A Resumption Monad Transformer and its Applications in the Semantics of Concurrency[*]

Nikolaos S. Papaspyrou
(nickie@softlab.ntua.gr)

National Technical University of Athens,
Department of Electrical and Computer Engineering,
Software Engineering Laboratory,
Polytechnioupoli, 15780 Zografou, Athens, Greece.

Abstract

Resumptions are a valuable tool in the analysis and design of semantic models for concurrent programming languages, in which computations consist of sequences of atomic steps that may be interleaved. In this paper we consider a general notion of resumption, parameterized by the kind of computations that take place in the atomic steps. We define a monad transformer which, given a monad M that represents the atomic computations, constructs a monad $\mathsf{R}(M)$ for interleaved computations. Moreover, we use this monad transformer to define the denotational semantics of a simple imperative language supporting non-determinism and concurrency.

1 Introduction

Modern computer architectures and operating systems have made it practical to execute different parts of a program simultaneously. From the programmer's point of view, it is often not important whether the parts of a program are executed by different physical processors or by a single processor using a time-sharing strategy. New tools are needed to define the semantics of *concurrent programming languages*, which allow the parts of a program that execute simultaneously to interact with one another, typically using the same memory variables.

Resumptions have long been suggested as a model of interleaved computation in the semantics of concurrent programming languages. In brief, a resumption is either a computed value of some domain D or an atomic computation that results in a new resumption. An extensive treatment is offered in [dBak96] using the theory of complete metric spaces as the mathematical framework for domains. Many variations of resumption domains (also called *branching domains*) for specific instances of atomic computations are investigated there.

In this paper, we propose a structured generalization of this technique. We allow the atomic steps to perform any type of computation, represented by an arbitrary monad M. Thus, we define the *resumption monad transformer* R, which transforms monad M to a new monad $\mathsf{R}(M)$ representing interleaved computations. Domains constructed by $\mathsf{R}(M)$ satisfy the isomorphism

$$\mathsf{R}(M)(D) \quad \simeq \quad D + M(\mathsf{R}(M)(D))$$

which defines the essence of resumption domains. By introducing $\mathsf{R}(M)$ we obtain a general framework for reasoning about such domains. For example, the domain

$$\mathbf{D} \quad \simeq \quad \mathbf{U} + (\mathbf{S} \to \mathsf{P}(\mathbf{S} \times \mathbf{D}))$$

that is used in [dBak96] for defining the semantics of non-uniform parallelism (ignoring some complexities related to the use of complete metric spaces) is exactly the same as the domain $\mathsf{R}(\mathsf{D}(\mathsf{P}))(\mathbf{U})$ that we use in Section 4 for the same purpose.

The rest of the paper is structured as follows. Section 2 contains a brief introduction to monads as part of the mathematical background that is required for this paper. In Section 3 we define the resumption monad transformer R and in Section 4 we use it to present the denotational semantics of a simple imperative language featuring non-determinism and concurrency. We conclude with Section 5.

2 Mathematical background

In this section we define part of the mathematical background that is necessary for the rest of the paper. We introduce monads and monad transformers and discuss their

[*]This paper is based on work supported by the National Technical University of Athens, under the Programme for the Support of Basic Research “Archimedes”. Project title “SynSemAL: Syntax and Semantics of Artificial Languages”.

connection with computations and the semantics of programming languages. The reader is referred to [Papa01] for a more complete and informative introduction, and to the literature related to category theory [Pier91, Aspe91, Barr96] and domain theory [Scot82, Gunt90].

2.1 Monads and monad transformers

The notion of *monad*, also called triple, is not new in the context of category theory. In Computer Science, monads became very popular in the 1990s. The categorical properties of monads are discussed in most books on category theory, e.g. in [Barr96]. For a comprehensive introductions to monads and their use in denotational semantics the user is referred to [Mogg90]. A somehow different approach to the definition of monads is found in [Wadl92], which expresses the current practice of monads in functional programming. The two approaches are equivalent. In this paper, the categorical approach (presented here) is used for the definition of monads, since it is much more elegant, and the functional approach (presented in Section 2.2) is used for describing the semantics of programming languages.

Definition 2.1 *A* monad *on a category* C *is a triple* $\langle M, \eta, \mu \rangle$*, where* $M : \mathsf{C} \to \mathsf{C}$ *is an endofunctor,* $\eta : \mathit{Id}_{\mathsf{C}} \dot{\to} M$ *and* $\mu : M^2 \dot{\to} M$ *are natural transformations. For all objects* x *in* C*, the following diagrams must commute.*

$$\begin{tikzcd}
M(x) \arrow[r, "\eta_{M(x)}"] \arrow[dr, "\mathit{id}_{M(x)}"'] & M^2(x) \arrow[d, "\mu_x"] & M(x) \arrow[l, "M(\eta_x)"'] \arrow[dl, "\mathit{id}_{M(x)}"] \\
 & M(x) &
\end{tikzcd}$$

$$\begin{tikzcd}
M^3(x) \arrow[r, "M(\mu_x)"] \arrow[d, "\mu_{M(x)}"'] & M^2(x) \arrow[d, "\mu_x"] \\
M^2(x) \arrow[r, "\mu_x"] & M(x)
\end{tikzcd}$$

The transformation η *is called the* unit *of the monad, whereas the transformation* μ *is called the* multiplication *or* join.

The commutativity of these two diagrams is equivalent to the following three equations, commonly called the three *monad laws*:

$$\begin{array}{lcll}
\mu_x \circ \eta_{M(x)} & = & \mathit{id}_{M(x)} & \textit{(1st Monad Law)} \\
\mu_x \circ M(\eta_x) & = & \mathit{id}_{M(x)} & \textit{(2nd Monad Law)} \\
\mu_x \circ M(\mu_x) & = & \mu_x \circ \mu_{M(x)} & \textit{(3rd Monad Law)}
\end{array}$$

Definition 2.2 *If* C *is a category, a* monad transformer *on* C *is a mapping between monads on* C.[1]

2.2 Monads and computations

An alternative approach to the definition of monads has become very popular in the functional programming community. According to this, a monad on category Dom is defined as a triple $\langle M, \mathit{unit}_{\mathsf{M}}, *_{\mathsf{M}} \rangle$. In this triple M is a domain constructor, $\mathit{unit}_{\mathsf{M}} : D \to M(D)$ is a continuous function and $*_{\mathsf{M}} : M(A) \times (A \to M(B)) \to M(B)$ is a binary operation.

In the semantics of programming languages, domains constructed by monad M typically denote *computations*, e.g. the domain $M(D)$ denotes computations returning values of the domain D. The result of $\mathit{unit}_{\mathsf{M}}\ v$ is simply a computation returning the value v and the result of $m\ *_{\mathsf{M}}\ f$ is the combined computation of m, returning v, followed by computation $f(v)$. Monad transformers are useful to transform between different types of computations [Lian95, Lian98].

