A fully abstract semantics for a nondeterministic functional language with monadic types

Alan Jeffrey [1]

School of Cognitive and Computing Sciences
University of Sussex, Brighton BN1 9QH, UK
alanje@cogs.susx.ac.uk

Abstract

This paper presents a functional programming language, based on Moggi's monadic metalanguage. In the first part of this paper, we show how the language can be regarded as a monad on a category of signatures, and that the resulting category of algebras is equivalent to the category of computationally cartesian closed categories. In the second part, we extend the language to include a nondeterministic operational semantics, and show that the lower powerdomain semantics is fully abstract for may-testing.

1 Introduction

Moggi has proposed *strong monads* as an appropriate way to model *computation*. In [9], he shows that any model of computation satisfying certain equations forms a strong monad. His work concentrates on the denotational properties of programs, whereas we shall show how his work can be applied to an operational semantics.

In the first section of this paper, we present a slight variant on his *functional monadic metalanguage* and show that its algebras are equivalent to strong monads with T-exponentials. This language differs from Moggi's in the way that pairing is handled, in particular our language has the properties:

- any closed term of unit type is (up to syntactic identity) the distinguished element $*$,
- any closed term of pairing type is (up to syntactic identity) a pair (e, f),
- any closed term of function type is (up to syntactic identity) a λ-term $\lambda x\,.\,e$.

[1] This work is funded by SERC project GR/H 16537, and is carried out in the context of Esprit BRA 7166 Concur 2.

Moggi's language has these properties, but only up to provable equality, and not syntactic identity. Having these properties true up to syntactic identity is very useful in the second section, where we present an operational semantics for a monadic language with nondeterminism, and show that the fully abstract semantics for this language is given by a powerdomain semantics.

The operational semantics for the monadic language is much simpler than the call-by-value language, since the type structure allows fine control over the syntactic form of terms. For example, the only operational rule required for function application is β-reduction. We do not need any operational rules for which contexts reduction is allowed in, since this is taken care of by the type discipline.

The monadic type system also makes it easier to show full abstraction for the nondeterministic language, since it gives contexts more power over how expressions are evaluated.

2 Algebras

In this section, we present three languages for data and computation, and show that their algebras correspond to well-known categorical structures.

2.1 Algebraic datatypes

A (*many-sorted*) *signature* (ranged over by Σ) is a set of *sorts* (ranged over by A, B and C) and a set of *constructors* (ranged over by c) together with a *sorting* $c : A_1, \ldots, A_n \to A$. A *signature morphism* is a mapping between sorts and constructors with respects sorting. Let **Sig** be the category of signatures with signature morphisms.

Given a signature Σ, we can define the language $\mathsf{ST}\,\Sigma$ of *syntax trees* over Σ as:

$$e ::= * \mid c(e_1, \ldots, e_n) \mid (e, e) \mid v \qquad v ::= x \mid v.\mathrm{L} \mid v.\mathrm{R}$$

where x ranges over a set of *variables*. We shall call expressions v *lvalues*. We can give $\mathsf{ST}\,\Sigma$ a static type system, with types:

$$\tau ::= I \mid [A] \mid \tau \otimes \tau$$

and type judgements of the form $\Gamma \vdash e : \tau$ given by rules:

$$\frac{}{\Gamma \vdash * : I} \qquad \frac{\Gamma \vdash e : \sigma \quad \Gamma \vdash f : \tau}{\Gamma \vdash (e,f) : \sigma \otimes \tau}$$

$$\frac{\Gamma \vdash e_1 : [A_1] \quad \cdots \quad \Gamma \vdash e_n : [A_n]}{\Gamma \vdash c(e_1,\ldots,e_n) : [A]}[c : A_1,\ldots,A_n \to A]$$

$$\frac{\Gamma \vdash v : (\sigma \otimes \tau)}{\Gamma \vdash v.\mathrm{L} : \sigma} \qquad \frac{\Gamma \vdash v : (\sigma \otimes \tau)}{\Gamma \vdash v.\mathrm{R} : \tau}$$

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \frac{\Gamma \vdash y : \tau}{\Gamma, x : \sigma \vdash y : \tau}[x \neq y]$$

where Γ ranges over *contexts* of the form $x_1 : \tau_1, \ldots, x_n : \tau_n$.

Note that we are only allowing projections $v.\mathrm{L}$ and $v.\mathrm{R}$ on lvalues, and *not* on arbitrary terms, since this would not allow us to have the following useful properties:

- any term of type I is either an lvalue or $*$,
- any term of type $[A]$ is either an lvalue or of the form $c(e_1,\ldots,e_n)$, and
- any term of type $\sigma \otimes \tau$ is either an lvalue or of the form (e,f).

However, whenever $\Gamma \vdash e : \sigma \otimes \tau$, we can define $\Gamma \vdash \pi e : \sigma$ and $\Gamma \vdash \pi' e : \tau$ as syntactic sugar, e is either an lvalue or a pair:

$$\pi v = v.\mathrm{L} \qquad \pi' v = v.\mathrm{R} \qquad \pi(f,g) = f \qquad \pi'(f,g) = g$$

$\mathsf{ST}\,\Sigma$ is itself a signature, with types as sorts and judgements $(x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash e : \tau)$ as constructors $\vec{\sigma} \to \tau$, viewed up to the congruence given by (when y is fresh):

$$\begin{aligned}(\Gamma \vdash x : I) &= (\Gamma \vdash * : I)\\ (\Gamma \vdash (v.\mathrm{L}, v.\mathrm{R}) : \sigma \otimes \tau) &= (\Gamma \vdash v : \sigma \otimes \tau)\\ (\Gamma, x : \sigma, \Gamma' \vdash e : \tau) &= (\Gamma, y : \sigma, \Gamma' \vdash e[y/x] : \tau)\end{aligned}$$

Note that these equations only involve open terms, so closed terms are viewed up to syntactic identity.

