Abstract Modularity

Michael Abbott[1], Neil Ghani[1] and Christoph Lüth[2]

[1] Department of Mathematics and Computer Science, University of Leicester
michael@araneidae.co.uk, ng13@mcs.le.ac.uk
[2] FB 3 — Mathematik und Informatik, Universität Bremen
cxl@informatik.uni-bremen.de

Abstract. Modular rewriting seeks criteria under which rewrite systems inherit properties from their smaller subsystems. This divide and conquer methodology is particularly useful for reasoning about large systems where other techniques fail to scale adequately. Research has typically focused on reasoning about the modularity of specific properties for specific ways of combining specific forms of rewriting.
This paper is, we believe, the first to ask a much more general question. Namely, what can be said about modularity independently of the specific form of rewriting, combination and property at hand. A priori there is no reason to believe that anything can actually be said about modularity without reference to the specifics of the particular systems etc. However, this paper shows that, quite surprisingly, much can indeed be said.

1 Introduction

The key properties of term rewriting systems (TRSs) are *confluence* and *strong normalisation*. One technique for establishing these properties is *modularity* which seeks criteria under which TRSs inherit properties from their smaller (and hence easier to reason about) subsystems. This divide and conquer methodology is particularly useful for reasoning about large systems where other techniques fail to scale adequately.

Research originally focused on *disjoint unions* of term rewrite systems where the systems do not share any operators. Here, confluence is modular [15] and strong normalisation is modular for *non-collapsing* TRSs and for *non-duplicating* TRSs [14]. Subsequently, a variety of alternative proof techniques have been developed [9, 5, 4, 12]. Modularity for *conditional term rewriting systems* (CTRSs) was first studied by Middeldorp [11] who showed that confluence is modular for certain types of CTRS while strong normalisation is again only modular in the presence of extra syntactic restrictions. Several unions permitting the sharing of term constructors have been proposed but, for each of these, confluence and strong normalisation are only modular again in the presence of various syntactic restrictions [13, 10].

These examples demonstrate how modularity is typically studied for specific combinations of rewrite systems, or specific notions of rewriting, or specific properties. This paper, we believe, is the first to ask what can be said about modularity independently of these specifics. That is we do not ask about the modularity of confluence, or termination, or weak termination, but rather seek a general modularity theorem applicable to all of these properties. Similarly, we wish to get away from modularity for specific notions of rewriting such as term rewriting, or graph rewriting, or equational rewriting and instead prove general modularity results applicable to as many forms of rewriting as possible. And finally, we wish to avoid commitment to modularity for specific ways of combining rewrite systems but rather extract conditions that are uniformly applicable to a variety of different such mechanisms.

A priori, there is no reason to believe that such an abstract theory of modularity should exist. Certainly it is hard to see how the conditions on the modularity of confluence, strong normalisation *etc.* are instances of the same general theme. This paper demonstrates that such a theory is indeed possible. Of course, it will not be able to magically prove the most general results for any specific situation. Rather, its contribution is to provide a platform of general results which can be instantiated for a specific situation as the need arises. In order to develop such a theory of abstract modularity, we have to build it upon a theme which unifies key features of specific modularity results. We believe the key concept in modularity is the notion of *layer structure* on the terms of the combined TRS which describes how rewrites in the combined TRS decompose into rewrites from the component TRSs. If rewrites do not preserve this layer structure (i.e. if there are collapsing rewrites), then non-trivial interactions between the layers may occur and modularity may fail.

Our results show that, providing rewrite systems preserve the layer structure, properties are inherently modular. We were quite surprised to find such a powerful result by using the techniques we have developed in our previous work [6, 8, 7]. To this end, we generalised our approach by treating not just term rewriting systems, but all rewrite systems that arise as *monads*, and by abstracting from specific properties to properties in general, given by a subcategory of the base.

Our general modularity result requires two conditions: i) that the rewrite systems do not collapse layers which is reflected in a condition on the monad representing the rewrite system; and ii) that the semantic and syntactic treatment of properties coincide. The latter condition ought to be automatic in the sense that it should hold for any reasonable property; it does for all well-known ones, such as confluence, termination and weak termination. Overall, we believe that this paper delivers on the promise of clean and simple results in rewriting based upon the categorical methodology.

The paper is structured as follows: In Sect. 2, we explain our abstract notion of the data structures we rewrite. In Sect. 3, we show how to model the actual rewriting by monads. In Sect. 4, we develop our semantic notion of properties of rewriting systems, and show they coincide with the well-known syntactic properties. Sect. 5 introduces an abstract notion of combining systems modelled by monads and shows the general modularity results, the key result being Thm. 28.

For this paper, we assume a very basic knowledge of category theory (comprising concepts such as categories, functors, push-outs and adjoints), but will explain all more sophisticated concepts as they are needed, and concentrate on examples and intuition rather than technical categorical proofs. For an introduction to category theory, see [**?**].

2 Abstract Data Structures

To enable our modularity results to as many forms of rewriting as possible, we need to extract their common feature. One possibility are abstract reduction systems (ARSs) which model a rewrite system by the one step reduction relation it induces. This semantics therefore throws away the structure of the data being rewritten, including the key concepts of substitution and layer which are central to modularity. Thus, it is unlikely that the ARS semantics of rewriting can be used as the basis of an abstract theory of modularity.

For us, rewriting consists of a data structure where subterms can be replaced with other terms and, as such, substitution is the fundamental property. Thus, we propose to use monads as they take as primitive an abstract notion of data endowed with a well behaved notion of substitution.

