The Formal Semantics of Programming Languages

Yuxin Deng

East China Normal University

http://basics.sjtu.edu.cn/∼yuxin/

April 1, 2016

Reading materials

1. Glynn Winskel. The Formal Semantics of Programming Languages: An Introduction. The MIT Press, 1993.
2. Peter Selinger. Lecture Notes on the Lambda Calculus. `http://www.mathstat.dal.ca/~selinger/papers/lambdanotes.pdf`
3. Benjamin C. Pierce et al. Software Foundations. `http://www.cis.upenn.edu/~bcpierce/sf/current/index.html`
4. John C. Mitchell. Foundations for Programming Languages. The MIT Press, 1996.
5. Robert Harper. Practical Foundations for Programming Languages. `http://www.cs.cmu.edu/~rwh/plbook/book.pdf`

Why formal semantics?

- To understand how programs behave
- To build a mathematical model useful for program analysis and verification

Three kinds of semantics (1/3)

- Operational semantics: describing the meaning of a programming language by specifying how it executes on an abstract machine.
  Gordon Plotkin
- Denotational semantics: defining the meaning of programming languages by mathematical concepts.
  Christopher Strachey, Dana Scott
- Axiomatic semantics: giving the meaning of a programming construct by axioms or proof rules in a program logic.
  R.W. Floyd, C.A.R. Hoare

Three kinds of semantics (2/3)

- Operational semantics: very helpful in implementation
- Denotational semantics: provides deep and widely applicable techniques for various languages
- Axiomatic semantics: useful in developing and verifying programs

Three kinds of semantics (3/3)

Different styles of semantics are dependent on each other. E.g.

- To show the proof rules of an axiomatic semantics are correct, use an underlying denotational or operational semantics.
- To show an implementation correct wrt denotational semantics, need to show the operational and denotational semantics agree.
- To justify an operational semantics, use a denotational semantics to abstract away from unimportant implementation details so to understand high-level computational behavior.

Chapter 1. Basic set theory

1.1 Logical notation

Let A and B be statements

- $A \ \& \ B$: the conjunction of A and B
- $A||B$: the disjunction of A and B
- $A \Rightarrow B$: if A then B
- $A \Leftrightarrow B$: logical equivalence of A and B
- $\exists x.P(x)$: there exists some x such that $P(x)$ holds
- $\exists!x.P(x)$: there exists q unique x such that $P(x)$ holds
- $\forall x.P(x)$: for all x, $P(x)$ holds

1.2 Sets (1/3)

- $\{x \mid P(x)\}$: specify a set with property $P(x)$
- Russell's paradox: $R = \{x \mid x \notin x\}$ is not a set.
- So we assume all sets in the textbook are properly constructed.
- $\emptyset$: the *null* or *empty* set
- $\omega = \{0, 1, 2, ...\}$

1.2 Sets (2/3)

- Powerset: $\mathcal{P}ow(X) = \{Y \mid Y \subseteq X\}$.
- Indexed set: $\{x_i \mid i \in I\}$.
- Big union: Let X be a set of sets. $\bigcup X = \{a \mid \exists x \in X . a \in x\}$
- When $X = \{x_i \mid i \in I\}$ for some indexing set I we write $\bigcup X$ as $\bigcup_{i \in I} x_i$.
- Big intersection: Let X be a nonempty set of sets. $\bigcap X = \{a \mid \forall x \in X . a \in x\}$
- When $X = \{x_i \mid i \in I\}$ for a nonempty indexing set I we write $\bigcap X$ as $\bigcap_{i \in I} x_i$.

1.2 Sets (3/3)

- Product: $X \times Y = \{(a, b) \mid a \in X \ \& \ b \in Y\}$.
- More generally, $X_1 \times X_2 \times ... \times X_n$ consists of the set of n-tuples $(x_1, x_2, ..., x_n) = (x_1, (x_2, (x_3, ...)))$.
- Disjoint union:
  $X_0 \uplus X_1 \uplus \cdots \uplus X_n = (\{0\} \times X_0) \cup (\{1\} \times X_1) \cup \ldots \cup (\{n\} \times X_n)$
- Set difference: $X \backslash Y = \{x \mid x \in X \ \& \ x \notin Y\}$
- The axiom of foundation: Any descending chain of memberships

$$\ldots b_n \in \ldots \in b_1 \in b_0$$

  must be finite. Thus no set can be a member of itself. It is an assumption generally made in set theory.

1.3 Relations and functions (1/3)

- A binary relation between X and Y is an element of $\mathcal{P}ow(X \times Y)$.
- When R is a relation $R \subseteq X \times Y$, we write xRy for $(x, y) \in R$.
- A partial function from X to Y is a relation $f \subseteq X \times Y$ with

$$\forall x, y, y'.(x, y) \in f \;\&\; (x, y') \in f \Rightarrow y = y'$$

  We write $f(x) = y$ when $(x, y) \in f$ for some y and say $f(x)$ is *defined*, otherwise $f(x)$ is *undefined*. Sometimes we write $f : x \mapsto y$ or $x \mapsto y$ when f is understood, for $y = f(x)$

- A (total) function from X to Y is a special partial function such that $\forall x \in X.\exists y \in Y.f(x) = y$.
- Write $(X \rightharpoonup Y)$ for the set of all partial function from X to Y, and $(X \rightarrow Y)$ for the set of all total functions.

1.3 Relations and functions (2/3)

- Lambda notation To write a function without naming it.
  $\lambda x \in X.e \;=\; \{(x, e) \mid x \in X\}$

- Let $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ be two relations. Their composition is
  $S \circ R =_{def} \{(x, z) \in X \times Z \mid \exists y \in Y.(x, y) \in R \;\&\; (y, z) \in S\}$

- For functions $f : X \to Y$ and $g : Y \to Z$, their composition is the function $g \circ f : X \to Z$.

- Each set X is associated with an identity function
  $Id_X = \{(x, x) \mid x \in X\}$.

- A function $f : X \to Y$ has an inverse $g : Y \to X$ iff $g(f(x)) = x$ for all $x \in X$ and $f(g(y)) = y$ for all $y \in Y$. Then X and Y are said to be in $1 - 1$ correspondence.

1.3 Relations and functions (3/3)

- Let $R : X \times Y$ and $A \subseteq X$. The direct image of A under R $RA = \{y \in Y \mid \exists x \in A.\ (x, y) \in R\}$
- Let $B \subseteq Y$. The inverse image of B under R $R^{-1}B = \{x \in X \mid \exists y \in B.\ (x, y) \in R\}$
- If R is an equivalence relation on X, then the $(R-)$equivalence class of an element $x \in X$ is $\{x\}_R =_{def} \{y \in X \mid yRx\}$.
- Let $R^0 = Id_X$, define $R^{n+1} = R \circ R^n$ for all $n \geq 0$. The transitive closure of R is $R^+ = \bigcup_{n\in\omega} R^{n+1}$. The reflexive, transitive closure of R is $R^* = Id_X \cup R^+ = \bigcup_{n\in\omega} R^n$.

1.3 Georg Cantor's diagonal argument (1/2)

Theorem 0.1 Let X be any set, X and $\mathcal{P}ow(X)$ are never in $1-1$ correspondence.

Proof: Suppose there exists a 1-1 correspondence $\theta : X \rightarrow \mathcal{P}ow(X)$. Form the set $Y = \{x \in X \mid x \notin \theta(x)\}$. Now $Y \in \mathcal{P}ow(X)$ and is in correspondence with some $y \in X$, i.e. $\theta(y) = Y$.

- If $y \in Y$ then $y \notin \theta(y) = Y$.
- If $y \notin Y = \theta(y)$ then $y \in Y$.

So the correspondence θ does not exist at all. $\square$

1.3 Georg Cantor's diagonal argument (2/2)

Theorem 0.2 $\mathbb{N}$ and $\mathcal{P}ow(\mathbb{N})$ are never in $1-1$ correspondence.

$$
\begin{array}{c|cccccc}
 & \theta(x_0) & \theta(x_1) & \theta(x_2) & \cdots & \theta(x_j) & \cdots \\
\hline
x_0 & 0 & 1 & 1 & \cdots & 1 & \cdots \\
x_1 & 1 & 1 & 1 & \cdots & 1 & \cdots \\
x_2 & 0 & 0 & 0 & \cdots & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & & \vdots & \\
x_i & 0 & 1 & 0 & \cdots & 1 & \cdots \\
\vdots & \vdots & \vdots & \vdots & & \vdots & \\
x_i & 0 & 1 & 0 & \cdots & 1 & \cdots
\end{array}
$$

In the ith row and jth column is placed 1 if $x_i \in \theta(x_j)$ and 0 otherwise.

