Domain Theory II

October 29, 2015
Outline

Meaning of recursive definitions
- Compositionality and well-definedness
- Extensionality of recursive functions
- Least fixed-point construction
- Internal structure of domains
Recall: a **denotational semantics** must be **compositional**

- a term’s denotation is built from the denotations of its parts

**Example: integer expressions**

\[
i \in \text{Int} ::= \text{(any integer)}\\
e \in \text{Exp} ::= i \mid \text{add } e \ e \mid \text{mul } e \ e
\]

\[
\begin{align*}
[\text{Exp}] & : \text{Int} \\
[i] & = i \\
[\text{add } e_1 \ e_2] & = [e_1] + [e_2] \\
[\text{mul } e_1 \ e_2] & = [e_1] \times [e_2]
\end{align*}
\]

Compositionality ensures the semantics is **well-defined** by **structural induction**

Each AST has **exactly one** meaning
A non-compositional (and ill-defined) semantics

Anti-example: while statement

\[ t \in \text{Test} ::= \ldots \]
\[ c \in \text{Cmd} ::= \ldots \mid \text{while } t \ c \]

\[ T[\text{Test}] : S \rightarrow \text{Bool} \]
\[ C[\text{Cmd}] : S \rightarrow S \]
\[ C[\text{while } t \ c] = \lambda s. \text{if } T[t](s) \text{ then } C[\text{while } t \ c](C[c](s)) \text{ else } s \]

Meaning of \texttt{while } t \ c \ in \ state \ s:

1. evaluate \( t \) in state \( s \)
2. if true:
   a. run \( c \) to get updated state \( s' \)
   b. re-evaluate \texttt{while} in state \( s' \)
      (not compositional)
3. otherwise return \( s \) unchanged

A particular \texttt{while} statement may have \texttt{infinitely many} meanings!
Outline

Meaning of recursive definitions
  Compositionality and well-definedness
  Extensionality of recursive functions
  Least fixed-point construction
  Internal structure of domains
Extensional vs. operational definitions of a function

**Mathematical function**
Defined *extensionally*:
- a relation between inputs and outputs

**Computational function** (e.g. Haskell)
Usually defined *operationally*:
- compute output by sequence of reductions

**Example (intensional definition)**
\[
fac(n) = \begin{cases} 
  1 & n = 0 \\
  n \cdot fac(n - 1) & \text{otherwise}
\end{cases}
\]

**Extensional meaning**
\{..., (2, 2), (3, 6), (4, 24), ...\}

**Operational meaning**
\[
\begin{align*}
fac(3) & \leadsto 3 \cdot fac(2) \\
& \leadsto 3 \cdot 2 \cdot fac(1) \\
& \leadsto 3 \cdot 2 \cdot 1 \cdot fac(0) \\
& \leadsto 3 \cdot 2 \cdot 1 \cdot 1 \\
& \leadsto 6
\end{align*}
\]
Extensional meaning of recursive functions

\[ \text{grow}(n) = \begin{cases} 1 & n = 0 \\ \text{grow}(n + 1) - 2 & \text{otherwise} \end{cases} \]

Best extension (use \( \bot \) if undefined):
- \( \{(0, 1), (1, \bot), (2, \bot), (3, \bot), (4, \bot), \ldots \} \)

Other valid extensions:
- \( \{(0, 1), (1, 2), (2, 4), (3, 6), (4, 8), \ldots \} \)
- \( \{(0, 1), (1, 5), (2, 7), (3, 9), (4, 11), \ldots \} \)
- \( \ldots \)

Goal: best extension = only extension
A **function space domain** is a set of **mathematical functions**

**Anti-example: while statement**

\[
\begin{align*}
  t & \in \text{Test} \quad ::= \quad \ldots \\
  c & \in \text{Cmd} \quad ::= \quad \ldots \quad | \quad \textbf{while} \quad t \quad c
\end{align*}
\]

\[
\begin{align*}
  T[\text{Test}] & : S \rightarrow \text{Bool} \\
  C[\text{Cmd}] & : S \rightarrow S \\
  C[\textbf{while} \ t \ c] & = \lambda s. \text{if } T[t](s) \text{ then } \\
  & \quad C[\textbf{while} \ t \ c](C[c](s)) \quad \text{else } s
\end{align*}
\]

**Ideal semantics of Cmd:**

- semantic domain: \( S \rightarrow S_\bot \)
- contains \((s, s')\) if \(c\) terminates
- contains \((s, \bot)\) if \(c\) diverges
Outline

