Nordic Journal of Computing

Rewriting Via Coinserters

NEIL GHANI
University of Leicester, Department of Mathematics and Computer Science
University Road, Leicester LE1 7RH, United Kingdom
ng13@mcs.le.ac.uk

CHRISTOPH LÜTH
Universität Bremen, FB 3 — Mathematik und Informatik
Postfach 330 440, D-28334 Bremen, Germany
cxl@informatik.uni-bremen.de

Abstract. This paper introduces a semantics for rewriting that is independent of the data being rewritten and which, nevertheless, models key concepts such as substitution which are central to rewriting algorithms. We demonstrate the naturalness of this construction by showing how it mirrors the usual treatment of algebraic theories as coequalizers of monads. We also demonstrate its naturalness by showing how it captures several canonical forms of rewriting.

CR Classification: F.4.m [Mathematical Logic and Formal Languages]: Miscellaneous — Rewrite Systems, Term Rewriting Systems; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic — Lambda Calculus and Related Systems.

Key words: Rewriting, Monads, Computation, Category Theory.

1. Introduction

Rewrite systems offer a flexible and general notion of computation that has appeared in various guises over the last hundred years. The first rewrite systems were used to study the word problem in group theory at the start of the twentieth century [Epstein 1992], and were subsequently further developed in the 1930s in the study of the λ-calculus and combinatory logic [Church 1940]. In the 1950s, Gröbner basis were developed to solve the ideal membership problem for polynomials [Buchberger and Winkler 1998]. In the 1970s, starting with Knuth-Bendix completion [Knuth and Bendix 1970], term rewriting systems (TRSs) have played a prominent rôle in the design and implementation of logical and functional programming languages, the theory of abstract data types, and automated theorem proving [Baader and Nipkow 1998].

Each of the above forms of rewriting uses rewriting over a different data structure i) computational group theory uses rewriting over free monoids or, equivalently, strings; ii) Gröbner basis uses rewriting over free rings or polynomials; iii) TRSs use rewriting over first order terms; and iv) higher

Received January 15, 2004.

order rewriting uses rewriting over data structures where variable binding is present. This proliferation of data structures to which rewriting can be applied has lead to the search for a meta-theory of rewriting which is independent of the data being rewritten. For example, the construction of a Gröbner basis is clearly equivalent to the Knuth-Bendix completion of a term rewriting systems, yet we lack a mathematical formalism within which this observation can be expressed.

This paper seeks to provide a model of rewriting independent of the data being rewritten. Our ideas can be traced back to Gordon Plotkin who remarked to us, "Rewrite systems are anti-symmetric equational logic". For example, if one takes the standard presentation of equational logic and deleted the symmetry rule, one would get precisely the definition of a term rewriting system. However, the real insight in this remark is that it holds for all rewrite systems, not only term rewriting systems. Thus one can provide a model of rewriting independent of the underlying data structure if we can generalise equational logic to arbitrary data structures, and avoid the symmetry in this construction.

Consequently, there are two ingredients to our model, namely *monads* and *coinserters*:

- *Why monads?* If we are to provide a model of rewriting independent of the data being rewritten, we have to ask what properties the data structure must posses for rewriting to take place. The essence of rewriting and equational logic is the substitution of a part of a term in some formal language with some other term. *Monads* provide an abstraction of the notion of a term calculus equipped with a well behaved substitution and as such covers strings, rings, first and higher order terms etc. Thus, by axiomatizing the data to be rewritten as a monad, our models cover all of the examples above.
- *Why coinserters?* Equational logic can be seen as axiomatizing quotients and hence a general treatment of equational logic is as the coequaliser of two monads. A coequaliser of a pair $s, t : T_E \longrightarrow T_\Sigma$ of monad morphisms is the monad T_Σ quotiented by the least equivalence relation identifying terms in the image of s and t — exactly what an algebraic theory does. An asymmetric variant of the coequaliser is the *coinserter* where the coinserter of $s, t : T_E \longrightarrow T_\Sigma$ is the smallest monad including T_S where terms in the image of s are bigger (i.e. have a morphism to) the corresponding term in the image of t. This is of course what one does when constructing the rewrite relation.

Thus, whenever we can define a notion of equational logic we can also define a notion of rewriting by simply replacing the coequaliser of two monads with their coinserter. This is precisely the sense in which our model is independent of the data being rewritten. Explaining the relationship between rewriting and equational logic as the relationship between the coequaliser and the coinserter is, for us, a very elegant observation. The simplicity and naturality of this perspective is made concrete by the examples included in this paper.

Other abstract models of rewriting have been proposed with the simplest being *Abstract Reduction Systems* (ARSs) based upon relations [Baader and Nipkow 1998]. These models throw away the structure of the data being rewritten and solely focus on properties of the reduction relation itself. Consequently, while useful for certain results, ARSs lack sufficient structure to adequately model key concepts such as substitution, context and the layer structure whereby terms from one system are layered over terms from another in modularity problems. Hence ARSs are mainly used as an organisational tool with the difficult results proved directly at the syntactic level.

Category theory has been used to provide a semantics for term rewriting systems at an intermediate level of abstraction between the actual syntax and the relational model. Research originally focused on structures such as *2-categories* [Rydeheard and Stell 1987, Seely 1987], *Sesqui-categories* [Stell 1994] and *ordered categories* [Jay 1990]. Significant applications of this work to the theory of rewriting include generalised forms of Knuth-Bendix Completion [Stokkermans 1992, Reichel 1991], expansionary rewrite rules [Ghani 1995] and the theory of residuals [Melliès 1998]. This paper builds upon previous results by the authors who used monads to model term rewriting systems and consequently gave a monadic proof of Toyama's theorem concerning the modularity of confluence for TRSs [Lüth 1998, Lüth and Ghani 1997]. The 2-categorical models then arise essentially as the Kleisli categories of these monads.

We would like to thank John Power for his help and the referees for their comments which have helped make this paper accessible to a wider audience. We assume a passing knowledge of rewriting as found in [Baader and Nipkow 1998] or [Klop 1992] and the basic concepts of category theory, such as categories, functors, natural transformations, limits and colimits, adjunctions and monads [MacLane 1971]. In some places we have to employ more advanced categorical constructions which are summarised in Appendix A allowing readers not familiar with these constructions to follow the thrust of our argument. Further, a running example is included to which the reader may refer while reading the main body of the paper.

The rest of the paper is structured as follows: Section 2 recaps the literature on categorical, and specifically monadic, approaches to universal algebra while Section 3 shows how we can form an equational theory via a coequaliser, and a rewrite system via a coinserter. Section 4 contains examples showing how many common forms of rewriting arise as coinserters while section 5 contains some concluding remarks.

2. Signatures, Equations and Monads

Since our approach to rewriting generalises the categorical treatment of universal algebra, where algebraic theories are equivalent to monads on the category of sets, we give a presentation of this equivalence. However, since this standard material, we omit proofs and instead refer the reader to [MacLane 1971] or [Barr and Wells 1985].

2.1 Signatures and Terms

DEFINITION 1. (SIGNATURE) *A (single-sorted)* signature *consists of a function* $\Sigma : \mathbb{N} \to \mathbf{Set}$. *The set of* n-ary operations *of* Σ *is defined* $\Sigma_n \stackrel{def}{=} \Sigma(n)$. □

As a running example, we use the following signature for addition:

EXAMPLE 1. (ADDITION) The signature $\Sigma_{Add} : \mathbb{N} \to \mathbf{Set}$ for the theory of addition is defined as $\Sigma_{Add}(0) = \{0\}, \Sigma_{Add}(1) = \{S\}, \Sigma_{Add}(2) = \{+\}$ and $\Sigma_{Add}(n) = \emptyset$ for all other $n \in \mathbb{N}$. Thus Σ_{Add} declares one operation 0 of arity 0 (a constant), one unary operation S and one binary operation $+$ (written in infix notation). □

Given a signature, we construct terms as follows:

DEFINITION 2. (TERM ALGEBRA) *Given a signature* Σ *and a set of variables* X, *the* term algebra $T_\Sigma(X)$ *is defined inductively:*

$$\frac{x \in X}{x \in T_\Sigma(X)} \qquad\qquad \frac{f \in \Sigma(n) \quad t_1, \dots t_n \in T_\Sigma(X)}{f(t_1, \dots, t_n) \in T_\Sigma(X)}$$

□

Of course, the term algebra $T_{\Sigma_{Add}}(X)$ only constructs terms and does not enforce any equations; for example, in the addition signature from Example 1, if $x \in X$, then x and $0 + x$ are different elements of $T_{\Sigma_{Add}}(X)$. Adding equations to signatures to form algebraic theories is covered in Section 2.4.