The following equations connect a monad $\langle M, \mathit{unit}_{\mathsf{M}}, *_{\mathsf{M}} \rangle$ defined using the functional approach with a monad $\langle M, \eta, \mu \rangle$ defined using the categorical approach.

$$\begin{array}{lcl}
\mathit{unit}_{\mathsf{M}} & = & \eta \\
m\ *_{\mathsf{M}}\ f & = & (\mu \circ M(f))\, m
\end{array}$$

$$\begin{array}{lcl}
\eta & = & \mathit{unit}_{\mathsf{M}} \\
\mu & = & \lambda m.\, m\ *_{\mathsf{M}}\ \mathit{id} \\
M(f) & = & \lambda m.\, m\ *_{\mathsf{M}}\ (\mathit{unit}_{\mathsf{M}} \circ f)
\end{array}$$

In the functional approach, the three monad laws can be formulated as follows.

$$\begin{array}{lcl}
m\ *_{\mathsf{M}}\ \mathit{unit}_{\mathsf{M}} & = & m \\
(\mathit{unit}_{\mathsf{M}}\ v)\ *_{\mathsf{M}}\ f & = & f\ v \\
m\ *_{\mathsf{M}}\ (\lambda v.\, (f\ v)\ *_{\mathsf{M}}\ g) & = & (m\ *_{\mathsf{M}}\ f)\ *_{\mathsf{M}}\ g
\end{array}$$

An interesting remark is that these three laws are enough to prove that the equivalent (M, η, μ), as defined above, is indeed a monad, i.e. that M is a functor (preserves function identities and composition) and η and μ are natural transformations.

In this setting, it is useful to define two special classes of monads, equipped with additional operations that are

[1]Many options for the definition of monad transformers have been suggested in literature. Given a category C, monads on C and *monad morphisms* (which have not been defined in this paper) form a category $\mathsf{Mon}(\mathsf{C})$. Monad transformers can be defined as mappings between objects in $\mathsf{Mon}(\mathsf{C})$, as endofunctors on $\mathsf{Mon}(\mathsf{C})$, as premonads on $\mathsf{Mon}(\mathsf{C})$ (i.e. endofunctors with a unit), and as monads on $\mathsf{Mon}(\mathsf{C})$. In this paper we have selected the first option.

useful for modeling the semantics of concurrency in programming languages.

Definition 2.3 *A* multi-monad *is a monad M with a binary operation $\|_{\mathsf{M}} : M(D) \times M(D) \to M(D)$, where D is a domain.*

Definition 2.4 *A* strong monad *is a monad M with a binary operation $\bowtie_{\mathsf{M}} : M(A) \times M(B) \to M(A \times B)$, where A and B are domains.*

The binary operation $\|$ of a multi-monad is used to express disjunction in computations. In other words, if M is a multimonad, D is a domain and $m_1, m_2 \in M(D)$ are two computations, the computation $m_1 \; \| \; m_2$ indicates a (possibly non-deterministic) option between m_1 and m_2. Moreover, the binary operation $\bowtie$ of a strong monad is used to express conjunction in computations. Let M be a strong monad, let A and B be domains. If $m_1 \in M(A)$ and $m_2 \in M(B)$ are two computations, the computation $m_1 \bowtie m_2$ indicates that both m_1 and m_2 will be performed and their results will be combined. The option here relates to the order, if any, in which the two computations will be performed.

3 Resumption monad transformer

The notion of execution *interleaving* is a well known one in the theory of concurrency. In this context, computations are considered to be sequences of *atomic steps* the nature of which depends on our notion of computation. In isolation, these atomic steps are performed one after another until the computation is complete. Given two computations A and B, an interleaved computation of A and B consists of an arbitrary merging of the atomic steps that constitute A and B. Interleaving easily extends to more than two computations. The atomic steps of any computation must still be executed in the right order, but this process can be interrupted by the execution of atomic steps belonging to other computations.

Our primary goal is to define a monad transformer R capable of modelling generic interleaved computations. In this way, if we are given a monad M which models the computations taking place at the atomic steps, we can obtain a monad $\mathsf{R}(M)$ which models interleaved computations of such atomic steps. One possible solution to this problem is to use the long suggested technique of *resumptions*, illustrated in [Schm86, dBak96] for specific instances of M.

Generalizing this technique, the domain $\mathsf{R}(M)(D)$ of resumptions must satisfy the following isomorphism:

$$\mathsf{R}(M)(D) \quad \simeq \quad D + M(\mathsf{R}(M)(D))$$

In this domain, atomic steps are arbitrary computations defined by M. The left part of the sum represents an already evaluated result, i.e. a computation that consists of zero atomic steps. The right part represents a computation that requires at least one atomic step. The result of this atomic step is a new element of the resumption domain.

We start by considering an arbitrary locally continuous monad M on Dom. The rest of the section is organized as follows. In Section 3.1 we define an endofunctor $\mathbf{R}_M : \mathsf{Dom} \to \mathsf{Dom}$. In Section 3.2 we define two natural transformations $\mathit{unit} : \mathit{Id} \dot{\to} \mathbf{R}_M$ and $\mathit{join} : \mathbf{R}_M^2 \dot{\to} \mathbf{R}_M$ and in Section 3.3 we prove that $(\mathbf{R}_M, \mathit{unit}, \mathit{join})$ satisfies the three monad laws. In this way we define the monad transformer R. Next, in Section 3.4 we prove that $\mathsf{R}(M)(D)$ satisfies the aforementioned isomorphism by constructing the two components h^e and h^p of the isomorphism. Finally, in Section 3.5 we define a few additional operations on domains constructed by $\mathsf{R}(M)$.

Most proofs of the results contained in this section have been omitted due to space restrictions. The reader is referred to [Papa01] for more details.

3.1 Functor $\mathbf{R}_M$

We start by defining for each domain D an endofunctor $\mathbf{F}_{M,D} : \mathsf{Dom} \to \mathsf{Dom}$, and some auxiliary functions. The domain $\mathsf{R}(M)(D)$ that we are trying to define is a fixed point of $\mathbf{F}_{M,D}$.

Definition 3.1 *Let D, A and B be domains and $f : A \to B$ a continuous function. We define the following mappings:*

$$\begin{aligned} \mathbf{F}_{M,D}(X) &= D + M(X) \\ \mathbf{F}_{M,D}(f) &= [\,\mathit{inl}, \mathit{inr} \circ M(f)\,] \end{aligned}$$

Theorem 3.1 $\mathbf{F}_{M,D} : \mathsf{Dom} \to \mathsf{Dom}$ *is a functor.*

Definition 3.2 *Let D be a domain. We define the pair of functions $\iota^e : \mathbf{O} \to \mathbf{F}_{M,D}(\mathbf{O})$ and $\iota^p : \mathbf{F}_{M,D}(\mathbf{O}) \to \mathbf{O}$ to be equal to $\bot$.*

We proceed by defining a mapping of objects and a mapping of functions, which will define the endofunctor $\mathbf{R}_M : \mathsf{Dom} \to \mathsf{Dom}$ at the end of this section.

Definition 3.3 *Let D be a domain. The domain $\mathbf{R}_M(D)$ is the set*

$$\begin{aligned} \mathbf{R}_M(D) \;=\; \{\, (x_n)_{n\in\omega} \;\mid\; & \forall n \in \omega.\; x_n \in \mathbf{F}^n_{M,D}(\mathbf{O}) \\ & \wedge\, x_n = \mathbf{F}^n_{M,D}(\iota^p)(x_{n+1}) \,\} \end{aligned}$$

with its elements ordered pointwise:

$$\begin{aligned} &(x_n)_{n\in\omega} \sqsubseteq_{\mathbf{R}_M(D)} (y_n)_{n\in\omega} \;\Leftrightarrow \\ &\qquad \forall n \in \omega.\; x_n \sqsubseteq_{\mathbf{F}^n_{M,D}(\mathbf{O})} y_n \end{aligned}$$

Definition 3.4 *Let D be a domain. For all $m, n \in \omega$, we define a function $f^D_{m,n} : \mathbf{F}^m_{M,D}(\mathbf{O}) \to \mathbf{F}^n_{M,D}(\mathbf{O})$ by:*

$$\begin{array}{lcll} f^D_{m,n} & = & id_{\mathbf{F}^n_{M,D}(\mathbf{O})} & \text{, if } m = n \\ f^D_{m,n+1} & = & f^D_{m,n} \circ \mathbf{F}^n_{M,D}(\iota^p) & \text{, if } m \leq n \\ f^D_{m+1,n} & = & \mathbf{F}^m_{M,D}(\iota^e) \circ f^D_{m,n} & \text{, if } m \geq n \end{array}$$

