Denotational Semantics

CS 152 (Spring 2021)

Harvard University

Tuesday, February 11, 2021

Today, we will learn about

- Modeling programs as functions from input stores to output stores
- Denotational model
- Fixed point of a function

Program models

- Operational model
- Denotational model

Program models

- Operational model (executions)
- Denotational model (mathematical functions)

Compositionality (Mitchell 4.3)

An important principle of denotational semantics is that the meaning of a program is determined from its text compositionally. This means that the meaning of a program must be defined from the meanings of its parts, not something else, such as the text of its parts or the meanings of related programs obtained by syntactic operations. For example, the denotation of a program such as if B then P else Q must be explained with only the denotations of B, P, and Q; it should not be defined with programs constructed from B, P, and Q by syntactic operations such as substitution.

Importance of Compositionality (Mitchell 4.3)

The importance of compositionality, which may seem rather subtle at first, is that if two program pieces have the same denotation, then either may safely be substituted for the other in any program. More specifically, if B, P, and Q have the same denotations as B', P', and Q', respectively, then if B then P else Q must have the same denotation as if B' then P' else Q'. Compositionality means that the denotation of an expression or program statement must be detailed enough to capture everything that is relevant to its behavior in larger programs. This makes denotational semantics useful for understanding and reasoning about such pragmatic issues as program transformation and optimization, as these operations on programs involve replacing parts of programs without changing the overall meaning of the whole program.

Programs as functions

Programs as functions

For a program c, we write $\mathcal{C}[\![c]\!]$ for the *denotation* of c, that is, the mathematical function that c represents:

$$\mathcal{C}[\![c]\!] : \textbf{Store} \rightharpoonup \textbf{Store}.$$

Programs as functions

For a program c, we write $\mathcal{C}[\![c]\!]$ for the *denotation* of c, that is, the mathematical function that c represents:

$$\mathcal{C}[\![c]\!] : \textbf{Store} \rightharpoonup \textbf{Store}.$$

We write $\mathcal{C}[\![c]\!]\sigma$ for the result of applying this function to to the store σ.

Expressions as functions

Expressions as functions

Similarly, $\mathcal{A}[\![a]\!]$ and $\mathcal{B}[\![b]\!]$ are denotations for arithmetic and boolean expressions.

$$\mathcal{A}[\![a]\!] : \textbf{Store} \to \textbf{Int}$$
$$\mathcal{B}[\![b]\!] : \textbf{Store} \to \{\textbf{true}, \textbf{false}\}$$

Expressions as functions

Note that $\mathcal{A}[\![a]\!]$ and $\mathcal{B}[\![b]\!]$ are total functions.

$$\mathcal{A}[\![a]\!] : \textbf{Store} \to \textbf{Int}$$
$$\mathcal{B}[\![b]\!] : \textbf{Store} \to \{\textbf{true}, \textbf{false}\}$$

Arithmetic expressions

$$\mathcal{A}[\![n]\!] = \{(\sigma, n)\}$$

Arithmetic expressions

$$\mathcal{A}[\![n]\!] = \{(\sigma, n)\}$$
$$\mathcal{A}[\![x]\!] = \{(\sigma, \sigma(x))\}$$

Arithmetic expressions

$$
\begin{aligned}
\mathcal{A}[\![n]\!] &= \{(\sigma, n)\} \\
\mathcal{A}[\![x]\!] &= \{(\sigma, \sigma(x))\} \\
\mathcal{A}[\![a_1 + a_2]\!] &= \{(\sigma, n) \mid (\sigma, n_1) \in \mathcal{A}[\![a_1]\!] \\
&\qquad \wedge (\sigma, n_2) \in \mathcal{A}[\![a_2]\!] \\
&\qquad \wedge n = n_1 + n_2\}
\end{aligned}
$$

