Natural Transformations as Rewrite Rules and Monad Composition

Dexter Kozen
Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA
kozen@cs.cornell.edu

July 2, 2004

Abstract

Eklund et al. [6] present a graphical technique aimed at simplifying the verification of various category-theoretic constructions, notably the composition of monads. In this note we take a different approach involving string rewriting. We show that a given tuple (T, μ, η) is a monad if and only if T is a terminal object in a certain category of functors and natural transformations, and that this fact can be established by proving confluence of a certain string rewriting system. We illustrate the technique on the monad composition problem of Eklund et al.

1 Introduction

As common constructions in the theory of data types and programming language semantics become better understood, there is a natural tendency toward generality. One desires to isolate common underlying principles, to unify related notions in a common framework, and to provide powerful abstract tools for reasoning and understanding.

A good example of one successful such enterprise is the use of monads in functional and logic programming [10, 16, 2, 6]. Monads provide a clean way to combine modules or extend functionality of programming languages or data structures with new features such as continuations, state, and concurrency [10, 7, 16, 5]. They have been applied to parsing and type checking [15], semantics of nondeterministic and probabilistic computation [10, 12, 11], and unification in logic programming [13].

Unfortunately, with abstraction often comes reduced accessibility. Many abstract constructions, although well motivated by applications, may at times be difficult to navigate when presented in more abstract form. In particular, reasoning about the basic properties of monads—such as *monad composition*,

the construction that underlies many of the applications above—relies on the combinatorial manipulation of functors and natural transformations. Verification often requires a complicated process of arrow chasing in large diagrams. Specialized verification tasks such as the following example from [3] are not uncommon.

$$\begin{tikzcd}
T_1^2T_2 \arrow[r, "T_1\eta_2T_1T_2\eta_1"] \arrow[d, "T_1^2T_2\eta_1"'] & T_1T_2T_1T_2T_1 \arrow[r, "T_1\mu"] \arrow[d, "\eta_2T_1T_2T_1T_2T_1"'] & T_1T_2T_1 \arrow[d, "\eta_2T_1T_2T_1"] \\
T_1^2T_2T_1 \arrow[r, "\eta_2T_1\eta_2T_1T_2T_1"'] \arrow[d, "\mu_1T_2T_1"'] & T_2T_1T_2T_1T_2T_1 \arrow[r, "T_2T_1\mu"'] \arrow[d, "\mu T_2T_1"'] & T_2T_1T_2T_1 \arrow[d, "\mu"] \\
T_1T_2T_1 \arrow[r, "\eta_2T_1T_2T_1"'] & T_2T_1T_2T_1 \arrow[r, "\mu"'] & T_2T_1
\end{tikzcd}$$

There are several general accounts of monad composition in the literature [9, 2, 6]. In particular, Eklund et al. [6] make note of the difficulty of monad composition proofs and present a graphical technique aimed at simplifying the process. Their technique provides a pictorial representation of various constructions.

In this note we take a different approach involving string rewriting. We show that a given tuple (T, μ, η) is a monad if and only if T is a terminal object in a certain category of functors and natural transformations, and that this fact can be established by proving confluence of a certain easily-described string rewriting system. Unlike the approaches of [9, 2, 6], this gives a general technique for verifying the commutativity of diagrams of arbitrary size, not just particular instances that (one hopes) fit on a page. We illustrate the technique on the monad composition problem studied by Eklund et al.

2 Functors and Natural Transformations

Let C, D be categories. Recall that a *natural transformation* $\varphi : F \to G$ between functors $F, G : \mathsf{C} \to \mathsf{D}$ is a collection of morphisms $\varphi_A : FA \to GA$ of D, one for each object A of C, such that for any morphism $h : A \to B$ of C, the following diagram commutes:

$$\begin{tikzcd}
FA \arrow[r, "Fh"] \arrow[d, "\varphi_A"'] & FB \arrow[d, "\varphi_B"] \\
GA \arrow[r, "Gh"'] & GB
\end{tikzcd} \tag{2.1}$$

The functors $\mathsf{C} \to \mathsf{D}$ and natural transformations $\varphi : F \to G$ form a category. Composition of natural transformations is defined by $(\varphi \circ \psi)_A = \varphi_A \circ \psi_A$. We omit the composition symbol $\circ$ when applied to functors, writing ST for $S \circ T$, but retain it when applied to natural transformations.