Definition 3.5 *Let D be a domain, $n \in \omega$, $z \in \mathbf{F}^n_{M,D}(\mathbf{O})$ and $(x_m)_{m\in\omega} \in \mathbf{R}_M(D)$. We define the pair of functions $\mu^e_n : \mathbf{F}^n_{M,D}(\mathbf{O}) \to \mathbf{R}_M(D)$ and $\mu^p_n : \mathbf{R}_M(D) \to \mathbf{F}^n_{M,D}(\mathbf{O})$ as follows:*

$$\begin{array}{lcl} \mu^e_n\ z & = & (f^D_{m,n}\ z)_{m\in\omega} \\ \mu^p_n\ (x_m)_{m\in\omega} & = & x_n \end{array}$$

Definition 3.6 *Let A and B be domains and $f : A \to B$ a continuous function. For all $n \in \omega$ we define a continuous function $\zeta^{A,B}_n\ f : \mathbf{F}^n_{M,A}(\mathbf{O}) \to \mathbf{F}^n_{M,B}(\mathbf{O})$ by:*

$$\begin{array}{lcl} \zeta^{A,B}_0\ f & = & \perp \\ \zeta^{A,B}_{n+1}\ f & = & [\,inl \circ f, inr \circ M(\zeta^{A,B}_n\ f)\,] \end{array}$$

Definition 3.7 *Let A and B be domains and $f : A \to B$ a continuous function. We define a continuous function $\mathbf{R}_M(f) : \mathbf{R}_M(A) \to \mathbf{R}_M(B)$ by:*

$$\mathbf{R}_M(f)\ (x_n)_{n\in\omega} \quad = \quad (\zeta^{A,B}_n\ f\ x_n)_{n\in\omega}$$

The central result of this section is Theorem 3.2 in which we prove that $\mathbf{R}_M$ is a functor. For doing so, we make use of the following lemmata.

Lemma 3.1 *For all $m, n \in \omega$, $\mu^p_m \circ \mu^e_n = f^D_{m,n}$.*

Lemma 3.2 *Let A be a domain. Then for all $n \in \omega$,*

$$\zeta^{A,A}_n\ id_A = id_{\mathbf{F}^n_{M,A}(\mathbf{O})}$$

Lemma 3.3 *Let A, B and C be domains, $f : A \to B$ and $g : B \to C$ continuous functions. Then for all $n \in \omega$,*

$$\zeta^{A,C}_n\ (g \circ f) = \zeta^{B,C}_n\ g \circ \zeta^{A,B}_n\ f$$

We can now proceed with the proof of Theorem 3.2.

Theorem 3.2 $\mathbf{R}_M$: Dom $\to$ Dom *is a functor.*

Proof We must prove that $\mathbf{R}_M$ preserves identities and the composition of continuous functions.

1. Let X be a domain and $(x_n)_{n\in\omega} \in \mathbf{R}_M(X)$.

$$\begin{array}{l} \quad \mathbf{R}_M(id_X)\ (x_n)_{n\in\omega} \\ = \langle\ \textit{Definition of } \mathbf{R}_M\ \rangle \\ \quad (\zeta^{X,X}_n\ id_X\ x_n)_{n\in\omega} \\ = \langle\ \textit{Lemma 3.2}\ \rangle \\ \quad (id_{\mathbf{F}^n_{M,X}(\mathbf{O})}\ x_n)_{n\in\omega} \\ = \langle\ \textit{Identity function}\ \rangle \\ \quad (x_n)_{n\in\omega} \\ = \langle\ \textit{Identity function}\ \rangle \\ \quad id_{\mathbf{R}_M(X)}\ (x_n)_{n\in\omega} \end{array}$$

2. Let A and B be domains, $f : A \to B$ and $g : B \to C$ continuous functions and $(x_n)_{n\in\omega} \in \mathbf{R}_M(X)$.

$$\begin{array}{l} \quad \mathbf{R}_M(g \circ f)\ (x_n)_{n\in\omega} \\ = \langle\ \textit{Definition of } \mathbf{R}_M\ \rangle \\ \quad (\zeta^{A,C}_n\ (g \circ f)\ x_n)_{n\in\omega} \\ = \langle\ \textit{Lemma 3.3}\ \rangle \\ \quad ((\zeta^{B,C}_n\ g \circ \zeta^{A,B}_n\ f)\ x_n)_{n\in\omega} \\ = \langle\ \textit{Composition}\ \rangle \\ \quad (\zeta^{B,C}_n\ g\ (\zeta^{A,B}_n\ f\ x_n))_{n\in\omega} \\ = \langle\ \textit{Definition of } \mathbf{R}_M\ \rangle \\ \quad \mathbf{R}_M(g)\ (\zeta^{A,B}_n\ f\ x_n)_{n\in\omega} \\ = \langle\ \textit{Definition of } \mathbf{R}_M\ \rangle \\ \quad \mathbf{R}_M(g)\ (\mathbf{R}_M(f)\ (x_n)_{n\in\omega}) \\ = \langle\ \textit{Composition}\ \rangle \\ \quad (\mathbf{R}_M(g) \circ \mathbf{R}_M(f))\ (x_n)_{n\in\omega} \qquad \Box \end{array}$$

3.2 Unit and join

Having defined $\mathbf{R}_M$ as a functor, we now define the two monad operations $unit$ and $join$. For each one, we prove that it is a natural transformation.

Definition 3.8 *Let D be a domain. For all $n \in \omega$ we define a continuous function $\eta^D_n : D \to \mathbf{F}^n_{M,D}(\mathbf{O})$ by:*

$$\begin{array}{lcl} \eta^D_0 & = & \perp \\ \eta^D_{n+1} & = & inl \end{array}$$

Definition 3.9 *Let D be a domain and $d \in D$. We define the function $unit_D : D \to \mathbf{R}_M(D)$ by:*

$$unit_D\ d \quad = \quad (\eta^D_n\ d)_{n\in\omega}$$

Lemma 3.4 *Let A and B be domains, $f : A \to B$ a continuous function. Then, for all $n \in \omega$,*

$$\zeta^{A,B}_n\ f \circ \eta^A_n = \eta^B_n \circ f$$

Theorem 3.3 $unit : Id \dot{\to} \mathbf{R}_M$ *is a natural transformation.*

Proof Let A and B be domains and $f : A \to B$ a continuous function. We must show that $unit_B \circ f = \mathbf{R}_M(f) \circ unit_A$. Let $a \in A$.

$$\begin{array}{l} \quad unit_B\ (f\ a) \\ = \langle\ \textit{Definition of } unit\ \rangle \\ \quad (\eta^D_n\ (f\ a))_{n\in\omega} \\ = \langle\ \textit{Composition}\ \rangle \\ \quad ((\eta^D_n \circ f)\ a)_{n\in\omega} \\ = \langle\ \textit{Lemma 3.4}\ \rangle \\ \quad ((\zeta^{A,B}_n \circ \eta^A_n)\ a)_{n\in\omega} \\ = \langle\ \textit{Composition}\ \rangle \\ \quad (\zeta^{A,B}_n\ (\eta^A_n\ a))_{n\in\omega} \\ = \langle\ \textit{Definition of } \mathbf{R}_M\ \rangle \end{array}$$

