Monads as notions of computation

Fabio Zanasi

March 21, 2012
Outline

- Warmup: semantics for typed $\lambda$-calculus.
- Monads and Kleisli triples.
- Monadic metalanguage.
- Sketch: monadic-style translation and direct-style interpretation.
\( \lambda \)-calculus
\( \lambda x \rightarrow \) - syntax

- Types

\[
a ::= \text{a} \mid a'
\]
\[
A ::= a \mid A \to B \mid A \times B
\]

- Terms

\[
x ::= x \mid x'
\]
\[
M ::= x \mid \lambda x : A.M \mid MN \mid < M, N > \mid \pi_1 M \mid \pi_2 N
\]
\(\lambda\rightarrow\) - typing

- Environment

\[\Gamma = \{x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n\}\]

- Type judgement

\[\Gamma \vdash M : A\]

- Typing rules

- Typing derivations = derivations of type judgements by means of the typing rules
\( \lambda \times \) - typing rules

\[
\begin{align*}
\Gamma & \vdash x : A \quad \text{Var} \\
\Gamma, x : A & \vdash M : B \\
\Gamma & \vdash \lambda x : A. M : A \to B \quad \text{Int} \rightarrow \Gamma & \vdash M : A \to B \quad \Gamma & \vdash N : A \\
\Gamma & \vdash MN : B \quad \text{El} \rightarrow \\
\Gamma & \vdash M : A \quad \Gamma & \vdash N : B \\
\Gamma & \vdash < N, M > : A \times B \quad \text{Int}_\times \\
\Gamma & \vdash \pi_1 M : A \quad \Gamma & \vdash \pi_2 M : B \\
\Gamma & \vdash M : A \times B \quad \text{El}_{l\times} \\
\Gamma & \vdash M : A \times B \quad \text{El}_{r\times}
\end{align*}
\]
Definition
A category is called **cartesian closed** \((CC)\) if it has all finite products and exponentials.
Definition

An exponential of objects $A$ and $B$ in a category $C$ is given by an object $B^A$ and a morphism $eval_{AB} : B^A \times A \to B$ with the following UMP.

- For any object $C \in C_{Ob}$ and morphism $f : C \times A \to B$ there is a unique morphism $curry_f : C \to B^A$ such that the following diagram commutes.

\[
\begin{array}{ccc}
B^A \times A & \xrightarrow{eval_{AB}} & B \\
\uparrow cur fy f & & \downarrow f \\
C \times A & & 
\end{array}
\]
Fix a $CC$ category $C$.

- Types are interpreted as objects.
- Environments are interpreted as products of objects.
- Typing derivations are interpreted as morphisms.
\(\lambda \rightarrow\) - semantics

- Types are interpreted as objects.

\[
\| a \| = \mathcal{A}_a \\
\| A \rightarrow B \| = \| B \| \| A \|
\]
\[
\| A \times B \| = \| A \| \times \| B \|
\]

- Environments are interpreted as products of objects.

\[
\| \emptyset \| = 1_C \\
\| \Gamma, x : A \| = \| \Gamma \| \times \| A \|
\]
Typing derivations are interpreted as morphisms.

\[
\begin{align*}
\| \Gamma, x : A \vdash x : A \| &= \pi_2 : (\| \Gamma \| \times \| A \|) \to \| A \| \\
\| \Gamma, x : A \vdash x' : B \| &= \| \Gamma \vdash x' : B \| \circ \pi_1 : (\| \Gamma \| \times \| A \|) \to \| B \| \\
\| \Gamma \vdash \lambda x : A. M : A \to B \| &= \text{curry}_{\| \Gamma, x : A \vdash M : B \|} : \| \Gamma \| \to \| B \|^{\| A \|} \\
\| \Gamma \vdash MN : B \| &= \text{eval}_{AB} \circ < \| \Gamma \vdash M : A \to B \|, \| \Gamma \vdash N : A \| > : \| \Gamma \| \to \| B \| \\
\| \Gamma \vdash < M, N > : A \times B \| &= < \| \Gamma \vdash M : A \|, \| \Gamma \vdash N : B \| > : \| \Gamma \| \to \| A \times B \| \\
\| \Gamma \vdash \pi_1 M : A \| &= \pi_1 \circ \| \Gamma \vdash M : A \times B \| : \| \Gamma \| \to \| A \| \\
\| \Gamma \vdash \pi_2 M : B \| &= \pi_2 \circ \| \Gamma \vdash M : A \times B \| : \| \Gamma \| \to \| B \|
\end{align*}
\]
Monads
Notions of computation