Definition 1 (Monad). *A* monad $\mathsf{T} = \langle T, \eta, \mu \rangle$ *on a category* $\mathcal{C}$ *is given by a functor* $T : \mathcal{C} \to \mathcal{C}$*, called the* action*, and two natural transformations,* $\eta : \mathsf{Id} \to T$*, called the* unit*, and* $\mu : TT \to T$*, called the* multiplication *of the monad, satisfying the* monad laws*:* $\mu \cdot T\eta = \mathsf{Id} = \mu \cdot \eta_T$*, and* $\mu \cdot T\mu = \mu \cdot \mu_T$*.*

Good introductions to the theory of monads in our sense are [**?**,**?**,**?**]. The canonical example of a monad is the one arising from the term algebra over a signature:

Definition 2 (Signature). *A (single-sorted)* signature *consists of a function* $\Sigma : \mathbb{N} \to \mathsf{Set}$*. The set of* n-ary operators *of* Σ *is defined* $\Sigma_n \stackrel{def}{=} \Sigma(n)$

Definition 3 (Term Algebra). *Given a signature* Σ *and a set of variables* X*, the* terms $T_\Sigma(X)$ *built over* X *are defined inductively:*

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

Lemma 4. *The map* $X \mapsto T_\Sigma(X)$ *defines a monad* T_Σ *on* Set*.*

Proof. Given a function $f : X \to Y$, renaming of variables defines a function $T_\Sigma(f) : T_\Sigma(X) \to T_\Sigma(Y)$. Every variable is a term, which gives us a family $\eta_X : X \to T_\Sigma(X)$ while substitution defines a family $\mu_X : T_\Sigma T_\Sigma(X) \to T_\Sigma(X)$. The monad laws state that substitution behaves correctly, i.e. is associative and has variables as left and right units, which is easily checked by induction. □

Our interest in monads is that they describe a number of other computationally interesting data structures possessing well behaved notions of substitutions, as the following examples show.

Example 5 (Strings). The map sending an alphabet X to the set X^* of words over X extends to a monad $\mathsf{T}^* : \mathsf{Set} \to \mathsf{Set}$. Substitution here takes a word consisting of words and flattens it into one word.

Example 6 (Groups and Rings). The map sending X to the free group $G(X)$ over X extends to a monad. Similarly, the map sending X to the set of free polynomials over X extends to a monad as well. In both cases, substitution is defined structurally, as for the term algebra above.

These examples can be generalised to any algebraic theories:

Example 7 (Algebraic theories). Given an algebraic theory $\langle \Sigma, E\rangle$ where Σ is a signature and E a set of equations, let $\sim_E$ be the congruence generated from E, and $T_{\langle \Sigma, E\rangle}(X) = T_\Sigma(X)/\sim_E$ be the term algebra quotiented by this congruence, then the map $X \mapsto T_{\langle \Sigma, E\rangle}(X)$ extends to a monad.

Furthermore, monads have another key advantage when applied to modularity, in that the interleaving of monads models the layer structure, e.g. $T_\Sigma(T_\Omega(X))$ consists of terms with a Σ-layer over a Ω-layer with variables built from X.

As a mild technical condition, we require these monads to be finitary which corresponds to the fact that the data structure under question is built inductively. Formally, a monad is finitary iff it preserves filtered colimits [**?**] (i.e. $T(X)$ is built from a finite subset of X). Finitary monads on a category $\mathcal{C}$ and monad morphisms form a category $\mathsf{Mon}(\mathcal{C})$. Motivated by all of this, we make our first definition of a rewrite structure, which is the structure containing the data over which rewriting takes place.

Definition 8. *A rewrite structure is a finitary monad* $\mathsf{T} : \mathsf{Set} \to \mathsf{Set}$.

To summarise, this section observed that in order to do rewriting, the fundamental properties required were the construction of some form of term calculus and a notion of substitution for that calculus. These concepts are perfectly captured by a rewrite structure.

3 Abstract Rewriting

A monad on Set builds a set of terms from a set of variables. Incorporating rewrites into this framework means that we are actually building a relation of terms and rewrites between them from a relation consisting of a set of variables and (what we consider to be) rewrites between these variables, called *variable rewrites*. That variables rewrite to other variables may seem odd from a rewriting perspective but in modularity these variables represent terms from sublayers, and terms in a sublayer certainly can rewrite to others. Further, as we shall see, adding variable rewrites does not affect properties such as confluence.

The exact nature of these relations depends upon what we are interested in studying. If we are interested in one-step reduction or one-step completion, we take relations, if we are interested in many-step reduction, we take preorders, or if we are interested in labelled rewriting, graphs or categories. For termination, we want to preserve reduction sequence, so we take transitive relations or well-orders. Here, the base category is the category Pre of preorders and monotone morphisms between them, but the reader should be aware that our general treatment can, and will be used with other base categories. We could make the following definitions parametric over the choice of Pre, and work with an arbitrary category $\mathcal{V}$ such that there is an adjunction as in Lemma 9 below, but we prefer a more concrete definition here.

