A Metalanguage for Guarded Iteration

Sergey Goncharov\textsuperscript{a} Christoph Rauch\textsuperscript{a} Lutz Schröder\textsuperscript{a}

University of Minho, Braga, July 31, 2018

\textsuperscript{a}Friedrich-Alexander-Universität Erlangen-Nürnberg
Prelude: Guarded Iteration
Monads formalize generalized functions $f : X \rightarrow TY$, like nondeterministic (with $T = \mathcal{P}X$) or partial (with $TX = X + 1$)\(^1\)

\(^1\)Moggi 1991, Notions of Computation and Monads
\(^2\)Plotkin and Power 2002, Notions of Computation Determine Monads
Monads formalize generalized functions $f : X \rightarrow TY$, like nondeterministic (with $T = \mathcal{P}X$) or partial (with $TX = X + 1$)

- $T$ is a type constructor, together with operations $\eta : X \rightarrow TX$ (unit) and $(f : X \rightarrow TY) \mapsto (f^* : TX \rightarrow TY)$ (lifting), inducing the Klesili category of $T$ under

  $$\text{id} = \eta : X \rightarrow TX \quad f \diamond g = (f : Y \rightarrow TZ)^* (g : X \rightarrow TY)$$

In Haskell’s point-full notation: do $x \leftarrow p; f(x) = f^*(p)$

---

1Moggi 1991, Notions of Computation and Monads
2Plotkin and Power 2002, Notions of Computation Determine Monads
Monads for Computations in One Slide

- Monads formalize generalized functions \( f : X \rightarrow TY \), like nondeterministic (with \( T = \mathcal{P}X \)) or partial (with \( TX = X + 1 \))\(^1\)

- \( T \) is a type constructor, together with operations \( \eta : X \rightarrow TX \) (unit) and \((f : X \rightarrow TY) \mapsto (f^* : TX \rightarrow TY)\) (lifting), inducing the \textit{Klesili category} of \( T \) under

\[
\text{id} = \eta : X \rightarrow TX \quad f \circ g = (f : Y \rightarrow TZ)^* (g : X \rightarrow TY)
\]

In Haskell’s point-full notation: \( \text{do } x \leftarrow p; f(x) = f^*(p) \)

- Duality of operations and effects\(^2\): e.g. for \( T = \mathcal{P} \),

\( \text{toss} = \{\text{head, tail}\} \)

\[
p + q = \text{do } x \leftarrow \text{toss}; \text{if } (x = \text{head}) \text{ then } p \text{ else } q.
\]

In this sense \( T_\Sigma \) extends \( T \) with \( \Sigma \)-operations, e.g. for \( \Sigma = A \times - \):

\[
a . p = \text{do } (\text{action}_a : 1 \rightarrow T_\Sigma 1); p
\]

\(^1\) Moggi 1991, Notions of Computation and Monads
\(^2\) Plotkin and Power 2002, Notions of Computation Determine Monads
Abstract guardedness for a monad $T$ is a relation between Kleisi morphisms $f : X \to TY$ and summands $\sigma : Y' \hookrightarrow Y$ satisfying

(\text{trv}) \quad \frac{f : X \to TY}{(T \text{in}_1) f : X \to_{\text{in}_2} T(Y + Z)}

(\text{sum}) \quad \frac{f : X \to_{\sigma} TZ \quad g : Y \to_{\sigma} TZ}{[f, g] : X + Y \to_{\sigma} TZ}

(\text{cmp}) \quad \frac{f : X \to_{\text{in}_2} T(Y + Z) \quad g : Y \to_{\sigma} TV \quad h : Z \to TV}{[g, h]^* f : X \to_{\sigma} TV}

where $f : X \to_{\sigma} TY$, equivalently $f \in \text{Hom}_{\sigma}(X, TY)$, means that $f$ and $\sigma$ are in the relation
Abstract Guardedness on Monads

A monad is **guarded Elgot** if it supports partial iteration operator sending each \( f : X \to T(Y + X) \) to \( f^\dagger : X \to TY \) satisfying the fixpoint law

\[
f^\dagger = [\eta, f^\dagger]^* f
\]

and other laws of iteration\(^3\)

