A Metalanguage for Guarded Iteration

Sergey Goncharov  Christoph Rauch  Lutz Schröder

Friedrich-Alexander-Universität Erlangen-Nürnberg

ICTAC 2018, October 15-19, Stellenbosch
Two Flavors of Computations

Domain theory

- Computations are identified with final result (if any)
- Programs either terminate with a value, or they converge
- Extensional paradigm†
Two Flavors of Computations

Domain theory

- Computations are identified with final result (if any)
- Programs either terminate with a value, or they diverge
- Extensional paradigm\(^\dagger\)

Process algebra

- Computations are processes unfolding in time
- Behavioural semantics, potentially disregarding final result
- Intensional paradigm\(^\dagger\)

\(^\dagger\)Abramsky 2014, Intensionality, Definability and Computation
Two Flavors of Computations

Domain theory

- Computations are identified with final result (if any)
- Programs either terminate with a value, or they diverge
- Extensional paradigm

Process algebra

- Computations are processes unfolding in time
- Behavioural semantics, potentially disregarding final result
- Intensional paradigm

Here

Unified semantic framework for iterative computations

†Abramsky 2014, Intensionality, Definability and Computation
General idea
Ensure progress or productivity in (co)recursive definitions

In process algebra

- Recursive process specification $X = t$ guarded if every occurrence of $X$ in $t$ is under an action
- E.g. in CCS under bisimulation semantics: guarded recursive specifications have unique solutions [Milner, 1989]
- For example,

  $$P = a.P$$

  keeps performing the action $a$
**Guardedness**

**General idea**
Ensure progress or productivity in (co)recursive definitions

**More recently: in (co)programming**

- Guardedness analysis in Coq for corecursive definitions, proofs by corecursion: Do corecursive calls occur under constructors? [Coquand, 1994]
- Guarded recursion by typing/functorial guardedness [Birkedal and Møgelberg, 2013], [Milius and Litak, 2017], [Clouston, Bizjak, Grathwohl, and Birkedal, 2015] and many others

**Abstract guardedness**
Unifying notion both for **guarded recursion** and for **guarded iteration** via guarded traced monoidal categories [Goncharov and Schröder, 2018]
Inference rules

\[
\begin{align*}
\text{in}_1 \ f : X & \rightarrow \text{in}_2 \ Y + Z \\
\text{in}_2 \ f : X & \rightarrow Y + Z
\end{align*}
\]

\[
\begin{align*}
[f, g] : X + Y & \rightarrow \sigma Z \\
[f, g] : X + Y & \rightarrow \sigma Z
\end{align*}
\]

Definition (Guarded co-Cartesian category)
A co-Cartesian category \( \mathbf{C} \) equipped with distinguished subsets \( \text{Hom}_\sigma(X, Y) \subseteq \text{Hom}(X, Y) \) of partially guarded morphisms for \( A, B \in |\mathbf{C}| \), and any summand \( \sigma : Y_1 \rightarrow Y_1 + Y_2 \cong Y \) satisfying the rules above is called guarded \( (f : X \rightarrow \sigma Y \text{ means } f \in \text{Hom}_\sigma(X, Y)) \)
Inference rules

\[
\begin{align*}
\text{f : } X & \rightarrow Y \\
\text{in}_1 \text{f : } X & \rightarrow_2 Y + Z \\
\text{f : } X & \rightarrow_\sigma Z \\
\text{g : } Y & \rightarrow_\sigma Z \\
[f, g] : X + Y & \rightarrow_\sigma Z \\
[\text{g, h}] \text{f : } X & \rightarrow_\sigma V
\end{align*}
\]

**Definition (Guarded co-Cartesian category)**

A co-Cartesian category \( \mathbf{C} \) equipped with distinguished subsets \( \text{Hom}_\sigma(X, Y) \subseteq \text{Hom}(X, Y) \) of partially guarded morphisms for \( A, B \in |\mathbf{C}| \), and any summand \( \sigma : Y_1 \rightarrow Y_1 + Y_2 \simeq Y \) satisfying the rules above is called **guarded** \( (f : X \rightarrow_\sigma Y \text{ means } f \in \text{Hom}_\sigma(X, Y)) \)
(Abstractly) Guarded Symmetric Monoidal Categories
Monads for Computations

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

\[\text{Duality of operations and effects; e.g. for } T = \mathcal{P}, \text{toss} \text{ head, tail } \]
\[\text{do } x \triangleq \text{toss}; \text{if } p(x) = \text{head} \text{ then } p \text{ else } q.\]