- ‘Pure’ programming language
  - Functions in the mathematical sense: value $\mapsto$ value
- ‘Impure’ programming language
  - Function computation can have side effects: value $\mapsto$ computation
- Different notions of computation:
  - I/O
  - Exception throwing.
  - Modify the state.
Eugenio Moggi’s insight

Pure program  =  $A \to B$

Impure program  =  $A \to TB$

Semantics of $T$ given by a monad.
Notions of computation

- $TA = A + E$ - programs with exceptions
- $TA = A + \{\bot\}$ - possibly non-terminating programs
- $TA = \wp_{fin} A$ - non-deterministic programs
- $TA = (A \times S)^S$ - imperative programs
- $TA = A \times N$ - programs with timers
- $TA = R^{RA}$ - programs with a continuation
Monads

Definition
A monad on $\mathcal{C}$ is a triple $< T, \eta, \mu >$ where $T : \mathcal{C} \to \mathcal{C}$ is an endofunctor, $\eta : \text{Id} \to T$, $\mu : T^2 \to T$ are natural transformations such that the following diagrams commute.
Kleisli triples

Definition
A Kleisli triple in a category $C$ is a triple $< T, \eta, \cdot^* >$ where

- $T : C_{Ob} \to C_{Ob}$ expresses the type of computations,
- $\eta = \{ \eta_A : A \to TA \}_{A \in C_{Ob}}$ expresses the inclusion of values into computations,
- $\cdot^* : (f : A \to TB) \mapsto (f^* : TA \to TB)$ expresses the extension of $f$ to act on computations,

and the following equations hold.

\[
\begin{align*}
\eta^*_A &= Id_{TA} \\
 f^* \circ \eta_A &= f \\
g^* \circ f^* &= (g^* \circ f)^*
\end{align*}
\]
Kleisli Category

Definition
Given a category $\mathcal{C}$, a Kleisli triple $< T, \eta, \cdot^* >$ in $\mathcal{C}$, the Kleisli Category $\mathcal{C}_T$ is defined as follows.

- $\mathcal{C}_T Ob := \mathcal{C} Ob$
- $\text{Hom}_{\mathcal{C}_T}(A, B) := \text{Hom}_{\mathcal{C}}(A, TB)$
- $g \circ f$ in $\mathcal{C}_T := g^* \circ f$ in $\mathcal{C}$
Monadic metalanguage
The idea

- The syntactic counterpart of a category equipped with a monad $T$
- Implementation: a $\lambda$-calculus parametrized to a notion of computation $T$
  - Syntactically: a new type constructor $T$ and associated term constructors $val$ and $let$.
  - Semantically: $CCC$ equipped with a monad to interpret $T$. 
**ML - syntax**

- **New type constructor** $T$
  
  $$A ::= a \mid TA \mid A \rightarrow B \mid A \times B$$

- **New term constructors** `val` and `let`
  
  $$M ::= x \mid \lambda x : A.M \mid MN \mid < M, N > \mid \pi_1 M \mid \pi_2 N \mid valM \mid let \ x = M \ in \ N$$
**ML - typing rules**

Additional typing rules

\[
\begin{align*}
\Gamma \vdash M : A & \quad \text{Int val} \\
\Gamma \vdash \text{val}M : TA & \\
\Gamma \vdash M : TA, x : A \vdash N : TB & \quad \text{Int let} \\
\Gamma \vdash \text{let } x = M \text{ in } N : TB & 
\end{align*}
\]
The axioms for *val* and *let* capture the intended interpretation of the new constructors by matching the equations of a Kleisli triple.