The term algebra construction, which for every set X gives us the set of terms $T_\Sigma(X)$, extends to a functor $T_\Sigma : \mathbf{Set} \to \mathbf{Set}$ which maps a function $f : X \to Y$ to the function $T_\Sigma(X) \to T_\Sigma(Y)$ which renames variables appropriately. Further, every variable $x \in X$ gives a term $x \in T_\Sigma(X)$, so the variables are given by a function $\eta_X : X \to T_\Sigma(X)$. Lastly, substitution takes terms built over terms and flattens them, as described by a function $\mu_X : T_\Sigma(T_\Sigma(X)) \to T_\Sigma(X)$. Thus, we have all the data for a monad:

DEFINITION 3. (MONAD) *A* monad $T = \langle T, \eta, \mu \rangle$ *on a category* $\mathcal{C}$ *is given by an endofunctor* $T : \mathcal{C} \longrightarrow \mathcal{C}$, *called the* action, *and two natural transformations,* $\eta : 1 \longrightarrow T$, *called the* unit, *and* $\mu : TT \longrightarrow T$, *called the* multiplication *of the monad, satisfying the* monad laws*:*

$$\mu \cdot T\eta = 1 = \mu \cdot \eta_T \qquad \mu \cdot T\mu = \mu \cdot \mu_T .$$

□

As suggested above, the term algebra construction forms a monad:

LEMMA 1. *For every signature* Σ, $\mathsf{T}_\Sigma = \langle T_\Sigma, \eta, \mu \rangle$ *is a monad.*

PROOF. We have already given the data. Naturality of unit and multiplication is easily verified while the monad laws boil down to associativity of substitution and that variables form left and right units for substitution. □

Monads also model a number of other important structures in computer science, such as (many-sorted) algebraic theories, non-well-founded syntax [Moss 2001], term graphs [Ghani *et al.* 2002], calculi with variable binders [Fiore *et al.* 1999], term rewriting systems [Lüth 1998], and, via computational monads [Moggi 1989], state-based computations, exceptions, continuations etc.

The fact that the term algebra $T_\Sigma(X)$ is inductively defined is expressed categorically by the fact that $T_\Sigma(X)$ is the initial algebra of the functor $X + F_\Sigma$ which sends Y to $X + F_\Sigma Y$, where F_Σ is the polynomial endofunctor

$$F_\Sigma(Y) = \coprod_{n\in\mathbb{N}, f\in\Sigma(n)} Y^n = \coprod_{n\in\mathbb{N}} Y^n \times \Sigma(n) \tag{2.1}$$

The structure map $X + F_\Sigma T_\Sigma(X) \longrightarrow T_\Sigma(X)$ is equivalent to two maps $X \longrightarrow T_\Sigma(X)$ and $F_\Sigma T_\Sigma(X) \longrightarrow T_\Sigma(X)$ stating that all variables are contained in $T_\Sigma(X)$, and all operations can be interpreted in $T_\Sigma(X)$. Initiality of this algebra says that $T_\Sigma(X)$ is the smallest such set. For our running example, we have

$$F_{\Sigma_{Add}}(Y) = Y^0 + Y^1 + Y^2 = 1 + Y + Y^2$$

Note that we can explicitly describe F_Σ as calculating the terms of depth one over a set of variables; for example,

$$F_{\Sigma_{Add}}(\{x, y\}) = \{0, Sx, Sy, x + x, x + y, y + x, y + y\}. \tag{2.2}$$

2.2 Models and Algebras

Given a signature, we have seen how the term algebra construction can be expressed abstractly using the categorical concept of a monad. This subsection shows how models of signatures can also be treated categorically. Recall that if Σ is a signature, a Σ-algebra is a set together with an interpretation for each of the operations while a Σ-algebra homomorphism is a function on the underlying sets which respects the interpretations of the operations [Manes 1976]. These definitions are precisely captured by the following definition.

DEFINITION 4. (ALGEBRA OF AN ENDOFUNCTOR) *Let* $F : \mathcal{C} \longrightarrow \mathcal{C}$ *be a functor. An* F-algebra *is a pair* (X, h) *where* X *is in* $\mathcal{C}$, *called the* carrier, *and* $h : FX \longrightarrow X$ *is called the* structure map. *The category* $F{-}\mathbf{alg}$ *has* *as objects F-algebras and a map from the F-algebra (X, h) to the F-algebra (X', h') is a map $f : X \longrightarrow X'$ such that*

$$\begin{tikzcd} FX \arrow[r, "h"] \arrow[d, "Ff"'] & X \arrow[d, "f"] \\ FX' \arrow[r, "h'"'] & X' \end{tikzcd}$$

□

For example, an $F_{\Sigma_{Add}}$-algebra is a set A and a map $1 + A + A \times A \longrightarrow A$, which corresponds to a constant in A, a map $A \longrightarrow A$ (the unary operation) and a map $A \times A \longrightarrow A$ as expected. A more general notion of model which works for arbitrary monads rather than just those arising from signatures are given by the following definition.

Definition 5. (Eilenberg-Moore Algebra) *Let $T = \langle T, \eta, \mu \rangle$ be a monad on $\mathcal{C}$. An Eilenberg-Moore algebra for T is an algebra (X, h) of the underlying functor T which commutes with the unit and multiplication, i.e.*

$$h \cdot Th = h \cdot \mu_X \qquad h \cdot \eta_X = 1_X$$

A morphism of Eilenberg-Moore algebras is a morphism of the underlying algebras of the endofunctors. This gives the category of Eilenberg-Moore algebras $T-\mathbf{Alg}$, which is a full subcategory of $T-\mathbf{alg}$. □

In future, we follow standard practice and call Eilenberg-Moore algebras simply algebras when there is no danger of confusion. The functor $U^T : T-\mathbf{Alg} \longrightarrow \mathcal{C}$ mapping an algebra (X, h) to its underlying object X has a left adjoint $F^T : \mathcal{C} \longrightarrow T-\mathbf{Alg}$, mapping an object X to the *free Eilenberg-Moore algebra* (TX, μ_X). The adjunction $U^T F^T$ results in a monad which is in turn isomorphic to T (see [MacLane 1971], Chapter VI.2). Crucially, this means that a monad is determined by its algebras and vice versa. We shall use this fact later to reason about a monad T by reasoning about the associated category $T-\mathbf{Alg}$.

Example 2. (Algebras for Σ_{Add}) The set $\mathbb{N}$ of natural numbers forms a $T_{\Sigma_{Add}}$-algebra, with the structure map $h : T_{\Sigma_{Add}}(\mathbb{N}) \to \mathbb{N}$ defined inductively as follows:

$$h(n) = n, h(0) = 0, h(S(t)) = h(t) + 1, h(s + t) = h(s) + h(t).$$

Because of the first equation $(\mathbb{N}, h)$ is also a Eilenberg-Moore algebra (i.e. a T_Σ-algebra). □

Indeed the inductive nature of T_Σ means that the category $T_\Sigma-\mathbf{Alg}$ is isomorphic to $F_\Sigma-\mathbf{alg}$ and hence we could have picked any non-empty set Z with arbitrary constant, unary and binary operations.

2.3 From **Set** *to lfp categories*

The definition of signature and our running example was formulated in the category of sets, whereas the definitions of monads and algebras were given more generally for any category $\mathcal{C}$. This allows us to generalise away from the category of sets so as to account for other data structures and to replace equations by reductions. In particular, we can extend the definition of signature, polynomial endofunctor and free monad to any *locally finitely presentable* (lfp) category — see Appendix A for their definition. In such a setting, we define a signature to be a functor $\Sigma : \mathcal{N} \longrightarrow \mathcal{A}$ where $\mathcal{N}$ is the discrete subcategory of the representing set of finitely presentable objects (the appropriate categorical counterpart of the natural numbers).

Definition 6. *Let $\mathcal{A}$ be an lfp category. The category of signatures in $\mathcal{A}$ is written $\mathbf{Sig}(\mathcal{A})$ and is the functor category $\mathbf{Sig}(\mathcal{A}) = [\mathcal{N}, \mathcal{A}]$ where $\mathcal{N}$ is the discrete subcategory formed by the representing set of finitely presentable objects of $\mathcal{A}$.* □

Equation (2.1) generalises to such generalised signatures by setting F_Σ to be the left *Kan extension* (again see Appendix A) along the inclusion $J : \mathcal{N} \longrightarrow \mathcal{A}$, i.e. $F_\Sigma = \mathsf{Lan}_J\Sigma$. As a generalised polynomial, F_Σ is always finitary, i.e. preserves filtered colimits. The associated term algebra $T_\Sigma : \mathcal{A} \to \mathcal{A}$ is then the free monad over the endofunctor F_Σ, which can be constructed in a number of ways [Kelly 1980]:

Lemma 2. *Let F be a finitary endofunctor over an lfp category $\mathcal{A}$. Then the free monad H_F on F satisfies the following:*

(1) For every X in $\mathcal{A}$, $H_F X$ is the carrier of the initial $X + F$-algebra.

(2) The forgetful functor $U : F\mathbf{-alg} \longrightarrow \mathcal{A}$ from the category of F-algebras to $\mathcal{A}$ has a left adjoint L, and $H_F \cong UL$.

