An introduction to monadic semantics for computational effects

Fabio Zanasi
ILLC - Universiteit van Amsterdam

April 12, 2012

Background

Denotational semantics of programming languages.

- Programs $\approx$ functions in the mathematical sense: value$\mapsto$value
- Program computation can have side effects: value $\overset{\textit{effects}}{\mapsto}$ value
- Computational effects:
  - Modify the state.
  - I/O
  - Exception throwing.

Outline

- Warmup: semantics for typed λ-calculus.
- Monads and Kleisli triples.
- Monadic metalanguage.
- Sketch: monadic-style translation and direct-style interpretation.

λ-calculus

$\lambda^{\times}_{\rightarrow}$ - syntax

- Types

$$a ::= \mathbf{a} \mid a'$$

$$A ::= a \mid A \rightarrow B \mid A \times B$$

- Terms

$$x ::= \mathbf{x} \mid x'$$

$$M ::= x \mid \lambda x : A.M \mid MN \mid \; < M, N > \; \mid \pi_1 M \mid \pi_2 N$$

$\lambda_{\rightarrow}^{\times}$ - typing

- Environment

$$\Gamma \;=\; \{x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n\}$$

- Type judgement

$$\Gamma \;\vdash\; M : A$$

- Typing rules

$\lambda^{\times}_{\rightarrow}$ - typing rules

$$\frac{}{\Gamma x : A \vdash x : A}\ Var$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightarrow B}\ Int\rightarrow \qquad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ El\rightarrow$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash < N, M >: A \times B}\ Int\times \quad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A}\ El_l\times \quad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}\ El_r\times$$

$\lambda^{\times}_{\rightarrow}$ - semantics

Definition

A category is called <u>cartesian closed</u> (*CC*) if it has all finite products and exponentials.

$\lambda^{\times}_{\rightarrow}$ - semantics

Definition

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

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

$$
\begin{tikzcd}
B^A \times A \arrow[r, "eval_{AB}"] & B \\
C \times A \arrow[u, dotted, "curry_f \times Id_A"] \arrow[ur, "f"'] &
\end{tikzcd}
$$

$\lambda^{\times}_{\rightarrow}$ - semantics

Fix a *CC* category $\mathcal{C}$.

- Types are interpreted as objects.
- Environments are interpreted as products of objects.
- Type judgements are interpreted as morphisms.

$\lambda^{\times}_{\rightarrow}$ - semantics

- Types are interpreted as objects.

$$\begin{array}{rcl} \|a\| & = & A_a \\ \|A \rightarrow B\| & = & \|B\|^{\|A\|} \\ \|A \times B\| & = & \|A\| \times \|B\| \end{array}$$

- Environments are interpreted as products of objects.

$$\begin{array}{rcl} \|\varnothing\| & = & 1_{\mathcal{C}} \\ \|\Gamma, x : A\| & = & \|\Gamma\| \times \|A\| \end{array}$$

$\lambda^{\times}_{\rightarrow}$ - semantics

Type judgements are interpreted as morphisms.

$$
\begin{array}{rcl}
\|\Gamma, x:A \vdash x:A\| & = & \pi_2 : (\|\Gamma\| \times \|A\|) \rightarrow \|A\| \\
\|\Gamma, x:A \vdash x':B\| & = & \|\Gamma \vdash x':B\| \circ \pi_1 : (\|\Gamma\| \times \|A\|) \rightarrow \|B\| \\
\|\Gamma \vdash \lambda x:A.M : A \rightarrow B\| & = & curry_{\|\Gamma, x:A \vdash M:B\|} : \|\Gamma\| \rightarrow \|B\|^{\|A\|} \\
\|\Gamma \vdash MN : B\| & = & eval_{AB} \circ <\|\Gamma \vdash M : A \rightarrow B\|, \|\Gamma \vdash N:A\|>: \\
 & & \|\Gamma\| \rightarrow \|B\| \\
\|\Gamma \vdash <M,N>: A \times B\| & = & <\|\Gamma \vdash M:A\|, \|\Gamma \vdash N:B\|>: \|\Gamma\| \rightarrow \|A \times B\| \\
\|\Gamma \vdash \pi_1 M : A\| & = & \pi_1 \circ \|\Gamma \vdash M : A \times B\| : \|\Gamma\| \rightarrow \|A\| \\
\|\Gamma \vdash \pi_2 M : B\| & = & \pi_2 \circ \|\Gamma \vdash M : A \times B\| : \|\Gamma\| \rightarrow \|B\|
\end{array}
$$

Monads

Eugenio Moggi's insight

$$\begin{aligned} \text{Pure program} &= A \to B \\ \text{Impure program} &= A \to TB \end{aligned}$$

Semantics of T given by a monad.