Any signature morphism $f : \Sigma \to \Sigma'$ can be homomorphically extended to a signature morphism $\mathsf{ST}\,f : \mathsf{ST}\,\Sigma \to \mathsf{ST}\,\Sigma'$. It is routine to verify that $\mathsf{ST} : \mathbf{Sig} \to \mathbf{Sig}$ is a functor.

Whenever $\Gamma, \vec{x} : \vec{\sigma} \vdash e : \tau$ and $\Gamma \vdash \vec{f} : \vec{\sigma}$ we can define the *substitution* $\Gamma \vdash e[\vec{f}/\vec{x}] : \tau$ by its action on lvalues (when $x \neq y$):

$$v.\mathrm{L}[\vec{f}/\vec{x}] = \pi(v[\vec{f}/\vec{x}]) \qquad v.\mathrm{R}[\vec{f}/\vec{x}] = \pi'(v[\vec{f}/\vec{x}])$$

$$y[\vec{f}, f/\vec{x}, x] = y[\vec{f}/\vec{x}] \qquad x[\vec{f}, f/\vec{x}, x] = f$$

We can define $\eta_\Sigma : \Sigma \to \mathsf{ST}\,\Sigma$ as the *injection*:

$$A \mapsto [A] \qquad (c : A_1, \ldots, A_n \to A) \mapsto (x_1 : [A_1], \ldots, x_n : [A_n] \vdash c(x_1, \ldots, x_n) : [A])$$

and $\mu_\Sigma : \mathsf{ST}^2\,\Sigma \to \mathsf{ST}\,\Sigma$ as the *substitution* map given homomorphically by:

$$[\tau] \mapsto \tau \qquad (\vec{x} : \vec{\sigma} \vdash e : \tau)(\vec{f}) \mapsto e[\mu\vec{f}/\vec{x}]$$

It is routine to verify that ST is a monad. Since we have defined η by injection and μ by substitution, it is reasonable to view the denotational models for ST as being ST-*algebras*, that is a signature Σ with a morphism $[\![\_]\!] : \mathsf{ST}\,\Sigma \to \Sigma$ such that:

$$\eta; [\![\_]\!] = \mathsf{id} \qquad \mu; [\![\_]\!] = \mathsf{ST}[\![\_]\!]; [\![\_]\!]$$

The first equation says that the denotation of each constructor in Σ should be itself, and the second that the semantics respects substition, and so is *denotational*. Let ST-**Alg** be the category of all ST-algebras, together with morphisms which respect $[\![\_]\!]$.

Let **CCat** be the category of small categories with distinguished finite products, and functors which respect the product structure.

Proposition 2.1 ST-**Alg** *is equivalent to* **CCat**.

2.2 Monadic metalanguage

We shall now add a notion of *computation* to our language of data, using Moggi's [9] typed monadic language.

To do this, we extend $\mathsf{ST}\,\Sigma$ to the *monadic metalanguage*, $\mathsf{MML}\,\Sigma$ by adding two new expression constructions:

$$e ::= \cdots \mid [e] \mid \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, e$$

These are:

- $[e]$ is a computation which immediately terminates with result e. This is similar to 'exit' in LOTOS [1], and 'return' in Concurrent ML [13,14].
- $\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f$ is a computation which evaluates e until it returns a value, which is then bound to x in f. For example, $\mathsf{let}\, x \Leftarrow [\mathsf{zero}]\, \mathsf{in}\, [\mathsf{succ}\, x]$ is the same as $[\mathsf{succ}\,\mathsf{zero}]$.

We also extend the type system by adding a new type constructor for computations:

$$\tau ::= \cdots \mid \mathsf{C}\,\tau$$

and statically typing $\mathsf{MML}\Sigma$ as:

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash [e] : \mathsf{C}\tau} \qquad \frac{\Gamma \vdash e : \mathsf{C}\sigma \quad \Gamma, x : \sigma \vdash f : \mathsf{C}\tau}{\Gamma \vdash \mathsf{let}\, x \Leftarrow e \,\mathsf{in}\, f : \mathsf{C}\tau}$$

Then MML forms a monad in the same way as ST does, with the addition of Moggi's [9] axioms (when x is not free in g):

$$\begin{aligned}
(\Gamma \vdash \mathsf{let}\, y \Leftarrow f \,\mathsf{in}\, g : \mathsf{C}\tau) &= (\Gamma \vdash \mathsf{let}\, x \Leftarrow f \,\mathsf{in}\, g[x/y] : \mathsf{C}\tau) \\
(\Gamma \vdash \mathsf{let}\, x \Leftarrow [e] \,\mathsf{in}\, f : \mathsf{C}\tau) &= (\Gamma \vdash f[e/x] : \mathsf{C}\tau) \\
(\Gamma \vdash \mathsf{let}\, x \Leftarrow e \,\mathsf{in}\, [x] : \mathsf{C}\tau) &= (\Gamma \vdash e : \mathsf{C}\tau) \\
(\Gamma \vdash \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow e \,\mathsf{in}\, f) \,\mathsf{in}\, g : \mathsf{C}\tau) &= (\Gamma \vdash \mathsf{let}\, x \Leftarrow e \,\mathsf{in}\, (\mathsf{let}\, y \Leftarrow f \,\mathsf{in}\, g) : \mathsf{C}\tau)
\end{aligned}$$

Let **SMon** be the category of small categories with strong monads, together with functors which respect the monadic structure. The next proposition shows that the MML-algebras are precisely strong monads (hence the name 'monadic metalanguage'). This result is due largely to Moggi [9].