Chapter 2. Operational semantics

2.1 IMP- a simple imperative language

Some syntactic sets in **IMP**.

- numbers **N**, consisting of all integer numbers, ranged over by metavariables n, m
- truth values **T**=$\{$**true**, **false**$\}$,
- locations **Loc**, ranged over by X, Y
- arithmetic expressions **Aexp**, ranged over by a
- boolean expressions **Bexp**, ranged over by b
- commands **Com**, ranged over by c

Sometimes we use metavariable which are primed or subscripted, e.g. X', X_0 for locations.

2.1 IMP- a simple imperative language

The syntax of **IMP**defined by BNF (Backus-Naur form).

- For **Aexp**: $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
- For **Bexp**: $b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
- For **Com**:
  $c ::= \mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$

2.1 IMP- a simple imperative language

The syntax of **IMP**defined by BNF (Backus-Naur form).

- For **Aexp**: $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
- For **Bexp**: $b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
- For **Com**:
  $c ::= \mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$
- From set-theoretic point of view, this notation gives an inductive definition of the syntactic sets, the least sets closed under the formation rules.
- Syntactic equivalence $\equiv$. e.g. $3 + 4 \not\equiv 4 + 3$.

2.2 The evaluation of arithmetic expressions

- The set of states consists of functions $\sigma : \mathbf{Loc} \rightarrow \mathbf{N}$.
- A configuration is a pair $\langle a, \sigma \rangle$, where a is an arithmetic expression and σ a state.
- An evaluation relation between pairs and numbers $\langle a, \sigma \rangle \rightarrow n$

2.2 Structural operational semantics

Evaluation of numbers $\langle n, \sigma \rangle \to n$

Evaluation of locations $\langle X, \sigma \rangle \to \sigma(X)$

Evaluation of sums

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the sum of } n_0 \text{ and } n_1}{\langle a_0 + a_1, \sigma \rangle \to n}$$

Evaluation of subtractions

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the result of subtracting } n_1 \text{ from } n_0}{\langle a_0 - a_1, \sigma \rangle \to n}$$

Evaluation of products

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the product of } n_0 \text{ and } n_1}{\langle a_0 \times a_1, \sigma \rangle \to n}$$

2.2 Derivation tree

$$\dfrac{\dfrac{\dfrac{}{\langle Init, \sigma_0\rangle \to 0} \qquad \dfrac{}{\langle 5, \sigma_0\rangle \to 5}}{\langle (Init + 5), \sigma_0\rangle \to 5} \qquad \dfrac{\dfrac{}{\langle 7, \sigma_0\rangle \to 7} \qquad \dfrac{}{\langle 9, \sigma_0\rangle \to 9}}{\langle 7 + 9, \sigma_0\rangle \to 16}}{\langle (Init + 5) + (7 + 9), \sigma_0\rangle \to 21}$$

2.2 Equivalence of arithmetic expressions

Two arithmetic expressions are equivalent if they evaluate to the same value in all states.

$a_0 \sim a_1 \quad \text{iff} \quad \forall\sigma \in \Sigma\ \forall n \in \mathbf{N}.\ \langle a_0, \sigma\rangle \rightarrow n \Leftrightarrow \langle a_1, \sigma\rangle \rightarrow n$

2.3 The evaluation of boolean expressions

$$\langle \mathbf{true}, \sigma \rangle \to \mathbf{true} \qquad \langle \mathbf{false}, \sigma \rangle \to \mathbf{false}$$

$$\frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to n}{\langle a_0 = a_1, \sigma \rangle \to \mathbf{true}} \qquad \frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to m \qquad n \not\equiv m}{\langle a_0 = a_1, \sigma \rangle \to \mathbf{false}}$$

$$\frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to m \qquad \text{if } n \text{ is less than or equal to } m}{\langle a_0 \leq a_1, \sigma \rangle \to \mathbf{true}}$$

$$\frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to m \qquad \text{if } n \text{ is not less than or equal to } m}{\langle a_0 \leq a_1, \sigma \rangle \to \mathbf{false}}$$

$$\frac{\langle b, \sigma \rangle \to \mathbf{true}}{\langle \neg b, \sigma \rangle \to \mathbf{false}} \qquad \frac{\langle b, \sigma \rangle \to \mathbf{false}}{\langle \neg b, \sigma \rangle \to \mathbf{true}}$$

$$\frac{\langle b_0, \sigma \rangle \to t_0 \qquad \langle b_1, \sigma \rangle \to t_1 \qquad \text{if } t \text{ is } \mathbf{true} \text{ iff } t_0 \equiv t_1 \equiv \mathbf{true}}{\langle b_0 \wedge b_1, \sigma \rangle \to t}$$

$$\frac{\langle b_0, \sigma \rangle \to t_0 \qquad \langle b_1, \sigma \rangle \to t_1 \qquad \text{if } t \text{ is } \mathbf{false} \text{ iff } t_0 \equiv t_1 \equiv \mathbf{false}}{\langle b_0 \vee b_1, \sigma \rangle \to t}$$

2.4 The execution of commands

A (command) configuration is a pair $\langle c, \sigma \rangle$ where c is a command and σ a state. The execution of commands are defined via relations $\langle c, \sigma \rangle \to \sigma'$

Notation. Write $\sigma[m/X]$ for the state satisfying

$$\sigma[m/X](Y) = \begin{cases} m & \text{if } Y = X \\ \sigma(Y) & \text{if } Y \neq X \end{cases}$$

2.4 The execution of commands

Atomic commands

$$\langle \mathbf{skip}, \sigma\rangle \rightarrow \sigma \qquad \frac{\langle a, \sigma\rangle \rightarrow m}{\langle X := a, \sigma\rangle \rightarrow \sigma[m/X]}$$

Sequencing $$\frac{\langle c_0, \sigma\rangle \rightarrow \sigma'' \qquad \langle c_1, \sigma''\rangle \rightarrow \sigma'}{\langle c_0; c_1, \sigma\rangle \rightarrow \sigma'}$$

Conditionals

$$\frac{\langle b, \sigma\rangle \rightarrow \mathbf{true} \qquad \langle c_0, \sigma\rangle \rightarrow \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \rightarrow \sigma'} \qquad \frac{\langle b, \sigma\rangle \rightarrow \mathbf{false} \qquad \langle c_1, \sigma\rangle \rightarrow \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \rightarrow \sigma'}$$

While-loops

$$\frac{\langle b, \sigma\rangle \rightarrow \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \rightarrow \sigma}$$

$$\frac{\langle b, \sigma\rangle \rightarrow \mathbf{true} \qquad \langle c, \sigma\rangle \rightarrow \sigma'' \qquad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma''\rangle \rightarrow \sigma'}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \rightarrow \sigma'}$$

2.4 Big step semantics

To see the semantics just defined is a big step semantics, consider the following program:

$$
\begin{aligned}
Factorial \quad \equiv \quad & Y := 1; \\
& \textbf{while } X > 1 \textbf{ do} \\
& \quad \{Y := Y \times X;\ X := X - 1\}; \\
& Z := Y
\end{aligned}
$$

Let σ be a state with $\sigma(X) = 3$, what's the state σ' such that $\langle Factorial, \sigma\rangle \to \sigma'$? Construct the derivation tree.