(3) H_F is the colimit of the sequence

$$S_0 = 1 \quad S_{n+1} = 1 + FS_n \quad H_F = \operatorname*{colim}_{i<\omega} S_i \tag{2.3}$$

The sequence S_i in (2.3) in Lemma 2 is called the *free algebra sequence* [Kelly 1980] and can be seen as a uniform calculation of the initial $X + F-$ algebra. Lemma 2 and the adjunction inherent in the definition of the left Kan extension along J give the following adjunctions:

$$\begin{tikzcd} \mathbf{Sig}(\mathcal{A}) \arrow[r, shift left=2, "\mathsf{Lan}_{J\_}"] \arrow[r, phantom, "\perp"] & {[\mathcal{A},\mathcal{A}]_f} \arrow[l, shift left=2, "\_\circ J"] \arrow[r, shift left=2, "H_\_"] \arrow[r, phantom, "\perp"] & \mathbf{Mon}(\mathcal{A}) \arrow[l, shift left=2, "V"] \end{tikzcd} \tag{2.4}$$

where $[\mathcal{A},\mathcal{A}]_f$ is the category of finitary endofunctors over $\mathcal{A}$ and $\mathbf{Mon}(\mathcal{A})$ is the category of finitary monads over $\mathcal{A}$ (see Appendix A for the definition this category). Composing the two adjunctions in (2.4), we obtain the adjunction

$$\mathbf{Sig}(\mathcal{A}) \underset{U}{\overset{T_{\_}}{\underset{\perp}{\rightleftarrows}}} \mathbf{Mon}(\mathcal{A}) \tag{2.5}$$

This adjunction maps a signature Σ to the *free monad* T_Σ on Σ. In summary, we have given a general treatment of signatures, and term algebras as initial algebras, for any lfp category $\mathcal{A}$. The full benefit of this generalisation will become apparent in Section 3, when we move from the category of sets to the category of preorders, replacing equations with reductions. But first, we need to treat equations.

2.4 Equations

DEFINITION 7. (EQUATIONS AND ALGEBRAIC THEORIES) *Given a signature Σ, a Σ-equation is of the form $(Y \vdash l = r)$ where Y is a set of variables and $l, r \in T_\Sigma(Y)$. An* algebraic theory *$\langle \Sigma, E\rangle$ consists of a signature Σ and a set E of Σ-equations.* □

The term algebra construction generalises from signatures to algebraic theories by mapping a set X to the term algebra quotiented by the equivalence relation $\sim$ generated from the equations. That is, $T_{\langle \Sigma,E\rangle}(X) = T_\Sigma(X)/\!\sim$.

$T_{\langle \Sigma,E\rangle}$ is actually a monad with unit and multiplication given as for T_Σ — crucially, once this result is established, one can then reason algebraically using monad theoretic arguments without having to resort to picking representatives of equivalence classes etc. Although $T_{\langle \Sigma,E\rangle}$ is not the free monad over any signature, it does have a universal property. To see this, note that an algebraic theory $\langle \Sigma, E\rangle$ specifies a set of equations, each of which has a left hand side and a right hand side. Thus we can reformulate a Σ-algebraic theory as a signature $E : \mathbb{N} \longrightarrow \mathbf{Set}$, and a family of pairs of maps

$$E(c) \underset{r_c}{\overset{l_c}{\rightrightarrows}} T_\Sigma(c) \tag{2.6}$$

in $\mathbf{Set}$ for $c \in \mathbb{N}$. One should regard E as giving the context or arity of the equations, and l and r the actual left-hand sides and right-hand sides. Resuming our running example:

EXAMPLE 3. (ADDITION CONTINUED) Let Σ_{Add} be the signature defined in Example 1. The algebraic theory $Add = \langle \Sigma_{Add}, \Phi_{Add}\rangle$ implements addition on the natural numbers and has equations $\Phi_{Add} = \{e_1, e_2\}$ where

$$\begin{aligned} e_1 &= (\{x\} \vdash 0 + x = x) \\ e_2 &= (\{x, y\} \vdash Sx + y = S(x + y)) \end{aligned}$$

The first equation has a context of one variable, and hence arity 1; the second equation has a context of two variables, and hence arity 2. Hence, the signature E_{Add} is $E_{Add}(1) = \{e_1\}$ and $E_{Add}(2) = \{e_2\}$. Since $E(c)$ is empty for $c \neq 1, 2$, the family of morphisms $l_c, r_c : E_{Add}(c) \longrightarrow T_{\Sigma_{Add}}(c)$ is defined by

$$\begin{array}{rclcrcl} l_1(e_1) & = & 0 + x & \qquad & l_2(e_2) & = & S(x + y) \\ r_1(e_1) & = & x & & r_2(e_2) & = & Sx + y \end{array}$$

□

Since $\mathbb{N}$ is discrete, the families of l_c and r_c of maps form two natural transformations

$$E \overset{l}{\underset{r}{\rightrightarrows}} UT_\Sigma.$$

Under adjunction (2.5), these natural transformations correspond to a pair of monad morphisms, which we call a *presentation*:

$$T_E \overset{l'}{\underset{r'}{\rightrightarrows}} T_\Sigma. \tag{2.7}$$

The action T_E contains proof terms for equalities, with l and r giving the actual equalities. In Example 3 for instance, one may have the proof term $\phi = e_2(e_1(z), z) \in T_\Sigma(Z)$ if $z \in Z$. This corresponds to equation e_2 applied at top level, with the first variable instantiated with the equation e_1 and the second variable instantiated with just z. The maps l' and r' map this proof term to two different terms in $T_\Sigma(Z)$, by replacing e_1 and e_2 with the corresponding left and right-hand sides: $l(\phi) = S(0 + z) + z$ and $r(\phi) = S(z + 0)$.

In the monad representing the equational theory of Add, $l(\phi)$ and $r(\phi)$ should be mapped to the same term. This suggests that the representing monad $T_{\langle\Sigma,E\rangle}$ of Add should be the *coequalizer* of the presentation (2.7) in the category $\mathbf{Mon}(\mathbf{Set})$. By taking this coequalizer in the category of monads, the coequalizing map q is a monad morphism. In particular, it commutes with substitution, and thus is a congruence. The universal property of the coequalizer makes it the smallest such.

The above discussion generalises to lfp categories. Moreover, the adjunction (2.5) is of *descent type* [Kelly and Power 1993], which means that each component of the counit of the adjunction is a coequaliser. In other words, *every* finitary monad T on $\mathcal{A}$ is a coequaliser of two free monads over two signatures $B, E : \mathcal{N} \longrightarrow \mathcal{A}$, or equivalently, every such monad is represented by equations in this general sense. As a technical note, when we move from the category of sets to an lfp category $\mathcal{A}$, we need to *enrich* the construction and consider $\mathcal{A}$ as enriched over a monoidal category; for us we use the case of **Pre** enriched over itself as we need the hom-sets to also be preorders. However, we de-emphasize this here for better readability, but note that the constructions in the following are actually enriched, even if we do not say so explicitly.

To sum up this section, we have seen how the key concepts of universal algebra (terms, substitution, equations) can be modelled in any lfp category $\mathcal{A}$. Signatures are functors $\mathcal{N}$ to $\mathcal{A}$, from which we get an endofunctor over $\mathcal{A}$ by a left Kan extension, and the term algebra is the free monad over this endofunctor. Algebraic theories are given by coequalizers of free monads and all finitary monads arise in this way. We will now move from sets to preorders in order to model reductions, making use of the more general framework developed in this section.

3. From Universal Algebra to Rewriting

The aim of this section is to give an abstract characterisation of rewriting. This is achieved by replacing sets with preorders according the general scheme presented in the last section. Crucially, just as coequalisers are a general notion for algebraic theories (operations and equations, closed under congruence), the categorical construction of a *coinserter* is a general notion of rewriting systems (operations and rewrite rules, closed under anti-symmetric congruence).

3.1 Term Rewriting Systems and How to Model Them

DEFINITION 8. (REWRITE RULES AND TRSS) *Given a signature Σ, a Σ-rewrite rule is of the form $(Y \vdash l \to r)$ where Y is a set of variables and $l, r \in T_\Sigma(Y)$. A* term rewriting system $\mathcal{R} = \langle \Sigma, R \rangle$ *consists of a signature Σ and a set R of Σ-rewrite rules.* □

Usually one requires that l is not a variable and that the free variables of r are contained in the free variables of l, but semantically these restrictions are not necessary and hence omitted here.

EXAMPLE 4. (ADDITION VIA REWRITING) The TRS $\mathcal{R}_{Add} = \langle \Sigma_{Add}, R_{Add} \rangle$ implements addition on the natural numbers and has Σ_{Add} as above (Ex. 1) and $R_{Add} = \{r_1, r_2\}$ where

$$
\begin{aligned}
r_1 &= (\{x\} \vdash 0 + x \to x) \\
r_2 &= (\{x, y\} \vdash Sx + y \to S(x + y))
\end{aligned}
$$

□

Notice how according to Definitions 7 and 8, algebraic theories and TRSs are exactly the same thing. The difference between an algebraic theory and a TRS is the semantics we attach to them. While one constructs an equivalence relation from an algebraic theory, one constructs a *reduction relation* $\to_{\mathcal{R}}$ from a TRS by closing the rewrite rules under contexts and substitutions (for the exact definitions, see [Klop 1992] or [Baader and Nipkow

1998]). The *many-step reduction relation* $\twoheadrightarrow_{\mathcal{R}}$ is the reflexive-transitive closure of the one-step reduction relation.

