]) failed to provide a simple recipe[7], there was no simple theorem one could use without getting through the rest of the paper. The reaction of my fellow colleagues was: "I don't want to become an expert in termination of HRS. I only want to prove my system to be terminating."

Van de Pol interprets β-equivalent terms equally. This has the advantage that the method remains solely semantical: the interpretation is unaffected by reduction and he can compose substitutions and environments — in my approach the normalisation provides a barrier which in some cases fails to disappear and consequently forces us to consider both environments and substitutions simultaneously. The price van de Pol pays for this elegance is a rather odd semantic domain in which functions are split into three classes of monotonic, strictly monotonic, and "strict" functions. This split occurs, because λ-calculus forces the presence of constant functions, which — for reasons we have discussed at length — are very undesirable in the context of termination proofs. So for each type one considers the set of all values (including constant functions) M and its subset $S \subseteq M$ of desirable, "strict", values. For a function to be strict it has to preserve the (strict) order, and its range must only contain strict values. Constant functions do not satisfy the first condition. The categorical treatment of van de Pol's approach is difficult if not impossible, because the second condition excludes the identity functions on higher types from S, as they map constant functions to themselves rather than to strict values.

If all function symbols are interpreted as strict values then all contexts are strict as well, and so the problem of proving the rewrite relation to be terminating is reduced to showing that the left-hand sides are bigger than the right-hand sides under the given interpretation. The problem with van de Pol's first attempt (prliminary version of [38]) was that it was actually difficult to find a strict interpretation, i.e. the approach was difficult to apply. A slight technical modification in the definition of strict value (due to Schwichtenberg) made the approach a lot more practical [40]: by "adding" a strict value to an arbitrary one always gets a strict value, and another lemma provided a particular strict value for every type. These lemmas constitute a big step towards the simple recipe the first version was lacking. They also link the approach with this article: the addition corresponds to our operation $\boxplus$ and the particular values to our decoding maps.

Having a solely semantic approach has a rather surprising weakness, because the semantic domains contain values that do not correspond to terms or substitutions. Here is an example that shows the weakness:

$$\mathtt{A}\,(f\;(f\;x\;\mathtt{C})\;x) \to f\;x\;x$$

[6]To be precise: I am talking about the version that appeared in the preliminary proceedings, the only version I had access to at the time. The version in the final proceedings was an improvement.

[7]When I discussed this problem with Jaco van de Pol, he complained that this argument was rather *ad hominem*. It is indeed and so it should be.

This rule is terminating since for every substitution instance the left-hand side has more $\mathtt{A}$'s than the right-hand side. We can also prove this rule terminating with our method: we interpret the type of $\mathtt{C}$ as ω, and set $[\![\mathtt{C}]\!] = 0$ and $[\![\mathtt{A}]\!](x) = x + 1$. W.l.o.g. we can assume that the substitute for f is $\lambda y{:}\,\sigma.\,\lambda z{:}\,\sigma.\,t$ with t in normal form. We have to consider 4 cases for t, i.e. whether the result value depends on y, on z, on neither, or on both. If t does not contain y then the interpretation of the left-hand side is just one bigger than the interpretation of the right-hand side. If it contains x but not y then $[\![(\theta(f)\,y\,z)\!\downarrow]\!] = g\,y$ for some $g \in \mathsf{ARS}(\omega, \omega)$; we can exploit the fact that any such g is pointwise greater or equal than the identity function and thus $[\![l^\theta\!\downarrow]\!] = 1 + g(g\,x) > g(g\,x) \geq g\,x = [\![r^\theta\!\downarrow]\!]$. Finally, if both x and y occur in t then $[\![(\theta(f)\;y\;z)\!\downarrow]\!] = h\;y\;z$ for some $h : \omega \otimes \omega \to \omega$, and again any such h is pointwise greater or equal than addition [15]: we get $h\,x\,0 \geq x + 0 = x$ and by monotonicity $h\;(h\,x\,0)\;x \geq h\;x\;x$. The result follows easily.

However, this very example cannot be shown terminating using van de Pol's approach, because the semantic domain he considers has too many "nasty" values. Given any interpretation $[\![\_]\!]$, we can define an environment υ as follows:

$$\begin{array}{lcl} \upsilon(x) & = & [\![\mathtt{A}\;\mathtt{C}]\!] \\ \upsilon(f) & = & \underline{\lambda} y.\underline{\lambda} z.\text{if } y \geq \upsilon(x) \wedge z \geq \upsilon(x) \text{ then } \upsilon(x) \text{ else } [\![\mathtt{C}]\!] \end{array}$$

The notation used to describe $\upsilon(f)$ is informal and operates on the semantic level, the underlined $\underline{\lambda}$ indicate abstraction on the meta-level. By instantiating the variable f with the first projection we have $\mathtt{A}\;x \to_R x$ for any x (especially $x = \mathtt{C}$) and thus we would need $[\![\mathtt{A}\;\mathtt{C}]\!] > [\![\mathtt{C}]\!]$ in any semantic termination proof. But then the above environment υ is well-defined ($\upsilon(f)$ is monotonic w.r.t. $\geq$) and we have $[\![\mathtt{A}\;(f\;(f\;x\;\mathtt{C})\;x)]\!]_\upsilon = \upsilon(x) = [\![f\;x\;x]\!]_\upsilon$. The example is typical for showing that a Poset-model is not fully abstract for a sequential language, as $\upsilon(f)$ is a kind of parallel-or.

Jaco van de Pol himself found another, even simpler example of this kind

$$\mathtt{A}\,(f\;(f\;x)) \to f\;x$$

which again (for similar reasons) can be proved terminating in my approach but not in his. Both examples make use of the half-syntactic nature of my interpretations. This is typical for such examples, at least at the lower end of the type hierarchy.

