Semantics of Higher-Order Functional Programming

Petros Barbagiannis

\[ \mu \Pi \lambda \forall \]

July 14, 2014
Higher-order functions are functions that either accept other functions as arguments or they return functions (i.e., functions are first-class objects).

Higher-order functions have been used widely in:

- Mathematics, ($\int$, $\sum$, etc.)
- Computability Theory (Kleene’s $T$-Predicate, Church’s $\lambda$-calculus, etc.)
- Programming Languages (Lisp, Haskell, C#, etc.)
A Declarative Language

Types

\[
\tau ::= \texttt{bool} \mid \texttt{nat} \quad \text{data types}
\]

\[
\theta ::= \texttt{val}[\tau] \mid \theta \rightarrow \theta' \mid (\theta) \quad \text{phrase types}
\]

A type of the form \( \theta \rightarrow \theta' \) is a functional type.

A functional type \( \theta \rightarrow \theta' \) is called higher-order if \( \theta \) is a functional type or \( \theta' \) is higher-order.

Example:

\[
(\texttt{val[nat]} \rightarrow \texttt{val[bool]}) \rightarrow \texttt{val[nat]}
\]
Syntax Rules

Bracketing \( \frac{X : \theta}{(X)} : \theta \)

Zero \( 0 : \text{val[nat]} \)

Succ \( \text{succ} \ N : \text{val[nat]} \)

Cond \( \frac{B : \text{val[bool]} \ X_0 : \theta \ X_1 : \theta}{\text{if} \ B \ \text{then} \ X_0 \ \text{else} \ X_1 : \theta} \)

Truth \( \text{true} : \text{val[bool]} \)

Negation \( B : \text{val[bool]} \)

\( \text{not} \ B : \text{val[bool]} \)

Conjunction \( \frac{B_0 : \text{val[bool]} \ B_1 : \text{val[bool]}}{B_0 \ \text{and} \ B_1 : \text{val[bool]}} \)

Application \( \frac{P : \theta \rightarrow \theta' \ Q : \theta}{P \ Q : \theta'} \)

Local Definition \( \frac{P : \theta \ Q : \theta'}{\text{let} \ \iota \ \text{be} \ P \ \text{in} \ Q : \theta'} \)

Petros Barbagiannis  Semantics of Higher-Order Functional Programming  July 14, 2014 4 / 18
An environment \( u \) is a function which assigns meanings (values) to the identifiers in a phrase.

A function \( \pi \) is a phrase-type assignment which maps identifiers to types.

\[
\begin{align*}
\llbracket \text{val}[\tau] \rrbracket &= \llbracket \tau \rrbracket \text{ where } \llbracket \tau \rrbracket \text{ is the set of values for data type } \tau. \\
\llbracket \theta \rightarrow \theta' \rrbracket &= \llbracket \theta \rrbracket \rightarrow \llbracket \theta' \rrbracket \\
\llbracket (\theta) \rrbracket &= \llbracket \theta \rrbracket
\end{align*}
\]

Example:
\[
\llbracket \pi(\iota) \rrbracket \text{ is the set of values } \iota \text{ takes on.}
\]
Semantic Equations

For an identifier $\nu$, $\llbracket \nu \rrbracket(u) = u(\nu)$

$\llbracket B_0 \text{ and } B_1 \rrbracket u = \begin{cases} 
  \text{true,} & \text{if } \llbracket B_0 \rrbracket u = \text{true} \text{ and } \llbracket B_1 \rrbracket u = \text{true} \\
  \text{false,} & \text{otherwise}
\end{cases}$

$\llbracket \text{if } B \text{ then } X_0 \text{ else } X_1 \rrbracket(u) = \begin{cases} 
  \llbracket X_0 \rrbracket(u) & \text{if } \llbracket B \rrbracket u = \text{true} \\
  \llbracket X_1 \rrbracket(u) & \text{if } \llbracket B \rrbracket u = \text{false}
\end{cases}$

$\llbracket P \ Q \rrbracket u = (\llbracket P \rrbracket u)(\llbracket Q \rrbracket u)$