**Example:** \( TX = (X \times Nat^*) \cup Nat^\omega \), equivalently, \( TX \) is a final \((X + Nat \times -)\)-coalgebra

\( TX \) contains

- pairs \((x, \tau)\) of a result \( x \in X \) and a finite trace \( \tau \in Nat^* \), and
- infinite traces \( \pi \in Nat^\omega \)

\(^3\)Bloom and Ésik 1993, Iteration theories: The equational logic of iterative processes
A Monad of (In)Finite Traces

- The unit of $TX = (X \times \text{Nat}^*) \cup \text{Nat}^\omega$ sends $x$ to $(x, \langle \rangle)$
- Given $f : X \rightarrow TY$,
  \[ f^*(x, \tau) = \begin{cases} 
  (y, \tau + \tau') & \text{if } f(x) = (y, \tau'), \\
  \tau + \pi & \text{if } f(x) = \pi,
  \end{cases} \quad f^*(\pi) = \pi. \]
- $f : X \rightarrow_{\text{inr}} (Y + Z) \times \text{Nat}^* \cup \text{Nat}^\omega$ if for every $x \in X$,
  \[ f(x) \in Z \times \text{Nat}^* \quad \text{implies} \quad f(x) \in Z \times \text{Nat}^+. \]
- Given $f : X \rightarrow_{\text{inr}} T(Y + X) = (Y + X) \times \text{Nat}^* \cup \text{Nat}^\omega$,
  \[ f^\dagger(x) = \begin{cases} 
  (y, \tau_1 + \cdots + \tau_n) & \text{if } f(x) = (\text{in}_2 x_1, \tau_1), \ldots, f(x_n) = (\text{in}_1 y, \tau_n), \\
  \tau_1 + \cdots + \tau_{n-1} + \pi & \text{if } f(x) = (\text{in}_2 x_1, \tau_1), \ldots, f(x_n) = \pi, \\
  \tau_1 + \tau_2 + \cdots & \text{if } f(x) = (\text{in}_2 x_1, \tau_1), f(x_1) = (\text{in}_1 x_2, \tau_2), \ldots
  \end{cases} \]
The Metalanguage
Metalanguages for (Guarded) Iteration: Motivation

- Guardedness is a fundamental notion: Just like Moggi’s computational metalanguage is a metalanguage of abstract effects, the metalanguage for guarded iteration is a metalanguage of abstract guardedness.

- The metalanguage for guarded iteration can be used as a ‘core programming language’ for effects associated with monads. The stock of examples is growing: various process semantic domains, hybrid monads, etc.
Geron and Levy\textsuperscript{4} observed that

- modelling iteration directly would amount to syntax like

\[
\text{return inr . . . inr inl . . .}
\]

which is like using De Bruijn indexes instead of variables

- they also proposed to use labels to index coproduct summands in

\[
f : X \rightarrow T(\sum_i X_i),
\]

so as to be able to point the branch in which to iterate

Here, we assume labels = exceptions, for they can be uniformly used in three constructs

<table>
<thead>
<tr>
<th>exception raising</th>
<th>exception handling</th>
<th>iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{raise}_e \nu</td>
<td>\text{handle } x \text{ in } p \text{ with } q</td>
<td>\text{handleit } x = \nu \text{ in } p</td>
</tr>
</tbody>
</table>

\textsuperscript{4}Geron and Levy 2016, Iteration and labelled iteration
Quick Example

```plaintext
handleit e = *) in
  handle u in
    (print ("think of a number") & raise u *)
  with
    (do y ← random();
     z ← read();
     if (y = z) then ret *) else raise e *)
```
Syntax

Types:

\[ A, B, \ldots ::= C \mid 0 \mid 1 \mid A + B \mid A \times B \quad (C \in \text{Base}) \]

Signatures:

- value signature \( \Sigma_v \) of \( f : A \rightarrow B \) (e.g. \( + : \text{Nat} \times \text{Nat} \rightarrow \text{Nat} \))
- effect signature \( \Sigma_c \) of \( f : A \rightarrow B[C] \) (e.g. put : \( \text{Nat} \rightarrow 0[1] \))