$\mathbf{R}_M(f)\ (\eta_n^A\ a)_{n\in\omega}$
= ⟨ *Definition of* unit ⟩
$\mathbf{R}_M(f)\ (\mathit{unit}_A\ a)$ □

Definition 3.10 *Let D be a domain. For all $n \in \omega$ we define a continuous function $\xi_n^D : \mathbf{F}^n_{M,\mathbf{R}_M(D)}(\mathbf{O}) \to \mathbf{F}^n_{M,D}(\mathbf{O})$ by:*

$$\begin{array}{lcl}\xi_0^D & = & \bot \\ \xi_{n+1}^D f & = & [\,\mu^p_{n+1}, \mathit{inr} \circ M(\xi_n^D)\,]\end{array}$$

Definition 3.11 *Let D be a domain and $(x_n)_{n\in\omega} \in \mathbf{R}^2_M(D)$. We define the function $\mathit{join}_D : \mathbf{R}^2_M(D) \to \mathbf{R}_M(D)$ by:*

$$\mathit{join}_D\ (x_n)_{n\in\omega} \quad = \quad (\xi_n^D\ x_n)_{n\in\omega}$$

Lemma 3.5 *Let A and B be domains, $f : A \to B$ a continuous function. Then for all $n \in \omega$,*

$$\xi_n^B \circ \zeta_n^{\mathbf{R}_M(A),\mathbf{R}_M(B)}\ (\mathbf{R}_M(f)) = \zeta_n^{A,B}\ f \circ \xi_n^A$$

Theorem 3.4 *$\mathit{join} : \mathbf{R}^2_M \dot\to \mathbf{R}_M$ is a natural transformation.*

Proof Let A and B be domains and $f : A \to B$ a continuous function. We must show that $\mathit{join}_B \circ \mathbf{R}_M(\mathbf{R}_M(f)) = \mathbf{R}_M(f) \circ \mathit{join}_A$. Let $(x_n)_{n\in\omega} \in \mathbf{R}^2_M(A)$.

$\mathit{join}_B\ (\mathbf{R}_M(\mathbf{R}_M(f))\ (x_n)_{n\in\omega})$
= ⟨ *Definition of* $\mathbf{R}_M$ ⟩
$\mathit{join}_B\ (\zeta_n^{\mathbf{R}_M(A),\mathbf{R}_M(B)}\ (\mathbf{R}_M(f))\ x_n)_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$(\xi_n^B\ (\zeta_n^{\mathbf{R}_M(A),\mathbf{R}_M(B)}\ (\mathbf{R}_M(f))\ x_n))_{n\in\omega}$
= ⟨ *Composition* ⟩
$((\xi_n^B \circ \zeta_n^{\mathbf{R}_M(A),\mathbf{R}_M(B)}\ (\mathbf{R}_M(f)))\ x_n)_{n\in\omega}$
= ⟨ *Lemma 3.5* ⟩
$((\zeta_n^{A,B}\ f \circ \xi_n^A)\ x_n)_{n\in\omega}$
= ⟨ *Composition* ⟩
$(\zeta_n^{A,B}\ f\ (\xi_n^A\ x_n))_{n\in\omega}$
= ⟨ *Definition of* $\mathbf{R}_M(f)$ ⟩
$\mathbf{R}_M(f)\ (\xi_n^A\ x_n)_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$\mathbf{R}_M(f)\ (\mathit{join}_A\ (x_n)_{n\in\omega})$ □

3.3 Monad $\mathsf{R}(M)$

In this section we prove that functor $\mathbf{R}_M$ together with the natural transformations unit and join defines a monad. The three theorems of this section verify the three monad laws. The following lemmata are necessary for proving the monad laws. Let D be a domain.

Lemma 3.6 *For all $n \in \omega$, $\xi_n^D \circ \eta_n^{\mathbf{R}_M(D)} = \mu_n^p$.*

Lemma 3.7 *For all $n \in \omega$,*

$$\xi_n^D \circ \zeta_n^{D,\mathbf{R}_M(D)}\ \mathit{unit}_D = \mathit{id}_{\mathbf{F}^n_{M,D}(\mathbf{O})}$$

Lemma 3.8 *For all $d \in \omega$,*

$$\xi_n^D \circ \zeta_n^{\mathbf{R}^2_M(D),\mathbf{R}_M(D)}\ \mathit{join}_D = \xi_n^D \circ \xi_n^{\mathbf{R}_M(D)}$$

We can now proceed by proving the three monad laws.

Theorem 3.5 (1st Monad Law)

$$\mathit{join}_D \circ \mathit{unit}_{\mathbf{R}_M(D)} = \mathit{id}_{\mathbf{R}_M(D)}$$

Proof Let $(x_n)_{n\in\omega} \in \mathbf{R}_M(D)$. Then

$\mathit{join}_D\ (\mathit{unit}_{\mathbf{R}_M(D)}\ (x_n)_{n\in\omega})$
= ⟨ *Definition of* unit ⟩
$\mathit{join}_D\ (\eta_n^{\mathbf{R}_M(D)}\ (x_m)_{m\in\omega})_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$(\xi_n^D\ (\eta_n^{\mathbf{R}_M(D)}\ (x_m)_{m\in\omega}))_{n\in\omega}$
= ⟨ *Composition* ⟩
$((\xi_n^D \circ \eta_n^{\mathbf{R}_M(D)})\ (x_m)_{m\in\omega})_{n\in\omega}$
= ⟨ *Lemma 3.6* ⟩
$(\mu_n^p\ (x_m)_{m\in\omega})_{n\in\omega}$
= ⟨ *Definition of* μ_n^p ⟩
$(x_n)_{n\in\omega}$ □

Theorem 3.6 (2nd Monad Law)

$$\mathit{join}_D \circ \mathbf{R}_M(\mathit{unit}_D) = \mathit{id}_{\mathbf{R}_M(D)}$$

Proof Let $(x_n)_{n\in\omega} \in \mathbf{R}_M(D)$. Then

$\mathit{join}_D\ (\mathbf{R}_M(\mathit{unit}_D)\ (x_n)_{n\in\omega})$
= ⟨ *Definition of* $\mathbf{R}_M$ ⟩
$\mathit{join}_D\ (\zeta_n^{D,\mathbf{R}_M(D)}\ \mathit{unit}_D\ x_n)_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$(\xi_n^D\ (\zeta_n^{D,\mathbf{R}_M(D)}\ \mathit{unit}_D\ x_n))_{n\in\omega}$
= ⟨ *Composition* ⟩
$((\xi_n^D \circ \zeta_n^{D,\mathbf{R}_M(D)}\ \mathit{unit}_D)\ x_n)_{n\in\omega}$
= ⟨ *Lemma 3.7* ⟩
$(\mathit{id}_{\mathbf{F}^n_{M,D}(\mathbf{O})}\ x_n)_{n\in\omega}$
= ⟨ *Identity* ⟩
$(x_n)_{n\in\omega}$ □

Theorem 3.7 (3rd Monad Law)

$$\mathit{join}_D \circ \mathbf{R}_M(\mathit{join}_D) = \mathit{join}_D \circ \mathit{join}_{\mathbf{R}_M(D)}$$

Proof Let $(x_n)_{n\in\omega} \in \mathbf{R}^3_M(D)$. Then

$\mathit{join}_D\ (\mathbf{R}_M(\mathit{join}_D)\ (x_n)_{n\in\omega})$
= ⟨ *Definition of* $\mathbf{R}_M$ ⟩
$\mathit{join}_D\ (\zeta_n^{\mathbf{R}^2_M(D),\mathbf{R}_M(D)}\ \mathit{join}_D\ x_n)_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$(\xi_n^D\ (\zeta_n^{\mathbf{R}^2_M(D),\mathbf{R}_M(D)}\ \mathit{join}_D\ x_n))_{n\in\omega}$
= ⟨ *Composition* ⟩
$((\xi_n^D \circ \zeta_n^{\mathbf{R}^2_M(D),\mathbf{R}_M(D)}\ \mathit{join}_D)\ x_n)_{n\in\omega}$

= ⟨ *Lemma 3.8* ⟩
$((\xi_n^D \circ \xi_n^{\mathbf{R}_M(D)})\ x_n)_{n\in\omega}$
= ⟨ *Composition* ⟩
$(\xi_n^D\ (\xi_n^{\mathbf{R}_M(D)}\ x_n))_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$\mathit{join}_D\ (\xi_n^{\mathbf{R}_M(D)}\ x_n)_{n\in\omega}$
= ⟨ *Definition of join* ⟩
$\mathit{join}_D\ (\mathit{join}_{\mathbf{R}_M(D)}\ (x_n)_{n\in\omega})$ □

Having established that $\mathbf{R}_M$ satisfies the three monad laws, we can now conclude the definition of the resumption monad transformer R.