Arithmetic expressions

$$
\begin{aligned}
\mathcal{A}[\![n]\!] &= \{(\sigma, n)\} \\
\mathcal{A}[\![x]\!] &= \{(\sigma, \sigma(x))\} \\
\mathcal{A}[\![a_1 + a_2]\!] &= \{(\sigma, n) \mid (\sigma, n_1) \in \mathcal{A}[\![a_1]\!] \\
&\qquad \wedge\, (\sigma, n_2) \in \mathcal{A}[\![a_2]\!] \\
&\qquad \wedge\, n = n_1 + n_2\} \\
\mathcal{A}[\![a_1 \times a_2]\!] &= \{(\sigma, n) \mid (\sigma, n_1) \in \mathcal{A}[\![a_1]\!] \\
&\qquad \wedge\, (\sigma, n_2) \in \mathcal{A}[\![a_2]\!] \\
&\qquad \wedge\, n = n_1 \times n_2\}
\end{aligned}
$$

Boolean expressions

$$\mathcal{B}[\![\textbf{true}]\!] = \{(\sigma, \textbf{true})\}$$
$$\mathcal{B}[\![\textbf{false}]\!] = \{(\sigma, \textbf{false})\}$$

Boolean expressions

$$
\begin{aligned}
\mathcal{B}[\![\textbf{true}]\!] &= \{(\sigma, \textbf{true})\} \\
\mathcal{B}[\![\textbf{false}]\!] &= \{(\sigma, \textbf{false})\} \\
\mathcal{B}[\![a_1 < a_2]\!] &= \{(\sigma, \textbf{true}) \mid (\sigma, n_1) \in \mathcal{A}[\![a_1]\!] \\
&\qquad \wedge\, (\sigma, n_2) \in \mathcal{A}[\![a_2]\!] \wedge n_1 < n_2\} \\
&\qquad \cup\ \{(\sigma, \textbf{false}) \mid (\sigma, n_1) \in \mathcal{A}[\![a_1]\!] \\
&\qquad \wedge\, (\sigma, n_2) \in \mathcal{A}[\![a_2]\!] \\
&\qquad \wedge\, n_1 \geq n_2\}
\end{aligned}
$$

Denotations of some programs

Denotations of some programs

$$\mathcal{C}[\![\textbf{skip}]\!] = \{(\sigma, \sigma)\}$$

$$\mathcal{C}[\![x := a]\!] = \{(\sigma, \sigma[x \mapsto n]) \mid (\sigma, n) \in \mathcal{A}[\![a]\!]\}$$

$$\mathcal{C}[\![c_1; c_2]\!] = \{(\sigma, \sigma') \mid \exists \sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c_1]\!] \\ \wedge\, (\sigma'', \sigma') \in \mathcal{C}[\![c_2]\!])\}$$

Composition of relations

Suppose $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$.
The *composition* of relations $R_2 \circ R_1 \subseteq A \times C$ is

$$R_2 \circ R_1 = \{(a, c) \mid \exists b \in B.\, (a, b) \in R_1 \wedge (b, c) \in R_2\}.$$

Composition of relations

We have $\mathcal{C}[\![c_1; c_2]\!] = \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!]$, where $\circ$ is the composition of relations.

Definition of a function

A function is a set of input-output pairs s.t. each input has a unique output.

if b **then** c_1 **else** c_2

The mathematical function represented by
if b **then** c_1 **else** c_2 is the set

$$\begin{aligned}\mathcal{C}[\![\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2]\!] = \{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \\ \wedge\, (\sigma,\sigma') \in \mathcal{C}[\![c_1]\!]\} \;\cup \\ \{(\sigma,\sigma') \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!] \\ \wedge\, (\sigma,\sigma') \in \mathcal{C}[\![c_2]\!]\}\end{aligned}$$

These are the input-output pairs of our function.

$\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$

$\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$

$$\begin{aligned}\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!] = \{(\sigma,\sigma) \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!]\} \ \cup \\ \{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \\ \wedge \exists \sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \\ \wedge\ (\sigma'',\sigma') \in \mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!])\}\end{aligned}$$

Computing $f = \mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$

So far we only have a recursive equation

$$\begin{aligned} f = \{(\sigma, \sigma) \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!]\} \ \cup \\ \{(\sigma, \sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \\ \wedge \exists \sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'', \sigma') \in f)\} \end{aligned}$$

What is f, as a subset $\textbf{Store} \times \textbf{Store}$?