Proposition 2.2 MML-**Alg** *is equivalent to* **SMon**.

2.3 Partial functions

We extend $\mathsf{MML}\Sigma$ to the *functional* monadic metalanguage, $\mathsf{MML}\lambda\Sigma$ by adding λ-binding and function application:

$$e ::= \cdots \mid \lambda x\,.\,e \mid ee$$

We also extend the type system by adding a new type constructor for functions:

$$\tau ::= \cdots \mid \tau \to \mathsf{C}\tau$$

and statically typing $\mathsf{MML}\lambda\Sigma$ as:

$$\frac{\Gamma, x : \sigma \vdash e : \mathsf{C}\tau}{\Gamma \vdash \lambda x\,.\,e : \sigma \to \mathsf{C}\tau} \qquad \frac{\Gamma \vdash e : \sigma \to \mathsf{C}\tau, f : \sigma}{\Gamma \vdash ef : \mathsf{C}\tau}$$

Note that we are only allowing functions to return computations, for example there is no type $I \to I$, only $I \to \mathsf{C}I$. This corresponds to our intuition that the only terms which involve computation are terms of type $\mathsf{C}\tau$, and this would not be true if we allowed functions to return arbitrary type. This restriction also allows us to show that:

– any term of type $\sigma \to \mathsf{C}\tau$ is either an lvalue or of the form $\lambda x\,.\,e$.

Note that we have *no* similar result about terms of type $\mathsf{C}\tau$.

Then $\mathsf{MML}\lambda$ forms a monad in the same way as MML does, with the addition of the standard α, β and η axioms for functions (when y is not free in e):

$$\begin{aligned}(\Gamma \vdash \lambda x.\, e : \sigma \to \mathsf{C}\tau) &= (\Gamma \vdash \lambda y.\, e[y/x] : \sigma \to \mathsf{C}\tau)\\ (\Gamma \vdash (\lambda x.\, e)\, f : \mathsf{C}\tau) &= (\Gamma \vdash e[f/x] : \mathsf{C}\tau)\\ (\Gamma \vdash \lambda y.\, (ey) : \sigma \to \mathsf{C}\tau) &= (\Gamma \vdash e : \sigma \to \mathsf{C}\tau)\end{aligned}$$

A category $\mathbf{C}$ is *computationally cartesian closed* iff it has a strong monad $T : \mathbf{C} \to \mathbf{C}$, and for each objects X and Y there is an object TY^X such that there is a natural isomorphism:

$$\mathsf{curry} : \mathbf{C}[X \times Y, TZ] \to \mathbf{C}[X, TZ^Y]$$

Let **CCCC** be the category of small computationally cartesian closed categories together with functors which respect the monadic and T-exponential structure.

Proposition 2.3 $\mathsf{MML}\lambda$-**Alg** *is equivalent to* **CCCC**.

3 Nondeterminism

In this section, we extend the monadic metalanguage with the structure of a nondeterministic programming language. We present an operational semantics for this language, and show that a powerdomain semantics is fully abstract for may-testing for this language.

3.1 Syntax

A signature has *booleans* iff it has a sort bool with constructors $\mathsf{true}, \mathsf{false} : \to \mathsf{bool}$. A signature has *deconstructors* iff it has a set of deconstructors ranged over by d with sorting $d : \vec{A} \to A$. Let **SigBD** be the category of signatures with booleans and deconstructors, together with morphisms which respect the booleans, constructors, deconstructors, and sorting.

Given a signature Σ with deconstructors and booleans, the *nondeterministic* monadic metalanguage $\mathsf{NMML}\Sigma$ extends $\mathsf{MML}\lambda\Sigma$ with expressions:

$$e ::= \cdots \mid \mathsf{if}\, e\, \mathsf{then}\, e\, \mathsf{else}\, e \mid d\vec{e} \mid \delta \mid e \,\square\, e \mid \mathsf{fix}(x = e)$$

and type judgements:

$$\frac{\Gamma \vdash e : [\mathsf{bool}] \quad \Gamma \vdash f : \mathsf{C}\,\tau \quad \Gamma \vdash g : \mathsf{C}\,\tau}{\Gamma \vdash \mathsf{if}\, e\, \mathsf{then}\, f\, \mathsf{else}\, g : \mathsf{C}\,\tau}$$

$$\frac{\Gamma \vdash e_1 : [A_1] \cdots \Gamma \vdash e_n : [A_n]}{\Gamma \vdash d(e_1, \ldots, e_n) : \mathsf{C}[A]}[d : A_1, \ldots, A_n \to A]$$

$$\frac{}{\Gamma \vdash \delta : \mathsf{C}\,\tau} \qquad \frac{\Gamma \vdash e : \mathsf{C}\,\tau \Gamma \vdash f : \mathsf{C}\,\tau}{\Gamma \vdash e \,\square\, f : \mathsf{C}\,\tau} \qquad \frac{\Gamma, x : \mathsf{C}\,\tau \vdash e : \mathsf{C}\,\tau}{\Gamma \vdash \mathsf{fix}(x = e) : \mathsf{C}\,\tau}$$

Note that deconstructors and if-statements are of *computation* type.