Our categorical approach instantiates the category $\mathcal{A}$ in the framework of Section 2 with the category **Pre** of preorders (transitive-reflexive orders) and monotone functions between them. **Set** and **Pre** are connected by a series of three adjunctions:

LEMMA 3. *Consider the functors* $D : \mathbf{Set} \longrightarrow \mathbf{Pre}$, *which maps a set to the discrete preorder over it, and* $V : \mathbf{Pre} \longrightarrow \mathbf{Set}$, *which maps a preorder to its underlying set. Then* D *is left adjoint to* V.

PROOF. The relevant natural isomorphisms are easily established. □

In fact, D has left adjoint $C : \mathbf{Pre} \longrightarrow \mathbf{Set}$, which maps a preorder to its connected components, and V has a further right adjoint $T : \mathbf{Set} \longrightarrow \mathbf{Pre}$, which maps a set to the total order over it, giving a chain of three adjunctions

$$C \dashv D \dashv V \dashv T$$

but we do not need C and T here.

One can then model Σ-rewrite systems by imposing a reduction order on Σ-algebras — again see [Baader and Nipkow 1998]. A Σ-algebra with a reduction order is called a preordered algebra:

DEFINITION 9. (PREORDERED ALGEBRA) *Let* Σ *be a signature over* **Set**. *A preordered* Σ*-algebra consists of a preorder* X *and a* Σ*-algebra structure over* VX *for which the interpretation of each operation is monotone.*

A morphism between preordered Σ*-algebras is an algebra morphism on the underlying sets which is also monotone, i.e. a morphism in* **Pre**. □

Given a signature Σ, the *free preordered* Σ*-algebra* on a preorder $(X, \Rightarrow)$ is the set of terms $T_\Sigma(X)$, ordered by the smallest preorder which contains the order on X and makes the operations in Σ monotone. It is easily verified that this is indeed free, i.e. for any other preordered Σ-algebra A and monotone function $f : X \longrightarrow A$ there is exactly one preordered Σ-algebra morphism $!_f : T_\Sigma(X) \longrightarrow A$, given by the homomorphic extension of f. In our running example, let $X = \{x, y\}$ with $x \Rightarrow y$. Then in the free preordered Σ_{Add}-algebra, we have $Sx \Rightarrow Sy$, $x + y \Rightarrow y + y$, $S(x + y) \Rightarrow S(y + y)$, etc. but not e.g. $S(x + 0) \Rightarrow Sx$. Another, non-free, preordered Σ_{Add}-algebra are the natural numbers, ordered by $\leq$.

There are now two questions we need to address. Firstly, we have given a categorical account of Σ-algebras but what are preordered Σ-algebras? We would like them to be algebras of a monad on **Pre** but T_Σ is a monad on **Set** — thus we define a functor $L : \mathbf{Mon}(\mathbf{Set}) \longrightarrow \mathbf{Mon}(\mathbf{Pre})$. The second question is how to account for the reduction rules. We shall use L to convert a TRS such as R_{Add} from Example 4 into a pair of monad morphisms in $\mathbf{Mon}(\mathbf{Pre})$. However, if we take their coequalizer, it will force the left-hand side of the rule to be *equal* to the right-hand side, which is not what we want. What we want is to force the left-hand side to reduce to the right-hand side. This is exactly what a *coinserter* [Kelly 1989]) does. For rewriting, one only needs an preorder-enriched version of the coinserter, which considerably simplifies the presentation. A category is preorder enriched iff the hom-sets are equipped with a preorder and composition is monotone. Note that **Pre** is preorder-enriched with the order on homs given by $f \Rightarrow g : (X, \Rightarrow) \longrightarrow (Y, \Rightarrow)$ iff $\forall x \in X.\, fx \Rightarrow gx$.

Definition 10. (Coinserter) *Let $\mathcal{C}$ be a preorder-enriched category with a pair of arrows $f, g : A \longrightarrow B$. Their coinserter is given by an object $\mathsf{coin}(f,g)$ and a morphism $k : B \longrightarrow \mathsf{coin}(f,g)$, such that $kf \Rightarrow kg$*

$$\begin{tikzcd}
& B \arrow[dr, "k"] & \\
A \arrow[ur, "f"] \arrow[dr, "g"'] \arrow[rr, phantom, "\Downarrow"] & & \mathsf{coin}(f,g) \\
& B \arrow[ur, "k"'] &
\end{tikzcd}$$

Furthermore, for any other object P and morphism $p : B \longrightarrow P$ such that $pf \Rightarrow pg$, there exists a unique morphism $!_p : \mathsf{coin}(f,g) \longrightarrow P$ such that $!_p k = p$; and this assignment is monotone, i.e. if there is another $p' : B \longrightarrow P$ such that $p' \Rightarrow p$ and $p'f \Rightarrow p'g$, then $!_{p'} \Rightarrow !_p$. □

As we shall see, the coinserter is the smallest reduction relation generated by the rules, just as the coequaliser is the smallest equivalence relation generated by the equations. In fact, if we consider the category **Set** to be trivially preorder enriched with the discrete order on the hom-sets, the coinserter of two monads on **Set** is the coequaliser since if $kf \Rightarrow kg$ then $kf = kg$. The use of preorder enrichment is crucial in allowing non-discrete orders on the hom-sets and hence permitting the coinserter to differ from the coequaliser. The rest of this section implements the general plan outlined above. We begin by relating the categories **Sig**(**Set**) and **Sig**(**Pre**), and **Mon**(**Set**) and **Mon**(**Pre**), respectively.

3.2 From Sets to Preorders and Back

Lemma 3 allow us to move between sets and preorders. This lifts to an ability to translate between signatures on **Set** and **Pre**. First a trivial observation:

Lemma 4. *Let $\mathcal{N}$ be a representing set of finitely presentable objects in* **Set** *(i.e. finite sets), and $\mathcal{N}_P$ a representing set of finitely presentable objects in* **Pre** *(i.e. of those preorders with a finite carrier set). $K : \mathcal{N} \longrightarrow \mathcal{N}_P$ is given by restricting D to $\mathcal{N}$, and J, J_P are inclusions. Then the following is* *a commuting square in* **Cat***:*

$$\begin{tikzcd}
\mathcal{N} \arrow[r, "J"] \arrow[d, "K"'] & \mathbf{Set} \arrow[d, "D"] \\
\mathcal{N}_P \arrow[r, "J_P"'] & \mathbf{Pre}
\end{tikzcd}$$

We can use these functors to form an adjunction between signatures in **Set** and signatures in **Pre** as follows.

LEMMA 5. *The following are true:*

- *The mapping sending a signature* Σ *over* **Pre** *to the signature* $V\Sigma K$ *over* **Set** *extends to a functor* $V\_K : \mathbf{Sig}(\mathbf{Pre}) \longrightarrow \mathbf{Sig}(\mathbf{Set})$.
- *If* Ω *is a signature over* **Set**, *then* $\mathsf{Lan}_K D\Omega$ *is a signature over* **Pre**. *Indeed, this defines a functor* $\mathsf{Lan}_K D\_ : \mathbf{Sig}(\mathbf{Set}) \longrightarrow \mathbf{Sig}(\mathbf{Pre})$.
- *There is an adjunction* $\mathsf{Lan}_K D\_ \dashv V\_K : \mathbf{Sig}(\mathbf{Set}) \longrightarrow \mathbf{Sig}(\mathbf{Pre})$.

PROOF. That V_K and $\mathsf{Lan}_K D\_$ are functors with the given domain and codomain is obvious. For the adjunction, to give a natural transformation $\mathsf{Lan}_K D\Omega \longrightarrow \Sigma$ is by definition of the left Kan extension precisely the same thing as a natural transformation $D\Omega \longrightarrow \Sigma K$. Since $D \dashv V$, this is exactly a natural transformation $\Omega \longrightarrow V\Sigma K$ as required. □

Now that we can move from signatures on **Set** to signatures on **Pre**, we extend this to monads. The crucial lemma is the following:

LEMMA 6. *Let* $F \dashv G$ *be an adjunction with* $F : \mathcal{C} \longrightarrow \mathcal{D}$. *If* T *is a monad on* $\mathcal{D}$, *then* GTF *is a monad on* $\mathcal{C}$. *Similarly if* $\sigma : T \longrightarrow S$ *is a monad morphism, then so is* $G\sigma F : GTF \longrightarrow GSF$. *Thus we have a functor* $G\_F : \mathbf{Mon}(\mathcal{D}) \longrightarrow \mathbf{Mon}(\mathcal{C})$.

PROOF. Given a monad T, we compose the adjunction $F \dashv G$ with the adjunction $F^T \dashv U^T$, yielding an adjunction $F^T F \dashv GU^T$ which gives the desired monad on $\mathcal{C}$.