1. \[ \text{let } x = M \text{ in } \text{val}x = M \]  
2. \[ \text{let } x = \text{val}M \text{ in } \text{val}N = N[x := M] \]  
3. \[ \text{let } y = L \text{ in } (\text{let } x = M \text{ in } N) = \text{let } x = (\text{let } y = L \text{ in } M) \text{ in } N \]  

With \( y \) not free in \( N \) in (3).

**Remark:** the axioms are part of an equational theory for *ML* that is fully displayed in *MOGGI* 1991. Here we consider only the ‘non-equational’ fragment.
Interpretation of types.

\[
\| a \| = A_a \\
\| TA \| = T \| A \|
\]

\[
\| A \rightarrow B \| = B \| ^{A} \\
\| A \times B \| = | A | \times | B |
\]
**ML - semantics**

**Interpretation of val**

Intuitively $\text{val}M : TA$ expresses the view of a value $M$ of type $A$ as a ‘special case’ of computation of type $TA$.

By type derivation of $\Gamma \vdash \text{val}M : TA$ we can assume a morphism $\| \Gamma \vdash M : A \|$.

\[
\| \Gamma \vdash \text{val}M : TA \| := \eta_{\| A \|} \circ \| \Gamma \vdash M : A \|
\]
**ML - semantics**

**Interpretation of let**

- Intuitively `let x = M in N` stands for the application of `λx.N` to `M` when `N` and `M` are not just values, but computations.

  \[
  \frac{\Gamma \vdash M : TA \quad \Gamma, x : A \vdash N : TB}{\Gamma \vdash \text{let } x = M \text{ in } N : TB} \quad \text{Int let}
  \]

- Suppose that \( \| \Gamma \vdash M : TA \| \) is \( f : \| \Gamma \| \to \| TA \| \) and \( \| \Gamma, x : A \vdash N : TB \| \) is \( g : \| \Gamma \| \times \| A \| \to \| TB \| \). Then \( \| \Gamma \vdash \text{let } x = M \text{ in } N \| : TB \) should be some morphism \( h : \| \Gamma \| \to \| TB \| \).

- Take the Kleisli composition of \( g \) and \( < ld_{\| \Gamma \| \times} f > \).

- So intuitively we want \( \| \text{let } x = M \text{ in } N \| = g^* \circ f \).
**ML - semantics**

A problem

- Complication given by the presence of a non-empty environment $\Gamma$.

$$
g^* = \|\Gamma, x : A \vdash N : TB\|^* : T(\|\Gamma\| \times \|A\|) \rightarrow T\|B\|
$$

$$
< Id\|\Gamma\| \times f > : \|\Gamma\| \rightarrow \|\Gamma\| \times \|TA\|
$$

- Domain-range mismatch in $g^* \circ < Id\|\Gamma\| \times f >$.

$$
\begin{align*}
\|\Gamma\| & \quad \|\Gamma \vdash M : TA\| \\
\downarrow < Id\|\Gamma\|, \|\Gamma \vdash M : TA\|> & \quad \|\Gamma\| \times \|A\| \quad \eta\|\Gamma\| \times \|A\| \\
\|\Gamma\| \times \|TA\| & \quad ? \\
\|\Gamma\| \times \|TA\| & \quad T(\|\Gamma\| \times \|A\|) \\
\|\Gamma, x : A \vdash N : TB\|^* & \quad \|TB\|
\end{align*}
$$
A solution

We enforce the existence of a natural transformation $t$ from pairs (value, computation) to computations of pairs.

\[ \| \Gamma \| \times \| TA\| \xrightarrow{t_{\| \Gamma \|, \| TA\|}} T(\| \Gamma \| \times \| A\|) \]

This is given under the assumption that the corresponding monad is strong.
Strong monad

Definition

A strong monad on a category $\mathcal{C}$ is a tuple $\langle T, \eta, \mu, t \rangle$ where $\langle T, \eta, \mu \rangle$ is a monad and $t$ is a natural transformation $t_{A,B}: A \times TB \rightarrow T(A \times B)$ such that the following equations hold in $\mathcal{C}$.