2.4, 2.5 Equivalence of commands

Definition 0.3 $c_0 \sim c_1$ iff $\forall \sigma, \sigma' \in \Sigma.\ \langle c_0, \sigma\rangle \to \sigma' \Leftrightarrow \langle c_1, \sigma\rangle \to \sigma'$

Proposition 0.4 Let $w \equiv \textbf{while } b \textbf{ do } c$ with $b \in \textbf{Bexp}$ and $c \in \textbf{Com}$. Then

$$w \ \sim\ \textbf{if } b \textbf{ then } c; w \textbf{ else skip}.$$

Proof: Show that $\langle w, \sigma\rangle \to \sigma'$ iff $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma\rangle \to \sigma'$ for all states σ, σ'. Inspecting the rules with matching conclusions. cf. Page 21. $\square$

2.6 Small step semantics

For example,

$$\frac{\langle a_0, \sigma\rangle \rightarrow_1 \langle a_0', \sigma\rangle}{\langle a_0 + a_1, \sigma\rangle \rightarrow_1 \langle a_0' + a_1, \sigma\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_1 \langle a_1', \sigma\rangle}{\langle n + a_1, \sigma\rangle \rightarrow_1 \langle n + a_1', \sigma\rangle}$$

$$\langle n + m, \sigma\rangle \rightarrow_1 \langle p, \sigma\rangle \qquad p \text{ is the sume of } n \text{ and } m$$

$$\langle X := 5; Y := 1, \sigma\rangle \rightarrow_1 \langle Y := 1, \sigma[5/X]\rangle \rightarrow_1 \sigma[5/X][1/Y]$$

Chapter 3. Some principles of induction

3.1 Mathematical induction

The principle of mathematical induction: Let $P(n)$ be a property of the natural number n. To show $P(n)$ holds for all natural numbers n it is sufficient to show

- $P(n)$ is true
- If $P(m)$ is true then so is $P(m+1)$ for any natural number m.

I.e. $(P(0) \;\; \& \;\; (\forall m \in \omega.\ P(m) \Rightarrow P(m+1))) \Rightarrow \forall n \in \omega.\ P(n)$ where

- $P(0)$ is the induction basis
- $P(m)$ the induction hypothesis
- $(\forall m \in \omega.\ P(m) \Rightarrow P(m+1))$ the induction step.

3.1 Course-of-values induction

If a property Q's truth at $m+1$ depends on not just its truth at m but also its truth at other numbers preceding m as well, we strengthen the induction hypothesis to be $\forall k < m.\ Q(k)$. Then

- the basis: $\forall k < 0.\ Q(k)$ — vacuously true.
- the induction step: $\forall m \in \omega.\ ((\forall k < m.\ Q(k)) \Rightarrow (\forall k < m+1.\ Q(k)))$ — equivalent to $\forall m \in \omega.\ (\forall k < m.\ Q(k)) \Rightarrow Q(m)$

So as a special form of mathematical induction is course-of-values induction: $(\forall m \in \omega.\ (\forall k < m.\ Q(k)) \Rightarrow Q(m)) \Rightarrow \forall n \in \omega.\ Q(n)$.

3.2 Structural induction

Let $P(a)$ be a property of arithmetic expression a. To show $P(a)$ holds for all arithmetic expressions a it is sufficient to show:

- For all numerals m, $P(m)$ holds.
- For all locations X, $P(X)$ holds.
- For all arithmetic expressions a_0 and a_1, if $P(a_0)$ and $P(a_1)$ hold then so does $P(a_0 + a_1)$.
- Similarly with $P(a_0 - a_1)$ and $P(a_0 \times a_1)$.

3.2 Structural induction: an example

Proposition 0.5 For all arithmetic expressions a, states σ and numbers m, m', $\langle a, \sigma\rangle \to m \ \ \& \ \ \langle a, \sigma\rangle \to m' \Rightarrow m = m'$.

Proof: By structural induction on arithmetic expressions a using induction hypothesis $P(a)$ where
$P(a)$ iff $\forall \sigma, m, m'. \ (\langle a, \sigma\rangle \to m \ \ \& \ \ \langle a, \sigma\rangle \to m' \Rightarrow m = m')$

- $a \equiv n$: since there is only one rule for evaluating $\langle n, \sigma\rangle$, trivial.
- $a \equiv a_0 + a_1$: Again one rule for evaluating $\langle a_0 + a_1, \sigma\rangle$. So $\langle a_0, \sigma\rangle \to m_0$ and $\langle a_1, \sigma\rangle \to m_1$ with $m = m_0 + m_1$ and $\langle a_0, \sigma\rangle \to m_0'$ and $\langle a_1, \sigma\rangle \to m_1'$ with $m' = m_0' + m_1'$. By induction hypothesis applied to a_0, a_1 we obtain $m_0 = m_0'$ and $m_1 = m_1'$. Thus $m = m'$.
- The remaining cases are similar.

$\square$

3.3 Well-founded relation

A well-founded relation is a binary relation $\prec$ on a set A such that there are no infinite descending chains $\cdots \prec a_i \prec \cdots \prec a_1 \prec a_0$. If $a \prec b$ then a is a predecessor of b.

3.3 Well-founded relation

Proposition 0.6 The relation $\prec$ on set A is well-founded iff any nonempty subset Q of A has a minimal element, i.e. an element m with $m \in Q \;\; \& \;\; \forall b \prec m.b \notin Q$.

Proof: ($\Leftarrow$) Suppose every nonempty subset of A has a minimal element, but there is an infinite chain $\cdots \prec a_1 \prec a_0$. The set $\{a_i \mid i \in \omega\}$ would have no minimal element, a contradiction.

($\Rightarrow$) Take any element a_0 from Q. Inductively, assume a chain $a_n \prec \cdots \prec a_0$ has been constructed inside Q. If there is $b \prec a_n$ with $b \in Q$, take $a_{n+1} = b$, otherwise stop the construction. As $\prec$ is well-founded, the chain is finite whose least element is minimal in Q. $\square$

3.3 The principle of well-founded induction

Proposition 0.7 Let $\prec$ be well founded on set A, and P be a property. Then $\forall a.P(a)$ iff $\forall a \in A.((\forall b \prec a.\ P(b)) \Rightarrow P(a))$.

Proof: $(\Rightarrow)$ Trivial.

$(\Leftarrow)$ Suppose $\forall a \in A.((\forall b \prec a.\ P(b)) \Rightarrow P(a))$ but $\neg P(a)$ for some $a \in A$. The set $\{a \in A \mid \neg P(a)\}$ has a minimal element m. Then $\forall b \prec m.P(b)$ but $\neg P(m)$, contradicting the assumption. $\square$

In mathematics this principle is called Noetherian induction after the German algebraist Emmy Noether.

3.3 The principle of well-founded induction

Proposition 0.6 provides an alternative to proofs by well-founded induction. To show property P holds for every element in a well-founded set A, it is sufficient to show the subset of counterexamples $\{a \in A \mid \neg P(a)\}$ is empty. Suppose it's nonempty, there is a minimal element m contradicting the assumption $(\forall b \prec m.P(b)) \Rightarrow P(m)$.

3.3 The principle of well-founded induction: an example

Euclid's algorithm for the greatest common divisor of M, N.

$$Euclid \equiv \textbf{while } \neg(M = N) \textbf{ do}$$
$$\textbf{if } M \leq N \textbf{ then } N := N - M \textbf{ else } M := M - N$$

Theorem 0.8 For all states σ,
$\sigma(M) \geq 1 \;\;\&\;\; \sigma(N) \geq 1 \Rightarrow \exists \sigma'. \, \langle Euclid, \sigma \rangle \to \sigma'$.

Proof: Let $S = \{\sigma \in \Sigma \mid \sigma(M) \geq 1 \;\;\&\;\; \sigma(N) \geq 1\}$ and $\prec$ by

$$\sigma' \prec \sigma \quad \text{iff} \quad (\sigma'(M) \leq \sigma(M) \;\;\&\;\; \sigma'(N) \leq \sigma(N)) \;\;\&$$
$$\neg(\sigma'(M) = \sigma(M) \;\&\; \sigma'(N) = \sigma(N)).$$

Then $\prec$ is well-founded. Let $P(\sigma) = \exists \sigma'. \langle Euclid, \sigma \rangle \to \sigma'$. Suppose $\forall \sigma' \prec \sigma. P(\sigma')$, we show $P(\sigma)$ with two cases: (i) $\sigma(M) = \sigma(N)$, (ii) $\sigma(M) \neq \sigma(N)$. Argue in both cases that $\langle Euclid, \sigma \rangle \to \sigma'$ for some σ'. Then conclude $\forall \sigma \in S. P(\sigma)$ by well-founded induction. $\square$

3.4 Induction on derivations

A rule instance is a pair X/y with premises X and conclusion y. Usually we write X/y as $\dfrac{}{y}$ if $X = \emptyset$, and $\dfrac{x_1, \cdots, x_n}{y}$ if $X = \{x_1, \cdots, x_n\}$

Let R be a set of rule instances. An R-derivation of y is either a rule instance $\emptyset/y$ or a pair $\{d_1, \cdots, d_n\}/y$ where $\{x_1, \cdots, x_n\}/y$ is a rule instance and d_i an R-derivation of x_i for all $1 \leq i \leq n$. Write $d \Vdash_R y$ to mean d is an R-derivation of y.