Meaning of recursive definitions
- Compositionality and well-definedness
- Extensionality of recursive functions
- Least fixed-point construction
- Internal structure of domains
Least fixed points

Basic idea:

1. a **recursive** function defines a **set** of **non-recursive, finite** subfunctions
2. its meaning is the “**union**” of the meanings of its subfunctions

Iteratively grow the extension until we reach a **fixed point**

- essentially encodes computational functions as mathematical functions
Example: unfolding a recursive definition

**Recursive definition**

\[
\text{fac}(n) = \begin{cases} 
  1 & n = 0 \\ 
  n \cdot \text{fac}(n - 1) & \text{otherwise} 
\end{cases}
\]

**Non-recursive, finite subfunctions**

\[
\begin{align*}
\text{fac}_0(n) &= \bot \\
\text{fac}_1(n) &= \begin{cases} 
  1 & n = 0 \\ 
  n \cdot \text{fac}_0(n - 1) & \text{otherwise} 
\end{cases} \\
\text{fac}_2(n) &= \begin{cases} 
  1 & n = 0 \\ 
  n \cdot \text{fac}_1(n - 1) & \text{otherwise} 
\end{cases} \\
&\ldots \\
\end{align*}
\]

\[
\text{fac} = \bigcup_{i=0}^{\infty} \text{fac}_i
\]

Fine print:
- each \( \text{fac}_i \) maps all other values to \( \bot \)
- \( \bigcup \) operation prefers non-\( \bot \) mappings
Computing the fixed point

In general

\[ \text{fac}_0(n) = \bot \]

\[ \text{fac}_i(n) = \begin{cases} 1 & n = 0 \\ n \cdot \text{fac}_{i-1}(n-1) & \text{otherwise} \end{cases} \]

A template to represent all \( \text{fac}_i \) functions:

\[
F = \lambda f. \lambda n. \begin{cases} 1 & n = 0 \\ n \cdot f(n - 1) & \text{otherwise} \end{cases}
\]

\( F \) takes \( \text{fac}_{i-1} \) as input

Fixpoint operator

\[
\text{fix} : (A \to A) \to A \\
\text{fix}(g) = \text{let } x := g(x) \text{ in } x
\]

\[
\text{fix}(h) = h(h(h(h(...))))
\]

Factorial as a fixed point

\[
\text{fac} = \text{fix}(F)
\]
Outline

Meaning of recursive definitions
  - Compositionality and well-definedness
  - Extensionality of recursive functions
  - Least fixed-point construction

Internal structure of domains
Internal structure of domains supports the least fixed-point construction

Recall fine print from factorial example:

- each $fac_i$ maps all other values to $\perp$
- $\cup$ operation prefers non-$\perp$ mappings

How can we generalize and formalize this idea?
Partial orderings and joins

Partial ordering: \( \sqsubseteq : D \times D \to \mathbb{B} \)

- reflexive: \( \forall x \in D. \quad x \sqsubseteq x \)
- antisymmetric: \( \forall x, y \in D. \quad x \sqsubseteq y \land y \sqsubseteq x \implies x = y \)
- transitive: \( \forall x, y, z \in D. \quad x \sqsubseteq y \land y \sqsubseteq z \implies x \sqsubseteq z \)

Join: \( \sqcup : D \times D \to D \)

\( \forall a, b \in D, \) the element \( c = a \sqcup b \in D, \) if it exists,

is the smallest element that is larger than both \( a \) and \( b \)

i.e. \( a \sqsubseteq c \) and \( b \sqsubseteq c, \) and there is no \( d = a \sqcup b \in D \) where \( d \sqsubseteq c \)
Directed-complete partial orderings

A **directed subset** of domain is a subset of elements related by the ordering relation $\sqsubseteq$

Every domain is a **directed-complete partial ordering** (dcpo):
- every directed subset $D$ has a **least element**

A function is **continuous** if it preserves the least element

*Finally*, the meaning of a continuous recursive function $f$ is:

$$\bigcap_{i=0}^{\infty} f_i$$

where $f_i$ are the finite approximations of $f$  

*whew!*
Well-defined semantics for the while statement

Syntax

\[ t \in \text{Test} ::= \ldots \]
\[ c \in \text{Cmd} ::= \ldots \mid \text{while } t \ c \]

Semantics

\[ T [\text{Test}] : S \rightarrow \text{Bool} \]
\[ C [\text{Cmd}] : S \rightarrow S \]
\[ C [\text{while } t \ c] = \text{fix}(\lambda f. \lambda s. \text{if } T[t](s) \ \text{then } f(C[c](s)) \ \text{else } s) \]