A simpler example

A simpler example

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ f(x-1) + 2x - 1 & \text{otherwise} \end{cases} \tag{1}$$

$$
\begin{aligned}
f_0 &= \emptyset \\
f_1 &= \begin{cases} 0 & \text{if } x = 0 \\ f_0(x-1) + 2x - 1 & \text{otherwise} \end{cases} \\
&= \{(0,0)\} \\
f_2 &= \begin{cases} 0 & \text{if } x = 0 \\ f_1(x-1) + 2x - 1 & \text{otherwise} \end{cases} \\
&= \{(0,0),(1,1)\} \\
f_3 &= \begin{cases} 0 & \text{if } x = 0 \\ f_2(x-1) + 2x - 1 & \text{otherwise} \end{cases} \\
&= \{(0,0),(1,1),(2,4)\} \\
&\vdots
\end{aligned}
$$

Consider this higher-order function F:

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$
$$F(g) = g'$$

where

$$g'(x) = \begin{cases} 0 & \text{if } x = 0 \\ g(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$
$$F(g) = g'$$

where

$$g'(x) = \begin{cases} 0 & \text{if } x = 0 \\ g(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$

- E.g. $F(\emptyset) = \{(0,0)\}$,
  $F(\{(1,0)\}) = \{(0,0),(2,3)\}$, and
  $F(\{(3,1),(0,1)\}) = \{(0,0),(4,8),(1,0)\}$.
- Note, however, that $F(f) = f$.
- In other words, f is a *fixed point* of F.

$$
\begin{aligned}
f &= \mathsf{fix}(F) \\
&= f_0 \cup f_1 \cup f_2 \cup f_3 \cup \ldots \\
&= \emptyset \cup F(\emptyset) \cup F(F(\emptyset)) \cup F(F(F(\emptyset))) \cup \ldots \\
&= \bigcup_{i \geq 0} F^i(\emptyset)
\end{aligned}
$$

Fixed-point semantics for loops

$$F_{b,c} : (\mathbf{Store} \rightharpoonup \mathbf{Store}) \to (\mathbf{Store} \rightharpoonup \mathbf{Store})$$

$$\begin{aligned} F_{b,c}(f) = \{(\sigma,\sigma) \mid (\sigma, \mathbf{false}) \in \mathcal{B}[\![b]\!]\} \ \cup \\ \{(\sigma,\sigma') \mid (\sigma, \mathbf{true}) \in \mathcal{B}[\![b]\!] \\ \wedge \exists \sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \\ \wedge\ (\sigma'',\sigma') \in f)\} \end{aligned}$$

$$
\begin{aligned}
\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!] &= \bigcup_{i \geq 0} {F_{b,c}}^i(\emptyset) \\
&= \emptyset \cup F_{b,c}(\emptyset) \cup F_{b,c}(F_{b,c}(\emptyset)) \\
&\quad \cup F_{b,c}(F_{b,c}(F_{b,c}(\emptyset))) \cup \ldots \\
&= \mathsf{fix}(F_{b,c})
\end{aligned}
$$

Let's consider an example:
while foo $<$ bar **do** foo $:=$ foo $+ 1$.
Here $b =$ foo $<$ bar and $c =$ foo $:=$ foo $+ 1$.

$$
\begin{aligned}
F_{b,c}(\emptyset) = \{(\sigma,\sigma) \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!]\} \;\cup \\
\{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \\
\wedge \exists \sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'',\sigma') \in \emptyset)\} \\
= \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\}
\end{aligned}
$$

$$
\begin{aligned}
{F_{b,c}}^2(\emptyset) = &\{(\sigma,\sigma) \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!]\} \ \cup \\
&\{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \\
&\wedge \exists \sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'',\sigma') \in F_{b,c}(\emptyset))\} \\
= &\{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo})+1]) \mid \sigma(\mathsf{foo}) < \sigma(\mathsf{bar}) \\
&\wedge \sigma(\mathsf{foo})+1 \geq \sigma(\mathsf{bar})\}
\end{aligned}
$$