Value and Computation Term Judgements:

\[ \Gamma \vdash_v v : A \quad \text{and} \quad \Delta \mid \Gamma \vdash_c p : A \]

where

\[ \Gamma = (x_1 : A_1, \ldots, x_n : A_n) \quad \text{(variable context)} \]
\[ \Delta = (e_1 : E_1^{\alpha_1}, \ldots, e_m : E_m^{\alpha_m}) \quad \text{(exception context)} \]

and \( \alpha_i \in \{g, u\} \) indicate (un-)guardedness
Some Derivation Rules

\[
\begin{align*}
e : E^g & \text{ in } \Delta \quad f : A \rightarrow 0[1] \in \Sigma_c \quad \Gamma \vdash_{\nu} p : A \quad \Gamma \vdash_{\nu} q : E \\
\Delta \mid \Gamma & \vdash_{c} f(p) \& \text{raise}_e q : D
\end{align*}
\]

\[
\begin{align*}
\Delta, e : E^g \mid \Gamma & \vdash_{c} p : A \quad \Delta' \mid \Gamma, e : E \vdash_{c} q : A \quad |\Delta| = |\Delta'| \\
\Delta \mid \Gamma & \vdash_{c} \text{handle } e \text{ in } p \text{ with } q : A
\end{align*}
\]

\[
\begin{align*}
e : E^u & \text{ in } \Delta \quad \Gamma \vdash_{\nu} q : E \\
\Delta \mid \Gamma & \vdash_{c} \text{raise}_e q : D
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash_{\nu} p : E \quad \Delta, e : E^g \mid \Gamma, e : E & \vdash_{c} q : A \\
\Delta \mid \Gamma & \vdash_{c} \text{handleit } e = p \text{ in } q : A
\end{align*}
\]
Generic Denotational Semantics
Typing the Semantics

Types:

\[ 0 = \emptyset, \quad 1 = 1, \quad A + B = A + B, \quad A \times B = A \times B. \]
\[ \Gamma = A_1 \times \ldots \times A_n \quad \text{for} \quad \Gamma = (x_1 : A_1, \ldots, x_n : A_n) \]
\[ \Delta = E_1 + \ldots + E_m \quad \text{for} \quad \Delta = (e_1 : E_1^{\alpha_1}, \ldots, e_m : E_m^{\alpha_m}) \]

Signatures:

\[ [f] \in \text{Hom}(A, B) \quad \text{for} \quad f : A \rightarrow B \in \Sigma_v \]
\[ [f] \in \text{Hom}_{\text{inr}}(A, T(B + C)) \quad \text{for} \quad f : A \rightarrow B[C] \in \Sigma_c \]

Terms:

\[ [\Gamma \vdash_{\nu} \nu : A] \in \text{Hom}(\Gamma, A) \quad [\Delta \mid \Gamma \vdash_{\epsilon} \rho : A] \in \text{Hom}_{! + \sigma_{\Delta}}(\Gamma, T(A + \Delta)) \]

where \( \sigma_{\Delta} : \Delta' \hookrightarrow \Delta \) corresponds to removal of all unguarded exceptions \( e : E^u \) from \( \Delta \)
Assumptions on the Model

The underlying model consists of a category $\mathbf{C}$ and a monad $\mathbb{T}$ on $\mathbf{C}$, such that

- $\mathbf{C}$ is **distributive**, i.e. essentially supports a distributivity isomorphism
  \[ \text{dist} : A \times (B + C) \cong A \times B + A \times C \]

- $\mathbb{T}$ is strong, i.e. equipped with **tensorial strength**
  \[ \tau_{A,B} : A \times TB \to T(A \times B) \]

- $\mathbb{T}$ supports guarded iteration, and additionally validated the rule
  \[
  \text{(str)} \quad \frac{f : X \to \sigma TY}{\tau (\text{id}_Z \times f) : Z \times X \to \text{id} \times \sigma T(Z \times Y)}
  \]
  (this yields **strong iteration**:
  \[
  f : W \times X \to \text{inr} T(Y + X)
  \]
  \[
  f^\dagger = (T(\text{snd} + \text{id})(T \text{dist})\tau \langle \text{fst}, f \rangle)^\dagger : W \times X \to TY
  \]

- Function spaces $A \to A \Delta B$ can be added at little cost

11/16
Operational Semantics and Adequacy
Geron and Levy\textsuperscript{5} elaborated the maybe monad \(\textit{+1}\) on \textbf{Set} as the simplest monad for unguarded iteration. Incidentally, it is an initial Elgot monad on \textbf{Set}\textsuperscript{6}.