3.2 Operational semantics

In order to give an operational semantics for $\mathsf{NMML}\,\Sigma$, we need an operational semantics for the deconstructors of Σ. This is given as a *higher-order unlabeled value production system*, that is:

– an *internal transition* relation $e \xrightarrow{\imath} e'$, and
– a *termination* relation $e \xrightarrow{\surd} e'$

such that:

– if $e \xrightarrow{\imath} e'$ then $\vdash e : \mathsf{C}\,\tau$ and $\vdash e' : \mathsf{C}\,\tau$ for some τ,
– if $e \xrightarrow{\surd} e'$ then $\vdash e : \mathsf{C}\,\tau$ and $\vdash e' : \tau$ for some τ,
– $\xrightarrow{\surd}$ is deterministic, and
– if $e \xrightarrow{\surd}$ then $e \not\xrightarrow{\imath}$.

Given an operational semantics for terms of the form $d\vec{e}$, we can extend it to an operational semantics for closed terms of $\mathsf{NMML}\,\Sigma$ with:

$$\frac{}{[e] \xrightarrow{\surd} e} \qquad \frac{e \xrightarrow{\imath} e'}{\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f \xrightarrow{\imath} \mathsf{let}\, x \Leftarrow e'\, \mathsf{in}\, f} \qquad \frac{e \xrightarrow{\surd} g}{\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f \xrightarrow{\imath} f[g/x]}$$

$$\frac{}{\mathsf{if}\, \mathsf{true}\, \mathsf{then}\, f\, \mathsf{else}\, g \xrightarrow{\imath} f} \qquad \frac{}{\mathsf{if}\, \mathsf{false}\, \mathsf{then}\, f\, \mathsf{else}\, g \xrightarrow{\imath} g}$$

$$\frac{}{(\lambda x\,.\,e)\, f \xrightarrow{\imath} e[f/x]} \qquad \frac{}{\mathsf{fix}(x = e) \xrightarrow{\imath} e[\mathsf{fix}(x = e)/x]}$$

$$\frac{e \xrightarrow{\imath} e'}{e \,\square\, f \xrightarrow{\imath} e' \,\square\, f} \qquad \frac{f \xrightarrow{\imath} f'}{e \,\square\, f \xrightarrow{\imath} e \,\square\, f'} \qquad \frac{e \xrightarrow{\surd} e'}{e \,\square\, f \xrightarrow{\imath} [e']} \qquad \frac{f \xrightarrow{\surd} f'}{e \,\square\, f \xrightarrow{\imath} [f']}$$

A (*higher order, weak*) *simulation* on $\mathsf{NMML}\Sigma$ is a type-indexed family of relations $\mathcal{R}_\tau \subseteq \{(e,f) \mid \vdash e, f : \tau\}$ such that:

– if $e\ \mathcal{R}_{[A]}\ f$ then $e = f$.
– if $(e,e')\ \mathcal{R}_{\sigma\otimes\tau}\ (f,f')$ then $e\ \mathcal{R}_\sigma\ f$ and $e'\ \mathcal{R}_\tau\ f'$,
– if $(\lambda x.\, e)\ \mathcal{R}_{\sigma\to \mathsf{C}\tau}\ (\lambda y.\, f)$ then for all $\vdash g : \sigma$ we have $e[g/x]\ \mathcal{R}_{\mathsf{C}\tau}\ f[g/y]$,
– if $e\ \mathcal{R}_{\mathsf{C}\tau}\ f$ and $e \xrightarrow{\tau} e'$ then $f \xrightarrow{\tau}{}^{*}\ f'$ and $e'\ \mathcal{R}_{\mathsf{C}\tau}\ f'$, and
– if $e\ \mathcal{R}_{\mathsf{C}\tau}\ f$ and $e \xrightarrow{\surd} e'$ then $f \xrightarrow{\tau}{}^{*} \xrightarrow{\surd} f'$ and $e'\ \mathcal{R}_\tau\ f'$.

A bisimulation is a weak simulation whose inverse is a weak simulation. Write $\models e = f : \tau$ iff there is a bisimulation $\mathcal{R}$ such that $e\ \mathcal{R}_\tau\ f$. Write $\vec{x} : \vec{\sigma} \models e = f : \tau$ iff for every $\vdash \vec{g} : \vec{\sigma}$ we have $\models e[\vec{g}/\vec{x}] = f[\vec{g}/\vec{x}] : \tau$.

Howe [6] has shown a technique for proving that simulation for a class of lazy functional languages is substitutive. In an unpublished paper [5], Howe has also shown that bisimulation is a congruence (this result was communicated to the author by Andy Pitts). This technique can be used to show that bisimulation is a congruence for $\mathsf{NMML}\Sigma$.

Proposition 3.1 *Bisimulation is a congruence for* $\mathsf{NMML}\Sigma$.

We can show that $\mathsf{NMML}\Sigma$ forms a signature in the same way as $\mathsf{MML}\lambda\Sigma$, except that we view terms up to bisimulation. It is routine to verify that NMML is a monad on **SigBD**. Any NMML-algebra is an $\mathsf{MML}\lambda$-algebra since we can exhibit bisimulations for (when y is not free in g):

$$\begin{aligned}
\Gamma &\models x = * : I \\
\Gamma &\models (v.\mathrm{L}, v.\mathrm{R}) = v : \sigma \otimes \tau \\
\Gamma &\models \mathsf{let}\, x \Leftarrow [e]\, \mathsf{in}\, f = f[e/x] : \mathsf{C}\tau \\
\Gamma &\models \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, [x] = e : \mathsf{C}\tau \\
\Gamma &\models \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f)\, \mathsf{in}\, g = \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, (\mathsf{let}\, y \Leftarrow f\, \mathsf{in}\, g) : \mathsf{C}\tau \\
\Gamma &\models (\lambda x.\, e) f = e[f/x] : \mathsf{C}\tau \\
\Gamma &\models \lambda y.\, (gy) = g : \sigma \to \mathsf{C}\tau
\end{aligned}$$

For any $\Gamma \vdash e, f : \tau$, define the *may-testing preorder* as $\Gamma \models e \sqsubseteq_O f : \tau$ iff $C[e] \stackrel{\surd}{\Longrightarrow} *$ implies $C[f] \stackrel{\surd}{\Longrightarrow} *$ for any closing context C of type $\mathsf{C}I$.