Lemma 9. *The functor* $\mathsf{D} : \mathsf{Set} \to \mathsf{Pre}$*, which maps a set to the discrete preorder over it, is left adjoint to the functor* $\mathsf{V} : \mathsf{Pre} \to \mathsf{Set}$*, which maps a preorder to its underlying set.*

Proof. The adjunction is established by the isomorphism $\mathsf{Pre}(\mathsf{D}X, P) \cong \mathsf{Set}(X, \mathsf{V}P)$ for any set X and preorder P. □

In fact, D also has a left adjoint $\mathsf{C} : \mathsf{Pre} \to \mathsf{Set}$, which maps preorder to its set of connected components, and V has a further left adjoint, which maps a set to the total order on it. Now, there are a number of different ways of adding rewrites to a rewrite structure, that is to turn a monad on Set into a monad on Pre:

1. We can define the monad $M_R : \mathsf{Pre} \to \mathsf{Pre}$ to send a preorder X to the preorder defined as the abstract reduction semantics where there are no variables, but constants from X with associated variable rewrites. However, we still need to define the ARS semantics for each form of rewriting.
2. We can define the action of the monad concretely as in our previous work on term rewriting systems, e.g. [6, 7]. The advantage of this is that it gives a precise description of the rewrite monad, but at the cost of having to repeat the exercise every time we change the data structure.
3. We can define a rewrite presentation to be a parallel pair in $\mathsf{Mon}(\mathsf{Set})$, lift to $\mathsf{Mon}(\mathsf{Pre})$ and take the *coinserter*. This was the approach in [**?**]. The advantage of this approach is that it gives a precise and abstract formation of the rewrite system associated to any presentation, but at the cost of the technical overhead of coinserters.

In this paper, we choose an axiomatic approach which allows us to derive as many results as possible on a general level, and then instantiate them.

Definition 10. *Let M be a rewrite structure. An M-rewrite system is a finitary monad* $M_R : \mathsf{Pre} \to \mathsf{Pre}$ *such that M_R is a lifting of M, i.e. the following diagram commutes:*

$$\begin{tikzcd}
\mathsf{Pre} \arrow[r, "M_R"] \arrow[d, "\mathsf{V}"'] & \mathsf{Pre} \arrow[d, "\mathsf{V}"] \\
\mathsf{Set} \arrow[r, "M"'] & \mathsf{Set}
\end{tikzcd}$$

The condition says that the monad M_R which calculates terms and rewrites agrees with the monad M on the terms. Thus one can think of M_R as acting as M on terms, but adding in extra rewrites.

We have been speaking informally of the ARS semantics for rewrite systems, but now we make this precise. If R is a rewrite system, one usually fixes a countable infinite set of variables X, and considers the resulting ARS (which would be $M_R(X)$ here). But we can be more specific, as all countable infinite sets are isomorphic to the set $\mathbb{N}$ of natural numbers, so we take the discrete preorder $\mathsf{D}\mathbb{N}$ as the canonical representation of all variables, and call $M_R(\mathsf{D}\mathbb{N})$ the *representing ARS*. Thus, the difference between the monadic semantics and the ARS semantics is that the monadic semantics builds terms and rewrites over an arbitrary, not fixed, preorder of variables and variable rewrites. This extra flexibility is precisely what is required by modular rewriting as we can instantiate the variables and variable rewrites to be the terms and rewrites from a sublayer.

Given a rewrite structure M, there is always an empty (or discrete) M-rewrite system $M_\emptyset$ with no rewrite rules.

Lemma 11. *For a rewrite structure M, there is a free M-rewrite system $M_\emptyset$.*

Proof. The functor $\mathsf{V\_D} : \mathsf{Mon}(\mathsf{Pre}) \to \mathsf{Mon}(\mathsf{Set})$ has a left adjoint, denoted L, as shown in [**?**]. This computes the free lifting $M_\emptyset = L(M)$. $\square$

If M_R is an M-rewrite system, then by definition $M\mathsf{V} = \mathsf{V}M_R$. Precomposing with D, and noting $\mathsf{VD} = 1$, we get $M = \mathsf{V}M_R\mathsf{D}$, and hence a canonical embedding $\kappa : M_\emptyset \to M_R$ which embeds the empty M-rewrite system in any other M-rewrite system. Given a M-rewrite system, we will often want to abstractly use the idea that rewrites created by M_R are either created by an underlying rewrite system or by the variable rewrites. This is captured by asking that the diagram (1) be a push-out, where ε is the counit of the adjunction of Lemma 9.

$$\begin{tikzcd} M_\emptyset(\mathsf{DV}X) \arrow[r, "M_\emptyset\varepsilon"] \arrow[d, "\kappa_{\mathsf{DV}X}"'] & M_\emptyset X \arrow[d, "\kappa_X"] \\ M_R(\mathsf{DV}X) \arrow[r, "M_R\varepsilon"'] & M_R X \end{tikzcd} \tag{1}$$

We say an M-rewrite system M_R is *cocartesian* iff $\kappa : M_\emptyset \to M_R$ is a cocartesian natural transformation, i.e. all components form push-out squares. Most M-rewrite systems are cocartesian, because $M_R(X)$ is the coproduct of the monads representing R and representing the rewrites of X.

We finish this section with some examples.

Example 12 (Term Rewriting). A term rewriting system $\langle \Sigma, R\rangle$ has as a rewrite structure the term algebra monad T_Σ and as a T_Σ-rewrite system the monad $\mathsf{T}_{\langle\Sigma,R\rangle}$ which sends a preorder X to the smallest ordered Σ-algebra $T_{\langle\Sigma,R\rangle}(X)$ containing X for which R is sound [**?**]. Cocartesianness follows from the inductive construction of $T_{\langle\Sigma,R\rangle}(X)$ [6].

Example 13 (String Rewriting and Gröbner Bases). String rewriting can be regarded as rewriting over the free monoid, i.e. words, while Gröbner bases can be regarded as rewriting over free rings, i.e. polynomials. The rewrite structure here is given by Example 5 and 6, with M_R adding in a reduction structure between the words and polynomials respectively.

Thus, for example, Gröbner bases have as a rewrite system the monad that computes for a preorder X, the smallest preorder on the free ring over X containing X for which the ring operations are monotone and for which R is sound.