Computational effects

- $TA = A + E$ - programs with exceptions
- $TA = A + \{\perp\}$ - 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
- $TA = R^{R^A}$ - 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, $\eta : Id \to T$, $\mu : T^2 \to T$ are natural transformations such that the following diagrams commute.

$$
\begin{tikzcd}
T^3 \arrow[r, "\mu_T"] \arrow[d, "T\mu"] & T^2 \arrow[d, "\mu"'] \\
T^2 \arrow[r, "\mu"] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
T \arrow[r, "\eta_T"] \arrow[dr, "Id_T"'] & T^2 \arrow[d, "\mu"'] & T \arrow[l, "\eta_T"'] \arrow[dl, "Id_T"] \\
& T &
\end{tikzcd}
$$

Kleisli triples

Definition

A Kleisli triple in a category $\mathcal{C}$ is a triple $< T, \eta, \cdot^\star >$ where

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

and the following equations hold.

$$\begin{array}{rcl} \eta_A^\star & = & Id_{TA} \\ f^\star \circ \eta_A & = & f \\ g^\star \circ f^\star & = & (g^\star \circ f)^\star \end{array}$$

Kleisli Category

Definition

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

- $\mathcal{C}_{\mathcal{T}\,Ob} \;:=\; \mathcal{C}_{Ob}$
- $Hom_{\mathcal{C}_T}(A, B) \;:=\; Hom_{\mathcal{C}}(A, TB)$
- $g \circ f$ in $\mathcal{C}_T \;:=\; g^{\star} \circ f$ in $\mathcal{C}$

Monadic metalanguage

The idea

- The syntactic counterpart of a category equipped with a monad T
- Implementation: a λ-calculus parameterized to an effect 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

$$A ::= a \mid TA \mid A \to B \mid A \times B$$

- New term constructors *val* and *let*

$$M ::= x \mid \lambda x : A.M \mid MN \mid < M, N > \mid \pi_1 M \mid \pi_2 N \mid val M \mid let\ x = M\ in\ N$$

ML - typing rules

Additional typing rules

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathit{val} M : TA}\ \mathit{Int\ val} \qquad \frac{\Gamma \vdash M : TA \quad \Gamma, x : A \vdash N : TB}{\Gamma \vdash \mathit{let}\ x = M\ \mathit{in}\ N : TB}\ \mathit{Int\ let}$$

ML - axioms

- The axioms for *val* and *let* capture the intended interpretation of the new constructors by matching the equations of a Kleisli triple.

$$
\begin{aligned}
\mathit{let}\ x = M\ \mathit{in}\ \mathit{val}\,x &= M && (1)\\
\mathit{let}\ x = \mathit{val}\,M\ \mathit{in}\ \mathit{val}\,N &= N[x := M] && (2)\\
\mathit{let}\ y = L\ \mathit{in}\ (\mathit{let}\ x = M\ \mathit{in}\ N) &= \mathit{let}\ x = (\mathit{let}\ y = L\ \mathit{in}\ M)\ \mathit{in}\ N && (3)
\end{aligned}
$$

  With y not free in N in (3).
- **Remark**: A complete definition of *ML* includes equation judgements and inference rules. The three axioms can be formulated as equations that are derivable in *ML*. See MOGGI 1991 for more details.

ML - semantics

Interpretation of types.

$$
\begin{aligned}
\|a\| &= A_a \\
\|TA\| &= T\|A\| \\
\|A \to B\| &= \|B\|^{\|A\|} \\
\|A \times B\| &= \|A\| \times \|B\|
\end{aligned}
$$

ML - semantics

Interpretation of *val*

Intuitively $val M : TA$ expresses the view of a value M of type A as a 'special case' of computation of type TA.
By type derivation of $\Gamma \vdash val M : TA$ we can assume a morphism $\|\Gamma \vdash M : A\|$.

$$\|\Gamma \vdash val M : TA\| := \eta_{\|A\|} \circ \|\Gamma \vdash M : A\|$$

$$\begin{tikzcd}
 & \|TA\| \\
\|\Gamma\| \arrow[ur, "\|\Gamma \vdash val M:TA\|"] \arrow[r, "\|\Gamma \vdash M:A\|"] & \|A\| \arrow[u, "\eta_{\|A\|}"']
\end{tikzcd}$$

ML - semantics

Interpretation of *let*

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

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

- 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 $\|\Gamma \vdash let\ x = M\ in\ N\| : TB$ should be some morphism $h : \|\Gamma\| \to \|TB\|$.
- Take the Kleisli composition of g and $< Id_{\|\Gamma\|} \times f >$.
- So intuitively we want $\|let\ x = M\ in\ N\| \ = \ g^{\star} \circ f$.