If $T : \mathsf{C} \to \mathsf{C}$ is an endofunctor and $\varphi : F \to G$ a natural transformation, then $\varphi T : FT \to GT$ with components $\varphi T_A = \varphi_{TA}$ is also a natural transformation. Similarly, if $S : \mathsf{D} \to \mathsf{D}$ is an endofunctor, then $S\varphi : SF \to SG$ with components $(S\varphi)_A = S(\varphi_A)$ is also a natural transformation. Note that if $\varphi : F \to G$, then $S\varphi T : SFT \to SGT$. We refer to this operation as *scalar multiplication*.

Scalar multiplication satisfies the following properties:

$$I\varphi = \varphi = \varphi I \tag{2.2}$$

$$(SS')\varphi = S(S'\varphi) \qquad \varphi(TT') = (\varphi T)T' \tag{2.3}$$

$$S(\varphi \circ \psi) = S\varphi \circ S\psi \qquad (\varphi \circ \psi)T = \varphi T \circ \psi T. \tag{2.4}$$

Thus we can write $SS'\varphi$ and $\varphi TT'$ without ambiguity.

It follows from (2.4) that commutative diagrams remain commutative under scalar multiplication. For example, if the left-hand diagram in (2.5) commutes, then so does the right:

$$\begin{tikzcd} U \arrow[r, "\varphi"] \arrow[d, "\psi"'] & V \arrow[d, "\tau"] \\ X \arrow[r, "\sigma"'] & Y \end{tikzcd} \qquad \begin{tikzcd} PUQ \arrow[r, "P\varphi Q"] \arrow[d, "P\psi Q"'] & PVQ \arrow[d, "P\tau Q"] \\ PXQ \arrow[r, "P\sigma Q"'] & PYQ \end{tikzcd} \tag{2.5}$$

Let $\mathcal{F}$ be a collection of endofunctors on C closed under composition. Let Σ be a collection of natural transformations on $\mathcal{F}$. We define a certain subcategory $(\mathcal{F}\,;\,\Sigma)$ of the category of endofunctors on C and natural transformations. The objects of $(\mathcal{F}\,;\,\Sigma)$ are $\mathcal{F}$. The arrows of $(\mathcal{F}\,;\,\Sigma)$ are those natural transformations generated by Σ under scalar multiplication with elements of $\mathcal{F}$ and composition. These include all $U\sigma V$ for $U, V \in \mathcal{F}$ and $\sigma \in \Sigma$ and their (well-typed) compositions, including the identities $U \to U$.

For a set R, let R^* and R^+ denote the sets of finite-length strings and nonnull finite-length strings over R, respectively.

2.1 Monads

A *monad* on a category C is a triple (T, μ, η), where T is an endofunctor on C and $\mu : T^2 \to T$ and $\eta : I \to T$ are natural transformations such that $\mu \circ \mu T = \mu \circ T\mu$ and $\mu \circ \eta T = \mu \circ T\eta = \iota$; that is, the diagrams

$$\begin{tikzcd} T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\ T^2 \arrow[r, "\mu"'] & T \end{tikzcd} \qquad \begin{tikzcd} T \arrow[r, "T\eta"] \arrow[d, "\eta T"'] \arrow[dr, "\iota"] & T^2 \arrow[d, "\mu"] \\ T^2 \arrow[r, "\mu"'] & T \end{tikzcd} \tag{2.6}$$

commute. A typical example is the *powerset monad* (P, μ^P, η^P) on Set, where PA is the powerset of A, $\eta^P_A(x) = \{x\}$, and $\mu^P_A(\mathcal{C}) = \bigcup \mathcal{C}$.