But $\sigma(\mathsf{foo}) < \sigma(\mathsf{bar}) \wedge \sigma(\mathsf{foo})+1 \geq \sigma(\mathsf{bar})$ if and only if $\sigma(\mathsf{foo})+1 = \sigma(\mathsf{bar})$ so we get:

$$
\begin{aligned}
= &\{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo})+1]) \mid \sigma(\mathsf{foo})+1 = \sigma(\mathsf{bar})\}
\end{aligned}
$$

$$
\begin{aligned}
{F_{b,c}}^3(\emptyset) = \{&(\sigma,\sigma) \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!]\} \ \cup \\
&\{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \wedge \exists \sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \\
&\qquad \wedge (\sigma'',\sigma') \in {F_{b,c}}^2(\emptyset))\} \\
= \{&(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma, \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 1]) \mid \sigma(\mathsf{foo}) + 1 = \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma, \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 2]) \mid \sigma(\mathsf{foo}) + 2 = \sigma(\mathsf{bar})\}
\end{aligned}
$$

$$
\begin{aligned}
{F_{b,c}}^4(\emptyset) = \{&(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma, \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 1]) \mid \sigma(\mathsf{foo}) + 1 = \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma, \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 2]) \mid \sigma(\mathsf{foo}) + 2 = \sigma(\mathsf{bar})\} \ \cup \\
&\{(\sigma, \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 3]) \mid \sigma(\mathsf{foo}) + 3 = \sigma(\mathsf{bar})\}
\end{aligned}
$$

$$\mathcal{C}[\![\textbf{while}\ \mathsf{foo} < \mathsf{bar}\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 1]\!] =$$
$$\{(\sigma, \sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup$$
$$\{(\sigma, \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + n]) \mid \sigma(\mathsf{foo}) + n = \sigma(\mathsf{bar}) \wedge n \geq 1\}$$

Break

- How are denotional semantics different from large-step operational semantics?
- Let $w = \textbf{while } b \textbf{ do } c$. Prove

$$\mathcal{C}[\![w]\!] = \mathcal{C}[\![\textbf{if } b \textbf{ then } c; w \textbf{ else skip}]\!]$$

.

- Why/when can we take the least fixed point?

Contrast...

$$c \sim c' \triangleq$$

$$\forall \sigma, \sigma'. < c, \sigma > \Downarrow \sigma' \iff < c', \sigma > \Downarrow \sigma'$$

vs

$$\{(\sigma, \sigma') | < c, \sigma > \Downarrow \sigma'\} = \{(\sigma, \sigma') | < c', \sigma > \Downarrow \sigma'\}$$

vs

$$\mathcal{C}[\![c]\!] = \mathcal{C}[\![c']\!]$$

Contrasting Meta Language

- operational semantics: execution on abstract machine defined by stylized inductive sets
- denotational semantics: language of mathematics

Unfolding

Let $w = \textbf{while}\ b\ \textbf{do}\ c$. Prove

$$\mathcal{C}[\![w]\!] = \mathcal{C}[\![\textbf{if}\ b\ \textbf{then}\ c; w\ \textbf{else skip}]\!]$$

.

Unfolding

$$F_{b,c} : (\mathbf{Store} \rightharpoonup \mathbf{Store}) \to (\mathbf{Store} \rightharpoonup \mathbf{Store})$$

$$\begin{aligned} F_{b,c}(f) = \{(\sigma,\sigma) \mid (\sigma, \mathbf{false}) \in \mathcal{B}[\![b]\!]\} \ \cup \\ \{(\sigma,\sigma') \mid (\sigma, \mathbf{true}) \in \mathcal{B}[\![b]\!] \\ \wedge \exists \sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \\ \wedge\ (\sigma'',\sigma') \in f)\} \end{aligned}$$

Unfolding

$$F_{b,c} : (\mathbf{Store} \rightharpoonup \mathbf{Store}) \to (\mathbf{Store} \rightharpoonup \mathbf{Store})$$

$$F_{b,c}(f) = \{(\sigma, \sigma) \mid (\sigma, \mathbf{false}) \in \mathcal{B}[\![b]\!]\} \cup \{(\sigma, \sigma') \mid (\sigma, \mathbf{true}) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \sigma') \in f \circ \mathcal{C}[\![c]\!]\}$$