Note: Since $\llbracket P \rrbracket u$ is a function and $\llbracket Q \rrbracket u \in \text{dom}\llbracket P \rrbracket u$ we can rewrite this equation as $\llbracket P \rrbracket u(\llbracket Q \rrbracket u)$

$\llbracket \text{let } \nu \text{ be } P \text{ in } Q \rrbracket u = \llbracket Q \rrbracket(u|\nu \mapsto \llbracket P \rrbracket u)$
The above construct has the following semantic equation:

\[
\left[ \text{let } \nu(\nu_0 : \theta) = P \text{ in } Q : \theta' \right] \equiv \left[ Q \right](u_j^n f)
\]

where \( f : \left[ \right] \rightarrow \left[ \right] \) is the function defined by

\[
f(a) = \left[ P \right](u_j^n a)
\]

for all \( a \in \theta \).
The above construct has the following semantic equation:

\[
\llbracket \text{let } \nu(\nu_0 : \theta) = P \text{ in } Q \rrbracket u = \llbracket Q \rrbracket(u|_\nu \mapsto f)
\]

where \( f : \llbracket \theta \rrbracket \rightarrow \llbracket \theta' \rrbracket \) is the function defined by

\[
f(a) = \llbracket P \rrbracket(u|_\nu \mapsto a)
\]

for all \( a \in \theta_0 \)
Lambda Expressions

\[
\begin{array}{l}
\text{Abstraction} \\
\frac{[\cdot : \theta]}{P : \theta'} \\
\quad \lambda \cdot : \theta. P : \theta \rightarrow \theta'
\end{array}
\]

Example:

\[(\lambda n : \text{val}[\text{nat}]. n + m) (3)\]

The semantic equation for the lambda expression is:

\[
[\lambda \cdot : \theta. P](u) = f \in [\theta] \rightarrow [\theta']
\]

where \( f(a) = [P](u | \cdot \mapsto a) \) for all \( a \in [\theta] \).
Prelude> let myfunction f = f 10 in 5 * myfunction add2

\[
\begin{align*}
\llbracket \text{let } & \text{myfunction } f = f \ 10 \ \text{in } 5 \ast \text{myfunction add2}\rrbracket \ u \\
= \llbracket 5 \ast \text{myfunction add2}\rrbracket(u\mid \text{myfunction} \mapsto \llbracket f \ 10\rrbracket(u\mid f \mapsto \text{add2})) \\
= \llbracket 5 \ast \text{myfunction add2}\rrbracket(u\mid \text{myfunction} \mapsto \llbracket \text{add2 } 10\rrbracket u) \\
= \llbracket 5 \ast 12\rrbracket u = 5 \ast 12 = 60
\end{align*}
\]
A pair \((D, \sqsubseteq)\) consisting of a set \(D\) and a partial order is called **partially-ordered set** (poset).

A poset \(D\) is **complete** iff for every chain \(d \in D^\omega\), the least upper bound \(\bigsqcup_{i \in \omega} d_i\) exists in \(D\).

A **domain** is any poset that is complete.

If \(A\) and \(D\) are domains, then the set \(A \times D\) is a domain where 
\[(a, d) \sqsubseteq_{A \times D} (a', d')\] if \(a \sqsubseteq_A a'\) and \(d \sqsubseteq_D d'\).

If \(D\) is a domain, then so is \(D_\bot\), i.e., the set with a least element \(\bot\) added to \(D\).

If \(A\) is a set and \(D\) is a domain, then \(A \to D\) is a domain where 
\(f \sqsubseteq_{A \to D} f'\) if \(f(a) \sqsubseteq_D f'(a)\) for every \(a \in A\).
Continuous Functions

**Definition**

Let $A$ and $D$ be domains. A function $f : A \rightarrow D$ is called:

- **monotonic** if $f(a) \sqsubseteq_D f(a')$ when $a \sqsubseteq_A a'$.
- **continuous** if, for every chain $d \in D^\omega$,

\[
\bigcup_{i \in \omega} f(d_i) = \bigcup_{i \in \omega} f(d_i)
\]