3 Natural Transformations as Rewrite Rules

Let $\mathcal{F}$ be a set of endofunctors $\mathsf{C} \to \mathsf{C}$. Every functor in the submonoid of $\mathsf{Cat}(\mathsf{C}, \mathsf{C})$ generated by $\mathcal{F}$ has a representation as a string in $\mathcal{F}^*$ under the canonical interpretation in which string concatenation in $\mathcal{F}^*$ is interpreted as composition of functors.

If $X, Y \in \mathcal{F}^*$ and $\varphi : X \to Y$ is a natural transformation, we can view φ as a rewrite rule $X \to Y$ on $\mathcal{F}^*$. Given the string UXV, we can rewrite the indicated occurrence of X to Y by applying the natural transformation $U\varphi V : UXV \to UYV$. Thus the strings U and V determine which occurrence of X to rewrite. An application of φ in the form of a scalar multiple $U\varphi V$ to the string UXV is called a *reduction*. The *redex* of the reduction $U\varphi V$ is the substring X of UXV. A sequence of reductions from Y to Z is called a *derivation*. We write $\pi : Y \xrightarrow{*} Z$ if π is such a derivation. A string is in *normal form* if no rules apply.

Let R be a set of natural transformations $X \to Y$ for various $X, Y \in \mathcal{F}^*$. Viewing these as rewrite rules and applying them to strings in $\mathcal{F}^*$, we can generate diagrams in the category of endofunctors on C and natural transformations. We say that the system R

- is *confluent* if for any two derivations $X \xrightarrow{*} U$ and $X \xrightarrow{*} V$, there is a word W and derivations $U \xrightarrow{*} W$ and $V \xrightarrow{*} W$ such that the resulting diagram commutes;
- is *locally confluent* if for any two single reductions $X \to U$ and $X \to V$, there is a word W and derivations $U \xrightarrow{*} W$ and $V \xrightarrow{*} W$ such that the resulting diagram commutes; and
- has the *diamond property* if for any two reductions $X \to U$ and $X \to V$, there is a word W and reductions $U \to W$ and $V \to W$ such that the resulting diagram commutes. This might be a pushout in the category of endofunctors on C and natural transformations, but not always.

Local confluence does not imply confluence, but the diamond property does. See [1] for a thorough treatment of these concepts.

Note, however, that our reductions have semantic content as well as syntactic. In our definitions of confluence, local confluence, and the diamond property, it is not enough that two derivations derive the same word; the resulting diagrams must also commute. We say that two derivations $X \xrightarrow{*} Y$ are *equivalent* if the composition of the natural transformations along the two paths are equal.

One of the defining properties for monads, namely the left-hand diagram of (2.6), says that μ as a reduction rule can be applied to the string T^3 in two ways to obtain T^2: one way as μT to the leftmost two occurrences of T (the left arrow of the diagram) and the other as $T\mu$ to the rightmost (the top arrow), in both cases giving T^2. By applying μ again to the two occurrences of T^2, we obtain a commutative diamond.

By (2.5), we can compose on the left and right with any strings in T^*:

$$\begin{tikzcd} T^{m+n+3} \arrow[r, "T^{m+1}\mu T^n"] \arrow[d, "T^m\mu T^{n+1}"'] & T^{m+n+2} \arrow[d, "T^m\mu T^n"] \\ T^{m+n+2} \arrow[r, "T^m\mu T^n"'] & T^{m+n+1} \end{tikzcd} \tag{3.7}$$

This says that any two reductions involving the rewrite rule μ with overlapping redexes, applied anywhere in a string of length at least three, can be completed to a commutative diamond.

For nonoverlapping redexes, the diamond property holds by virtue of the fact that μ is a natural transformation. More generally,

Lemma 3.1 *Any two applications of rewrite rules with disjoint redexes can be applied in either order, and the resulting diagram commutes. That is, any diamond of the form*

$$\begin{tikzcd} PQRST \arrow[r, "PQR\tau T"] \arrow[d, "P\sigma RST"'] & PQRYT \arrow[d, "P\sigma RYT"] \\ PXRST \arrow[r, "PXR\tau T"] & PXRYT \end{tikzcd} \tag{3.8}$$

commutes, where P, Q, R, S, T, X, Y are functors, $\sigma : Q \to X$, and $\tau : S \to Y$.