A derivation d' is an immediate subderivation of d, written $d' \prec_1 d$, iff d has the form D/y with $d' \in D$. Let $\prec$ be the transitive closure of $\prec_1$ $(\prec_1^+)$. We say d' is a proper subderivation of d iff $d' \prec d$.

Since derivations are finite, both $\prec_1$ and $\prec$ are well-founded.

3.4 Induction on derivations

Theorem 0.9 Let c be a command and σ_0 a state. If $\langle c, \sigma_0 \rangle \to \sigma$ and $\langle c, \sigma_0 \rangle \to \sigma'$, then $\sigma = \sigma_1$.

Proof: By well-founded induction on the proper subderivation relation $\prec$. For any derivation d, let $P(d)$ be the following property

$\forall c \in \mathbf{Com}, \sigma_0, \sigma, \sigma_1 \in \Sigma.\ d \Vdash \langle c, \sigma_0 \rangle \to \sigma \ \ \& \ \ \langle c, \sigma_0 \rangle \to \sigma_1 \Rightarrow \sigma = \sigma_1.$

Show that $\forall d' \prec d.P(d')$ implies $P(d)$ by inspecting the structure of c. cf. Page 37. $\square$

3.4 Induction on derivations

Proposition 0.10 $\forall c \in \mathbf{Com}, \sigma, \sigma' \in \Sigma.\ \langle \mathbf{while\ true\ do}\ c, \sigma\rangle \not\to \sigma'$.

Proof: Abbreviate $w \equiv$ **while true do** c. Suppose the set $\{d \mid \exists \sigma, \sigma' \in \Sigma.\ d \Vdash \langle w, \sigma\rangle \to \sigma'\}$ is nonempty. By Proposition 0.6 there is a minimal derivation d in the form

$$\dfrac{\dfrac{\vdots}{\langle \mathbf{true}, \sigma\rangle \to \mathbf{true}} \qquad \dfrac{\vdots}{\langle c, \sigma\rangle \to \sigma''} \qquad \dfrac{\vdots}{\langle w, \sigma''\rangle \to \sigma'}}{\langle w, \sigma\rangle \to \sigma'}$$

But this contains a proper subderivation $d' \Vdash \langle w, \sigma''\rangle \to \sigma'$, contradicting the minimality of d. $\square$

3.5 Definition by induction

Definition by well-founded induction, also called well-founded recursion, e.g.

$$size(a) = \begin{cases} 1 & \text{if } a \equiv n \text{ or } X \\ 1 + size(a_0) + size(a_1) & \text{if } a = a_0 + a_1, \\ \vdots & \end{cases}$$

Chapter 4. Inductive definitions

4.1 Rule induction

Viewed abstractly, instances of rules have the form $\emptyset/y$ or $\{x_1, \cdots, x_n\}/y$. Let R be a set of rule instances, let I_R be the set of all elements with a R-derivation, i.e. $I_R = \{x \mid \ \Vdash_R x\}$.

The general principle of rule induction:

Let I_R be defined by rule instances R and P a property. Then
$\forall y \in I_R.\ P(y)$ iff for all rule instances X/y in R for which $X \subseteq I_R$,
$(\forall x \in X.\ P(x)) \Rightarrow P(y)$.

4.1 Rule induction

The general principle of rule induction says: for rule instances R we have $\forall y \in I_R.\ P(y)$ iff

- for all instances of axioms $\dfrac{}{y}$, $P(y)$ is true, and
- for all rule instances $\dfrac{x_1, \cdots, x_n}{y}$, if $\forall 1 \leq i \leq n.\ x_i \in I_R \ \ \&\ \ P(x_i)$ then $P(y)$ is true.

4.1 R-closure

A set Q is closed under rule instances R, or R-closed, iff for all rule instances X/y, we have $X \subseteq Q \Rightarrow y \in Q$.

Proposition 0.11 With respect rule instances R,

1. I_R is R-closed.
2. If Q is an R-closed set, then $I_R \subseteq Q$.

Proof: 1. By definition, if $\{x_1, \cdots, x_n\}/y$ is a rule instance, then each x_i has derivation d_i. Combining these d_i with the rule instance gives a derivation of y.

2. Each element in I_R has a derivation. So we do an induction on the subderivation relation $\prec$ to show $\forall y \in I_R.\ d \Vdash_R y \Rightarrow y \in Q$ for all R-derivations d.

$\square$

4.1 Rule induction

Let P be a property. To show P is true of all elements of I_R, define the set $Q = \{x \in I_R \mid P(x)\}$, and Proposition 0.11 says it's sufficient to show Q is R-closed, i.e. for all rule instances X/y,
$(\forall x \in X.\ x \in I_R \ \&\ P(x)) \Rightarrow P(y)$.

4.2 Special rule induction

Consider the rule for commands

$$\frac{X : \mathbf{Loc} \quad a : \mathbf{Aexp}}{X := a \ : \ \mathbf{Com}}$$

In general a rule instance may not be homogeneous, then it's awkward to directly use rule induction.

The special principle of rule induction:
Let I_R be defined by rule instances R and $A \subseteq I_R$. Let Q be a property. Then $\forall a \in A.\ Q(a)$ iff for all rule instances X/y with $X \subseteq I_R$ and $y \in A$, $(\forall x \in X \cap A.\ Q(x)) \Rightarrow Q(y)$.

4.2 Special vs. general rule induction

The special principle follows from the general one.

Let $Q(x)$ be a property we are interested in showing is true of all elements of A. Define property $P(x)$ by $P(x) \Leftrightarrow (x \in A \Rightarrow Q(x))$. Then $(\forall x \in A.\ Q(x)) \Leftrightarrow (\forall x \in I_R.\ P(x))$.

The general principle says for all rule instance X/y in R,

$$\begin{array}{ll} & (\forall x \in X.\ x \in I_R \ \&\ P(x)) \Rightarrow P(y) \\ \Leftrightarrow & (\forall x \in X.\ x \in I_R \ \&\ (x \in A \Rightarrow Q(x))) \Rightarrow (y \in A \Rightarrow Q(y)) \\ \Leftrightarrow & ((\forall x \in X.\ x \in I_R) \ \&\ (\forall x \in X.\ (x \in A \Rightarrow Q(x))) \ \&\ y \in A) \Rightarrow Q(y) \\ \Leftrightarrow & X \subseteq I_R \ \&\ y \in A \ \&\ (\forall x \in X.\ (x \in A \Rightarrow Q(x))) \Rightarrow Q(y) \\ \Leftrightarrow & X \subseteq I_R \ \&\ y \in A \ \&\ (\forall x \in X \cap A.\ Q(x)) \Rightarrow Q(y) \end{array}$$

4.3 Rule induction for arithmetic expressions

$\forall a \in \mathbf{Aexp}, \sigma \in \Sigma, n \in \mathbf{N}.\ \langle a, \sigma \rangle \to n \Rightarrow P(a, \sigma, n)$

iff

$(\ \forall n \in \mathbf{N}, \sigma \in \Sigma.\ P(n, \sigma, n)$

$\&$

$\forall X \in \mathbf{Loc}, \sigma \in \Sigma.\ P(X, \sigma, \sigma(X))$

$\&$

$\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, n_0, n_1 \in \mathbf{N}.$

$\langle a_0, \sigma \rangle \to n_0 \ \ \&\ \ P(a_0, \sigma, n_0) \ \ \&\ \ \langle a_1, \sigma \rangle \to n_1 \ \ \&\ \ P(a_1, \sigma, n_1)$

$\Rightarrow P(a_0 + a_1, \sigma, n_0 + n_1)$

$\&$

$\cdots\ )$

4.3 Rule induction for boolean expressions

$\forall b \in \mathbf{Bexp}, \sigma \in \Sigma, t \in \mathbf{T}.\ \langle b, \sigma \rangle \to t \Rightarrow P(b, \sigma, t)$

iff

$(\ \forall \sigma \in \Sigma.\ P(\mathbf{false}, \sigma, \mathbf{false})\ \ \&\ \ \forall \sigma \in \Sigma.\ P(\mathbf{false}, \sigma, \mathbf{false})$