Definition 3.12 *The* resumption monad transformer R *is defined by the mapping* $\mathsf{R}(M) = \mathbf{R}_M$.

3.4 Isomorphism

Let D be a domain. In this section, we define the pair of functions h^e and h^p that establish the isomorphism between domains $\mathbf{R}_M(D)$ and $D + M(\mathbf{R}_M(D))$. Using these functions, it is possible to define an operation in one of these two domains and obtain the corresponding operation on the other domain by applying h^e and h^p appropriately.

The definition of the embedding function h^e is straightforward.

Definition 3.13 *For all* $n \in \omega$ *we define a function* $\theta_n^D : \mathbf{F}_{M,D}(\mathbf{R}_M(D)) \to \mathbf{F}_{M,D}^n(\mathbf{O})$ *by:*

$$\begin{array}{lcl} \theta_0^D & = & \bot \\ \theta_{n+1}^D & = & [\,\mathit{inl}, \mathit{inr} \circ M(\mu_n^p)\,] \end{array}$$

Definition 3.14 *Let* $z \in \mathbf{F}_{M,D}(\mathbf{R}_M(D))$*. We define the function* $h^e : \mathbf{F}_{M,D}(\mathbf{R}_M(D)) \to \mathbf{R}_M(D)$ *by:*

$$h^e\ z \quad = \quad (\theta_n^D\ z)_{n\in\omega}$$

On the other hand, the definition of the projection function h^p is more complicated. It first requires the definition of an additional domain $\mathbf{Q}_M(D)$. Furthermore, it requires the proof of Lemma 3.9, which states that elements of $\mathbf{R}_M(D)$ come in three distinct forms. This lemma is crucial in the definition of h^p and in the proofs of several theorems that follow.

Definition 3.15 *The domain* $\mathbf{Q}_M(D)$ *is the set*

$$\begin{array}{rcl} \mathbf{Q}_M(D) & = & \{\,(z_n)_{n\in\omega} \mid \forall n \in \omega.\ z_n \in M(\mathbf{F}_{M,D}^n(\mathbf{O})) \\ & & \wedge\ z_n = M(\mathbf{F}_{M,D}^n(\iota^p))(z_{n+1})\,\} \end{array}$$

with its elements ordered pointwise:

$$\begin{array}{l} (z_n)_{n\in\omega} \sqsubseteq_{\mathbf{Q}_M(D)} (w_n)_{n\in\omega} \ \Leftrightarrow \\ \quad \forall n \in \omega.\ z_n \sqsubseteq_{M(\mathbf{F}_{M,D}^n(\mathbf{O}))} w_n \end{array}$$

Definition 3.16 *Let* $(z_m)_{m\in\omega} \in \mathbf{Q}_M(D)$*. For all* $n \in \omega$*, we define a function* $\sigma_n^D : \mathbf{Q}_M(D) \to \mathbf{F}_{M,D}^n(\mathbf{O})$ *by:*

$$\begin{array}{lcl} \sigma_0^D\ (z_m)_{m\in\omega} & = & \bot \\ \sigma_{n+1}^D\ (z_m)_{m\in\omega} & = & \mathit{inr}\ z_n \end{array}$$

Lemma 3.9 *Let* $(x_n)_{n\in\omega} \in \mathbf{R}_M(D)$*. Then exactly one of the following is true:*

1. *For all* $n \in \omega$, $x_n = \bot$.
2. *There exists a* $t \in D$ *such that for all* $n \in \omega$, $x_n = \eta_n^D\ t$.
3. *There exists a* $(z_m)_{m\in\omega} \in \mathbf{Q}_M(D)$ *such that for all* $n \in \omega$, $x_n = \sigma_n^D\ (z_m)_{m\in\omega}$.

Definition 3.17 *We define the function* $h^p : \mathbf{R}_M(D) \to \mathbf{F}_{M,D}(\mathbf{R}_M(D))$ *by case analysis on its argument* $(x_n)_{n\in\omega}$ *based on Lemma 3.9:*

1. *If for all* $n \in \omega$, $x_n = \bot$*, then*
   $h^p\ (x_n)_{n\in\omega} \ = \ \bot$
2. *If there exists a* $t \in D$ *such that for all* $n \in \omega$, $x_n = \eta_n^D\ t$*, then*
   $h^p\ (x_n)_{n\in\omega} \ = \ \mathit{inl}\ t$
3. *If there exists a* $(z_m)_{m\in\omega} \in \mathbf{Q}_M(D)$ *such that for all* $n \in \omega$, $x_n = \sigma_n^D\ (z_m)_{m\in\omega}$*, then*
   $$h^p\ (x_n)_{n\in\omega} \ = \ \mathit{inr}\ \left(\bigsqcup_{n\in\omega} M(\mu_n^e)\ z_n\right)$$

In order to ensure that the least upper bound in the third case of the previous definition exists, we prove Lemma 3.10 which states that $M(\mu_n^e)\ z_n$ form an ω-chain.

Lemma 3.10 *Let* $(z_n)_{n\in\omega} \in \mathbf{Q}_M(D)$*. For all* $n \in \omega$*,*
$M(\mu_n^e)\ z_n \sqsubseteq M(\mu_{n+1}^e)\ z_{n+1}$

The following lemmata are necessary for proving the central theorems of this section.

Lemma 3.11 *For all* $t \in D$*, for all* $n \in \omega$*,*
$\theta_n^D\ (\mathit{inl}\ t) = \eta_n^D\ t$

Lemma 3.12 *For all* $w \in M(\mathbf{R}_M(D))$*, for all* $n \in \omega$*,*
$\theta_n^D\ (\mathit{inr}\ w) = \sigma_n^D\ (M(\mu_m^p)\ w)_{m\in\omega}$

Lemma 3.13 *For all* $w \in M(\mathbf{R}_M(D))$*,*
$$\bigsqcup_{n\in\omega} M(\mu_n^e \circ \mu_n^p)\ w = w$$

Lemma 3.14 *Let* $(z_m)_{m\in\omega} \in \mathbf{Q}_M(D)$*. For all* $m \in \omega$*,*
$$\bigsqcup_{n\in\omega} M(f_{m,n}^D)\ z_n = z_m$$

And now we can proceed to Theorem 3.8 and Theorem 3.9, which conclude that the two functions h^e and h^p define indeed an isomorphism between the domains $\mathbf{R}_M(D)$ and $D + M(\mathbf{R}_M(D))$.