A monad morphism $\sigma : T \longrightarrow S$ gives a natural transformation $\sigma' : GTF \Rightarrow GSF$. That this is a monad morphism, and the functoriality of the whole construction, is a simple diagram chase. □

We can now apply this lemma to the adjunction at hand:

LEMMA 7. *Consider the adjunction* $D \dashv V$ *between* **Set** *and* **Pre**. *The assignment sending a monad* T *on* **Pre** *to a monad* VTD *on* **Set** *extends to a functor* $V\_D : \mathbf{Mon}(\mathbf{Pre}) \longrightarrow \mathbf{Mon}(\mathbf{Set})$.

PROOF. Apply Lemma 6 to the adjunction $D \dashv V$ from Lemma 3. □

Recall our desire to define a functor $L : \mathbf{Mon}(\mathbf{Set}) \longrightarrow \mathbf{Mon}(\mathbf{Pre})$ so as to characterise preordered Σ-algebras as Eilenberg-Moore algebras and to turn parallel pairs in $\mathbf{Mon}(\mathbf{Set})$ into to parallel pairs in $\mathbf{Mon}(\mathbf{Pre})$ whose coinserter we can take. This can be achieved by Butler's Theorem which lifts adjunctions between categories to adjunctions between categories monadic over the original categories. It is formally stated as follows:

THEOREM 1. (BUTLER) *Consider the following (not necessarily commuting) diagram of categories, functors and adjunctions:*

$$\begin{tikzcd}
\mathcal{B} \arrow[r, "W"] \arrow[d, shift left, "U"] & \mathcal{B}' \arrow[d, shift left, "U'"] \\
\mathcal{C} \arrow[u, shift left, "T"] \arrow[r, shift left, "V"] \arrow[u, phantom, "\dashv"] & \mathcal{C}' \arrow[u, shift left, "T'"] \arrow[l, shift left, "G"] \arrow[l, phantom, "\top"] \arrow[u, phantom, "\dashv"]
\end{tikzcd}$$

Assume further the following:

- $VU \cong U'W$,
- U' *is of descent type,*
- $\mathcal{B}$ *has coequalizers.*

Then W has a left adjoint L.

PROOF. L will map objects of the form $T'X$ to TGX while the action of L on maps $h : T'X \longrightarrow T'Y$ follows from Yoneda. L has now been defined on the full subcategory of free objects of $\mathcal{B}'$. Since U' is of descent type and $\mathcal{B}$ has coequalizers and, as a left adjoint, L must preserve coequalizers, this suffices to define L on the all of $\mathcal{B}'$. See [Barr and Wells 1985] for the full proof. □

We can now instantiate Butler's theorem, with the adjunction (2.5) on the left and the right, and the adjunction from Lemma 7 on the bottom, as follows:

$$\begin{tikzcd}
\mathbf{Mon}(\mathbf{Pre}) \arrow[r, shift left, "V_D"] \arrow[r, phantom, "\top"] \arrow[d, shift left, "U"] & \mathbf{Mon}(\mathbf{Set}) \arrow[l, shift left, "L"] \arrow[d, shift left, "U"] \\
\mathbf{Sig}(\mathbf{Pre}) \arrow[u, shift left, "T"] \arrow[u, phantom, "\dashv"] \arrow[r, shift left, "V_K"] \arrow[r, phantom, "\top"] & \mathbf{Sig}(\mathbf{Set}) \arrow[l, shift left, "\mathsf{Lan}_K D_"] \arrow[u, shift left, "T"] \arrow[u, phantom, "\dashv"]
\end{tikzcd}$$

The conditions of Butler's theorem are satisfied, because

- for a monad T on **Pre**, we have $U(VTD) = VTDJ$ and $V(UT)K = VTJ_PK = VTDJ$ since, by Lemma 4, $DJ \cong J_PK$;
- U is of descent type (as remarked at the end of Section 2);
- **Mon**(**Pre**) has coequalisers (in fact, all colimits; see below).

We call the adjoint $L \dashv V\_D : \mathbf{Mon}(\mathbf{Set}) \longrightarrow \mathbf{Mon}(\mathbf{Pre})$. From the proof of Butler's theorem, the action of L on free monads is described by the equation $L(T_\Sigma) = T_{\mathsf{Lan}_K D\Sigma}$ while monads which are coequalizers are mapped to coequalizers. We can characterise the action of LT_Σ in more elementary terms. Intuitively, LT_Σ should be the smallest extension of Σ to preorders, or in other words the free preordered Σ-algebra:

LEMMA 8. *Let Σ be a signature over* **Set**. *LT_Σ maps a preorder X to the free preordered Σ-algebra over X.*

PROOF. Recall that LT_Σ is the free monad over the **Pre**-signature $\mathsf{Lan}_K D\Sigma$. By the definition of the left Kan extension, we get the signature

$$\mathsf{Lan}_K D\Sigma(c) = \begin{cases} D(\Sigma(n)) & \text{if } c = K(n), \text{ i.e. if } c \text{ is discrete and finite} \\ \emptyset & \text{otherwise} \end{cases}$$

Call this signature $\Sigma' : \mathcal{N}_P \longrightarrow \mathbf{Pre}$. Since Σ' contains exactly the same operations as Σ, we can further calculate

$$\begin{aligned} F_{\Sigma'}(X) &= \mathsf{Lan}_J \mathsf{Lan}_K D\Sigma(X) \\ &= \mathsf{Lan}_{JK} D\Sigma(X) \\ &= \coprod_{n \in \mathbb{N}, f \in D\Sigma(n)} X^{JK(n)} \end{aligned}$$

Hence $VF_{\Sigma'} = F_\Sigma V$, i.e. the carriers are the same but, of course, $F_{\Sigma'}(X)$ has an order on it which includes X (thus ensuring the unit will be monotone) and ensures the operations are also monotone. The free monad construction closes this under composition, preserving the order structure under substitution. □

Thus we have that the category of preordered Σ-algebras is the category $F_{\Sigma'}-\mathbf{alg}$, which as we have remarked earlier, is also $LT_\Sigma-\mathbf{Alg}$. In our example, if $X = \{x, y\}$ with $x \Rightarrow y$, then we have $F_{\Sigma'_{Add}}(X) = F_{\Sigma_{Add}}(\{x, y\})$ as in (2.2) with the order structure $Sx \Rightarrow Sy$, $x + x \Rightarrow y + x$, $x + x \Rightarrow x + y$, $x + y \Rightarrow y + y$, $y + x \Rightarrow y + y$, $x + x \Rightarrow y + y$, etc. Having defined L and used it to characterise preordered Σ-algebras, we can now use it to define the representing monad for a rewriting system.

3.3 Presentations and Representing Monads

DEFINITION 11. (PRESENTATIONS, REPRESENTING MONADS)

- *An* equational presentation *is given by two finitary monads T, E on* **Set** *and a pair of arrows $l, r : E \longrightarrow T$ in* $\mathbf{Mon}(\mathbf{Set})$.

  The coequaliser $\mathit{coeq}(s,t)$ *is the representing monad for the corresponding algebraic theory.*

- *A* rewrite presentation *is given by two finitary monads T, E on* **Pre** *and a pair of arrows $l, r : E \longrightarrow T$ in* $\mathbf{Mon}(\mathbf{Pre})$.

  The coinserter $\mathit{coin}(l,r)$ *is the representing monad for the corresponding rewrite system.*

□

Definition 11 doesn't ask that the monads be free. They are in the case of algebraic theories and TRSs but other applications, e.g. equational rewriting and higher order rewriting, involve non-free monads.

The coequaliser and coinserter of two monads always exists because the category of finitary monads on $\mathcal{A}$ is finitarily monadic over the category of finitary endofunctors $[\mathcal{A},\mathcal{A}]_f$ (as shown by [Kelly and Power 1993]). In other words, every finitary monad is the algebra of a finitary monad W on $[\mathcal{A},\mathcal{A}]_f$. With $[\mathcal{A},\mathcal{A}]_f$ cocomplete, the category of W-algebras has all coequalizers ([Barr and Wells 1985], Theorem 9.3.8), and hence all colimits ([Linton 1969], Corollary 2). Unfortunately, this rather high-level proof does not give us a good way to construct the coequalisers and coinserters. However, the key idea that we can understand monads through their algebras gives a more concrete characterisation of these monads and this underlies the following result [Kelly 1980].

For a presentation $l, r : E \longrightarrow T$, a (E,T)-*coequalizer algebra* is a T-algebra (A, h) such that $h{\cdot}s = h{\cdot}t$, and similarly a (E,T)-*coinserter algebra* is a T-algebra (A, h) such that $h{\cdot}s \Rightarrow h{\cdot}t$. Morphisms between these are morphisms in $T\!-\!\mathbf{Alg}$, defining the categories $(E,T)_{\mathit{Coeq}}\!-\!\mathbf{Alg}$ and $(E,T)_{\mathrm{Coin}}\!-\!\mathbf{Alg}$ of coequalizer-algebras and coinserter-algebras.