$\&$

$\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, m, n \in \mathbf{N}.$

$\langle a_0, \sigma \rangle \to m\ \ \&\ \ \langle a_1, \sigma \rangle \to n\ \ \&\ \ m = n\ \ \Rightarrow\ \ P(a_0 = a_1, \sigma, \mathbf{true})$

$\&$

$\forall b_0, b_1 \in \mathbf{Bexp}, \sigma \in \Sigma, t_0, t_1 \in \mathbf{T}.$

$\langle b_0, \sigma \rangle \to t_0\ \ \&\ \ P(b_0, \sigma, t_0)\ \ \&\ \ \langle b_1, \sigma \rangle \to t_1\ \ \&\ \ P(b_1, \sigma, t_1)$

$\Rightarrow P(b_0 \wedge b_1, \sigma, t_0 \wedge t_1)$

$\&$

$\cdots\ )$

4.3 Rule induction for commands

$\forall c \in \mathbf{Com}, \sigma, \sigma' \in \Sigma.\ \langle c, \sigma \rangle \to \sigma' \Rightarrow P(c, \sigma, \sigma')$

iff

$(\ \forall \sigma \in \Sigma.\ P(\mathbf{skip}, \sigma, \sigma)\ \ \&$

$\ldots$

$\ \&$

$\forall c \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma \in \Sigma.$

$\langle b, \sigma \rangle \to \mathbf{false}\ \Rightarrow\ P(\mathbf{while}\ b\ \mathbf{do}\ c, \sigma, \sigma)$

$\ \&$

$\forall c \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma, \sigma', \sigma'' \in \Sigma.$

$\langle b, \sigma \rangle \to \mathbf{true}\ \ \&\ \ \langle c, \sigma \rangle \to \sigma''\ \ \&\ \ P(c, \sigma, \sigma'')\ \ \&$

$\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \to \sigma'\ \ \&\ \ P(\mathbf{while}\ b\ \mathbf{do}\ c, \sigma'', \sigma')$

$\Rightarrow P(\mathbf{while}\ b\ \mathbf{do}\ c, \sigma, \sigma')\ )$

4.3 Rule induction for commands: an example

Proposition 0.12 Let $Y \in \mathbf{Loc}$. For all commands c and states σ, σ', $(Y \notin loc_L(c) \;\; \& \;\; \langle c, \sigma \rangle \to \sigma') \;\Rightarrow\; \sigma(Y) = \sigma'(Y)$.

Proof: Let P be the property given by:
$P(c, \sigma, \sigma') \;\Leftrightarrow\; (Y \notin loc_L(c) \Rightarrow \sigma(Y) = \sigma'(Y))$. Then use rule induction on commands to show that
$\forall c \in \mathbf{Com}, \sigma, \sigma' \in \Sigma.\ \langle c, \sigma \rangle \to \sigma' \Rightarrow P(c, \sigma, \sigma')$. $\square$

4.4 Operators and their least fixed points

A set of rule instances R determines an operator $\hat{R}$ on sets by $\hat{R}(B) = \{y \mid \exists X \subseteq B.\ (X/y) \in R\}$.

Proposition 0.13 1. A set B is closed under R iff $\hat{R}(B) \subseteq B$

2. $\hat{R}$ is monotonic.

Proof: Directly from definitions. $\square$

4.4 Operators and their least fixed points

Let $A_0 = \emptyset,\ A_{n+1} = \hat{R}^{n+1}(\emptyset),\ A = \bigcup_{n\in\omega} A_n$.

Proposition 0.14 1. A is R-closed

2. $\hat{R}(A) = A$

3. A is the least R-closed set.

Proof:

1. Suppose $(X/y) \in R$ with $X \subseteq A$. As X is a finite set, say $\{x_1, \cdots, x_k\}$, with $X \subseteq A$, then $\forall 1 \leq i \leq k.\ x_i \in A_{n_i}$. Take n bigger than all n_i, we have $\forall 1 \leq i \leq k.\ x_i \in A_n$, i.e. $X \subseteq A_n$. Then $y \in \hat{R}(A_n) \subseteq A$.

4.4 Operators and their least fixed points

2 It's easy to see that A is R-closed, thus $\hat{R}(A) \subseteq A$. For the converse, let $y \in A$. Then $y \in A_n$ for some $n > 0$. Thus $y \in \hat{R}(A_{n-1})$. So there is some $(X/y) \in R$ with $X \subseteq A_{n-1} \subseteq A$, giving $y \in \hat{R}(A)$. Thus $A \subseteq \hat{R}(A)$.

3 Suppose B is R-closed, then $\hat{R}(B) \subseteq B$. Show by mathematical induction that $\forall n \in \omega.\ A_n \subseteq B$. For the induction step, assume $A_n \subseteq B$. Then $A_{n+1} = \hat{R}(A_n) \subseteq \hat{R}(B) \subseteq B$. Thus, $A \subseteq B$. $\square$

4.4 Operators and their least fixed points

- It's essential in Proposition 0.14 that all rule instances are finitary, i.e. all premises X are finite sets.
- Parts 1 and 3 of Proposition 0.14 say $A = I_R$.
- Parts 2 and 3 of Proposition 0.14 say I_R is the least fixed point of $\hat{R}$.

The lambda calculus

Computability

A question in the 1930's: what does it mean for a function $f : \mathbb{N} \to \mathbb{N}$ to be computable?

Informally, there should be a pencil-and-paper method allowing a trained person to calculate $f(n)$, for any given n.

- Turing defined a Turing machines and postulated that a function is computable if and only if it can be computed by such a machine.
- Gödel defined the class of general recursive functions and postulated that a function is computable if and only if it is general recursive.
- Church defined the lambda calculus and postulated that a function is computable if and only if it can be written as a lambda term.

Church, Kleene, Rosser, and Turing proved that all three computational models were equivalent to each other.

The untyped lambda calculus

Def. Assume an infinite set $\mathcal{V}$ of variables, denoted by $x, y, z....$ The set of lambda terms are defined by the Backus-Naur Form:

$$M, N ::= x \mid (MN) \mid (\lambda x.M)$$

Alternatively, the set of lambda terms is the smallest set Λ satisfying:

- whenever $x \in \mathcal{V}$ then $x \in \Lambda$ (variables)
- whenever $M, N \in \Lambda$ then $(MN) \in \Lambda$ (applications)
- whenever $x \in \mathcal{V}$ and $M \in \Lambda$ then $(\lambda x.M) \in \Lambda$ (lambda abstractions)

E.g. $(\lambda x.x)$ $\quad$ $((\lambda x.(xx))(\lambda y.(yy)))$ $\quad$ $(\lambda f.(\lambda x.(f(fx))))$

Convention

- Omit outermost parentheses. E.g., write MN instead of (MN).
- Applications associate to the left, i.e. MNP means $(MN)P$.
- The body of a lambda abstraction (the part after the dot) extends as far to the right as possible. E.g, $\lambda x.MN$ means $\lambda x.(MN)$, and not $(\lambda x.M)N$.
- Multiple lambda abstractions can be contracted; E.g., write $\lambda xyz.M$ for $\lambda x.\lambda y.\lambda z.M$.

Free and bound variables

An occurrence of a variable x inside $\lambda x.N$ is said to be bound. The corresponding λx is called a binder, and the subterm N is the scope of the binder. A variable occurrence that is not bound is free.

E.g. in $M \equiv (\lambda x.xy)(\lambda y.yz)$, x is bound, z is free, variable y has both a free and a bound occurrence.