ML - semantics

A problem

- Complication given by the presence of a non-empty environment Γ.

$$
\begin{array}{rcl}
g^\star = \|\Gamma, x : A \vdash N : TB\|^\star & : & T(\|\Gamma\| \times \|A\|) \to T\|B\| \\
< Id_{\|\Gamma\|} \times f > & : & \|\Gamma\| \to \|\Gamma\| \times \|TA\|
\end{array}
$$

- Domain-range mismatch in $g^\star \circ < Id_{\|}\Gamma\| \times f >$.

$$
\begin{tikzcd}
\|\Gamma\| \arrow[r, "\|\Gamma \vdash M:TA\|"] \arrow[d, "<Id_{\|\Gamma\|}{,}\|\Gamma \vdash M:TA\|>"] & \|TA\| & \|\Gamma\| \times \|A\| \arrow[dl, "\eta_{\|\Gamma\| \times \|A\|}"'] \arrow[d, "\|\Gamma{,}x:A \vdash N:TB\|"] \\
\|\Gamma\| \times \|TA\| \arrow[r, "?"'] & T(\|\Gamma\| \times \|A\|) \arrow[r, "\|\Gamma{,}x:A \vdash N:TB\|^\star"'] & \|TB\|
\end{tikzcd}
$$

Strong monad

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 <u>strong</u>.

Strong monad

Definition

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

$$
\begin{array}{rcl}
T(r_A) \circ t_1 & = & r_{TA} \\
T(\alpha_A, B, C) \circ t_{A \times B, C} & = & t_{A, B \times C} \circ (Id_A \times t_{B,C} \circ \alpha_{A,B,TC} \\
t_{A,B} \circ (A \times \eta_B) & = & \eta_{A \times B} \\
t_{A,B} \circ (Id_A \times \mu_B) & = & \mu_{A \times B} \circ T(t_{A,B}) \circ t_{A,TB}
\end{array}
$$

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

See MOGGI 1989 for more details and motivation.

ML - semantics

Interpretation of *let*

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

$$\begin{aligned} \|\Gamma \vdash \mathit{let}\ x = M\ \mathit{in}\ N\| \quad := \quad & \|\Gamma, x : A \vdash N : TB\|^{\star} \circ \\ & t_{\|\Gamma\|,\|A\|} \circ < Id_{\|\Gamma\|}, \|\Gamma \vdash M : TA\| > \end{aligned}$$

$$\begin{tikzcd}
\|\Gamma\| \arrow[r, "\|\Gamma \vdash M:TA\|"] \arrow[d, "<Id_{\|\Gamma\|},\|\Gamma \vdash M:TA\|>"] & \|TA\| & \|\Gamma\| \times \|A\| \arrow[dl, "\eta_{\|\Gamma\| \times \|A\|}"'] \arrow[d, "\|\Gamma,x:A \vdash N:TB\|"] \\
\|\Gamma\| \times \|TA\| \arrow[r, "t_{\|\Gamma\|,\|A\|}"'] & T(\|\Gamma\| \times \|A\|) \arrow[r, "\|\Gamma,x:A \vdash N:TB\|^{\star}"'] & \|TB\|
\end{tikzcd}$$

Computational effects

Overview

- $TA = A + E$ - programs with exceptions
- $TA = A + \{\perp\}$ - 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
- $TA = R^{R^A}$ - programs with a continuation

Identity monad

Identity monad

A Kleisli triple $< Id, \eta, \cdot^{\star} >$ where $\eta \ := \ \{Id_A\}_{A \in \mathcal{C}_{Ob}}$ and $f^{\star} \ := \ f$.

$$
\begin{aligned}
\|\Gamma \vdash val_{Id} M\| &= \|\Gamma \vdash M\| \\
\|\Gamma \vdash let_{Id}\ x = M\ in\ N\| &= \|\Gamma \vdash N[x := M]\|
\end{aligned}
$$