Proof. The diagram (3.8), localized to an object C, is a special case of the diagram (2.1) with the following substitutions: $RSTC$ for A, $RYTC$ for B, PQ for F, PX for G, $R\tau T_C$ for h, and $P\sigma$ for φ. □

The following lemma and its proof introduce our approach at a basic level.

Lemma 3.2 *Let $T : \mathsf{C} \to \mathsf{C}$ be an endofunctor and $\mu : T^2 \to T$ a natural transformation. The following are equivalent:*

(i) *T is a terminal object in the category $(T^+ ; \mu)$.*

(ii) *The left-hand diagram in* (2.6) *commutes.*

Proof. Suppose that (ii) holds. Combining Lemma 3.1 with the observation (3.7), we have that the rewrite system consisting of the single rule μ on strings T^n for $n \geq 1$ satisfies the diamond property and is therefore confluent. It follows that any diagram starting from T^n, $n \geq 1$, and ending with the normal form T commutes. Moreover, there exists a reduction sequence from any such T^n to T. Thus there is a unique morphism $T^n \to T$ for $n \geq 1$, so T is a terminal object.

Conversely, if T is a terminal object, then the left-hand diagram of (2.6) must commute, since there is a unique morphism $T^3 \to T$. □

As previously observed, it is important that in the definitions of confluence, local confluence, and the diamond property, it is not enough that two reduction sequences derive the same word; the resulting diagrams must also commute. There certainly exist distinct parallel arrows (for example, $\mu T, T\mu : T^3 \to T^2$) giving noncommutative diagrams.

Now we add the rewrite rule η to the mix. This rule can be used to introduce a new occurrence of T anywhere in the string. In the presence of η, T is no longer a normal form, and in fact normal forms no longer exist. Nevertheless, T is still a terminal object. Moreover, $T^0 = I$ can now be included as an object, since there is an arrow $I \to T$.

Theorem 3.3 *Let $T : \mathsf{C} \to \mathsf{C}$ be an endofunctor and $\mu : T^2 \to T$ and $\eta : I \to T$ natural transformations. The following are equivalent:*

(i) *T is a terminal object in the category $(T^* \,;\, \mu, \eta)$.*

(ii) *(T, μ, η) is a monad.*

Proof. Assume (ii). We wish to show that T is a terminal object. Suppose we have a rewrite system in which the rules can be classified as either *bad rules* or *good rules* (in our application, a rule is *bad* if it increases the length of the string, e.g. η). A reduction or derivation (sequence of reductions) is *good* if it uses only good rules. Suppose further that

1. every pair of good reductions $X \to U$ and $X \to V$ can be completed to a diamond using only good reductions $U \to W$ and $V \to W$;

2. every pair of reductions $X \to U$ and $X \to V$, good or bad, are confluent using only good derivations $U \xrightarrow{*} W$ and $V \xrightarrow{*} W$.

These conditions hold for the system with μ and η. We have already argued condition 1 for μ above. The right-hand diagram in (2.6) implies condition 2 by immediately inverting any application of η whenever it is applied to a nonnull string. For example, consider applications of η and μ to a substring T^2, where η is applied between the two occurrences of T. The two redexes thus overlap.

$$\begin{tikzcd}
T^2 \arrow[r, "\mu"] \arrow[d, "T\eta T"'] & T \\
T^3 &
\end{tikzcd}$$

The top arrow is good, but the left arrow is bad. However, using the right-hand diagram of (2.6), the diagram can be completed using only good arrows:

$$\begin{tikzcd}
T^2 \arrow[r, "\mu"] \arrow[d, "T\eta T"'] \arrow[dr, "\mu"] & T \arrow[dr, "\mu"] & \\
T^3 \arrow[r, "T\mu"'] & T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}$$

Now we argue that any system satisfying 1 and 2 is confluent. Let $X \xrightarrow{*} U$ and $X \xrightarrow{*} V$ be any two derivations involving good or bad rules. Starting from the apex X, move down the two derivations, adding good diamonds to the diagram in the case 1 and good confluent derivations in case 2. All new transitions added to the diagram are good. When done, there are no more exposed bad rules, and the diagram can be completed by filling in with good diamonds.