The set of free variables of term M is $FV(M)$:

$$\begin{array}{rcl} FV(x) & = & \{x\} \\ FV(MN) & = & FV(M) \cup FV(N) \\ FV(\lambda x.M) & = & FV(M) \backslash \{x\} \end{array}$$

Renaming

Write $M\{y/x\}$ for the renaming of x as y in M.

$$
\begin{array}{rcl}
x\{y/x\} & \equiv & y \\
z\{y/x\} & \equiv & z, \qquad \text{if } x \neq z \\
(MN)\{y/x\} & \equiv & (M\{y/x\})(N\{y/x\}) \\
(\lambda x.M)\{y/x\} & \equiv & \lambda y.(M\{y/x\}) \\
(\lambda z.M)\{y/x\} & \equiv & \lambda z.(M\{y/x\}), \quad \text{if } x \neq z
\end{array}
$$

α-equivalence

$$\frac{}{M = M} \qquad \frac{M = M' \quad N = N'}{MN = M'N'}$$

$$\frac{M = N}{N = M} \qquad \frac{M = M'}{\lambda x.M = \lambda x.M'}$$

$$\frac{M = N \quad N = P}{M = P} \qquad \frac{y \notin M}{\lambda x.M = \lambda y.M\{y/x\}}$$

Substitution

The capture-avoiding substitution of N for free occurrences of x in M, in symbols $M[N/x]$ is defined below:

$$
\begin{array}{lll}
x[N/x] & \equiv N & \\
y[N/x] & \equiv y, & \text{if } x \neq y \\
(MP)[N/x] & \equiv (M[N/x])(P[N/x]) & \\
(\lambda x.M)[N/x] & \equiv \lambda x.M & \\
(\lambda y.M)[N/x] & \equiv \lambda y.(M[N/x]), & \text{if } x \neq y \text{ and } y \notin FV(N) \\
(\lambda y.M)[N/x] & \equiv \lambda y'.(M\{y'/y\}[N/x]), & \text{if } x \neq y,\ y \in FV(N), \text{ and } y' \text{ fresh.}
\end{array}
$$

β-reduction

Convention: we identify lambda terms up to α-equivalence.

A term of the form $(\lambda x.M)N$ is β-redex. It reduces to $M[N/x]$ (the reduct).

A lambda term without β-redex is in β-normal form.

$$\begin{aligned} (\lambda x.y)(\underline{(\lambda z.zz)(\lambda w.w)}) & \longrightarrow_\beta (\lambda x.y)(\underline{(\lambda w.w)(\lambda w.w)}) \\ & \longrightarrow_\beta \underline{(\lambda x.y)(\lambda w.w)} \\ & \longrightarrow_\beta y \end{aligned}$$

$$\underline{(\lambda x.y)((\lambda z.zz)(\lambda w.w))} \longrightarrow_\beta y$$

Observation

- reducing a redex can create new redexes,
- reducing a redex can delete some other redexes,
- the number of steps that it takes to reach a normal form can vary, depending on the order in which the redexes are reduced.

Evaluation

Write $\twoheadrightarrow_\beta$ for $\longrightarrow_\beta^*$, the reflexive transitive closure of $\longrightarrow_\beta$. If $M \twoheadrightarrow_\beta M'$ and M' is in normal form, then we say M evaluates to M'.

Not every term has a normal form.

$$\begin{array}{rcl}(\lambda x.x)(\lambda y.yyy) & \longrightarrow_\beta & (\lambda y.yyy)(\lambda y.yyy) \\ & \longrightarrow_\beta & (\lambda y.yyy)(\lambda y.yyy)(\lambda y.yyy) \\ & \longrightarrow_\beta & \ldots\end{array}$$

Formal definition of β-reduction

The single-step β-reduction is the smallest relation $\longrightarrow_\beta$ satisfying:

$$\frac{}{(\lambda x.M)N \longrightarrow_\beta M[N/x]}$$

$$\frac{M \longrightarrow_\beta M'}{MN \longrightarrow_\beta M'N}$$

$$\frac{N \longrightarrow_\beta N'}{MN \longrightarrow_\beta MN'}$$

$$\frac{M \longrightarrow_\beta M'}{\lambda x.M \longrightarrow_\beta \lambda x.M'}$$

Write $M =_\beta M'$ if M can be transformed into M' by zero or more reductions steps and/or inverse reduction steps. Formally, $=_\beta$ is the reflexive symmetric transitive closure of $\longrightarrow_\beta$.

Programming in the untyped lambda calculus

Booleans: let $\mathbf{T} = \lambda xy.x$ and $\mathbf{F} = \lambda xy.y$.

Let $\mathbf{and} = \lambda ab.ab\mathbf{F}$. Then

$$\begin{array}{rcl} \mathbf{and}\ \mathbf{TT} & \twoheadrightarrow_\beta & \mathbf{T} \\ \mathbf{and}\ \mathbf{TF} & \twoheadrightarrow_\beta & \mathbf{F} \\ \mathbf{and}\ \mathbf{FT} & \twoheadrightarrow_\beta & \mathbf{F} \\ \mathbf{and}\ \mathbf{FF} & \twoheadrightarrow_\beta & \mathbf{F} \end{array}$$

The above encoding is not unique. The "and" function can also be encoded as $\lambda ab.bab$.

Other boolean functions

$$
\begin{aligned}
\textbf{not} &= \lambda a.a\mathbf{FT} \\
\textbf{or} &= \lambda ab.a\mathbf{T}b \\
\textbf{xor} &= \lambda ab.a(b\mathbf{FT})b \\
\textbf{if-then-else} &= \lambda x.x
\end{aligned}
$$

$$\textbf{if-then-else}\ \mathbf{T}MN \twoheadrightarrow_\beta M$$

$$\textbf{if-then-else}\ \mathbf{F}MN \twoheadrightarrow_\beta N$$

Natural numbers

Write $f^n x$ for the term $f(f(\ldots(fx)\ldots))$, where f occurs n times. The bth Church numeral $\bar{n} = \lambda fx.f^n x$.

$$\begin{array}{rcl}
\bar{0} & = & \lambda fx.x \\
\bar{1} & = & \lambda fx.fx \\
\bar{2} & = & \lambda fx.f(fx) \\
 & \ldots &
\end{array}$$

The successor function

Let $\mathbf{succ} = \lambda n f x . f(n f x)$.

$$
\begin{array}{rcl}
\mathbf{succ}\ \bar{n} & = & (\lambda n f x . f(n f x))(\lambda f x . f^n x) \\
& \longrightarrow_\beta & \lambda f x . f((\lambda f x . f^n x) f x) \\
& \twoheadrightarrow_\beta & \lambda f x . f(f^n x) \\
& = & \lambda f x . f^{n+1} x \\
& = & \overline{n+1}
\end{array}
$$

Addition and mulplication

Let **add** $= \lambda nmfx.nf(mfx)$ and **mult** $= \lambda nmf.n(mf)$

Exercises: show that

$$\begin{aligned} \textbf{add}\ \bar{n}\bar{m} &\twoheadrightarrow_\beta \overline{n+m} \\ \textbf{mult}\ \bar{n}\bar{m} &\twoheadrightarrow_\beta \overline{n \cdot m} \end{aligned}$$

Exercise: Let **iszero** $= \lambda nxy.n(\lambda z.y)x$ and verify **iszero**$(0) = true$ and **iszero**$(n+1) = false$.

Fixed points and recursive functions

Thm. In the untyped lambda calculus, every term F has a fixed point.

Proof. Let $\Theta = AA$ where $A = \lambda xy.y(xxy)$.

$$\begin{array}{rcl} \Theta F & = & AAF \\ & = & (\lambda xy.y(xxy))AF \\ & \twoheadrightarrow_\beta & F(AAF) \\ & = & F(\Theta F) \end{array}$$

Thus ΘF is a fixed point of F.

The term Θ is called Turing's fixed point combinator.

The factorial function

$$\mathbf{fact}\ n \;=\; \mathbf{if\text{-}then\text{-}else}\ (\mathbf{iszero}\ n)(\bar{1})(\mathbf{mult}\ n(\mathbf{fact}\ (\mathbf{pred}\ n)))$$

$$\mathbf{fact} \;=\; \lambda n.\mathbf{if\text{-}then\text{-}else}\ (\mathbf{iszero}\ n)(\bar{1})(\mathbf{mult}\ n(\mathbf{fact}\ (\mathbf{pred}\ n)))$$

$$\mathbf{fact} \;=\; (\lambda f.\lambda n.\mathbf{if\text{-}then\text{-}else}\ (\mathbf{iszero}\ n)(\bar{1})(\mathbf{mult}\ n(f(\mathbf{pred}\ n)))\mathbf{fact}$$

$$\mathbf{fact} \;=\; \Theta(\lambda f.\lambda n.\mathbf{if\text{-}then\text{-}else}\ (\mathbf{iszero}\ n)(\bar{1})(\mathbf{mult}\ n(f(\mathbf{pred}\ n)))$$

Other data types: pairs

Define $\langle M, N\rangle = \lambda z.zMN$. Let $\pi_1 = \lambda p.p(\lambda xy.x)$ and $\pi_2 = \lambda p.p(\lambda xy.y)$. Observe that

$$\begin{aligned} \pi_1\langle M, N\rangle &\twoheadrightarrow_\beta M \\ \pi_2\langle M, N\rangle &\twoheadrightarrow_\beta N \end{aligned}$$

Tuples

Define $\langle M_1, ..., M_n \rangle = \lambda z.zM_1...M_n$ and the ith projection $\pi_1^n = \lambda p.p(\lambda x_1...x_n.x_i)$. Then

$$\pi_i^n \langle M_1, ..., M_n \rangle \twoheadrightarrow_\beta M_i$$

for all $1 \leq i \leq n$.