Unfolding

$$F_{b,c}(\mathcal{C}[\![w]\!]) = \{(\sigma,\sigma) \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!]\} \cup \{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \wedge (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}$$

Unfolding

$$
\begin{aligned}
F_{b,c}(\mathcal{C}[\![w]\!]) = \{(\sigma,\sigma') \mid (\sigma, \textbf{false}) \in \mathcal{B}[\![b]\!] \\
\wedge\, (\sigma,\sigma') \in \mathcal{C}[\![\textbf{skip}]\!]\} \;\cup \\
\{(\sigma,\sigma') \mid (\sigma, \textbf{true}) \in \mathcal{B}[\![b]\!] \\
\wedge\, (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}
\end{aligned}
$$

Unfolding

$$
\begin{aligned}
F_{b,c}(\mathcal{C}[\![w]\!]) = \{(\sigma,\sigma') \mid (\sigma, \mathbf{false}) \in \mathcal{B}[\![b]\!] \\
\wedge\, (\sigma,\sigma') \in \mathcal{C}[\![\mathbf{skip}]\!]\} \;\cup \\
\{(\sigma,\sigma') \mid (\sigma, \mathbf{true}) \in \mathcal{B}[\![b]\!] \\
\wedge\, (\sigma,\sigma') \in \mathcal{C}[\![c; w]\!]\}
\end{aligned}
$$

Unfolding

$$
\begin{aligned}
F_{b,c}(\mathcal{C}[\![w]\!]) = \{(\sigma,\sigma') \mid (\sigma, \mathbf{false}) \in \mathcal{B}[\![b]\!] \\
\wedge\, (\sigma,\sigma') \in \mathcal{C}[\![\mathbf{skip}]\!]\} \;\cup \\
\{(\sigma,\sigma') \mid (\sigma, \mathbf{true}) \in \mathcal{B}[\![b]\!] \\
\wedge\, (\sigma,\sigma') \in \mathcal{C}[\![c; w]\!]\} \\
= \mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c; w\ \mathbf{else\ skip}]\!]
\end{aligned}
$$

Unfolding

$$
\begin{aligned}
&\mathcal{C}[\![w]\!] && = \\
&F_{b,c}(\mathcal{C}[\![w]\!]) && = \\
&\mathcal{C}[\![\textbf{if } b \textbf{ then } c; w \textbf{ else skip}]\!]
\end{aligned}
$$

Domain Theory

Complete Partial Order (cpo)

A partial order which has a least upper bound $\bigsqcup_{n \in \omega} d_n$ in D of any ω-chain (infinite increasing chain) $d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n \sqsubseteq \ldots$ of elements of D.

The partial function **Store** $\rightharpoonup$ **Store** is a cpo, where the order is defined by as "less partial".

Monotonicity

A function $f : D \to E$ between cpo's D and E is monotonic iff

$$\forall d, d' \in D. d \sqsubseteq d' \longrightarrow f(d) \sqsubseteq f(d').$$

The higher-order function $F_{b,c}$ defining **while** b **do** c is monotonic.

Continuity

A function is *continuous* iff it is monotonic and for all chains $d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n \sqsubseteq \ldots$ in D we have

$$\bigsqcup_{n\in\omega} f(d_n) = f(\bigsqcup_{n\in\omega} d_n).$$

The higher-order function $F_{b,c}$ defining **while** b **do** c is continuous.

cpo *with bottom*

A cpo which has a least element $\bot$, that is an element which is less than every other element.

The cpo **Store** $\rightharpoonup$ **Store** has a least element $\bot$: the partial function defined nowhere.

Fixed Point Theorem

Let $f : D \to D$ be a continuous function on D a cpo with bottom $\bot$. Define

$$fix(f) = \bigsqcup_{n \in \omega} f^n(\bot).$$

Then $fix(f)$ is a least fixed point of f.