Example 14 (Equational Rewriting). The rewrite structure for an equational term rewriting system $\langle \Sigma, E, R\rangle$ is the monad $\mathsf{T}_{\langle \Sigma, E\rangle}$ from Example **??**, and as a $\mathsf{T}_{\langle \Sigma, E\rangle}$-rewrite system the monad $\mathsf{T}_{\langle \Sigma, E, R\rangle}$ which sends a preorder X to the smallest preorder on $T_{\langle \Sigma, E\rangle}(X)$ for which the Σ-constructors are monotone and for which R is sound. Cocartesianness follows from cocartesianness of $\mathsf{T}_{\langle \Sigma, R\rangle}$.

Further examples could be developed, e.g. the rational monad suffices as a rewrite structure to consider rational rewriting. Recent work on abstract syntax shows that structures with variable binding are monads. This monadic approach to higher order rewriting has been developed by Hamana [3].

These examples follow the general pattern. Given a rewrite structure M, a M-rewrite system is given by triples (Y, l, r) where Y is a set and l, r are elements of MY. The associated M-rewrite system M_R maps a preorder X to the smallest preorder on $M(X)$ for which the operations in M are monotone and for which the interpretations of l is greater than that of r. When this order relation is defined inductively, the cocartesianness of $\kappa : M_\emptyset \to M_R$ follows.

To summarise, M-rewrite systems provide a model of rewriting covering a large variety of different forms of rewriting. In fact, given any underlying data structure for rewriting which forms a monad, i.e. possesses a well behaved notion of substitution, we can model rewriting over that data structure by a monad which sends a preorder to the smallest preorder over $M\mathsf{V}(X)$ containing X, which forms an M-algebra which validates the rewrites.

4 Abstract Properties

Properties of rewrite systems are often given via properties of the associated abstract reduction system, e.g. a TRS is confluent iff the rewrites built from the TRS using a countably infinite set of variables form a confluent preorder. If we are going to reason about rewriting using M-rewrite systems, we need a definition of properties in terms of the representing monad M_R. The direct translation is that M_R satisfies P if the representing ARS $M(\mathsf{D}\mathbb{N})$ does. However, given the need for variable rewrites in modularity, it is only reasonable to ask the relation $M_R(X)$ to satisfy a property if the relation X does, and thus an alternative definition would be that M_R satisfies P iff it preserves P. We say that property is *monadic* if these two notions coincide:

Definition 15. *Let P be a property of preorders, characterising a subcategory $\mathcal{K}$ of Pre. We say P is* monadic *if the following holds: $M(\mathsf{D}\mathbb{N}) \in \mathcal{K}$ iff whenever $X \in \mathcal{K}$ then $M_R(X) \in \mathcal{K}$.*

If this definition is sensible it must be satisfied by the standard properties such as confluence and strong normalisation, so we first check if these two are monadic.

4.1 Abstract Confluence

In this section we prove that confluence is monadic according to Def. **??**, i.e. a finitary monad $M_R : \mathsf{Pre} \to \mathsf{Pre}$ preserves confluence iff its representing ARS $M_R(D\mathbb{N})$ is confluent. One direction is easy: if M_R preserves confluence, then since $\mathsf{D}\mathbb{N}$ is discrete and hence trivially confluent, $M_R(\mathsf{D}\mathbb{N})$ is also confluent.

Note that in the following cocartesianness is not required , and that previous results [6, 7] restricted to the case where M_R is the representing monad for a TRS and used an explicit inductive construction of this monad.

To prove our result, we use a characterisation of confluence in terms of maps.

Lemma 16. *If X is a finite preorder then it is confluent iff the map $f : X \to \mathsf{DC}X$ has a right adjoint $g : \mathsf{DC}X \to X$, denoted as $f \dashv g$.*

Proof. Let X be confluent. To each connected component of X assign an upper bound of the connected component which exists by confluence and finiteness. This defines a monotone function $g : \mathsf{DC}X \to X$ which satisfies $fg = 1$ and $1 \to gf$, establishing $f \dashv g$. Conversely, given such a right adjoint, it is obvious that each connected component has a minimal element, making X confluent. ☐

Adjoints like the above allow us to reflect confluence.

Lemma 17. *Let X, Y be preorders, and maps $f : X \to Y$, $g : Y \to X$ such that $1 \to gf$. Then if Y is confluent so is X.*

Proof. Let $b \leftarrow a \to c$ be a span in X with completion $fb \to d \leftarrow fc$ in Y, which has an image $gfb \to gd \leftarrow gfc$ in X. Since $b \to gfb$, and $c \to gfc$ in X, gd is a completion of $b \leftarrow a \to c$, hence X is confluent. ☐

Lemma 18. *Let $\mathsf{M} = \langle M, \eta, \mu \rangle$ be a monad such that $M\mathsf{D}\mathbb{N}$ is confluent, then $M\mathsf{D}X$ is confluent for every finite X.*

Proof. We proceed by assuming that $M\mathsf{D}X$ is inhabited, e.g. by $* \in M\mathsf{D}X$. We can assume this without loss of generality, as given any span $b \leftarrow a \to c$ in $M\mathsf{D}X$ which needs completing, we can take $* = a$.