Exception monad

Exception monad

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

$$T : A \mapsto A + E$$

$$\eta \ := \ \{\iota_A : A \to A + E\}_{A \in \mathcal{C}_Ob}$$

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

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

Exception monad

Exception monad

A Kleisli triple $< T, \eta, \cdot^{\star} >$ where we fix an 'exception' object E

$$
\begin{tikzcd}
 & B+E & \\
A \arrow[ur, "f"] \arrow[r, "\iota_1"] & A+E \arrow[u, dotted, "{[f,g]}"] & E \arrow[ul, "g"'] \arrow[l, "\iota_2"']
\end{tikzcd}
$$

$$
\begin{aligned}
\| \Gamma \vdash val_{exc} M \| &= \iota_1 \circ \| \Gamma \vdash M \| \\
\| \Gamma \vdash let_{exc}\ x = M\ in\ N \| &= [\, \| \Gamma \vdash N[x := M] \|, \iota_2]
\end{aligned}
$$

State monad

State monad

A Kleisli triple $< T, \eta, \cdot^\star >$ where we fix a 'state' object S and let

$$T : A \mapsto (A \times S)^S$$

$$\eta \ := \ \{\eta_A : a \mapsto (g_a : s \mapsto (a, s))\}_{A \in \mathcal{C}_{Ob}}$$

$$\begin{aligned}(f^\star : (A \times S)^S \to (B \times S)^S)(g_a : S \to (A \times S))(s : S) \ :=& \\ f(\pi_1(g_a(s)) : A)(\pi_2(g_a(s)) : S) : (B \times S)&\end{aligned}$$

Monadic semantics of programming languages

The monadic metalanguage as a ‘compiled language’

(Simplified picture)

$$
\begin{tikzcd}
PL \arrow[r, "MST"] \arrow[dr, "DSI"'] & ML \arrow[d, "\|\cdot\|"] \\
 & \textit{Categorical model}
\end{tikzcd}
$$

Example:

- *CPS* transformation can be seen as a monadic style transformation for the call-by-value λ-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 Σ of base types $\tau_1, \ldots, \tau_k$ and commands of the form $p : \tau_i \to \tau_j$. Programs e over Σ are defined by the following *BNF*

$$e \ ::= \ x \mid p(e) \mid \mu(e) \mid val\, e \mid let\ x = e_1\ in\ e_2$$

and the following typing rules.

$$\frac{}{\Gamma x : \tau \ \vdash \ x : \tau} \qquad \frac{\Gamma \ \vdash \ e : \tau_1}{\Gamma \ \vdash \ p(e) : \tau_2}\ p : \tau_1 \to \tau_2$$

$$\frac{\Gamma \ \vdash \ e : \tau}{\Gamma \ \vdash \ val\, e : T\tau} \quad \frac{\Gamma \ \vdash \ \mu e : T\tau}{\Gamma \ \vdash \ e : \tau} \quad \frac{\Gamma \ \vdash \ e_1 : \tau_1 \quad \Gamma, x : \tau_1 \ \vdash \ e_2 : \tau_2}{\Gamma \ \vdash \ let\ x = e_1\ in\ e_2 : \tau_2}$$

Remark Just as for the definition of *ML*, for the sake of simplicity we disregard the equational part of *PL*.

The monadic-style translation

The monadic-style translation

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

$$
\begin{array}{rcl}
x^\circ & := & val x \\
(p(e))^\circ & := & let\ x = e^\circ\ in\ p(x) \\
(val e)^\circ & := & val e^\circ \\
(\mu e)^\circ & := & let\ x = e^\circ\ in\ x \\
(let\ x = e_1\ in\ e_2)^\circ & := & let\ x = e_1^\circ\ in\ e_2^\circ
\end{array}
$$

Direct-style interpretation

The idea

- 'Interpretation without the compilation step'
- Model: not $\mathcal{C}$ but the Kleisli Category $\mathcal{C}_T$.

PL - semantics

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 $A, B \in \mathcal{C}_{Ob}$, an exponential $(TB)^A$ of A and TB.

$$
\begin{tikzcd}
TB^A \times A \arrow[r, "eval^T_{AB}"] & TB \\
C \times A \arrow[u, dotted, "curry_f \times Id_A"] \arrow[ur, "f"'] &
\end{tikzcd}
$$

Direct-style interpretation (sketch)

- Interpretation of types.
  - Objects in $\mathcal{C}_{\mathcal{T}Ob} = \mathcal{C}_{Ob}$
  - A functional type $\tau_1 \to \tau_2$ is interpreted as a T-exponential $(T\|\tau_2\|)^{\|\tau_1\|}$
- 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 computational effect given as a (strong) monad.
- The monadic-style translation can be flexibly (and monotonically) adapted for source programming languages with different features.