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

\[\text{do } p(a) : 1 \rightarrow T \Sigma 1 \text{ q}; p : M\text{oggi 1991, Notions of Computation and Monads}\]
\[\ddagger\text{Plotkin and Power 2002, Notions of Computation Determine Monads}\]
Monads for Computations

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

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

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

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

---

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

- \( T \) is a type constructor, plus \( \eta : X \to TX \) (unit) and \( (f : X \to TY) \mapsto (f^* : TX \to TY) \) (lifting), inducing the Klesili category of \( T \):

\[
\begin{align*}
\text{id} &= \eta : X \to TX \\
f \diamond g &= (f : Y \to TZ)^* (g : X \to TY)
\end{align*}
\]

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

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

\[
toss = \{ \text{head}, \text{tail} \}
\]

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

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

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

\( ^\dagger \) Moggi 1991, Notions of Computation and Monads

\( ^\ddagger \) 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

**(trv)**

\[
\frac{f : X \to TY}{(T \mathrm{in}_1) f : X \to_{\mathrm{in}_2} T(Y + Z)}
\]

**(sum)**

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

**(cmp)**

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

where $f : X \to_{\sigma} TY$, equivalently $f \in \mathrm{Hom}_{\sigma}(X, TY)$, means that $f$ and $\sigma$ are in the relation
A monad is guarded Elgot if it supports partial iteration operator sending each \( f : X \rightarrow T(Y + X) \) to \( f^\dagger : X \rightarrow TY \) satisfying the fixpoint law

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

and other laws of iteration\(^\dagger\) Roughly: Semantics of while-loops

**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 \( \pi \in Nat^\omega \)

\(^\dagger\)Bloom and Ésik 1993, Iteration theories: The equational logic of iterative processes
The unit of $TX = (X \times Nat^*) \cup Nat^\omega$ sends $x$ to $(x, \langle \rangle)$.

Given $f : X \rightarrow TY$,

$$f^*(x, \tau) = \begin{cases} 
(y, \tau + \tau') & \text{if } f(x) = (y, \tau'), \\
\tau + \pi & \text{if } f(x) = \pi,
\end{cases}$$

$$f^*(\pi) = \pi.$$

$f : X \rightarrow_{\text{inr}} (Y + Z) \times Nat^* \cup Nat^\omega$ if for every $x \in X$,

$$f(x) \in Z \times Nat^* \text{ implies } f(x) \in Z \times Nat^+.$$
The Metalanguage
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\(^\dagger\) observed that

- modelling iteration directly would amount to syntax like
  
  \[
  \text{return inr \ldots inr inl \ldots}
  \]

  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 v)</td>
<td>(\text{handle } x \text{ in } p \text{ with } q)</td>
<td>(\text{handleit } x = v \text{ in } p)</td>
</tr>
</tbody>
</table>

\(^\dagger\) Geron and Levy 2016, Iteration and labelled iteration
handleit $e = \ast$ in

handle $u$ in

($\text{print } ("\text{think of a number} \) \& \text{raise}_u \ast$)

with

($\text{do } y \leftarrow \text{random}() ;$
$z \leftarrow \text{read}() ;$
$\text{if } (y = z) \text{ then ret } \ast \text{ else raise}_{e} \ast$)
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:

\( \Gamma \vdash_v v : A \) and \( \Delta \mid \Gamma \vdash_c p : A \)

In \( \Delta \) types are tagged over \( \{g, u\} \) to indicate (un-)guardedness
Some Derivation Rules

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

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

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

\[
\begin{align*}
\Gamma \vdash_v p : E & \quad \Delta, e : E^g \mid \Gamma, e : E \vdash_c q : A \\
\Delta \mid \Gamma \vdash_c \text{handleit } e = p \text{ in } q : A
\end{align*}
\]
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:

\[ [f] \in \text{Hom}(A, B) \quad \text{for} \quad f : A \to B \in \Sigma_v \]

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

Terms:

\[ [\Gamma \vdash_{v} v : A] \in \text{Hom}(\Gamma, A) \quad [\Delta | \Gamma \vdash_{c} p : A] \in \text{Hom}_{1 + \sigma_{\Delta}}(\Gamma, T(A + \Delta)) \]
Operational Semantics and Adequacy
A Monad of (In)finite Traces

Geron and Levy\(^\dagger\) elaborated the maybe monad – \(+1\) on \(\textbf{Set}\) as the simplest monad for unguarded iteration. Incidentally, it is an initial Elgot monad on \(\textbf{Set}\)\(^\ddagger\).