LEMMA 9. *The following are true:*

- *The forgetful functor $V_1 : (E,T)_{\mathit{Coeq}}\!-\!\mathbf{Alg} \longrightarrow \mathbf{Set}$ (mapping each algebra to its underlying carrier object) has a left adjoint $G_1 \dashv V_1$, and the monad given by this adjunction is the coequalizer of $l, r : E \longrightarrow T$.*
- *The forgetful functor $V_2 : (E,T)_{\mathit{Coin}}\!-\!\mathbf{Alg} \longrightarrow \mathbf{Pre}$ (mapping each algebra to its underlying carrier object) has a left adjoint $G_2 \dashv V_2$, and the monad given by this adjunction is the coinserter of $l, r : E \longrightarrow T$.*

PROOF. Proposition 26.4 of [Kelly 1980] states, in a more general form for any diagram in the category $\mathbf{Mon}(\mathcal{A})$, that the left adjoints above exist and the resulting monads are the respective colimits if and only if the colimits exist, but we have shown that it does for these particular colimits. □

In other words, the monad Q is the coinserter of S and T if, for all X, QX is the free coinserter algebra over X: there is a structure map $h_X : TQX \longrightarrow QX$ and map $X \longrightarrow QX$ such that for any other coinserter algebra (A, k) with a map $f : X \longrightarrow A$, there is a unique algebra morphism $!_f : QX \longrightarrow A$.

We finish off this section by showing how these coinserter algebras can be used to show that the rewrite relation of our running example is indeed calculated by the coinserter of its rewrite representation.

Example 5. (Addition) We are mainly interested in the rewrite presentation and the representing monad. The rewrite presentation of Add can be given as

$$LT_{R_{Add}} \underset{Lr'}{\overset{Ll'}{\rightrightarrows}} LT_{\Sigma_{Add}}$$

We claim that the coinserter of this monad maps every preorder X to many-step reduction relation $\twoheadrightarrow_{\mathcal{R}_{Add}}$ generated by the term rewriting system from Example 4.

This is shown below in general, but here assume for simplicity that X is discrete. Using Lemma 9, we need to show that $\twoheadrightarrow_{\mathcal{R}_{Add}}$ is the free coinserter algebra. $LT_{\Sigma_{Add}}(X)$ and $\twoheadrightarrow_{\mathcal{R}_{Add}}$ have the same elements, namely $T_{\Sigma_{Add}}(X)$. $\twoheadrightarrow_{\mathcal{R}_{Add}}$ also is closed under congruence, making it a $LT_{\Sigma_{Add}}$-algebra. For every rule $e \in R_{Add}$, we have $l(t) \twoheadrightarrow_{\mathcal{R}_{Add}} r(t)$, making $\twoheadrightarrow_{\mathcal{R}_{Add}}$ into a coinserter algebra. That it is also the free one follows because it is the least such. □

4. Examples of Rewriting as Coinserters

Although we hope we have convinced the reader of the naturalness of regarding rewriting systems as coinserters, until now this has only been intuition. Therefore in this section we consider a number of forms of rewriting and show that their representing monad arises as coinserters deriving from a rewrite presentation $l, r : E \longrightarrow T$. This then validates the terminology of Definition 11. In each case the general pattern is the same:

- We define the representing monad R for the rewrite system as an axiomatisation of the rewrite relation.
- We choose T to represent the data being rewritten. The monad structure of T dictates how subterms may be replaced with other subterms.
- We choose E to construct the proof terms for the rewrites. This is similar to what happens in rewrite logic and 2-categorical models of rewriting.
- We choose l and r to assign to each proof term/label for a rewrite constructed by E the correct left- and right-hand side
- Finally we show that the representing monad satisfies the universal property of the coinserter of the presentation, that is $\mathcal{R} = \mathsf{coin}(l, r)$.

We give the proof of $R = \mathsf{coin}(l, r)$ for our first example, but since the proofs for the other examples are analogous, we concentrate only on the data for the latter examples.

4.1 Term Rewriting

We have seen how a TRS $\langle \Sigma, R \rangle$ as in Definition 8 is just an algebraic theory and hence an equational presentation $l, r : T_R \longrightarrow T_\Sigma$. Applying L gives a rewrite presentation $Ll, Lr : LT_R \longrightarrow LT_\Sigma$ (see Example 4).

Let $S = (S, \eta, \mu)$ be the monad which calculates the rewrite relation. Formally, S maps a preorder X to the smallest preordered Σ-algebra over X which also includes all instances of all rules in R, i.e. for all $(Y \vdash s \to t) \in R$ and all maps $\sigma : Y \longrightarrow SX$, $\overline{\sigma}(s) \Rightarrow \overline{\sigma}(t)$ in SX. Here $\overline{\sigma}$ is the substitution induced by σ, ie $\overline{\sigma} = \mu {\cdot} S\sigma$. The monad S has the same unit and multiplication as the monad T_Σ from Lemma 1, so to show S is a monad, we only need to verify that they are monotone (see [Lüth and Ghani 1997, Lüth 1998]).

Since SX is built over a non-discrete preorder X, it contains not only the many-step reductions $\twoheadrightarrow_{\mathcal{R}}$, but also the reductions from X, closed under congruence. This is the free preordered Σ-algebra on X, which by Lemma 8 is given by $LT_\Sigma(X)$. But is S the coinserter $\mathsf{coin}(Ll, Lr)$ derived from the presentation? Using Lemma 9, we need to show that SX is the free coinserter algebra. Since the elements of SX are the terms $T_\Sigma(X)$, there is an inclusion $k : LT_\Sigma(X) \longrightarrow SX$, making it a T_Σ-algebra. Since S also includes all rewrites generated from R, $k{\cdot}l \Rightarrow k{\cdot}r$. Thus SX is a coinserter algebra.

It remains to show freeness: suppose there is another coinserter algebra (A, h) such that $h : LT_\Sigma A \longrightarrow A$ with $h{\cdot}l \Rightarrow h{\cdot}r$ and $f : X \longrightarrow A$. We now have to show there is a unique algebra morphism $!_f : SX \longrightarrow A$. In fact, since the elements of SX are the terms $T_\Sigma(X)$, we have $!_f$ as the homomorphic extension of $f : X \longrightarrow A$, and it only remains to show that $!_f$ is monotone. This follows from the fact that SX computes the *least* Σ-algebra validating the rewrite rules, i.e. all rewrites in SX are sequences of steps in $\twoheadrightarrow_{\mathcal{R}}$ (which must be in A, since $h{\cdot}l \Rightarrow h{\cdot}r$) or rewrites from X which are preserved by $f : X \longrightarrow A$.

In summary, the crux of the proof is getting the monad of rewrites LT_R correct. That there are not too many elements of LT_R ensures that rewrite relation as embodied by SX is a coinserter algebra. That there are enough elements of LT_R ensures SX is the free coinserter algebra.

4.2 Equational Rewriting

Inherent in rewriting is the idea of solving equations by orienting them so as to form a rewrite relation. However, some equations are difficult to orient, such as commutativity axioms. For systems of equations involving those equations it is sometimes more convenient to quotient out these equations by considering the rewrite relation over the term algebra quotiented by the unorientable equations. Other uses include rewriting of data given by operations and equations, e.g. lists (a.k.a free monoids; see below).

An equational rewrite system is a triple $\langle\Sigma, E, R\rangle$ where $\langle\Sigma, E\rangle$ is an algebraic theory and $\langle\Sigma, R\rangle$ is a TRS. The equational rewrite system lifts the TRS $\langle\Sigma, R\rangle$ to a rewrite relation on the free algebra given by $\langle\Sigma, E\rangle$.

Formally, given an algebraic theory $\langle\Sigma, E\rangle$ a *preordered* $\langle\Sigma, E\rangle$-algebra is a preordered Σ-algebra A validating the equations in E, i.e. for all $(Y \vdash s = t) \in E$ and all assignments $\sigma : Y \longrightarrow A$, $\overline{\sigma}(s) = \overline{\sigma}(t)$. The rewrite relation defines a monad S which maps X to the least preordered $\langle\Sigma, E\rangle$-algebra validating the rules in R, i.e. for all $(Y \vdash s \to t) \in R$ and all maps $\sigma : Y \longrightarrow SX$, $\overline{\sigma}(s) \Rightarrow \overline{\sigma}(t)$ in SX. To see that S is a monad, note that the elements of SX are the elements of $T_{\langle\Sigma,E\rangle}$, so the unit and multiplication are as in Section 2.4, and we only need to verify that they are monotone.

Is there a rewriting presentation such that $SX = \mathsf{coin}(l, r)$? Clearly, the terms that are being rewritten are elements of the algebraic theory $T_{\langle\Sigma,E\rangle}$ which recall is a coequaliser whose quotient map we write $q : T_\Sigma \longrightarrow T_{\langle\Sigma,E\rangle}$. Thus, the codomain of the rewrite presentation should be $LT_{\langle\Sigma,E\rangle}$. Since the proof terms for the rewrites are the same as for the term rewriting system $\langle\Sigma, R\rangle$, we can take the domain of the presentation to be LT_R. From the term rewriting system $\langle\Sigma, R\rangle$ we also obtain maps $l', r' : T_R \longrightarrow T_\Sigma$, which we compose with the map $q : T_\Sigma \longrightarrow T_{\langle\Sigma,E\rangle}$ to define $l = q \cdot l'$ and $r = q \cdot r'$. Thus the desired rewriting presentation is $Ll, Lr : LT_R \longrightarrow LT_{\langle\Sigma,E\rangle}$.