Theorem 3.8 $h^p \circ h^e = id_{\mathbf{F}_{M,D}(\mathbf{R}_M(D))}$

Proof Let $z \in \mathbf{F}_{M,D}(\mathbf{R}_M(D)) = D + M(\mathbf{R}_M(D))$. By case analysis on z.

1. Case $z = \bot$. Then both sides are equal to $\bot$ according to the definitions of h^e and h^p

2. Case $z = inl\ t$ for some $t \in D$. Then

$$\begin{aligned}
& h^p\ (h^e\ (inl\ t)) \\
=\ & \langle \textit{ Definition of } h^e \rangle \\
& h^p\ (\theta_n^D\ (inl\ t))_{n\in\omega} \\
=\ & \langle \textit{ Lemma 3.11 } \rangle \\
& h^p\ (\eta_n^D\ t)_{n\in\omega} \\
=\ & \langle \textit{ Definition of } h^p \rangle \\
& inl\ t
\end{aligned}$$

3. Case $z = inr\ w$ for some $w \in M(\mathbf{R}_M(D))$. Then

$$\begin{aligned}
& h^p\ (h^e\ (inr\ w)) \\
=\ & \langle \textit{ Definition of } h^e \rangle \\
& h^p\ (\theta_n^D\ (inr\ w))_{n\in\omega} \\
=\ & \langle \textit{ Lemma 3.12 } \rangle \\
& h^p\ (\sigma_n^D\ (M(\mu_m^p)\ w)_{m\in\omega})_{n\in\omega} \\
=\ & \langle \textit{ Definition of } h^p \rangle \\
& inr\ \left(\bigsqcup_{n\in\omega} M(\mu_n^e)\ (M(\mu_n^p)\ w)\right) \\
=\ & \langle \textit{ Composition, } M \textit{ is a functor } \rangle \\
& inr\ \left(\bigsqcup_{n\in\omega} M(\mu_n^e \circ \mu_n^p)\ w\right) \\
=\ & \langle \textit{ Lemma 3.13 } \rangle \\
& inr\ w
\end{aligned}$$

□

Theorem 3.9 $h^e \circ h^p = id_{\mathbf{R}_M(D)}$

Proof Let $(x_n)_{n\in\omega} \in \mathbf{R}_M(D)$. By case analysis on $(x_n)_{n\in\omega}$ based on Lemma 3.9:

1. If for all $n \in \omega$, $x_n = \bot$, then

$$\begin{aligned}
& h^e\ (h^p\ (\bot)_{n\in\omega}) \\
=\ & \langle \textit{ Definition of } h^p \rangle \\
& h^e\ \bot \\
=\ & \langle \textit{ Definition of } h^e \rangle \\
& (\theta_n^D\ \bot)_{n\in\omega} \\
=\ & \langle \textit{ Definition of } \theta \rangle \\
& (\bot)_{n\in\omega}
\end{aligned}$$

2. If there exists a $t \in D$ such that for all $n \in \omega$, $x_n = \eta_n^D\ t$, then

$$\begin{aligned}
& h^e\ (h^p\ (\eta_n^D\ t)_{n\in\omega}) \\
=\ & \langle \textit{ Definition of } h^p \rangle \\
& h^e\ (inl\ t) \\
=\ & \langle \textit{ Definition of } h^e \rangle \\
& (\theta_n^D\ (inl\ t))_{n\in\omega} \\
=\ & \langle \textit{ Lemma 3.11 } \rangle \\
& (\eta_n^D\ t)_{n\in\omega}
\end{aligned}$$

3. If there exists a $(z_m)_{m\in\omega} \in \mathbf{Q}_M(D)$ such that for all $n \in \omega$, $x_n = \sigma_n^D\ (z_m)_{m\in\omega}$, then

$$\begin{aligned}
& h^e\ (h^p\ (\sigma_n^D\ (z_m)_{m\in\omega})_{n\in\omega}) \\
=\ & \langle \textit{ Definition of } h^p \rangle \\
& h^e\ \left(inr\ \left(\bigsqcup_{n\in\omega} M(\mu_n^e)\ z_n\right)\right) \\
=\ & \langle \textit{ Definition of } h^e \rangle \\
& \left(\theta_n^D\ \left(inr\ \left(\bigsqcup_{n\in\omega} M(\mu_n^e)\ z_n\right)\right)\right)_{n\in\omega} \\
=\ & \langle \textit{ Lemma 3.12 } \rangle \\
& \left(\sigma_n^D\ \left(M(\mu_m^p)\ \left(\bigsqcup_{n'\in\omega} M(\mu_{n'}^e)\ z_{n'}\right)\right)_{m\in\omega}\right)_{n\in\omega} \\
=\ & \langle\ M(\mu_m^p) \textit{ is continuous } \rangle \\
& \left(\sigma_n^D\ \left(\bigsqcup_{n'\in\omega} M(\mu_m^p)\ (M(\mu_{n'}^e)\ z_{n'})\right)_{m\in\omega}\right)_{n\in\omega} \\
=\ & \langle \textit{ Composition } \rangle \\
& \left(\sigma_n^D\ \left(\bigsqcup_{n'\in\omega} (M(\mu_m^p) \circ M(\mu_{n'}^e))\ z_{n'}\right)_{m\in\omega}\right)_{n\in\omega} \\
=\ & \langle\ M \textit{ is functor } \rangle \\
& \left(\sigma_n^D\ \left(\bigsqcup_{n'\in\omega} M(\mu_m^p \circ \mu_{n'}^e)\ z_{n'}\right)_{m\in\omega}\right)_{n\in\omega} \\
=\ & \langle \textit{ Lemma 3.1 } \rangle \\
& \left(\sigma_n^D\ \left(\bigsqcup_{n'\in\omega} M(f_{m,n'}^D)\ z_{n'}\right)_{m\in\omega}\right)_{n\in\omega} \\
=\ & \langle \textit{ Lemma 3.14 } \rangle \\
& (\sigma_n^D\ (z_m)_{m\in\omega})_{n\in\omega}
\end{aligned}$$

□

3.5 Additional operations

In this section we define two functions, *step* and *run*, which convert a non interleaved computation of type $M(A)$ to an interleaved computation of type $\mathsf{R}(M)(A)$ and vice-versa. The names of these functions indicate their behaviour. The first function converts a whole computation to a single atomic *step* in an interleaved computation. The second function *runs* the whole sequence of atomic steps of an interleaved computation without allowing other computations to intervene.

In the rest of this section, we assume that (M, η, μ) is a monad and that D is a domain.

Definition 3.18 $\mathit{step}_D : M(D) \to \mathsf{R}(M)(D)$ *is the continuous function defined by:*

$$\mathit{step}_D \;=\; h^e \circ \mathit{inr} \circ M(h^e \circ \mathit{inl})$$

Definition 3.19 $\mathit{run}_D : \mathsf{R}(M)(D) \to M(D)$ *is the continuous function defined by:*

$$\mathit{run}_D \;=\; \mathit{fix}\,(\lambda\, g.\, [\,\eta_D, \mu_D \circ M(g)\,] \circ h^p)$$

The following theorem states a property of run and step. The reverse composition does not yield identity, since it forces an interleaved computation to be executed in one atomic step (it will be used in Section 4 for defining the semantics of $\langle s \rangle$).