Thus any two derivations $T^n \xrightarrow{*} T$ are confluent via good derivations, which must be of the form $T \xrightarrow{*} T$. But the only good derivation $T \xrightarrow{*} T$ is the identity. It follows that any two derivations $T^n \xrightarrow{*} T$ are equivalent; in other words, T is a terminal object.

Conversely, if T is a terminal object, then all diagrams starting with any T^n and ending with T must commute, in particular those of (2.6), the defining conditions for monads. Therefore (T, μ, η) is a monad. □

4 Monad Composition

In this section we demonstrate the use of Theorem 3.3 in the verification of monad composition as presented by Eklund et al. [6].

Let (P, μ^P, η^P) and (T, μ^T, η^T) be monads on a category C connected by a *distributive law* (or *swapper* in the terminology of [6]), which is a natural transformation $\theta : TP \to PT$ satisfying the following properties:

$$\begin{tikzcd} TP^2 \arrow[r, "\theta P"] \arrow[d, "T\mu^P"'] & PTP \arrow[r, "P\theta"] & P^2T \arrow[d, "\mu^P T"] \\ TP \arrow[rr, "\theta"'] & & PT \end{tikzcd} \qquad \begin{tikzcd} T^2P \arrow[r, "T\theta"] \arrow[d, "\mu^T P"'] & TPT \arrow[r, "\theta T"] & PT^2 \arrow[d, "P\mu^T"] \\ TP \arrow[rr, "\theta"'] & & PT \end{tikzcd} \tag{4.9}$$

$$\begin{tikzcd} TP \arrow[dr, "\theta"] & \\ T \arrow[u, "T\eta^P"] \arrow[r, "\eta^P T"'] & PT \end{tikzcd} \qquad \begin{tikzcd} TP \arrow[dr, "\theta"'] & P \arrow[l, "\eta^T P"'] \arrow[d, "P\eta^T"] \\ & PT \end{tikzcd} \tag{4.10}$$

Distributive laws are discussed in depth in [3, §9.2]. A typical application is the construction of the *complex algebra* of an algebra, whose elements are sets of elements of the original algebra. Here P would be the powerset monad and T the term monad of some variety of algebras, and θ takes a term of sets $t(A_1, \ldots, A_n)$ and turns it into a set of terms $\{t(a_1, \ldots, a_n) \mid a_i \in A_i,\ 1 \leq i \leq n\}$. These constructions are discussed in [4, 8]. Another example would be the combination of the additive and multiplicative monoid structures in semirings. Here P would be the finite powerset monad and T the free monoid construction.

In light of the discussion of §3, it should be clear that the conditions (4.9) are nothing more than a way to establish local confluence in the case of overlapping

redexes between θ and μ^P (left-hand diagram) and between θ and μ^T (right-hand diagram).

The two monads P and T can be combined as follows. Define

$$\begin{aligned} \mu^{PT} &= \mu^P T \circ P^2 \mu^T \circ P\theta T : (PT)^2 \to PT \\ \eta^{PT} &= \eta^P T \circ \eta^T : I \to PT. \end{aligned}$$

Then $(PT, \mu^{PT}, \eta^{PT})$ is again a monad [3, 6]. To verify this using Theorem 3.3, it suffices to show that PT is a terminal object in the category $((PT)^* \,;\, \mu^{PT}, \eta^{PT})$. Most of the work is contained in the following lemma.

Lemma 4.1 *PT is terminal object in the category $(\{P,T\}^* \,;\, \mu^P, \mu^T, \eta^P, \eta^T, \theta)$.*

Proof. First we show that the rewrite system with rules $\mu^P, \mu^T, \eta^P, \eta^T, \theta$ on $\{P,T\}^*$ is confluent.