3.3 Denotational semantics

Let **Alg** be the category of algebraic dcpo's, together with continuous morphisms (we are not requiring dcpo's to have least elements). Let $\mathbf{Alg}_{\perp\vee}$ be the category of algebraic dcpo's with all finite joins, together with continuous morphisms which respect the joins. Let $\mathcal{P} : \mathbf{Alg} \to \mathbf{Alg}$ be the *lower powerdomain* functor given by the adjunction $\mathbf{Alg} \xrightarrow{F} \mathbf{Alg}_{\perp\vee} \xrightarrow{U} \mathbf{Alg}$. This forms a strong monad with $\mathcal{P}$-exponentials, where $\eta_X = \{\_\} : X \to \mathcal{P}X$ and $\mu_X = \bigcup : \mathcal{P}^2 X \to X$. (Note that these

exponentials exist even though **Alg** is not cartesian closed, since we are only considering functions whose target is an object in $\mathbf{Alg}_{\perp\vee}$.)

Alg is a signature with booleans and deconstructors, since it has objects as sorts, morphisms $f : X_1 \times \cdots \times X_n \to X$ as constructors, morphisms $f : X_1 \times \cdots \times X_n \to \mathcal{P}X$ as deconstructors, and a sort $1+1$ with constructors $\kappa, \kappa' : 1 \to 1+1$. Since $\mathcal{P}$ is a strong monad on **Alg** with $\mathcal{P}$-exponentials, we therefore have a denotational semantics $[\![\_]\!] : \mathsf{MML}\lambda\,\mathbf{Alg} \to \mathbf{Alg}$ given by Proposition 2.3.The semantics for $\mathsf{NMML}\,\mathbf{Alg}$ extends this with:

$$\begin{aligned}
[\![\Gamma \vdash \delta : \mathsf{C}\tau]\!] &= \bot \\
[\![\Gamma \vdash e \,\Box\, f : \mathsf{C}\tau]\!] &= [\![\Gamma \vdash e : \mathsf{C}\tau]\!] \vee [\![\Gamma \vdash f : \mathsf{C}\tau]\!] \\
[\![\Gamma \vdash \mathsf{fix}(x = e) : \mathsf{C}\tau]\!] &= \text{the least fixed pt of } f \mapsto \langle \mathsf{id}, f \rangle; [\![\Gamma, x : \mathsf{C}\tau \vdash e : \mathsf{C}\tau]\!] \\
[\![\Gamma \vdash \mathsf{if}\, e \,\mathsf{then}\, f \,\mathsf{else}\, g : \mathsf{C}\tau]\!] &= \langle \mathsf{id}, [\![\Gamma \vdash e : [\mathsf{bool}]]\!] \rangle; \mathsf{dist}; [[\![\Gamma \vdash f : \mathsf{C}\tau]\!], [\![\Gamma \vdash g\,\mathsf{C}\tau]\!]]
\end{aligned}$$

where $\mathsf{dist} : X \times (1+1) \to X + X$ is the distributivity morphism.

For any Σ, if there is a morphism $[\![\_]\!] : \Sigma \to \mathbf{Alg}$ then we can extend this to $\mathsf{NMML}\,\Sigma$ as:

$$\mathsf{NMML}\,\Sigma \xrightarrow{\mathsf{NMML}[\![\_]\!]} \mathsf{NMML}\,\mathbf{Alg} \xrightarrow{[\![\_]\!]} \mathbf{Alg}$$

A semantics $[\![\_]\!] : \Sigma \to \mathbf{Alg}$ is *adequate* iff:

$$[\![\vdash d\vec{e} : \mathsf{C}[A]]\!] = \bigvee \{ [\![\vdash [f] : \mathsf{C}[A]]\!] \mid d\vec{e} \stackrel{\checkmark}{\Longrightarrow} f \}$$

A semantics $[\![\_]\!] : \Sigma \to \mathbf{Alg}$ is *expressive* iff for any compact $a \in [\![A]\!]$ we can find terms is_a and test_a such that:

$$[\![\vdash \mathsf{is}_a : [A]]\!] = a \qquad [\![\vdash \mathsf{test}_a : [A] \to \mathsf{C}I]\!] = (a \Rightarrow \eta\bot)$$

A semantics $[\![\_]\!] : \mathsf{NMML}\,\Sigma \to \mathbf{Alg}$ is *correct* iff:

$$[\![\Gamma \vdash e : \tau]\!] \leq [\![\Gamma \vdash f : \tau]\!] \text{ implies } \Gamma \models e \sqsubseteq_O f : \tau$$

The semantics for $\mathsf{NMML}\,\Sigma$ is *fully abstract* iff:

$$[\![\Gamma \vdash e : \tau]\!] \leq [\![\Gamma \vdash f : \tau]\!] \text{ iff } \Gamma \models e \sqsubseteq_O f : \tau$$

The rest of this section shows that if a semantics for Σ is adequate then its extension to $\mathsf{NMML}\,\Sigma$ is correct, and that if a semantics for Σ is adequate and expressive, then its extension to $\mathsf{NMML}\,\Sigma$ is fully abstract.