With X finite, we have $f : X \hookrightarrow \mathbb{N}$ and hence $\mathsf{D}f : \mathsf{D}X \hookrightarrow \mathsf{D}\mathbb{N}$, and we can define a map $g : \mathsf{D}\mathbb{N} \to M\mathsf{D}X$ by cases so that the following commutes:

$$\begin{tikzcd} \mathsf{D}X \arrow[r, "\mathsf{D}f"] \arrow[dr, "\eta_{\mathsf{D}X}"'] & \mathsf{D}\mathbb{N} \arrow[d, "g"] \\ & M\mathsf{D}X \end{tikzcd} \qquad g(x) = \begin{cases} \eta_{\mathsf{D}X}(x) & \text{for } x \in \mathsf{D}X \\ * & \text{for } x \notin \mathsf{D}X \end{cases}$$

Now $g^\flat : M\mathsf{D}\mathbb{N} \to M\mathsf{D}X$ (the Kleisli extension of g) is defined as $g^\flat = \mu_{\mathsf{D}X} \cdot Mg$, and hence satisfies $g^\flat \cdot M\mathsf{D}f = \mu_{\mathsf{D}X} \cdot Mg \cdot M\mathsf{D}f = \mu_{\mathsf{D}X} \cdot M\eta_{\mathsf{D}X} = 1$. This allows us to reflect confluence of $M\mathsf{D}\mathbb{N}$ along $M\mathsf{D}f : M\mathsf{D}X \to M\mathsf{D}\mathbb{N}$ using Lemma 17, making $M\mathsf{D}X$ confluent. □

Lemma 19. *Let M be a functor $M : \mathsf{Pre} \to \mathsf{Pre}$ taking finite discrete preorders to confluent preorders. Then M also takes finite confluent preorders to confluent preorders.*

Proof. Let X be a finite confluent preorder. By Lemma 16 there are two adjoint maps $f \dashv g : X \to \mathsf{DC}X$, lifting to two adjoint maps $Mf \dashv Mg : MX \to M(\mathsf{DC}X)$. By Lemma 18, the preorder $M(\mathsf{DC}X)$ is confluent and hence by Lemma 17, MX is confluent. □

The following lemma uses the finite accessibility of Conf, which is a technical property and means that all confluent preorders are finitely generated. This allows us to deduce confluence of infinite preorders from the confluence of their finite suborders. We can then establish the monadicity of confluence as follows:

Lemma 20. *If M is a finitary monad and $M\mathsf{D}\mathbb{N}$ is confluent then MX is confluent whenever X is.*

Proof. By Lemma 18, $M\mathsf{D}X$ is confluent for finite X if $M\mathsf{D}\mathbb{N}$ be confluent. By Lemma 19, MP is confluent for every finite confluent preorder P. Finally, use finite accessibility of Conf to write any confluent P as a filtered colimit $P \cong \mathsf{colim}\, P_i$ of finite confluent preorders. We can now write

$$MP \cong M \ \mathsf{colim}\, P_i \cong \mathsf{colim}\, MP_i$$

concluding that MP can be written as a filtered colimit of confluent preorders and is therefore confluent. □

4.2 Abstract Strong Normalisation

Strong normalisation can be treated in a similar way. We do need a different base category though, as we need to exclude identity rewrites. First a few preliminaries. Let Trans be the category of transitive, but not necessarily reflexive, orders and monotone functions between them, and let WO_f be the full subcategory of well-founded, finitely branching orders. These are the strongly normalising orders that we are interested in. For any $X \in \mathsf{Set}$, we have the discrete order on X which is transitive but not reflexive (and of course in WO_f), which by abuse of language we call $\mathsf{D}X$; and similarly, for the underlying set of a transitive order Y we use $\mathsf{V}Y$. This overloading of notation makes sense, as we are now using a different instance of Definition 10 (with Trans for Pre).

To characterise WO_f algebraically, we use maps into and from ω, the natural numbers ordered by the strictly-greater relation $>$ (or strict reverse inclusion), and their dual ω^{op} with the reversed order, as follows.

Lemma 21. *If $X \in \mathsf{Trans}$, then $X \in \mathsf{WO}_f$ iff there is a map $X \to \omega$. If X is not in WO_f then there is a map $\omega^{op} \to X$.*

Note the finite branching is required to ensure that to each element of a well-founded order we can assign an element of ω (since each element only reduces to a finite number of direct successors), and that this assignment is monotone.

We now show that strong normalisation is monadic. According to Def. **??**, we need to show that given an M-rewriting system M_R, $M_R(\mathsf{D}\mathbb{N})$ is SN iff the monad M_R preserves strong normalisation, or equivalently, restricts to WO_f. One direction is easy: if M_R preserves SN, then since $\mathsf{D}\mathbb{N}$ is discrete and hence trivially well-founded, $M_R(\mathsf{D}\mathbb{N})$ is also SN. Our aim is to show the converse, and as with confluence, we build up to our result systematically. Note that opposed to confluence, we now need M_R to be cocartesian.

The first result shows that we can show strong normalisation by replacing all variables with a canonical element. For this, let $\mathbf{1}$ be the one-element set. Then, there is exactly one map $!_X : X \to \mathsf{D}\mathbf{1}$ in Trans if and only if X is discrete as the map needs to be monotone and $\mathsf{D}\mathbf{1}$ has an empty order structure.