Theorem 3.10 $\mathit{run}_D \circ \mathit{step}_D = \mathit{id}_{M(D)}$

Proof

$\mathit{run}_D \circ \mathit{step}_D$
= ⟨ *Unfolding fix in the definition of* run_D ⟩
$[\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ h^p \circ \mathit{step}_D$
= ⟨ *Definition of* step_D ⟩
$[\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ h^p \circ h^e \circ \mathit{inr} \circ M(h^e \circ \mathit{inl})$
= ⟨ *Theorem 3.8* ⟩
$[\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ \mathit{id}_{\mathbf{F}_{M,D}(\mathbf{R}_M(D))} \circ$
$\quad \mathit{inr} \circ M(h^e \circ \mathit{inl})$
= ⟨ *Composition with identity* ⟩
$[\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ \mathit{inr} \circ M(h^e \circ \mathit{inl})$
= ⟨ *Definition of selection* ⟩
$\mu_D \circ M(\mathit{run}_D) \circ M(h^e \circ \mathit{inl})$
= ⟨ M *is a functor* ⟩
$\mu_D \circ M(\mathit{run}_D \circ h^e \circ \mathit{inl})$
= ⟨ *Unfolding fix in the definition of* run_D ⟩
$\mu_D \circ M([\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ h^p \circ h^e \circ \mathit{inl})$
= ⟨ *Theorem 3.8* ⟩
$\mu_D \circ M([\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ \mathit{id}_{\mathbf{F}_{M,D}(\mathbf{R}_M(D))} \circ \mathit{inl})$
= ⟨ *Composition with identity* ⟩
$\mu_D \circ M([\,\eta_D, \mu_D \circ M(\mathit{run}_D)\,] \circ \mathit{inl})$
= ⟨ *Definition of selection* ⟩
$\mu_D \circ M(\eta_D)$
= ⟨ M *is a monad, 2nd Monad Law* ⟩
$\mathit{id}_{M(D)}$ □

The following definition is useful in the rest of this section, where we establish that $\mathsf{R}(M)(D)$ can be defined as a multi-monad and a strong-monad. These two properties of $\mathsf{R}(M)(D)$ will also be used in Section 4.

Definition 3.20 $\mathit{prom}_D : \mathsf{R}(M)(D) \to M(\mathsf{R}(M)(D))$ *is the continuous function defined by:*

$$\mathit{prom}_D \;=\; [\,\eta_{\mathbf{R}_M(D)} \circ \mathit{inl}, \mathit{id}_{M(\mathbf{R}_M(D))}\,] \circ h^p$$

Let us now assume that M is a multi-monad and that $\|_{\mathsf{M}}$ is a *non-deterministic option* operator for computations represented by monad M. It is easy to extend this behaviour to the monad $\mathsf{R}(M)$.

Definition 3.21 *Let M be a multi-monad. Let D be a domain. We define the binary operation* $\|_{\mathsf{R(M)}} : \mathsf{R}(M)(D) \times \mathsf{R}(M)(D) \to \mathsf{R}(M)(D)$ *by:*

$$x \;\|_{\mathsf{R(M)}}\; y \;=\; h^e\,(\mathit{inr}\,(\mathit{prom}\;x \;\|_{\mathsf{M}}\; \mathit{prom}\;y))$$

Monad $\mathsf{R}(M)$ *with* $\|_{\mathsf{R(M)}}$ *is a multi-monad.*

Furthermore, we can introduce a way to create a new interleaved computation of type $\mathsf{R}(M)(A \times B)$ given two existing computations of types $\mathsf{R}(M)(A)$ and $\mathsf{R}(M)(B)$. Here we prefer to use monads M and $\mathsf{R}(M)$ in the functional way. If one of the two computations does not require the execution of any atomic step, i.e. if one of the two computations has already been completed, then the other computation is executed and the two results are combined. Otherwise, if both computations require at least one atomic step, we choose non-deterministically which computation will start executing.

Definition 3.22 *Let M be a multi-monad. Let A and B be domains. We define the binary operation* $\bowtie_{\mathsf{R(M)}} : \mathsf{R}(M)(A) \times \mathsf{R}(M)(B) \to \mathsf{R}(M)(A \times B)$ *by:*

$$\begin{aligned}
\bowtie_{\mathsf{R(M)}} \;=\; & \mathit{fix}\,(\lambda\, g.\, \lambda\, \langle x, y\rangle.\\
& \quad [\,\lambda\, v_x.\, y \;*_{\mathsf{R(M)}}\; (\lambda\, v_y.\, \mathit{unit}_{\mathsf{R(M)}}\, \langle v_x, v_y\rangle), \lambda\, m_x.\\
& \quad [\,\lambda\, v_y.\, x \;*_{\mathsf{R(M)}}\; (\lambda\, v_x.\, \mathit{unit}_{\mathsf{R(M)}}\, \langle v_x, v_y\rangle), \lambda\, m_y.\\
& \qquad h^e\,(\mathit{inr}\,(m_x \;*_{\mathsf{M}}\; (\lambda\, x'.\, \mathit{unit}_{\mathsf{M}}\,(g\,\langle x', y\rangle)) \;\|_{\mathsf{M}}\\
& \qquad\qquad m_y \;*_{\mathsf{M}}\; (\lambda\, y'.\, \mathit{unit}_{\mathsf{M}}\,(g\,\langle x, y'\rangle))))\\
& \quad]\,(h^p\, y)\,]\,(h^p\, x))
\end{aligned}$$

Monad $\mathsf{R}(M)$ *with* $\bowtie_{\mathsf{R(M)}}$ *is a strong monad.*

4 Semantics of concurrency

Consider the simple imperative language whose abstract syntax is given below.

$$\begin{aligned}
s \;::=\; & \textbf{skip} \mid x \textbf{ := } e \mid s \textbf{ ; } s\\
\mid\; & \textbf{if } e \textbf{ then } s \textbf{ else } s \mid \textbf{while } e \textbf{ do } s
\end{aligned}$$

It features an empty statement, assignment, sequential composition of statements, a structure for conditional and one more for *while* loops. The symbol $x \in \mathbf{Var}$ represents a variable. The language of expressions e is not important for the purpose of this paper and has therefore been omitted.

We define the denotational semantics of this language, assuming that the values of expressions are elements of the semantic domain $\mathbf{V}$. The *program state*, mapping variables to their current values, is an element of the domain $\mathbf{S} = \mathbf{Var} \to \mathbf{V}$.

As a provision for what will follow, we define a monad transformer D implementing the *direct semantics* approach. If M is a monad, we define the monad $\mathsf{D}(M)$ as:

$$\begin{aligned}\mathsf{D}(M)(D) &= \mathbf{S} \to M(D \times \mathbf{S})\\ \mathit{unit}_{\mathsf{D(M)}}\, v &= \lambda\, s.\ \mathit{unit}_{\mathsf{M}}\ \langle v, s\rangle\\ m\ *_{\mathsf{D(M)}}\ f &= \lambda\, s.\, m\ s\ *_{\mathsf{M}}\ (\lambda\,\langle v, s'\rangle.\ f\ v\ s')\end{aligned}$$

State computations created by the direct semantics monad transformer are functions (elements of $\mathsf{D}(M)(D)$) that take the initial program state (an element of $\mathbf{S}$) and return a stateless computation that yields the computed value (an element of D) and the final program state (an element of $\mathbf{S}$). The implementations of $\mathit{unit}_{\mathsf{D(M)}}$ and $*_{\mathsf{D(M)}}$ carry out the propagation of the program state.