In order to show the relationship between the operational and denotational semantics of NMMLΣ, we shall use a *program logic* similar to that used by Abramsky [2] and Ong [11] in modelling the untyped λ-calculus, based on Abramsky's [3] *domain theory in logical form*.

This logic is similar to Ong's [10] logic for an untyped nondeterministic λ-calculus. Since we are looking at may-testing rather than simulation, we only have conjunction in the logic, and not disjunction, and only one modality rather than two.

The *program logic* for NMMLΣ has propositions:

$$\phi ::= * \mid (\phi, \psi) \mid |a| \mid \omega \mid \phi \wedge \psi \mid [\phi] \mid \phi \Rightarrow \psi$$

These can be statically typed, so the propositions for type τ are those where $\phi : \mathcal{L}\tau$:

$$\frac{}{* : \mathcal{L}I} \qquad \frac{\phi : \mathcal{L}\sigma \quad \psi : \mathcal{L}\tau}{(\phi, \psi) : \mathcal{L}(\sigma \otimes \tau)}$$

$$\frac{}{|a| : \mathcal{L}[A]}\,[a \in [\![A]\!], a \text{ is compact}]$$

$$\frac{}{\omega : \mathcal{L}(\mathsf{C}\tau)} \qquad \frac{\phi : \mathcal{L}(\mathsf{C}\tau) \quad \psi : \mathcal{L}(\mathsf{C}\tau)}{\phi \wedge \psi : \mathcal{L}(\mathsf{C}\tau)} \qquad \frac{\phi : \mathcal{L}\tau}{[\phi] : \mathcal{L}(\mathsf{C}\tau)}$$

$$\frac{}{\omega : \mathcal{L}(\sigma \to \mathsf{C}\tau)} \qquad \frac{\phi : \mathcal{L}(\sigma \to \mathsf{C}\tau) \quad \psi : \mathcal{L}(\sigma \to \mathsf{C}\tau)}{\phi \wedge \psi : \mathcal{L}(\sigma \to \mathsf{C}\tau)} \qquad \frac{\phi : \mathcal{L}\sigma \quad \psi : \mathcal{L}(\mathsf{C}\tau)}{\phi \Rightarrow \psi : \mathcal{L}(\sigma \to \mathsf{C}\tau)}$$

The operational characterization of the logic has judgements $\models e : \phi$ given by:

$$\frac{}{\models * : *} \qquad \frac{\models e : \phi \quad \models f : \psi}{\models (e, f) : (\phi, \psi)} \qquad \frac{a \leq [\![\vdash e : [A]]\!]}{\models e : |a|}$$

$$\frac{}{\models e : \omega} \qquad \frac{\models e : \phi \quad \models e : \psi}{\models e : \phi \wedge \psi}$$

$$\frac{e \xrightarrow{1} e' \quad \models e' : \phi}{\models e : \phi} \qquad \frac{e \xrightarrow{\surd} f \quad \models f : \phi}{\models f : [\phi]} \qquad \frac{\forall \models f : \phi \,.\, \models ef : \psi}{\models e : \phi \Rightarrow \psi}$$

This can be generalized to open terms as:

$$\vec{x} : \vec{\phi} \models e : \psi \text{ iff } \forall \models \vec{f} : \vec{\phi} \,.\, \models e[\vec{f}/\vec{x}] : \psi$$

Let Δ range over propositional contexts of the form $x_1 : \phi_1, \ldots, x_n : \phi_n$, and write

$\Delta : \mathcal{L}\Gamma$ for:

$$(x_1 : \phi_1, \ldots, x_n : \phi_n) : \mathcal{L}(x_1 : \tau_1, \ldots, x_n : \tau_n) \text{ iff } \phi_1 : \mathcal{L}\tau_1, \ldots, \phi_n : \mathcal{L}\tau_n$$

We can also define a denotational semantics for propositions, so that if $\phi : \mathcal{L}\tau$ then $[\![\phi]\!] \in [\![\tau]\!]$:

$$[\![*]\!] = \bot \quad [\![(\phi, \psi)]\!] = ([\![\phi]\!], [\![\psi]\!]) \quad [\![|a|]\!] = a$$
$$[\![\omega]\!] = \bot \quad [\![\phi \wedge \psi]\!] = [\![\phi]\!] \vee [\![\psi]\!] \quad [\![[\phi]]\!] = \eta[\![\phi]\!] \quad [\![\phi \Rightarrow \psi]\!] = [\![\phi]\!] \Rightarrow [\![\psi]\!]$$

Whenever $\Delta : \mathcal{L}\Gamma$, we can define $[\![\Delta]\!] \in [\![\Gamma]\!]$ as: $[\![x_1 : \phi_1, \ldots, x_n : \phi_n]\!] = ([\![\phi_1]\!], \ldots, [\![\phi_n]\!])$

Proposition 3.2 *$a \in [\![\tau]\!]$ is compact iff $\exists \phi : \mathcal{L}\tau \,.\, a = [\![\phi]\!]$.*

3.5 Proof system

In order to relate the denotational and operational characterizations of the program logic, we shall use an intermediate proof system. This is a sequent calculus with judgements of the form $\Delta \vdash e : \phi$ where $\Gamma \vdash e : \tau$, $\Delta : \mathcal{L}\Gamma$ and $\phi : \mathcal{L}\tau$.