The proof that $S = \mathsf{coin}(Ll, Lr)$ is essentially as above, we only need to additionally handle the equivalence induced by E.

4.3 Gröbner Bases and String Rewriting

String rewriting was developed in the early part of the 20th century to solve combinatorial problems in group theory. Within string rewriting one is not interested in rewriting terms, but rather in rewriting strings, i.e. elements of the free monoid monad over a set of generators. Since this is an algebraic theory, string rewriting is an instance of rewriting modulo an equational theory and hence is covered by the previous example where the algebraic theory is the free monoid monad.

Similarly, *Gröbner bases* were introduced in the 1950's to study the polynomial ideal membership problem. These days, Gröbner bases are often formulated in terms of rewrite systems over polynomial rings. Thus, Gröbner bases are another example of rewriting modulo an equational theory and are covered by the previous example where the algebraic theory is taken to be the free ring monad.

4.4 Infinite Rewriting

So far, all of the examples we have discussed, although canonical, are nevertheless instances of rewriting modulo an equational theory. This reflects the result that the forgetful functor $U : \mathbf{Mon}(\mathbf{Set}) \longrightarrow \mathbf{Sig}(\mathbf{Set})$ is of descent type. That is, all finitary monads, i.e. data structures where rewriting can take place, can be presented by equational theories.

We next consider infinitary rewriting which seeks to rewrite over potentially infinite terms and which is not an example of rewriting modulo equational theories. There have been several different attempts to define a useful notion of infinite term rewriting over the last decade. The key problem seems to be to identify which infinite sequences of rewrites should be accepted. On the one hand one wants to allow as many infinite rewrites as possible, while on the other one needs to exclude certain rewrite sequences to ensure a good meta-theory. The common forms are:

- **Rational Rewriting:** Rational equations are equations of the form $A = B(A, x)$ (for a binary operation B, a constant A and a variable x), which can be given as maps $E \to T_\Sigma(X + E)$ for some signature Σ and sets X of variables and E of unknowns. Rational terms are infinite terms which arise as solution of rational equations. Rational rewriting allows us to define infinite rewrites as the solution to rational equations over rewrites. For example, if $r : A \Rightarrow C$ is a finite rewrite, the equation $r' = B(r, r')$ would have as solution the infinite rewrite r' which performs an r rewrite on the left hand branch under each node labelled C.

  Can we give a presentation $l, r : E \longrightarrow S$ whose associated coinserter models the rational rewrites generated by a TRS $\langle \Sigma, R \rangle$? Recent work by [Ghani *et al.* 2002] and [Adamek *et al.* 2003] has shown that the rational terms over a signature Σ define a monad K_Σ and hence we may set $S = K_\Sigma$. Next we must choose the proof term monad E which generates the rewrites. As the above example suggests, the correct set of proof terms for the rewrites are again the rational terms over R, i.e. one should set $E = K_R$. The maps l, r are then easy to define.

- **Circular Rewriting:** Circular rewriting represents infinite terms as term graphs and then defines infinite rewrites as term graphs built over the signature and the rewrite rules. By taking the above coinserter and replacing the rational monads with term graph monads [Ghani *et al.* 2002] over a signature, one gets the semantics of circular rewriting.

- **Strongly Convergent Rewriting:** SC rewriting offers the widest choice of infinite rewrites by defining the infinite rewrite relation to contain all infinite sequences of rewrites where the depth of the redex contracted tends to infinity. Thus, given a TRS $\langle \Sigma, R \rangle$, we will be rewriting over the set of all finite and infinite Σ-terms. This set is known to be the final coalgebra of the polynomial endofunctor F_Σ and also a monad T_Σ^∞ [Aczel *et al.* 2003].

  However, unlike the previous examples, the proof terms are not simply T_R^∞ as this would allow arbitrary infinite rewrites. As proof terms for rewrites, we want to allow infinite terms built over the signature and the rewrite system such that all paths through the term do a finite amount of rewriting, then pass through a term constructor and then repeat. Having both finitary and infinitary properties, this structure is a final coalgebra built over an initial algebra, which maps a set X to $\nu Y.\mu Z.X + F_\Sigma Y + F_R Z$. That this actually defines a monad follows from [Matthes and Uustalu 2003].

Again the maps l, r and the proof that this coinserter actually computes the rewrite relation is as above.

4.5 Higher Order Rewriting

Higher Order Rewriting is a framework for rewriting over data structures with variable binding such as the λ-calculus. With the recent initial algebra semantics for such structures proposed by [Fiore *et al.* 1999], and the proof that they form monads by [Matthes and Uustalu 2003], we have all the ingredients to model higher order rewriting as a coinserter.

As an example we consider rewriting over the untyped λ-calculus. The papers cited above prove that there is a monad $M : \mathbf{Set} \longrightarrow \mathbf{Set}$ whose action maps a set X to the set of untyped λ-terms (upto α-equivalence) whose free variables belong to X. This is not a free monad but rather an initial algebra of a higher-order functor. Now let us assume we have a rewrite relation R given by pairs of untyped λ-terms analogous to the definition of a TRS. As before, this data gives a parallel pair $l, r : T_R \longrightarrow M$ and so the rewrite relation is computed by the coinserter of $Ll, Lr : LT_R \longrightarrow LM$. Note that the domain of the presentation is simply a free monad - there is nothing sophisticated such as bound rewrite variables. The fact that the rewrite relation will be closed under λ-abstractions follows from the fact that the coinserter will be an LM-algebra and hence closed under the operations of M, such as λ-abstraction.

5. Conclusions

In conclusion, we have proposed a semantics for rewriting which is independent of the data being rewritten but, via the use of monads, still models key concepts such as substitution or the layer structure in modularity problems. This builds on previous research using monads but adds the extra ingredient of a *coinserter*. Mirroring the relationship between algebraic theories and rewrite systems by the relationship between coequalizers and coinserters is particularly elegant and natural. As the examples show, defining rewrite relations by coinserters is fairly easy when one knows the terms being rewritten and the proof terms for the rewrites. The proof that the actual rewrite relation (given as the least preordered algebra over the data being rewritten which instantiates the rewrites) is the coinserter can then be checked using arguments similar to that for TRSs.

As it stands, this paper sets out a unifying framework for rewriting theory, and as such, will be of most interest to the general theoretical computer science community who are interested in learning more about the connections between universal algebra, rewriting and category theory. This is a broad community. Nevertheless, it is clear that there are many possible applications which will interest those more interested in the actual development in rewriting. Indeed, with the current unsettled nature of infinite rewriting, our work suggests that infinite rewriting may be done by choosing any set of infinite terms closed under substitution for rewriting and using some, potentially different, set of infinite terms as proof terms for the actual rewrites.

In a different direction, we want to extend the monadic proof of the modularity of confluence for TRSs to a general modularity proof using the framework developed here. Of course, confluence is not modular for all forms of rewriting so we expect certain conditions to be required in terms of the monads we use. This is still the subject of on-going research. Another result which can be tackled at this level of generality is that of generalised Knuth-Bendix completion and, here, no side conditions should be required.

References

ACZEL, P., ADMEK, J., MILIUS, S., AND VELEBIL, J. 2003. Infinite Trees and Completely Iterative Theories: A Coalgebraic View. *Theoretical Computer Science 300*, 1–45.

ADAMEK, J., MILIUS, S., AND VELEBIL, J. 2003. Free Iterative Theories: A coalgebraic view. *Mathematical Structures in Computer Science 13*, 259–320.

ADAMEK, JIŘI AND ROSICKÝ, JIŘI. 1994. *Locally Presentable and Accessible Categories*. Number 189 in London Mathematical Society Lecture Note Series. Cambridge University Press.

BAADER, FRANZ AND NIPKOW, TOBIAS. 1998. *Term Rewriting and All That*. Cambridge University Press.

BARR, M. AND WELLS, C. 1985. *Toposes, Triples and Theories*. Number 278 in Grundlehren der mathematischen Wissenschaften. Springer Verlag.

BUCHBERGER, B. AND WINKLER, F. 1998. Gröbner Bases and Applications: 33 Years of Gröbner Bases. In *Proc. London Math. Soc.*. Volume 251. Cambridge University Press.

CHURCH, ALONZO. 1940. A Formulation of The Simple Thoery of Types. *J. Symbolic Logic 5*, 56–68.

EPSTEIN, D. 1992. *Word processing in groups*. Jones and Bartlett.

FIORE, MARCELO, PLOTKIN, GORDON, AND TURI, DANIELE. 1999. Abstract Syntax and Variable Binding. In *Proc. 14th Annual Symposium on Logic in Computer Science (LICS'99)*. IEEE Computer Society Press, 193–202.