We also define an operation for the assignment of values to variables.

$$\begin{aligned}\mathit{store}_{\mathsf{D}} &: \mathbf{Var} \to \mathbf{V} \to \mathsf{D}(M)(\mathbf{U})\\ \mathit{store}_{\mathsf{D}}\, x\ v &= \lambda\, s.\ \mathit{unit}_{\mathsf{M}}\ \langle u, s\{x \mapsto v\}\rangle\end{aligned}$$

By taking the *identity monad* Id as the argument of D, we obtain the monad M that models our simple notion of computation (ordinary direct semantics).

$$\mathsf{M} = \mathsf{D}(\mathsf{Id})$$

The meaning of a statement s is a computation $[\![s]\!]$ of type $\mathsf{M}(\mathbf{U})$. Non-termination is represented by the bottom element. We also assume that the meaning $[\![e]\!]$ of an expression is a computation of type $\mathsf{M}(\mathbf{V})$.

$$\begin{aligned}&[\![\mathbf{skip}]\!] = \mathit{unit}\ u\\ &[\![x \mathbin{\mathbf{:=}} e]\!] = [\![e]\!] * (\mathit{store}\ x)\\ &[\![s_1\ \mathbf{;}\ s_2]\!] = [\![s_1]\!] * (\lambda\, u.\ [\![s_2]\!])\\ &[\![\mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2]\!] = [\![e]\!] * (\lambda\, b.\\ &\qquad \text{if } b \text{ then } [\![s_1]\!] \text{ else } [\![s_2]\!])\\ &[\![\mathbf{while}\ e\ \mathbf{do}\ s]\!] = \mathit{fix}\ (\lambda\, g.\ [\![e]\!] * (\lambda\, b.\\ &\qquad \text{if } b \text{ then } [\![s]\!] * (\lambda\, u.\ g) \text{ else } \mathit{unit}\ u))\end{aligned}$$

Let us now introduce non-determinism and concurrency in our language, by extending it with three new constructs.

$$s ::= \ldots \mid s + s \mid s \parallel s \mid \langle s \rangle$$

Operator + executes exactly one of the statements that are given as its operands. The selection is non-deterministic. On the other hand, operator $\parallel$ executes both statements that are given as its operands in an interleaved way. Finally, the construct $\langle s \rangle$ executes the statement s in a single atomic step, with no interleaving permitted during its execution.

Before we proceed with the semantics of our extended language, we have to modify the definition of M. By giving the powerdomain monad P as the argument of D, we obtain a multi-monad that can support non-determinism.

$$\mathsf{M} = \mathsf{D}(\mathsf{P})$$

The option operator $\parallel_{\mathsf{M}}$ is defined as:

$$m_1 \parallel_{\mathsf{M}} m_2 = \lambda\, s.\ (m_1\ s) \uplus^{\natural} (m_2\ s)$$

where $\uplus^{\natural}$ is the union operation on powerdomains.

In the semantics of the extended language, we use the monad $\mathsf{R(M)}$ to model interleaved computations. According to Definition 3.21, $\mathsf{R(M)}$ is a multi-monad equiped with a non-deterministic option operator $\parallel_{\mathsf{R(M)}}$. Also, according to Definition 3.22, $\mathsf{R(M)}$ is a strong monad and operator $\bowtie_{\mathsf{R(M)}}$ can be used to model the interleaving of computations. Furthermore, the store operation can easily be lifted to the new domain of computations.

$$\begin{aligned}\mathit{store}_{\mathsf{R}} &: \mathbf{Var} \to \mathbf{V} \to \mathsf{R}(\mathsf{D}(M))(\mathbf{U})\\ \mathit{store}_{\mathsf{R}}\, x\ v &= \mathit{step}\ (\mathit{store}_{\mathsf{D}}\, x\ v)\end{aligned}$$

The equations defining the meaning of existing language constructs do not require any changes, except for the implicit change that the meanings of statements and expressions are now elements of the semantic domains $\mathsf{R(M)}(\mathbf{U})$ and $\mathsf{R(M)}(\mathbf{V})$ respectively. On the other hand, the semantics of the additional constructs can be easily expressed in terms of $\mathsf{R(M)}$ operations.

$$\begin{aligned}&[\![s_1 + s_2]\!] = [\![s_1]\!] \parallel_{\mathsf{R(M)}} [\![s_2]\!]\\ &[\![s_1 \parallel s_2]\!] = [\![s_1]\!] \bowtie_{\mathsf{R(M)}} [\![s_2]\!] * (\lambda\, p.\ \mathit{unit}\ u)\\ &[\![\langle s \rangle]\!] = \mathit{step}\ (\mathit{run}\ [\![s]\!])\end{aligned}$$

In the companion technical report [Papa01], a semantics of an imperative concurrent programming language based on the resumption monad transformer is presented with more details and a few examples.

5 Conclusion

This paper defines a general theoretical framework for formalizing the semantics of interleaved computation in concurrent programming languages. The atomic steps in an interleaved computation may themselves be arbitrary computations represented by a given monad M. Furthermore, it is argued that the use of monads enhances the modularity and elegance of the semantics and facilitates the introduction of additional features in a principled way.

Apart from its application in the semantics of concurrency, the resumption monad transformer can be used in the semantics of deterministic languages with unspecified evaluation order, such as Algol and C. The present research was motivated by problems encountered in the formalization of ANSI C [Papa98]. A Haskell implementation of the resumption monad transformer, based on the isomorphism between $\mathsf{R}(M)(D)$ and $D + M(\mathsf{R}(M)(D))$, has been used in [Papa00] to define the denotational semantics of an expression language with side effects under a variety of possible evaluation strategies.

References

[Aspe91] A. Asperti and G. Longo, *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist*, Foundations of Computing Series, MIT Press, Cambridge, MA, 1991.

[Barr96] M. Barr and C. Wells, *Category Theory for Computing Science*, Prentice-Hall International Series in Computer Science, Prentice Hall, New York, NY, 2nd edition, 1996.

[dBak96] J. de Bakker and E. de Vink, *Control Flow Semantics*, Foundations of Computing Series, MIT Press, Cambridge, MA, 1996.

[Gunt90] C. A. Gunter and D. S. Scott, "Semantic Domains", in J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, chapter 12, pp. 633–674, Elsevier Science Publishers B.V., 1990.

[Lian95] S. Liang, P. Hudak and M. Jones, "Monad Transformers and Modular Interpreters", in *Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95)*, San Francisco, CA, January 1995.

[Lian98] S. Liang, *Modular Monadic Semantics and Compilation*, Ph.D. thesis, Yale University, Department of Computer Science, May 1998.

[Mogg90] E. Moggi, "An Abstract View of Programming Languages", Technical Report ECS-LFCS-90-113, University of Edinburgh, Laboratory for Foundations of Computer Science, 1990.

[Papa98] N. S. Papaspyrou, *A Formal Semantics for the C Programming Language*, Ph.D. thesis, National Technical University of Athens, Software Engineering Laboratory, February 1998.

[Papa00] N. S. Papaspyrou, "A Study of Evaluation Order Semantics in Expressions with Side Effects", *Journal of Functional Programming*, 2000, To appear.

[Papa01] N. S. Papaspyrou, "A Resumption Monad Transformer and its Applications in the Semantics of Concurrency", Technical Report CSD-SW-TR-2-01, National Technical University of Athens, Software Engineering Laboratory, April 2001.

[Pier91] B. Pierce, *Basic Category Theory for Computer Scientists*, Foundations of Computing Series, MIT Press, Cambridge, MA, 1991.

[Schm86] D. A. Schmidt, *Denotational Semantics: A Methodology for Language Development*, Allyn and Bacon, Newton, MA, 1986.

[Scot82] D. S. Scott, "Domains for Denotational Semantics", in *International Colloquium on Automata, Languages and Programs*, vol. 140 of *Lecture Notes in Computer Science*, pp. 577–613, Berlin, Germany, 1982, Springer Verlag.

[Wadl92] P. Wadler, "The Essence of Functional Programming", in *Proceedings of the 19th Annual Symposium on Principles of Programming Languages (POPL'92)*, pp. 1–14, January 1992.