Recall that a rule is *bad* if it increases length, *good* otherwise. The good rules are μ^P, μ^T, and θ, and the bad rules are η^P and η^T.

Every pair of good reductions can be completed to a good confluent diagram. If the redexes do not overlap, this follows from Lemma 3.1. For redexes that overlap, all cases are covered by the left-hand diagram of (2.6) and the two diagrams of (4.9).

Given any derivation $\pi : X \xrightarrow{*} Y$, possibly containing bad reductions, produce a new derivation π' as follows:

1. extend the derivation to derive PT;
2. rearrange the resulting derivation $X \xrightarrow{*} Y \xrightarrow{*} PT$ to obtain an equivalent derivation $\pi' : X \xrightarrow{*} PT$ in which all the bad rules are applied after all the good rules.

Step 1 can be accomplished by first introducing an occurrence of P and/or T using η^P and η^T if necessary, then moving all occurrences of P to the left of all occurrences of T using θ, then collapsing the P's using μ^P and the T's using μ^T.

Step 2 can be done without increasing the length of the derivation. For any bad reduction followed immediately by a good reduction, if the symbol introduced by the bad reduction is not part of the redex of the good reduction, then the two reductions can be switched by Lemma 3.1.

Otherwise, the symbol introduced by the bad reduction is part of the redex of the good reduction. Up to symmetry, there are only two ways this can happen:

$$T \xrightarrow{T\eta^T} T^2 \xrightarrow{\mu^T} T \qquad\qquad T \xrightarrow{T\eta^P} TP \xrightarrow{\theta} PT$$

Replace the first subsequence by T and the second by $T \xrightarrow{\eta^P T} PT$ in the derivation. These modifications are justified by the right-hand diagram of (2.6) and the two diagrams of (4.10), respectively. We can continue this process until there are no more bad reductions occurring before good reductions in the derivation.

If X contains at least one occurrence each of P and T, then this must also be true of any string derived from X, since all rules preserve this property. But then π' can contain no bad reductions at all! If it did, then the last reduction would be bad. But it is impossible to derive PT from such a reduction, since it would have to come from a string of length one, and no such string can be derived from X.

By a similar argument, if $X \in P^+$, then π' contains exactly one bad reduction to introduce T, and it occurs last in the derivation. This last reduction must be of the form $P \xrightarrow{P\eta^T} PT$. Similarly, if $X \in T^+$, the last reduction of π' is of the form $T \xrightarrow{\eta^P T} PT$, and this is the only bad reduction in the derivation.

If $X = I$, a similar argument shows that π' must be one of the following two derivations:

$$I \xrightarrow{\eta^T} T \xrightarrow{\eta^P T} PT \qquad\qquad I \xrightarrow{\eta^P} P \xrightarrow{P\eta^T} PT. \tag{4.11}$$

Now suppose we are given derivations $\pi : X \xrightarrow{*} U$ and $\rho : X \xrightarrow{*} V$, possibly using both good and bad rules. To show confluence of π and ρ, it suffices to show that $\pi' : X \xrightarrow{*} PT$ and $\rho' : X \xrightarrow{*} PT$ are confluent.

As argued above, if X contains at least one occurrence each of P and T, then π' and ρ' are good. Thus we can complete them to a good commutative diagram by filling in with good commutative diamonds and diagrams (4.9). This process must terminate, since there is a fixed upper bound, quadratic in the length of X, on the length of any good derivation from X, since each good reduction strictly decreases the string in length or lexicographic order relative to $P < T$ [1, Lemma 2.7.2].

If $X \in P^+$, then as argued above, π' and ρ' are both of the form $X \xrightarrow{*} P \xrightarrow{P\eta^T} PT$, where the prefixes $X \xrightarrow{*} P$ contain no occurrence of T. By Theorem 3.3, since P is a terminal object in (P^*, μ^P, η^P), the two prefixes $X \xrightarrow{*} P$ are equivalent, therefore so are π' and ρ'.