GHANI, NEIL. 1995. $\beta\eta$-Equality for Coproducts. In *Second Conference on Typed Lambda Calculus and its Applications*. Edinburgh.

GHANI, NEIL, LÜTH, CHRISTOPH, AND MARCHI, FEDERICO DE. 2002. Coalgebraic Monads. In *Coalgebraic Methods in Computer Science CMCS'02*, Volume 65(1) of *ENTCS*. Elsevier.

JAY, C. B. 1990. Modelling Reductions in Confluent Categories. In *Proceedings of the Durham Symposium on Applications of Categories in Computer Science*.

KELLY, G. M. 1980. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves and so on. *Bulletins of the Australian Mathematical Society 22*, 1– 83.

KELLY, G. M. 1989. Elementary Observations on 2-Categorical Limits. *Bulletins of the Australian Mathematical Society 39*, 301–317.

KELLY, G. M. AND POWER, A. J. 1993. Adjunctions whose counits are coequalizers, and presentations of finitary monads. *Journal for Pure and Applied Algebra 89*, 163–179.

Klop, J. W. 1992. Term Rewriting Systems. In *Handbook of Logic in Computer Science*, Abramsky, S., Gabbay, Dov M., and Maibaum, T. S. E., Editors. Volume 2 (Background: Computational Structures). Oxford University Press, 1–116.

Knuth, D. and Bendix, P. 1970. Simple Word Problems in Universal Algebra. In *Computational Problems in Universal Algebras*, Leech, J., Editor. Pergamon Press, 263–297.

Linton, F. E. J. 1969. Coequalizers in Categories of Algebras. In *Seminar on Triples and Categorical Homology Theory*, Number 80 in Lecture Notes in Mathematics. Springer Verlag, 75– 90.

Lüth, Christoph. 1998. *Categorical Term Rewriting: Monads and Modularity*. PhD thesis, University of Edinburgh.

Lüth, Christoph and Ghani, Neil. 1997. Monads and Modular Term Rewriting. In *Category Theory in Computer Science CTCS'97*, Number 1290 in Lecture Notes in Computer Science. Springer Verlag, Santa Margherita, Italy, 69– 86.

MacLane, S. 1971. *Categories for the Working Mathematician*. Volume 5 of *Graduate Texts in Mathematics*. Springer Verlag.

Manes, Ernest G. 1976. *Algebraic Theories*. Volume 26 of *Graduate Texts in Mathematics*. Springer Verlag.

Matthes, R. and Uustalu, T. 2003. Substitution in Non-Wellfounded Syntax with Variable Binding. In *Coalgebraic Methods in Computer Science CMCS'03*, Gumm, H. P., Editor. Volume 82(4) of *ENTCS*. Elsevier.

Melliès, Paul-André. 1998. A Stability Theorem in Rewriting Theory. In *14th Annual Symposium on Logic in Computer Science*. IEEE, Indianapolis, USA.

Moggi, E. 1989. Computational lambda-calculus and monads. In *Fourth Annual Symposium on Logic in Computer Science*. IEEE.

Moss, Lawrence S. 2001. Parametric corecursion. *Theoretical Computer Science 260*, 1–2, 139–163.

Reichel, H. 1991. A 2-Category Approach to Critical Pair Completion. In *Recent Trends in Data Type Specification*, Number 534 in Lecture Notes in Computer Science. Springer Verlag, 266–273.

Rydeheard, D. E. and Stell, J. G. 1987. Foundations of Equational Deduction: A Categorical Treatment of Equational Proofs and Unification Algorithms. In *Category Theory and Computer Science*, Number 283 in Lecture Notes in Computer Science. Springer Verlag, 114– 139.

Seely, R. A. G. 1987. Modelling Computations: A 2-Categorical Framework. In *Proceedings of the Second Annual Symposium on Logic in Computer Science*, 65–71.

Stell, John G. 1994. Modelling Term Rewriting Systems by Sesqui-Categories. Tech. Report TR94-02, Keele Unversity.

Stokkermans, K. 1992. A Categorical Formulation for Critical-Pair/Completion Procedures. In *Third International Workshop on Conditional Term Rewriting Systems*, Number 656 in Lecture Notes in Computer Science. Pout-à-Mousson, 171–175.

Appendix A. Presentability and Kan Extensions

This appendix gives a short definition and introduction into some constructions which go beyond basic category theory, namely finitariness, local presentability, Kan extensions. We give brief definitions here; more details can be found in MacLane 1971 or [Adamek and Rosický 1994].

Finitary Monads: A diagram $\mathcal{D}$ is *filtered* iff every subcategory with finite number of objects and morphisms has a compatible cocone in $\mathcal{D}$; the colimit of such a diagram is called filtered. A functor is *finitary* iff it preserves filtered colimits. A monad is finitary iff its action is finitary.

Given two monads $T = \langle T, \eta, \mu \rangle$ and $S = \langle S, \zeta, \xi \rangle$ on $\mathcal{C}$ a *monad morphism* is a natural transformation $\alpha : T \Rightarrow S$ between the actions commuting with unit and multiplication:

$$
\begin{tikzcd}
 & T \arrow[dd, "\alpha"] \\
1 \arrow[ur, "\eta"] \arrow[dr, "\zeta"'] & \\
 & S
\end{tikzcd}
\qquad
\begin{tikzcd}
 & TT \arrow[rr, "\mu"] \arrow[dl, "\alpha_T"'] \arrow[dr, "T\alpha"] & & T \arrow[dd, "\alpha"] \\
ST \arrow[dr, "S\alpha"'] & (*) & TS \arrow[dl, "\alpha_S"] & \\
 & SS \arrow[rr, "\xi"'] & & S
\end{tikzcd}
$$

where on the right, the diamond (*) commutes in any case because of naturality of α. The finitary monads on a category $\mathcal{C}$ and monad morphisms between them form a category $\mathbf{Mon}(\mathcal{C})$.

Locally Presentable and Accessible Categories: An object X of a category $\mathcal{A}$ is said to be *finitely presentable* iff the hom-functor $\mathcal{A}(X, \_)$ preserves filtered colimits. A category is *locally finitely presentable* (abbreviated as lfp) if it is cocomplete and has, up to isomorphism, a set N of finitely presentable objects such that every object is a filtered colimit of objects from N. The discrete category on N is denoted $\mathcal{N}$. The full subcategory of finitely presentable objects is denoted $\mathcal{A}_{fp}$. The inclusion functors are denoted $J : \mathcal{N} \to \mathcal{A}_{fp}$ and $I : \mathcal{A}_{fp} \to \mathcal{A}$, and the category of finitary functors from $\mathcal{A}$ to $\mathcal{B}$ by $[\mathcal{A}, \mathcal{B}]_f$.

Finite presentability is the categorical notion for finiteness. For example, for $\mathcal{A} = \mathbf{Set}$, the finitely presentable sets are precisely finite sets and the set N can be taken to be the natural numbers which we denote $\mathbb{N}$. Local finitely presentability is the categorical notion of when all the objects of a category can be generated from the finitely presentable objects of the category; for example, every set is the filtered colimit of the diagram of all its finite subsets ordered by inclusion. A finitary functor then is one who preserves this property in the sense that its action on all objects is given by the action on the generating objects. For example, a functor $F : \mathbf{Set} \longrightarrow \mathbf{Set}$ is finitary if its action on infinite set X is isomorphic to the the colimit of its images on finite subsets X_0 of X, ordered under inclusion: $F(X) = \cup_{X_0 \subseteq X} F(X_0)$.

Kan Extensions: Given a functor $I : \mathcal{A} \to \mathcal{B}$ and a category $\mathcal{C}$, precomposition with I defines a functor $\_ \circ I : [\mathcal{B}, \mathcal{C}] \to [\mathcal{A}, \mathcal{C}]$. The problem of left and right Kan extensions is the problem of finding left and right adjoints to $\_ \circ I$. More concretely, given a functor $F : \mathcal{A} \to \mathcal{C}$, the left and right Kan extensions satisfy the natural isomorphisms

$$[\mathcal{B}, \mathcal{C}](\mathrm{Lan}_I F, H) \cong [\mathcal{A}, \mathcal{C}](F, H \circ I) \qquad [\mathcal{B}, \mathcal{C}](H, \mathrm{Ran}_I F) \cong [\mathcal{A}, \mathcal{C}](H \circ I, F).$$

Thus, one can view the left and right Kan extension of a functor $F : \mathcal{A} \to \mathcal{C}$ along $I : \mathcal{A} \to \mathcal{B}$ as the canonical extensions of the domain of F to $\mathcal{B}$. Kan extensions can be given pointwise using colimits and limits, or more elegantly using ends and coends (see [MacLane 1971] for details). Since we work over the complete and cocomplete categories **Set** and **Pre**, all our Kan extensions exist.

In fact, given a lfp category $\mathcal{A}$, a functor $F : \mathcal{A} \longrightarrow \mathcal{B}$ is finitary precisely if it is (isomorphic to) the left Kan extension of its restriction to A_{fp} along the inclusion $A_{fp} \longrightarrow \mathcal{A}$.