Lists

Define $\mathbf{nil} = \lambda xy.y$ and $H :: T = \lambda xy.xHT$. Then the function of adding a list of numbers can be:

$$\mathbf{addlist}\ l = l(\lambda ht.\mathbf{add}\ h(\mathbf{addlist}\ t))(\bar{0})$$

Trees

A binary tree can be either a leaf, labeled by a natural number, or a node with two subtrees. Write $\mathbf{leaf}(n)$ for a leaf labeled n, and $\mathbf{node}(L, R)$ for a node with left subtree L and right subtree R.

$$\begin{array}{rcl} \mathbf{leaf}(n) & = & \lambda xy.xn \\ \mathbf{node}(L, R) & = & \lambda xy.yLR \end{array}$$

A program that adds all the numbers at the leaves of a tree:

$$\mathbf{addtree}\ t = t(\lambda n.n)(\lambda lr.\mathbf{add}\ (\mathbf{addtree}\ l)(\mathbf{addtree}\ r))$$

η-reduction

$$\lambda x.Mx \longrightarrow_\eta M, \text{ where } x \notin FV(M).$$

Define the single-step $\beta\eta$-reduction $\longrightarrow_{\beta\eta}=\longrightarrow_\beta \cup \longrightarrow_\eta$ and the multi-step $\beta\eta$-reduction $\twoheadrightarrow_{\beta\eta}$.

Church-Rosser Theorem

Thm. (Church and Rosser, 1936). Let $\twoheadrightarrow$ denote either $\twoheadrightarrow_\beta$ or $\twoheadrightarrow_{\beta\eta}$. Suppose M, N and P are lambda terms such that $M \twoheadrightarrow N$ and $M \twoheadrightarrow P$. Then there exists a lambda term Z such that $N \twoheadrightarrow Z$ and $P \twoheadrightarrow Z$.

This is the Church-Rosser property or confluence.

See Section 4.4 of the λ-calculus lecture notes for the detailed proof.

Some consequences of confluence

Cor. If $M =_\beta N$ then there exists some Z with $M, N \twoheadrightarrow_\beta Z$. Similarly for $\beta\eta$.

Cor. If N is a β-normal form and $M =_\beta N$, then $M \twoheadrightarrow_\beta N$, and similarly for $\beta\eta$.

Cor. If M and N are β-normal forms such that $M =_\beta N$, then $M =_\alpha N$, and similarly for $\beta\eta$.

Cor. If $M =_\beta N$, then neither or both have a β-normal form, and similarly for $\beta\eta$.

Simply-typed lambda calculus

Simple types: assume a set of basic types, ranged over by ι. The set of simple types is given by

$$A, B ::= \iota \mid A \longrightarrow B \mid A \times B \mid 1$$

- $A \longrightarrow B$ is the type of functions from A to B.
- $A \times B$ is the type of pairs $\langle x, y \rangle$
- 1 is a one-element type, considered as “void” or “unit” type in many languages: the result type of a function with no real result.

Convention: $\times$ binds stronger than $\longrightarrow$ and $\longrightarrow$ associates to the right. E.g. $A \times B \longrightarrow C$ is $(A \times B) \longrightarrow C$, and $A \longrightarrow B \longrightarrow C$ is $A \longrightarrow (B \longrightarrow C)$.

Raw typed lambda terms

$$M, N ::= x \mid MN \mid \lambda x^A.M \mid \langle M, N \rangle \mid \pi_1 M \mid \pi_2 M \mid *$$

Typing judgment

Write $M : A$ to mean "M is of type A". A typing judgment is an expression of the form

$$x_1 : A_1, x_2 : A_2, ..., x_n : A_n \vdash M : A$$

The meaning is: under the assumption that x_i is of type A_i, for $i = 1...n$, the term M is a well-typed term of type A. The free variables of M must be contained in $x_1, ..., x_n$

The sequence of assumptions $x_1 : A_1, x_2 : A_2, ..., x_n : A_n$ is a typing context, written as Γ. The notations Γ, Γ' and $\Gamma, x : A$ denote the concatenation of typing contexts, assuming the sets of variables are disjoint.

Typing rules

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma \vdash M : A \longrightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A}$$

$$\frac{\Gamma, x : A \vdash M : B}{\lambda x^A.M : A \longrightarrow B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \qquad \frac{}{\Gamma \vdash * : 1}$$

Typing derivation

$$
\dfrac{\dfrac{}{x : A \to A, y : A \vdash x : A \to A} \qquad \dfrac{\dfrac{}{x : A \to A, y : A \vdash x : A \to A} \quad \dfrac{}{x : A \to A, y : A \vdash y : A}}{x : A \to A, y : A \vdash xy : A}}{\dfrac{x : A \to A, y : A \vdash x(xy) : A}{\dfrac{x : A \to A \vdash \lambda y^{A}.x(xy) : A \to A}{\vdash \lambda x^{A \to A}.\lambda y^{A}.x(xy) : (A \to A) \to A \to A}}}
$$

Reductions in the simply-typed lambda calculus

β- and η-reductions:

$$\begin{array}{rcl} (\lambda x^A.M)N & \longrightarrow_\beta & M[N/x] \\ \pi_1\langle M, N\rangle & \longrightarrow_\beta & M \\ \pi_2\langle M, N\rangle & \longrightarrow_\beta & N \end{array}$$

$$\begin{array}{rcl} \lambda x^A.Mx & \longrightarrow_\eta & M \\ \langle \pi_1 M, \pi_2 M\rangle & \longrightarrow_\eta & M \\ M & \longrightarrow_\eta & *, \quad \text{if } M : 1 \end{array}$$

Subject reduction

Thm. If $\Gamma \vdash M : A$ and $M \longrightarrow_{\beta\eta} M'$, then $\Gamma \vdash M' : A$.

Proof: By induction on the derivation of $M \longrightarrow_{\beta\eta} M'$, and by case distinction on the last rule used in the derivation of $\Gamma \vdash M : A$. $\square$

Church-Rosser

The Church-Rosser theorem does not hold for $\beta\eta$-reduction in the simply-typed $\lambda^{\to,\times,1}$-calculus.

E.g. if x has type $A \times 1$, then

$$\langle \pi_1 x, \pi_2 x \rangle \longrightarrow_\eta x$$

$$\langle \pi_1 x, \pi_2 x \rangle \longrightarrow_\eta \langle \pi_1 x, * \rangle$$

Both x and $\langle \pi_1 x, * \rangle$ are normal forms.

If we omit all the η-reductions and consider only β-reductions, then the Church-Rosser property does hold.

Sum types

Simple types:

$$A, B ::= ... \mid A + B \mid 0$$

Sum type is also known as "union" or "variant" type. The type 0 is the empty type, corresponding to the empty set in set theory.

Raw terms:

$$\begin{array}{lcl} M, N, P & ::= & ... \mid in_1 M \mid in_2 M \\ & & \mid case\ M\ of\ x^A \Rightarrow N \mid y^B \Rightarrow P \\ & & \mid\ \Box_A M \end{array}$$

Typing rules for sums

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash in_1 M : A + B}$$

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash in_2 M : A + B}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N : C \quad \Gamma, y : B \vdash P : C}{\Gamma \vdash (case\ M\ of\ x^A \Rightarrow N \,|y^B \Rightarrow P) : C}$$

$$\frac{\Gamma \vdash M : 0}{\Gamma \vdash \Box_A M : A}$$

The booleans can be defined as $1+1$ with $\mathbf{T} = in_1 *,\ \mathbf{F} = in_2 *$, and **if-then-else** $MNP = case\ M\ of\ x^1 \Rightarrow N \,|\, y^1 \Rightarrow P$, where x and y don't occur in N and P. The term $\Box_A M$ is a simple type cast.