When $X = I$, we need only observe that the two derivations (4.11) form a commutative diamond by Lemma 3.1.

Finally we show that PT is a terminal object. We have already argued that there is at least one derivation of PT from every $X \in \{P, T\}^*$, so there is at least one arrow $X \to PT$ in the category. To show that there is at most one, let $\pi, \rho : X \xrightarrow{*} PT$ be any two derivations. By confluence, we can complete to a pair of equivalent derivations $\pi', \rho' : X \xrightarrow{*} PT \xrightarrow{*} PT$. Rearranging the final portions $PT \xrightarrow{*} PT$ of these two derivations by Step 2 above, we obtain good derivations. But the only good derivation $PT \xrightarrow{*} PT$ is the identity, therefore π and ρ were already equivalent. □

Theorem 4.2 ([3, 6]) *The tuple $(PT, \mu^{PT}, \eta^{PT})$ is a monad.*

Proof. By Theorem 3.3, it suffices to show that the functor PT is a terminal object in the category $((PT)^*\,;\, \mu^{PT}, \eta^{PT})$. By Lemma 4.1, it is terminal in the category $(\{P, T\}^*\,;\, \mu^P, \mu^T, \eta^P, \eta^T, \theta)$, thus for any $n \geq 0$, there is exactly one

arrow $(PT)^n \to PT$ in that category. It follows that there is at most one arrow $(PT)^n \to PT$ in the subcategory $((PT)^* \,;\, \mu^{PT}, \eta^{PT})$. But there is also at least one, since we can derive PT from $(PT)^0 = I$ by η^{PT} and from $(PT)^n$ for $n \geq 1$ by $n-1$ applications of the rule μ^{PT}. $\square$

References

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

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

[3] Michael Barr and Charles Wells. *Toposes, Triples and Theories*. November 7, 2002. http://www.cwru.edu/artsci/math/wells/pub/ttt.html.

[4] Chris Brink. Power structures. *Algebra Universalis*, 30:177–216, 1993.

[5] Koen Claessen. Functional pearl: A poor man's concurrency monad. *J. Functional Programming*, 9(3):313–323, May 1999.

[6] P. Eklund, M.A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde. A graphical approach to monad compositions. *Electronic Notes in Theoretical Computer Science*, 40, 2002. http://www.elsevier.nl/locate/entcs/volume40.html.

[7] William L. Harrison and Samuel N. Kamin. Modular compilers based on monad transformers. In *International Conference on Computer Languages*, volume 22. IEEE, 1998.

[8] Peter Jipsen. A note on complex algebras of semigroups. In R. Berghammer, B. Möller, and G. Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science: Proc. 7th Int. Sem. Relational Methods in Computer Science and 2nd Int. Workshop Applications of Kleene Algebra*, volume 3051 of *Lecture Notes in Computer Science*, pages 171–177. Springer-Verlag, May 2003.

[9] M. P. Jones and L. Duponcheel. Composing monads. Technical Report YALEU/DCS/RR-1004, Yale University, 1993. http://www.cse.ogi.edu/mpj/pubs/composing.html.

[10] Eugenio Moggi. Notions of computation and monads. *Inf. and Comp.*, 93, 1991.

[11] Prakash Panangaden. The category of markov kernels. In Christel Baier, Michael Huth, Marta Kwiatkowska, and Mark Ryan, editors, *Electronic Notes in Theoretical Computer Science*, volume 22. Elsevier, 2000.

[12] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 154–165. ACM Press, 2002.

[13] D. Rydeheard and R. Burstall. A categorical unification algorithm. In *Proc. Category Theory and Computer Programming*, pages 493–505, 1986.

[14] Alfred Tarski. On the calculus of relations. *J. Symbolic Logic*, 6:73–89, 1941.

[15] Philip Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2:461–493, 1992.

[16] Philip Wadler. Monads for functional programming. In Johan Jeuring and Erik Meijer, editors, *Advanced Functional Programming: 1st Int. School on Advanced Functional Programming Techniques*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer-Verlag, 1995.