\[
T(r_A) \circ t_1 = r_{TA} \\
T(\alpha_{A,B,C}) \circ t_{A \times B,C} = t_{A,B \times C} \circ (Id_A \times t_{B,C} \circ \alpha_{A,B,T,C}) \\
t_{A,B} \circ (A \times \eta_B) = \eta_{A \times B} \\
t_{A,B} \circ (Id_A \times \mu_B) = \mu_{A \times B} \circ T(t_{A,B}) \circ t_{A,TB}
\]

Where $r_A: (1 \times A) \rightarrow A$ and $\alpha_{A,B,C}: (A \times B) \times C \rightarrow A \times (B \times C)$ are natural isomorphisms.

With the focus on $\mathcal{C}_{Ob}$, intuitively they express various forms of commutativity between $\times$ and $T$.

With the focus on $\mathcal{C}_{Arr}$, the first two equations say that $t$ is a tensorial strength for the functor $T$; the last two equations say that $\eta$ and $\mu$ are natural transformations (see MOGGI 1989 for more details).

Interpretation of *let*

By type derivation of $\Gamma \vdash let \ x = M \ in \ N$ we can assume morphisms $\| \Gamma \vdash M : TA \|$ and $\| \Gamma, x : A \vdash N : TB \|$.

$$\| \Gamma \vdash let \ x = M \ in \ N \| \ := \ \| \Gamma, x : A \vdash N : TB \|^* \circ \ t_{\| \Gamma \|, \| A \|} < Id_{\| \Gamma \|}, \| \Gamma \vdash M : TA \| >$$
Notions of computation
Overview

- $TA = A + E$ - programs with exceptions
- $TA = A + \{\perp\}$ - possibly non-terminating programs
- $TA = \mathcal{O}_{\text{fin}} A$ - non-deterministic programs
- $TA = (A \times S)^S$ - imperative programs
- $TA = A \times N$ - programs with timers
- $TA = R^{R^A}$ - programs with a continuation
Identity monad

A Kleisli triple \(<Id, \eta, \cdot^* >\) where \(\eta := \{Id_A\}_{A \in C_{Ob}}\) and \(f^* := f\).

\[
\| val_{Id} M \| = \| M \|
\]

\[
\| let_{Id} x = M \text{ in } N \| = \| N[x := M] \|
\]
Exception monad

A Kleisli triple $< T, \eta, \cdot^* >$ where we fix an ‘exception’ object $E$ and let

$T : A \mapsto A + E$

$\eta := \{ \iota_A : A \to A + E \}_{A \in \text{Cob}}$

$(f^* : (A + E) \to (B + E))(x \in A + E) := \begin{cases} f(x) & \text{if } x \in A \\ x & \text{Otherwise } x \in E \end{cases}$

$A \xrightarrow{\iota_1} A + E \xleftarrow{\iota_2} E$
Exception monad

A Kleisli triple $< T, \eta, \cdot^* >$ where we fix an ‘exception’ object $E$

\[
A \xrightarrow{\iota_1} A + E \xleftarrow{\iota_2} E
\]

\[
\| val_{exc} M \| = \iota_1 \| M \|
\]

\[
\| let_{exc} x = M \text{ in } N \| = \begin{cases} \| N[x := M] \| & \text{if } \| M \| = (\iota_1 : A \to A + E) \\ \| M \| & \text{Oth.} \| M \| = (\iota_2 : E \to A + E) \end{cases}
\]
A Kleisli triple \( < T, \eta, \cdot^* > \) where we fix a ‘state’ object \( S \) and let

\[
T : A \mapsto (A \times S)^S
\]

\[
\eta := \{ \eta_A : a \mapsto (g_a : s \mapsto (a, s)) \}_{A \in \text{Ob} C}
\]

\[
(f^* : (A \times S)^S \to (B \times S)^S)(g_a : S \to (A \times S))(s : S) := f(\pi_1(g_a(s)) : A)(\pi_2(g_a(s)) : S) : (B \times S)
\]
Monadic semantics of programming languages
The monadic metalanguage as a ‘compiled language’

(Simplified picture)

Example:
- CPS transformation can be seen as a monadic style transformation for the call-by-value \( \lambda \)-calculus where \( T \) is the continuation monad.
A toy programming language

**PL - syntax**

*PL* is a call-by-value programming language consisting of a signature \( \Sigma \) of base types \( \tau_1, \ldots, \tau_k \) and commands of the form \( p : \tau_i \rightarrow \tau_j \).