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 \(\text{Nat}\)
- Value signature contains arithmetic operations
- Effect signature contains only \(\text{put : Nat} \to 0\)[1]

\(^\dagger\)Geron and Levy 2016, Iteration and labelled iteration
\(^\ddagger\)Goncharov, Rauch, and Schröder 2015, Unguarded recursion on coinductive resumptions
Values, Computations, Terminals:

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

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

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

Some Rules:

\[
\begin{align*}
\text{put} \ v \& \text{raise}_x \ w &\rightarrow \text{raise}_x \ w, \langle v \rangle \\
\end{align*}
\]

\[
\begin{align*}
v_0 = v \quad q[v_0/x] &\rightarrow \text{raise}_x \ v_1, \tau_1 \quad \ldots \quad q[v_{n-1}/x] &\rightarrow t, \tau_n \\
\text{handleit} \ x = v \text{ in } q &\rightarrow t, \tau_1 ++ \cdots ++ \tau_n
\end{align*}
\]

\[
\begin{align*}
v_0 = v \quad q[v_0/x] &\rightarrow \text{raise}_x \ v_1, \tau_1 \quad \ldots \quad q[v_{n-1}/x] &\rightarrow \pi \\
\text{handleit} \ x = p \text{ in } q &\rightarrow \tau_1 ++ \cdots ++ \tau_{n-1} ++ \pi
\end{align*}
\]

\[
\begin{align*}
v_0 = v \quad q[v_0/x] &\rightarrow \text{raise}_x \ v_1, \tau_1 \quad q[v_1/x] &\rightarrow \text{raise}_x \ v_2, \tau_2 \quad \ldots \\
\text{handleit} \ x = p \text{ in } q &\rightarrow \tau_1 + \tau_2 ++ \cdots
\end{align*}
\]
The Adequacy Theorem

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

1. If $p \Downarrow \text{ret } \nu, \tau$ then $\llbracket \Delta \models - \vdash_c p : B \rrbracket = (\text{in}_1 \nu, \tau) \in (B + \Delta) \times \text{Nat}^*$
2. If $p \Downarrow \text{raise}_x \nu, \tau$ and $x : E^g$ is in $\Delta$ then $\llbracket \Delta \models - \vdash_c p : B \rrbracket = (\text{in}_2 \text{in}_x \nu, \tau) \in (B + \Delta) \times \text{Nat}^+$
3. If $p \Downarrow \text{raise}_x \nu, \tau$ and $x : E^u$ is in $\Delta$ then $\llbracket \Delta \models - \vdash_c p : B \rrbracket = (\text{in}_2 \text{in}_x \nu, \tau) \in (B + \Delta) \times \text{Nat}^*$
4. If $p \Downarrow \pi$, then $\llbracket \Delta \models - \vdash_c p : B \rrbracket = \pi \in \text{Nat}^\omega$
Conclusions & Further Work

- 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$^\dagger$
- Hoare logic for guarded iteration:

\[
\begin{aligned}
\{ \phi \} \ x & \leftarrow \ p \ \{ \psi \} \\
\end{aligned}
\]

what are these?

\[\dagger\text{Goncharov, Jakob, and Neves 2018, A Semantics for Hybrid Iteration}\]
References


A Monad of (In)Finite Traces (with Iteration)

- The unit of $TX = (X \times \text{Nat}^*) \cup \text{Nat}^\omega$ sends $x$ to $(x, \langle \rangle)$

- Given $f : X \to TY$,

\[
f^*(x, \tau) = \begin{cases} 
(y, \tau + \tau') & \text{if } f(x) = (y, \tau'), \\
\tau + \pi & \text{if } f(x) = \pi,
\end{cases}
\]

- $f : X \rightarrow_{\text{inr}} (Y + Z) \times \text{Nat}^* \cup \text{Nat}^\omega$ 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 $f : X \rightarrow_{\text{inr}} T(Y + X) = (Y + X) \times \text{Nat}^* \cup \text{Nat}^\omega$,

\[
f^{\dagger}(x) = \begin{cases} 
(y, \tau_1 + \cdots + \tau_n) & \text{if } f(x) = (\text{in}_2 x_1, \tau_1), \ldots, f(x_n) = (\text{in}_1 y, \tau_n), \\
\tau_1 + \cdots + \tau_{n-1} + \pi & \text{if } f(x) = (\text{in}_2 x_1, \tau_1), \ldots, f(x_n) = \pi, \\
\tau_1 + \tau_2 + \cdots & \text{if } f(x) = (\text{in}_2 x_1, \tau_1), f(x_1) = (\text{in}_1 x_2, \tau_2), \ldots
\end{cases}
\]