While Gandy indicates how his method could be extended to other parametric types, van de Pol restricts his attention to atomic types, product types and function types; moreover the treatment of product types appears rather *ad hoc*. We have addressed this problem in a more general way by investigating the abstract properties the semantic object corresponding to a parametric type (i.e. a functor) has to satisfy in order to preserve the essential ingredients of the technique.

All three papers solely deal with natural numbers as the domain for atomic types, at least they do not offer any assistance for larger ordinals. As we have explained, this is an unnecessary restriction: the general techniques carry over to larger (indecomposable) ordinals than ω. Although the general techniques carry over, not everything else does; most problems that arise on larger ordinals originate in the difference between natural sum # and ordinal sum +. Most prominently, the function # is the smallest strictly monotonic function in $\omega \otimes \omega \to \omega$, but it loses this property on larger ordinals; dually, ordinal sum + is not a morphism in $\alpha \otimes \alpha \to \alpha$ for $\alpha > \omega$.

7 Conclusion and further work

We have presented a semantic approach to termination proofs for higher-order term rewriting systems. It consists of two parts. The first parst is an extended foundation for semantic termination proofs in general, based on the category of Abstract Reduction Systems. The second part links the semantic foundation with the syntax of Higher-Order Rewrite Systems.

To apply the method one does not have to go into the depths of the first part, theorem 3.4 lists the properties one needs to obtain the toolbox the approach provides.

The essence of the semantic set-up is this containing 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, together with a bunch of properties, in particular $n \boxplus x \geq x$ for any $x \in [\![\tau]\!]$, $n \in \boldsymbol{\alpha}$. This on its own is enough to tackle termination proofs for quite a few 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. For (term-) dependent types this should not be too difficult [11], e.g. using slice categories, but polymorphism appears to be a much harder feature to model.

Acknowledgements

This paper evolved over some period of time. People who helped developing it by providing feedback, advice, references, e-mail, etc. include (in alphabetical order): Randall Dougherty (he proved proposition 4.5), Marcello Fiore, Philippa Gardner, Wolfgang Gehrke, 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.

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

References

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

[2] Hendrik P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics.* North-Holland, 1984.

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

[4] Michael Barr and Charles Wells. *Toposes, Triples and Theories.* Springer, 1985.

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

[6] 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.

[7] Maria da Conceição Fernández Ferreira. *Termination of term rewriting.* PhD thesis, Universiteit Utrecht, 1995.

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

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

[10] 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.

[11] Wolfgang Gehrke. Monadic laws from purely functional programming as a canonical higher-order rewrite system. In *Higher-Order Algebra, Logic and Term Rewriting, Second International Workshop*, 1995. (Participants' Proceedings).

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

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

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

[15] Ernst Jacobsthal. Über den Aufbau der transfiniten Arithmetik. *Mathematische Annalen*, 66:145–194, 1909.

[16] Ernst Jacobsthal. Zur Arithmetik der transfiniten Zahlen. *Mathematische Annalen*, 67:130–144, 1909.

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

[18] Stefan Kahrs. Confluence of curried term-rewriting systems. *Journal of Symbolic Computation*, 19(6):601–623, 1995.

[19] Stefan Kahrs. Towards a domain theory for termination proofs. In *Rewriting Techniques and Applications*, pages 241–255, 1995. LNCS 914.

[20] Stefan Kahrs. The variable containment problem. In *Higher-Order Algebra, Logic and Term Rewriting, Second International Workshop, Selected Papers*, pages 109–123, 1995. LNCS 1074.

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

[22] Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21:15–39, 1996.

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

[24] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbai, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 2*, pages 1–116. Oxford University Press, 1992.

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

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

[27] 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.

[28] Pierre Lescanne. Termination of rewrite systems by elementary interpretation. *Formal Aspects of Computing*, 7:77–90, 1995.

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

[30] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. Technical Report TUM-I94333, Technische Universität München, 1994.

[31] Aart Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In *Rewriting Techniques and Applications*, pages 263–277, 1989. LNCS 355.

[32] J.C. Mitchell. Type systems for programming languages. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 8, pages 365–458. Elsevier, 1990.

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

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

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

[36] 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.

[37] 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.

[38] 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.

[39] Jaco van de Pol. *Termination of Higher-Order Rewrite Systems.* PhD thesis, Utrecht University, 1996.

[40] Jaco van de Pol and Helmut Schwichtenberg. Strict functionals for termination proofs. In *Typed Lambda Calculi and Applications*, pages 350–364, 1995. LNCS 902.

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

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

[43] J.J.M.M. Rutten. A calculus of transition systems. In Alban Ponse, Maarten de Rijke, and Yde Venema, editors, *Modal Logic and Process Algebra*, pages 231–256. CSLI pubplications, 1995.

[44] Richard Seifert. *Property Analysis of Term Rewriting Systems.* PhD thesis, Universität Bremen, 1993.

[45] Wacław Sierpiński. *Cardinal and Ordinal Numbers.* Polish Scientific Publishers, 1965.

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

[47] Roel de Vrijer. Exactly estimating functionals and strong normalisation. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschapen*, 90(4):479–493, 1987.

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

[49] Hans Zantema. Termination of term rewriting by interpretation. In *3rd International Workshop on Conditional Term Rewriting Systems*, pages 155–167, 1992.

[50] Hans Zantema. Type removal in term rewriting. In *3rd International Workshop on Conditional Term Rewriting Systems*, pages 148–154, 1992. LNCS 656.

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