Programs \( e \) over \( \Sigma \) are defined by the following *BNF*:

\[
e ::= x | p(e) | \mu(e) | vale | \text{let } x = e_1 \text{ in } e_2
\]

and the following typing rules.

\[
\begin{align*}
\frac{\Gamma \vdash x : \tau}{\Gamma \vdash x : \tau} & \quad \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash p(e) : \tau_2} & p : \tau_1 \rightarrow \tau_2 \\
\frac{\Gamma \vdash e : \tau}{\Gamma \vdash vale : T\tau} & \quad \frac{\Gamma \vdash \mu e : T\tau}{\Gamma \vdash e : \tau} & \frac{\Gamma \vdash e_1 : \tau_1}{\Gamma, x : \tau_1 \vdash e_2 : \tau_2} & \frac{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2}{\Gamma \vdash e_2 : \tau_2}
\end{align*}
\]

**Remark** Just as for the definition of *ML*, for the sake of simplicity we disregard the equational part of *PL*. 
The monadic-style translation

Consider the metalanguage $ML$ with the signature $\Sigma^\circ$ including the same base types of $\Sigma$ and a function $p : \tau_1 \rightarrow T\tau_2$ for every command $p : \tau_1 \rightarrow \tau_2$ in $\Sigma$. The translation from programs over $\Sigma$ to terms over $\Sigma^\circ$ is defined as follows.

$$
x^\circ := val x
$$

$$
(p(e))^\circ := let x = e^\circ in p(x)
$$

$$
(vale)^\circ := vale^\circ
$$

$$
(\mu e)^\circ := let x = e^\circ in x
$$

$$
(let x = e_1 in e_2)^\circ := let x = e_1^\circ in e_2^\circ
$$
Direct-style interpretation

The idea

- ‘Interpretation without the compilation step’
- Model: not $C$ but the Kleisli Category $C_T$. 
Models of $PL$

A model of $PL$ is a category $\mathcal{C}$ with finite products, a strong monad $T$ and $T$-exponentials, that is, for any $A, B \in \mathcal{C}_{\text{Ob}}$, an exponential $(TB)^A$ of $A$ and $TB$. 

$$TB^A \times A \xrightarrow{\text{eval}^T_{AB}} TB$$

$$\begin{array}{ccc}
TB^A \times A & \xrightarrow{\text{eval}^T_{AB}} & TB \\
\uparrow \text{curry}_f & & \\
C \times A & \xrightarrow{f} & \\
\end{array}$$
Direct-style interpretation (sketch)

- Interpretation of types.
  - Objects in $\mathcal{C}_{T\text{Ob}} = \mathcal{C}_{Ob}$
  - A functional type $\tau_1 \to \tau_2$ is interpreted as a $T$-exponential
    $\left( T \parallel \tau_2 \parallel \right) \parallel \tau_1 \parallel$

- Interpretations of terms.
  - Morphisms in $\mathcal{C}_T$

- Relation with the monadic-style translation

  Given $\Gamma \vdash_{pl} M : A$ in PL and $\Gamma \vdash_{ML} M^\circ : TA$ in ML we have that $\| \Gamma \vdash_{pl} M : A \| = \| \Gamma \vdash_{ML} M^\circ : TA \|$. 

The monadic-style translation

Advantages

- The monadic metalanguage is an interface hiding the categorical model.
- The same metalanguage can describe any notion of computation given as a (strong) monad.
- The monadic-style translation can be flexibly (and monotonically) adapted to compiler programming languages with different features.