Let $\leq$ be the preorder on propositions given by:

- ω is the top element, and $(\_ \wedge \_)$ is meet.
- $(\_, \_)$, $[\_]$ and $(\phi \Rightarrow \_)$ are monotone.
- $(\phi \Rightarrow \_)$ preserves ω and $\wedge$.
- $|\_|$ and $(\_ \Rightarrow \psi)$ are anti-monotone.

Proposition 3.3 *$\phi \leq \psi$ iff $[\![\phi]\!] \geq [\![\psi]\!]$.*

We can then define the proof system for $\mathsf{NMML}\,\Sigma$ as:

$$\frac{[\![\phi]\!] \leq [\![\Gamma \vdash c\vec{e} : [A]]\!][\![\Delta]\!]}{\Delta \vdash c\vec{e} : \phi} \quad \frac{[\![\phi]\!] \leq [\![\Gamma \vdash d\vec{e} : \mathsf{C}[A]]\!][\![\Delta]\!]}{\Delta \vdash d\vec{e} : \phi}$$

$$\frac{\Delta \vdash e : \psi}{\Delta \vdash e : \phi}[\psi \leq \phi] \quad \frac{}{\Delta, x : \phi \vdash x : \phi} \quad \frac{\Delta \vdash x : \phi}{\Delta, y : \psi \vdash x : \phi}[x \neq y]$$

$$\frac{}{\Delta \vdash * : *} \quad \frac{\Delta \vdash e : \phi \quad \Delta \vdash f : \psi}{\Delta \vdash (e, f) : (\phi, \psi)}$$

$$\frac{}{\Delta \vdash e : \omega} \quad \frac{\Delta \vdash e : \phi \quad \Delta \vdash e : \psi}{\Delta \vdash e : \phi \wedge \psi}$$

$$\frac{\Delta \vdash e : \phi}{\Delta \vdash [e] : [\phi]} \quad \frac{\Delta \vdash e : [\phi] \quad \Delta, x : \phi \vdash f : \psi}{\Delta \vdash \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f : \psi}$$

$$\frac{\Delta, x : \psi \vdash e : \chi}{\Delta \vdash \lambda x\,.\, e : \psi \Rightarrow \chi} \quad \frac{\Delta \vdash e : \psi \Rightarrow \chi \quad \Delta \vdash f : \psi}{\Delta \vdash ef : \chi} \quad \frac{\Delta \vdash e : \psi \quad \Delta \vdash f : \chi}{\Delta \vdash e \,\square\, f : \psi \wedge \chi}$$

$$\frac{\Delta \vdash e : |t| \quad \Delta \vdash f : \phi}{\Delta \vdash \mathsf{if}\, e\, \mathsf{then}\, f\, \mathsf{else}\, g : \phi} \quad \frac{\Delta \vdash e : |f| \quad \Delta \vdash g : \phi}{\Delta \vdash \mathsf{if}\, e\, \mathsf{then}\, f\, \mathsf{else}\, g : \phi}$$

$$\frac{\Delta \vdash \mathsf{fix}(x = e) : \psi \quad \Delta, x : \psi \vdash e : \chi}{\Delta \vdash \mathsf{fix}(x = e) : \chi}$$

Note that all of the structural rules for the proof system, such as weakening and contraction, have been absorbed into the definition of $\phi \leq \psi$.

Proposition 3.4 $\Delta \vdash e : \phi$ *iff* $[\![\phi]\!] \leq [\![\Gamma \vdash e : \tau]\!][\![\Delta]\!]$.

3.6 Full abstraction

We can now show that the semantics for $\mathsf{NMML}\,\Sigma$ is fully abstract. We begin by showing that if Σ is expressive, then so is $\mathsf{NMML}\,\Sigma$. Let $\mathsf{term}_\tau\,\phi$ be defined:

$$\begin{aligned}
\mathsf{term}_I\, * &= * \\
\mathsf{term}_{\sigma\otimes\tau}(\phi, \psi) &= (\mathsf{term}_\sigma\,\phi, \mathit{term}_\tau\psi) \\
\mathsf{term}_{[A]}\,|a| &= \mathsf{is}_a \\
\mathsf{term}_{\mathsf{C}\tau}\,\omega &= \delta \\
\mathsf{term}_{\mathsf{C}\tau}(\phi \wedge \psi) &= \mathsf{term}_{\mathsf{C}\tau}\,\phi \,\square\, \mathsf{term}_{\mathsf{C}\tau}\,\psi \\
\mathsf{term}_{\mathsf{C}\tau}[\phi] &= [\mathsf{term}_\tau\,\phi] \\
\mathsf{term}_{\sigma\to\mathsf{C}\tau}\,\omega &= \lambda x\,.\,\delta \\
\mathsf{term}_{\sigma\to\mathsf{C}\tau}(\phi \wedge \psi) &= \lambda x\,.\,(\mathsf{term}_{\sigma\to\mathsf{C}\tau}\,\phi)x \,\square\, (\mathsf{term}_{\sigma\to\mathsf{C}\tau}\,\psi)x \\
\mathsf{term}_{I\to\mathsf{C}\tau}(* \Rightarrow \chi) &= \lambda x\,.\,\mathsf{term}_{\mathsf{C}\tau}\,\chi
\end{aligned}$$