Lemma 22. *Let $T : \mathsf{Trans} \to \mathsf{Trans}$ and $T\mathsf{D}\mathbf{1} \in \mathsf{WO}_f$, then $T\mathsf{D}X \in \mathsf{WO}_f$ for all sets X.*

Proof. By Lemma **??**, there is a map $T\mathsf{D}\mathbf{1} \to \omega$. Since $\mathsf{D}X$ is discrete, there is a map $\mathsf{D}!_X : \mathsf{D}X \to \mathsf{D}\mathbf{1}$. Applying T and composing with the first map gives a map $T\mathsf{D}X \to \omega$, hence by Lemma **??**,$T\mathsf{D}X \in \mathsf{WO}_f$. ☐

Now observe that the free lifting $M_\emptyset$ of the monad M is strongly normalising because the only rewrites in $M_\emptyset$ are variable rewrites. The second main step in showing that normal SN implies monadic SN is to show that adding an order structure to the variables does not affect SN:

Lemma 23. *Let M_R be a cocartesian M-rewrite system with $\kappa : M_\emptyset \to M_R$ a cocartesian transformation. If $M_R(\mathsf{D}\mathbf{1})$ is SN and X is SN, then $M_R X$ is SN.*

Proof. We know that $M_\emptyset X$ and by Lemma 23 $M_R(\mathsf{DV}X)$ are well-founded and hence there are maps $\alpha_1 : M_R(\mathsf{DV}X) \to \omega$ and $\alpha_2 : M_\emptyset X \to \omega$. α_1 and α_2 do not form a cone over the square in Diagram (2), i.e. $\alpha_1 \cdot \kappa \neq \alpha_2 \cdot M^\dagger \varepsilon$, so we define new maps $\beta_1 : M_R(\mathsf{DV}X) \to \omega$ and $\beta_2 : M_\emptyset X \to \omega$ by

$$\beta_1(t) = \mathsf{max}\{\alpha_1(t), \alpha_2(t)\} \qquad \beta_2(t) = \mathsf{max}\{\alpha_1(t), \alpha_2(t)\}$$

This can be done as all the orders mentioned above have the same carrier. That the β_i are monotone is easily checked and, since they have the same underlying function, we have a cone over the square in diagram (2), and since this is a push-out square (because of the cocartesianness of M_R), we have a map $M_R X \to \omega$ as required.

$$
\begin{tikzcd}
M^\dagger \mathsf{D}\vee X \arrow[r, "M^\dagger\varepsilon"] \arrow[d, "\kappa"'] & M^\dagger X \arrow[d, "\kappa"] \arrow[ddr, bend left, "\beta_2"] & \\
M_R(\mathsf{D}\vee X) \arrow[r, "M_R\varepsilon"'] \arrow[drr, bend right, "\beta_1"'] & M_R(X) \arrow[dr, dashed] & \\
 & & \omega
\end{tikzcd}
\tag{2}
$$

□

Putting all the pieces together, we get the main result:

Lemma 24. *Let M_R be a cocartesian M-rewrite system. The ARS $M_R(\mathsf{D}\mathbb{N})$ is SN iff the monad M_R is SN.*

Summing up this section, we have shown that confluence and SN are monadic, i.e. their usual definition in terms of the representing ARS coincides with them being preserved by the monad representing an M-rewrite system.

5 Abstract Combinators

Modularity deals with combinations of systems, so we are now going to consider the combination of M-rewrite systems. We do so by defining combinators for putting together the representing monads. The appropriate categorical construction here is the *colimit*, but computing the colimit of monads in full generality is a technically involved exercise. Even if we restrict ourselves to the coproduct, corresponding to the disjoint union M-rewrite systems, the construction is very unwieldy, and hence much research has recently focused on developing simpler algorithms which are correct in specific situations. Ideal monads are one such situation which correspond to the idea of layers being non-collapsing.

5.1 Ideal Monads

Intuitively, ideal monads are monads whose variable part can be separated from the non-variable part. Formally:

Definition 25 (Ideal Monad). *A monad $\mathsf{T} = \langle T, \eta, \mu\rangle$ is* ideal *iff there is a functor T_0 such that $T = \mathsf{Id} + T_0$, the unit is the left injection and there is a natural transformation $\mu_0 : T_0 T \to T_0$ such that*

$$
\begin{tikzcd}
T_0 T \arrow[r, "\mathsf{inr}_T"] \arrow[d, "\mu_0"'] & TT \arrow[d, "\mu"] \\
T_0 \arrow[r, "\mathsf{inr}"'] & T
\end{tikzcd}
$$

where $\mathsf{inr} : T_0 \to \mathsf{Id} + T_0$ is the right injection into the coproduct.

We write ideal monads in the form $\mathsf{Id} + T_0$ for simplicity (where Id is the identity functor) and leave the restricted form of multiplication μ_0 implicit. A monad morphism $f : \mathsf{Id} + T_0 \to R$ whose source is an ideal monad has its action on Id forced by the monad laws and is hence of the form $[\eta^R, f_0]$ where $f_0 : T_0 \to R$. Examples of ideal monads over Set include the term monads T_Σ, the string and ring monads from Examples 5 and 6, and in general any algebraic theory $\mathsf{T}_{\langle \Sigma, E\rangle}$ where both sides of every equation are either variable terms or non-variable terms; hence, a counter-example is the group monad from Example 6.