**Definition**

If $A$ and $D$ are domains with least elements $\bot_A$ and $\bot_D$, a function $f : A \rightarrow D$ is called **strict** if $f(\bot_A) = \bot_D$. 
Continuous Functions

Definition
Let $D$ be a domain. $d \in D^{\omega \times \omega}$ is called a **double chain** if

\[ d_{i0} \sqsupseteq d_{i1} \sqsupseteq d_{i2} \sqsupseteq \ldots \text{ for every } i \in \omega \text{ and} \]

\[ d_{0j} \sqsupseteq d_{1j} \sqsupseteq d_{2j} \sqsupseteq \ldots \text{ for every } j \in \omega. \]

Lemma
If $d$ is a double chain in $D$, the following limits are well-defined and equivalent:

\[ \bigsqcup_{i \in \omega} \bigsqcup_{j \in \omega} d_{ij} \]

\[ \bigsqcup_{j \in \omega} \bigsqcup_{i \in \omega} d_{ij} \]
**Theorem**

Let $f$ be a chain of continuous functions from $A$ to $D$. Then $\bigsqcup_{i \in \omega} f_i$ is continuous.

**Proof**

Consider $d \in A$

$$\bigsqcup_{i \in \omega} f_i(\bigsqcup_{j \in \omega} d_j) = \bigsqcup_{i \in \omega} \bigsqcup_{j \in \omega} f_i(d_j) = \bigsqcup_{j \in \omega} \bigsqcup_{i \in \omega} f_i(d_j) = \bigsqcup_{j \in \omega} (\bigsqcup_{i \in \omega} f_i)(d_j)$$
Fixed Points

Theorem

Let $D$ be a domain with a least element $\bot$ and $f : D \rightarrow D$ be a continuous function. Then $\bigsqcup_{i \in \omega} f^i(\bot)$ is the least fixed point of $f$.

Proof

\[
f\left(\bigsqcup_{i \in \omega} f^i(\bot)\right) = \bigsqcup_{i \in \omega} f(f^i(\bot)) = \bigsqcup_{i \in \omega} f^{i+1}(\bot) = \bigsqcup_{i \in \omega} f^i(\bot)
\]
Domain-Theoretic Semantics

\[
\begin{array}{c}
[\iota : \theta] & [\iota : \theta] \\
\hline
P : \theta & Q : \theta' \\
\hline
\text{Recursion}
\end{array}
\]

\[\text{letrec } \iota : \theta \text{ be } P \text{ in } Q : \theta'\]

Example:

\[
\text{letrec } \text{double : val[nat] } \rightarrow \text{ val[nat] be}
\lambda n : \text{val[nat]}. \text{if } n = 0 \text{ then } 0 \text{ else } 2 + \text{double}(n - 1)
\text{ in...}
\]

The meaning defined for \(\iota\) by the letrec construct is the solution to the following equation:

\[
p = \sem{P}(u | \iota \mapsto p)
\]
Domain-Theoretic Semantics

We redefine sets to be domains as follows:

\[ [\text{val}[\tau]] \] is the flat domain \( [\tau] \)
\[ [\theta \to \theta'] = [\theta] \to [\theta'] \]
\[ [u] \] is the set of all environments.

We must also redefine valuation functions so that all phrases \( P \) are mapped to continuous functions. For all primitive operations (\texttt{not}, =, etc.) we can take their strict extension.

**Proposition**

For all phrases \( P \), \( [P] \) is continuous and when \( \pi(P) = \theta \to \theta' \), then \( [P](u) \) is continuous.

**Proof**

By structural induction on phrases.
Let $u \in \llbracket u \rrbracket$ and define a function $f : \theta \rightarrow \theta$ as follows:

$$f(p) = \llbracket P \rrbracket(u|i \mapsto p)$$

$f$ is continuous if $P$ is continuous.

$$\llbracket \text{letrec } \nu : \theta \text{ be } P \text{ in } Q \rrbracket = \llbracket Q \rrbracket(u|\nu \mapsto \bigsqcup f^i(\perp))$$
References