$$\begin{aligned}
\mathsf{term}_{\rho\otimes\sigma\to \mathsf{C}\tau}((\psi,\phi)\Rightarrow\chi) &= \lambda x.\ \mathsf{let}\, y \Leftarrow (\mathsf{term}_{\rho\to \mathsf{C}I}(\psi\Rightarrow[*]))(x.\mathrm{L}) \\
&\qquad \mathsf{in}(\mathsf{term}_{\sigma\to \mathsf{C}\tau}(\phi\Rightarrow\chi))(x.\mathrm{R}) \\
\mathsf{term}_{[A]\to \mathsf{C}\tau}(|a|\Rightarrow\chi) &= \lambda x.\, \mathsf{let}\, y \Leftarrow (\mathsf{test}_a x)\, \mathsf{in}\, \mathsf{term}_{\mathsf{C}\tau}\chi \\
\mathsf{term}_{\sigma\to \mathsf{C}\tau}(\omega\Rightarrow\chi) &= \lambda x.\, \mathsf{term}_{\mathsf{C}\tau}\chi \\
\mathsf{term}_{\sigma\to \mathsf{C}\tau}(\phi\wedge\psi\Rightarrow\chi) &= \lambda x.\ \mathsf{let}\, y \Leftarrow \mathsf{term}_{\sigma\to \mathsf{C}I}(\phi\Rightarrow[*])x \\
&\qquad \mathsf{in}\, \mathsf{term}_{\sigma\to \mathsf{C}\tau}(\psi\Rightarrow\chi)x \\
\mathsf{term}_{\mathsf{C}\sigma\to \mathsf{C}\tau}([\phi]\Rightarrow\chi) &= \lambda x.\, \mathsf{let}\, y \Leftarrow x\, \mathsf{in}\, \mathsf{term}_{\sigma\to \mathsf{C}\tau}y \\
\mathsf{term}_{(\rho\to \mathsf{C}\sigma)\to \mathsf{C}\tau}((\phi\Rightarrow\psi)\Rightarrow\chi) &= \lambda x.\, (\mathsf{term}_{\mathsf{C}\sigma\to \mathsf{C}\tau}(\psi\Rightarrow\chi))(x(\mathsf{term}_{\rho}\,\phi))
\end{aligned}$$

We can then verify that: $[\![\phi]\!] = [\![\vdash \mathsf{term}_\tau \phi : \tau]\!]$ This expressivity result is used in showing that the semantics for $\mathsf{NMML}\,\Sigma$ is fully abstract. The relationship between expressivity and full abstraction has been long known [8,12].

In Section 3.5 we showed that the denotational characterization and proof system for the program logic were equivalent:

$$\Delta \vdash e : \phi \text{ iff } [\![\phi]\!] \leq [\![\Gamma \vdash e : \tau]\!][\![\Delta]\!]$$

We can extend this to show (as long as the semantics for Σ is adequate and expressive) that:

$$\Delta \vdash e : \phi \text{ implies } \Delta \models e : \phi \text{ implies } [\![\phi]\!] \leq [\![\Gamma \vdash e : \tau]\!][\![\Delta]\!]$$

and so the operational characterization of the program logic is equivalent to the denotational characterization and to the proof system. From this we prove full abstraction.

Theorem 3.5 *[full abstraction]*

(i) *If a semantics for Σ is adequate, then its extension to $\mathsf{NMML}\,\Sigma$ is correct.*
(ii) *If a semantics for Σ is expressive and adequate then its extension to $\mathsf{NMML}\,\Sigma$ is fully abstract.*

4 Further work

The results given here are part of a larger paper [7], which builds on the results presented here to give an operational and fully abstract denotational semantics for a typed higher-order concurrent language based on Concurrent ML.

The techniques presented here can be applied to concurrent systems, and in particular the program logic for the concurrent language is a modal logic similar to Hennessy's program logic for untyped higher-order concurrency [4].

The author is currently working on applying these techniques to the ISO communications protocol specification language LOTOS [1], as part of the development of an extended LOTOS standard.

Acknowledgement

Many thanks to Bill Ferreira, Matthew Hennessy, and Edmund Robinson for long discussions on this material. Thanks to Andy Pitts for pointing out the proof that higher-order bisimulation is a congruence.

References

[1] ISO 8807. *LOTOS—A formal description technique based on the temporal ordering of observational behaviour*, 1989.

[2] Samson Abramsky. The lazy lambda calculus. In David Turner, editor, *Declarative Programming*. Addison-Wesley, 1989.

[3] Samson Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51:1–77, 1991.

[4] Matthew Hennessy. A denotational model for higher-order processes. Technical Report 6/92, University of Sussex, 1992.

[5] Douglas Howe. Proving congruence of simulation orderings in functional languages. Unpublished manuscript, 1989.

[6] Douglas J. Howe. Equality in lazy computation systems. In *Proc. LICS 89*, pages 198–203, 1989.

[7] Alan Jeffrey. A fully abstract semantics for a higher-order functional conurrent language. Technical report, University of Sussex, 1994. In preparation.

[8] Robin Milner. Fully abstract semantics of typed λ-calculi. *Theoret. Comput. Sci.*, 4:1–22, 1977.

[9] Eugenio Moggi. Notions of computation and mondad. *Inform. and Computing*, 93:55–92, 1991.

[10] C.-H. L. Ong. Non-determinism in a functional setting. In *Proc. LICS 93*, pages 275–286. IEEE Computer Soc. Press, 1993.

[11] C.-H. Luke Ong. *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College, London University, 1988.

[12] Gordon Plotkin. LCF considered as a programming language. *Theoret. Comput. Sci.*, 5:223–256, 1977.

[13] J. H. Reppy. A higher-order concurrent langauge. In *Proc. SIGPLAN 91*, pages 294–305, 1991.

[14] J. H. Reppy. *Higher-Order Concurrency*. Ph.D thesis, Cornell University, 1992.