The fundamental observation behind the construction of the coproduct $R+S$ of two ideal monads $R = \mathsf{Id} + R_0$ and $S = \mathsf{Id} + S_0$ is that $R + S$ should contain R and S as submonads, and further, that $R + S$ should be closed under the application of R_0 and S_0. Hence, $R + S$ should consist of alternating sequences beginning from R_0 or S_0, and we ask for the least functors satisfying the following mutually recursive equations:

$$T_1 \cong R_0(\mathsf{Id} + T_2) \qquad\qquad T_2 \cong S_0(\mathsf{Id} + T_1).$$

The solution is computed as the least fixpoint of a functor Φ on the product category $(\mathsf{Pre} \to \mathsf{Pre}) \times (\mathsf{Pre} \to \mathsf{Pre})$ (so Φ takes pairs of functors as arguments):

$$\langle T1, T2\rangle = \mu\Phi \qquad \Phi\langle F, G\rangle = \langle R_0 \cdot (\mathsf{Id} + G), S_0 \cdot (\mathsf{Id} + F)\rangle \tag{3}$$

To solve the fixpoint equation, note that the functor $c_0 : \mathsf{Pre} \to \mathsf{Pre}$ which constantly returns the initial object is initial in the functor category $\mathsf{Pre} \to \mathsf{Pre}$. We can then use the following standard construction: for a finitary functor $F : \mathcal{C} \to \mathcal{C}$, the least fixpoint μF is given by the colimit of the following chain (if it exists and there is an initial object 0, with $! : 0 \to X$ the unique map out of the initial object):

$$0 \xrightarrow{\;\;!\;\;} F0 \xrightarrow{\;F!\;} F^2 0 \xrightarrow{\;F^2!\;} F^3 0 \quad \ldots \tag{4}$$

Now, intuitively T_1 consists of elements in $R + S$ whose top layer is a non-variable R-layer (captured by the use of R_0) and whose next layers are either variables or a non-variable S layer, etc. In our opinion, this is a very elegant way of capturing the layer structure in the disjoint union of two systems. The following result proves our intuition correct and can be found in [2].

Theorem 26. *The action of the coproduct of ideal monads $\mathsf{Id} + R_0$ and $\mathsf{Id} + S_0$ is the functor $T = \mathsf{Id} + (T_1 + T_2)$, where T_1 and T_2 are defined as in (3).*

The central result of this paper is that those rewrite systems whose representing monad is ideal have good modularity properties and, further, that these are actually rather easy to derive. Note that Theorem 27 holds for all ideal monads, i.e. all ideal M-rewrite systems, not only term-generated ones. We now prove the central theorem from which all our modularity results can be uniformly derived.

Theorem 27. *Let P be a monadic property represented by a subcategory $\mathcal{K}$ of Pre. If $\mathcal{K}$ has coproducts, an initial object and ω-colimits, then P is modular for the disjoint union of ideal M-rewriting systems.*

Proof. Let R and S be ideal M-rewriting systems satisfying P. To show that their disjoint union has the property P, we have to show that, given $X \in \mathcal{K}$, $R + S(X) = X + T_1(X) + T_2(X)$ is in $\mathcal{K}$.

By Theorem 27, $T_1(X)$ and T_2X are given by the initial fixpoint of Φ in (3) at X; i.e. the colimit of the chain (**??**). We know that both R_0 and S_0 preserve

$$\langle 0, 0\rangle \xrightarrow{!} \langle R_0X, S_0X\rangle \longrightarrow \langle R_0(X + S_0X), S_0(X + R_0X)\rangle \longrightarrow \ldots \quad (5)$$

$\mathcal{K}$, and since $\mathcal{K}$ has coproducts and an initial object, all objects of the chain (**??**) are in $\mathcal{K}$, and since $\mathcal{K}$ has ω-colimits, so is the fixpoint, i.e. $T_1(X)$ and $T_2(X)$. With $\mathcal{K}$ having coproducts, we get that $X + T_1(X) + T_2(X) \in \mathcal{K}$. □

We finish this section by using Theorem 28 to uniformly derive a number of modularity results.

Example 28 (Confluence of Non-Collapsing TRSs). Take $\mathcal{K}$ to be the full subcategory Conf of Pre whose objects are confluent orders. A TRS Θ is confluent iff $T_\Theta(D\mathbb{N})$ is confluent, where T_Θ is its representing monad (Ex. 12). Using Lemma 20, this is equivalent to T_Θ being confluent.

It remains to show that confluence satisfies the preconditions of Theorem 28. Clearly, the empty preorder (the initial object) is confluent, and the disjoint union of two confluent preorders is confluent. Further, given an ω-chain of confluent preorders, their colimit (i.e. the least upper bound) will be confluent as well (this is the finite accessibility of Conf mentioned above), allowing us to conclude the result.

Example 29 (Strong Normalisation for Non-Collapsing TRS). In this example, we have to change the base category from Pre to Trans (as in Sect. **??**), and let $\mathcal{K}$ to be the category WO_f of finitely branching well-founded orders. As in the previous example, we can use Lemma 25 to show that termination of a TRS Θ and termination of the monad T_Θ coincide.

It remains to show that strong normalisation satisfies the preconditions of Theorem 28. The empty relation is SN. The disjoint union preserves SN. WO_f actually fails to have all filtered colimits, but fortunately it does have colimits of chains which preserve the normalisation rank, as is the case for the chain in (**??**). Thus, Theorem 28 applies.

Example 30 (Adding Equations; Modularity for Equational TRSs). Let R be a confluent non-collapsing TRS. Assume we want to add to R a fresh associative operator $\otimes$ and prove the resulting system remains confluent.

The monad $\mathsf{T}_{\langle \otimes, E\rangle}$ given by the algebraic theory with one operation $\otimes$ and the equation E stating associativity of $\otimes$ is confluent (trivially, as it contains no rewrites); note that the base category of this monad is Pre, not Set (in fact, we treat the algebraic theory as an equational rewrite system without rewrites). We have already established that confluent preorders satisfy the preconditions of Theorem 28. With T_R the monad representing the TRS R, can easily deduce that $\mathsf{T}_R + \mathsf{T}_{\langle \otimes, E\rangle}$ satisfies confluence, hence $R + \langle \otimes, E\rangle$ is confluent as well.

This can be generalised to two arbitrary, non-collapsing equational term rewriting systems: if both are confluent or SN, so will be their disjoint union.