We elaborate \(TX = (X \times \text{Nat}^*) \cup \text{Nat}^\omega\) as the simplest monad for properly guarded iteration on \textbf{Set}.

- The only base type is \textit{Nat}
- Value signature contains arithmetic operations
- Effect signature contains only \(\textit{put} : \text{Nat} \rightarrow 0[1]\)

\textsuperscript{5}Geron and Levy 2016, Iteration and labelled iteration
\textsuperscript{6}Goncharov, Rauch, and Schröder 2015, Unguarded recursion on coinductive resumptions
Big-Step Operational Semantics

Values, Computations, Terminals:

\[ v, w ::= x | \ast | 0 | \text{succ } v | \text{inl } v | \text{inr } v | \langle v, w \rangle | \ldots \]

\[ p, q ::= \text{ret } v | \text{pred } v | \text{put } v | \text{raise}_x v | \text{put } v \& \text{raise}_x w | \ldots \]

\[ t ::= \text{ret } v, \tau | \text{raise}_x v, \tau | \pi \quad (\tau \in \text{Nat}^*, \pi \in \text{Nat}^{\omega}) \]

Some Rules:

\[ \text{put } v \& \text{raise}_x w \Downarrow \text{raise}_x w, \langle v \rangle \]

\[ v_0 = v \quad q[v_0/x] \Downarrow \text{raise}_x v_1, \tau_1 \ldots \quad q[v_{n-1}/x] \Downarrow t, \tau_n \]

\[ \text{handleit } x = v \text{ in } q \Downarrow t, \tau_1 + \cdots + \tau_n \]

\[ v_0 = v \quad q[v_0/x] \Downarrow \text{raise}_x v_1, \tau_1 \ldots \quad q[v_{n-1}/x] \Downarrow \pi \]

\[ \text{handleit } x = p \text{ in } q \Downarrow \tau_1 + \cdots + \tau_{n-1} + \pi \]

\[ v_0 = v \quad q[v_0/x] \Downarrow \text{raise}_x v_1, \tau_1 \quad q[v_1/x] \Downarrow \text{raise}_x v_2, \tau_2 \ldots \]

\[ \text{handleit } x = p \text{ in } q \Downarrow \tau_1 + \tau_2 + \cdots \]
The Adequacy Theorem

**Theorem (Adequacy):** Let $\Delta \vdash_c p : B$. Then,

1. If $p \Downarrow \text{ret } \nu, \tau$ then $[\Delta \vdash_c p : B] = (\text{in}_1 \nu, \tau) \in (B + \Delta) \times \text{Nat}^*$
2. If $p \Downarrow \text{raise}_x \nu, \tau$ and $x : E^g$ is in $\Delta$ then
   \[ [\Delta \vdash_c p : B] = (\text{in}_2 \text{in}_x \nu, \tau) \in (B + \Delta) \times \text{Nat}^+ \]
3. If $p \Downarrow \text{raise}_x \nu, \tau$ and $x : E^u$ is in $\Delta$ then
   \[ [\Delta \vdash_c p : B] = (\text{in}_2 \text{in}_x \nu, \tau) \in (B + \Delta) \times \text{Nat}^* \]
4. If $p \Downarrow \pi$, then $[\Delta \vdash_c p : B] = \pi \in \text{Nat}^\omega$
• The metalanguage for guarded iteration provides an extensible platform for programming with guarded iteration
• More concrete monads $\Rightarrow$ more concrete operational semantics and more adequacy theorems
• Further case study: a monad for hybrid computation with guardedness as progressiveness$^7$
• Hoare logic for guarded iteration:

$\{ \phi \} \ x \leftarrow \ p \ \{ \psi \}$

what are these?

---

$^7$Goncharov, Jakob, and Neves 2018, A Semantics for Hybrid Iteration
References