6 Conclusion and Future work

We have demonstrated that there is indeed a theory of modularity which abstracts from the specific notion of rewriting, property and combination under consideration. Moreover, we believe that our use of monads has helped to establish these results in an elegant and straightforward way. Underlying this is the simple representation of the layer structure as the interleaving of monads and the use of variable rewrites to model rewrites in sublayers. As mentioned in the introduction, the point about these examples is not that they are the most general results for a specific modularity problem, but rather that we have a uniform principle that works in a variety of different situations.

In the above, we have used Pre as the base category, and switched to Trans when considering strong normalisation. The exact way to model this would have been define a rewrite system as parameterised over a base category $\mathcal{C}$, which has to satisfy certain properties, but we felt this would make the exposition more categorical and less rewriting. We have also omitted rewriting of infinite terms as the corresponding monads are not finitary. Our methodology still works, but requires us to work at a higher rank (with transfinite constructions), the technicalities of which we felt would distract from the concrete term rewriting contribution of the present paper.

The applications to graph rewriting need to be examined more closely. Graph rewriting has the dual modularity results then normal term rewriting (i.e. confluence is not modular but SN is). We can model term graphs with monads [**?**,**?**], but the precise relation of the monadic properties to the properties of term graph rewriting systems is not clear.

We would like to comment on the limitations of this work. Higher-order systems with variable binding are essentially not covered at all, because although this can be modelled in the monad framework [**?**], higher-order systems are not ideal monads (the reason is that free variables can be captured when building a new layer).

In future work we wish to make these ideas accessible to a wider audience by developing many more different examples and applications. We also plan to extend the methodology to other methods of combining rewrite systems than the disjoint union, in particular modelling constructor sharing systems, where first tentative steps have already been taken.

References

1. J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories.* London Mathematical Society Lecture Note Series 189. Cambridge University Press, 1994.
2. M. Barr and C. Wells. *Toposes, Triples and Theories.* Grundlehren der mathematischen Wissenschaften 278. Springer Verlag, 1985.
3. N. Ghani and C. Lüth. Rewriting via coinserters. *Nordic Journal of Computing*, 10:290– 312, 2004.
4. N. Ghani, C. Lüth, and F. de Marchi. Solving algebraic equations using coalgebra. *Journal of Theoretical Informatics and Applications*, 37:301–314, 2003.

5. N. Ghani, C. Lüth, and F. de Marchi. Monads of coalgebras: Rational terms and term graphs. To appear in *Mathematical Structures in Computer Science*.
6. N. Ghani and T. Uustalu. Explicit substitutions and higher-order syntax (extended abstract). In F. Honsell, M. Miculan, and A. Momigliano, editors, *Proc. of 2nd ACM SIGPLAN Wksh. on Mechanized Reasoning about Languages with Variable Binding, MERLIN'03*, pages 1–7. ACM Press, New York, 2003.
7. N. Ghani and T. Uustalu. Coproducts of ideal monads. *Journal of Theoretical Informatics and Applications*, 38:321–342, 2004.
8. M. Hamana. Term rewriting with variable binding: an initial algebra approach. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming*, pages 148–159. ACM Press, 2003.
9. J. W. Klop, A. Middeldorp, Y. Toyama, and R. de Vrijer. A simplified proof of Toyama's theorem. *Information Processing Letters*, 49:101–109, 1994.
10. M. Kurihara and I. Kaji. Modular term rewriting systems and the termination. *Information Processing Letters*, 34(1):1–4, Feb. 1990.
11. C. Lüth. *Categorical Term Rewriting: Monads and Modularity.* PhD thesis, University of Edinburgh, 1998.
12. C. Lüth and N. Ghani. Monads and modular term rewriting. In *Category Theory in Computer Science CTCS'97*, LNCS 1290, pages 69– 86. Springer Verlag, 1997.
13. C. Lüth and N. Ghani. Monads and modularity. In A. Armando, editor, *Frontiers of Combining Systems FroCos 2002, 4th International Workshop*, LNAI 2309, pages 18–32. Springer Verlag, 2002.
14. S. MacLane. *Categories for the Working Mathematician*, *Graduate Texts in Mathematics* 5. Springer Verlag, 1971.
15. E. G. Manes. *Algebraic Theories*, *Graduate Texts in Mathematics* 26. Springer Verlag, 1976.
16. M. Marchiori. Modularity of completeness revisited. In J. Hsiang, editor, *Proceedings of the 6^{th} International Conference on Rewriting Techniques and Applications RTA'95*, LNCS 914, pages 2– 10, Springer Verlag, 1995.
17. A. Middeldorp. *Modular Properties of Term Rewriting Systems*. PhD thesis, Vrije Universiteit te Amsterdam, 1990.
18. A. Middeldorp and Y. Toyama. Completeness of constructor systems. Technical Report CS-R9058, Centrum voor Wiskunde en Informatica, Amsterdam, 1990.
19. E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136:333– 360, 1994.
20. E. Ohlenbusch. On the modularity of confluence of constructor-sharing term rewriting systems. In S. Tison, editor, *Trees in Algebra and Programming — CAAP 94*, LNCS 787. Springer Verlag, 1994.
21. E. Robinson. Variations on algebra: monadicity and generalisations of equational theories. Manuscript. Available at `http://www.dcs.qmw.ac.uk/~edmundr/pubs/algebras/algebras.ps`, 1994.
22. M. Rusinowitch. On the termination of the direct sum of term-rewriting systems. *Information Processing Letters*, 26(2):65–70, 1987.
23. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.