Weak and strong normalization

Def. A term M is weakly normalizing if there exists a finite sequence of reductions $M \to M_1 \to ... \to M_n$ such that M_n is a normal form. It is strongly normalizing if there does not exist an infinite sequence of reductions starting from M, i.e., if every sequence of reductions starting from M is finite.

- $\Omega = (\lambda x.xx)(\lambda x.xx)$ is neither weakly nor strongly normalizing.
- $(\lambda x.y)\Omega$ is weakly normalizing, but not strongly normalizing.
- $(\lambda x.y)((\lambda x.x)(\lambda x.x))$ is strongly normalizing.
- Every normal form is strongly normalizing.

Strong normalization

Thm. In the simply-typed lambda calculus, all terms are strongly normalizing.

A proof is given in the following book: J.-Y.Girard, Y.Lafont, and P.Taylor. Proofs and Types. Cambridge University Press, 1989.

Chapter 5. The denotational semantics of IMP

5.1 Motivation

- Operational semantics is too concrete, built out of syntax, is hard to compare two programs written in different programming languages.
- E.g. $c_0 \sim c_1$ iff $(\forall \sigma, \sigma'.\ \langle c_0, \sigma\rangle \to \sigma') \Leftrightarrow \langle c_1, \sigma\rangle \to \sigma'$ iff $\{(\sigma, \sigma') \mid \langle c_0, \sigma\rangle \to \sigma'\} = \{(\sigma, \sigma') \mid \langle c_1, \sigma\rangle \to \sigma'\}$, i.e. c_0 and c_1 determine the same partial function on states.
- So we take the denotation of a command to be a partial function on states.

5.2 Denotations of Aexp

Define the semantic function $\mathcal{A} : \mathbf{Aexp} \to (\Sigma \to \mathbf{N})$

$$
\begin{array}{rcl}
\mathcal{A}[\![n]\!] & = & \{(\sigma, n) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![X]\!] & = & \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![a_0 + a_1]\!] & = & \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \;\;\&\;\; (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\} \\
\mathcal{A}[\![a_0 - a_1]\!] & = & \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \;\;\&\;\; (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\} \\
\mathcal{A}[\![a_0 \times a_1]\!] & = & \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \;\;\&\;\; (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}
\end{array}
$$

The "+" on the left-hand side represents syntactic sign in **IMP** whereas the sign on the right represents sum on numbers. Similarly for "-", "$\times$".

5.2 Denotations of Aexp

The denotation of arithmetic expressions are actually total functions. Using λ-notation,

$$
\begin{array}{rcl}
\mathcal{A}[\![n]\!] & = & \lambda\sigma \in \Sigma.\ n \\
\mathcal{A}[\![X]\!] & = & \lambda\sigma \in \Sigma.\ \sigma(X) \\
\mathcal{A}[\![a_0 + a_1]\!] & = & \lambda\sigma \in \Sigma.\ (\mathcal{A}[\![a_0]\!]\sigma + \mathcal{A}[\![a_1]\!]\sigma) \\
\mathcal{A}[\![a_0 - a_1]\!] & = & \lambda\sigma \in \Sigma.\ (\mathcal{A}[\![a_0]\!]\sigma - \mathcal{A}[\![a_1]\!]\sigma) \\
\mathcal{A}[\![a_0 \times a_1]\!] & = & \lambda\sigma \in \Sigma.\ (\mathcal{A}[\![a_0]\!]\sigma \times \mathcal{A}[\![a_1]\!]\sigma)
\end{array}
$$

5.2 Denotations of Bexp

Define the semantic function $\mathcal{B} : \mathbf{Bexp} \to (\Sigma \to \mathbf{T})$

$$
\begin{array}{rcl}
\mathcal{B}[\![\mathbf{true}]\!] & = & \{(\sigma, \mathbf{true}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![\mathbf{false}]\!] & = & \{(\sigma, \mathbf{false}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![a_0 = a_1]\!] & = & \{(\sigma, \mathbf{true}) \mid \sigma \in \Sigma \ \ \& \ \ \mathcal{A}[\![a_0]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma\} \cup \\
 & & \{(\sigma, \mathbf{false}) \mid \sigma \in \Sigma \ \ \& \ \ \mathcal{A}[\![a_0]\!]\sigma \neq \mathcal{A}[\![a_1]\!]\sigma\} \cup \\
\mathcal{B}[\![\neg b]\!] & = & \{(\sigma, \neg_T t) \mid \sigma \in \Sigma \ \ \& \ \ (\sigma, t) \in \mathcal{B}[\![b]\!]\} \\
\mathcal{B}[\![b_0 \wedge b_1]\!] & = & \{(\sigma, t_0 \wedge_T t_1) \mid \sigma \in \Sigma \ \ \& \ \ (\sigma, t_0) \in \mathcal{B}[\![b_0]\!] \ \ \& \ \ (\sigma, t_1) \in \mathcal{B}[\![b_1]\!]\} \\
 & \ldots &
\end{array}
$$

The sign "$\wedge_T$" is the conjunction operation on truth values.

5.2 Denotations of Com

Define the compositional semantic function $\mathcal{C} : \mathbf{Aexp} \to (\Sigma \to \Sigma)$

$$
\begin{array}{rcl}
\mathcal{C}[\![\mathbf{skip}]\!] & = & \{(\sigma,\sigma) \mid \sigma \in \Sigma\} \\
\mathcal{C}[\![X := a]\!] & = & \{(\sigma,\sigma[n/X]) \mid \sigma \in \Sigma \ \ \& \ \ n = \mathcal{A}[\![a]\!]\sigma\} \\
\mathcal{C}[\![c_0; c_1]\!] & = & \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!] \\
\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!] & = & \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \ \ \& \ \ (\sigma,\sigma') \in \mathcal{C}[\![c_0]\!]\} \cup \\
 & & \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false} \ \ \& \ \ (\sigma,\sigma') \in \mathcal{C}[\![c_1]\!]\} \\
\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!] & = & fix(\Gamma)
\end{array}
$$

where

$$
\begin{array}{rl}
\Gamma(\varphi) = & \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \ \ \& \ \ (\sigma,\sigma') \in \varphi \circ \mathcal{C}[\![c]\!]\} \cup \\
 & \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\}
\end{array}
$$

5.2 Denotation of while -loops

Let $w \equiv \textbf{while } b \textbf{ do } c$. Inspired by the equivalence $w \sim \textbf{if } b \textbf{ then } c; w \textbf{ else skip}$. We should have

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

We want a fixed point of Γ to be the denotation of w. But Γ is the operator $\hat{R}$ on sets where R is

$$\begin{aligned}R = \quad & \{\frac{(\sigma'',\sigma')}{(\sigma,\sigma')} \mid \mathcal{B}[\![b]\!]\sigma = \textbf{true} \;\;\&\;\; (\sigma,\sigma'') \in \mathcal{C}[\![c]\!]\} \;\cup \\ & \{\frac{}{(\sigma,\sigma)} \mid \mathcal{B}[\![b]\!]\sigma = \textbf{false}\}.\end{aligned}$$

5.3 Equivalence of the semantics

Lemma 0.15 For all $a \in \mathbf{Aexp}$, $\mathcal{A}[\![a]\!] = \{(\sigma, n) \mid \langle a, \sigma \rangle \to n\}$.

Proof: Define the property P by $P(a) =_{def} \mathcal{A}[\![a]\!] = \{(\sigma, n) \mid \langle a, \sigma \rangle \to n\}$ and proceed by structural induction on arithmetic expressions. cf. Page 61. $\square$

Lemma 0.16 For all $b \in \mathbf{Bexp}$, $\mathcal{B}[\![b]\!] = \{(\sigma, t) \mid \langle b, \sigma \rangle \to t\}$.

Proof: Similar to the proof of Lemma 0.15. cf. Page 62. $\square$

5.3 Equivalence of the semantics

Lemma 0.17 For all commands c and states σ, σ',
$\langle c, \sigma \rangle \to \sigma' \Rightarrow (\sigma, \sigma') \in \mathcal{C}[\![c]\!]$.

Proof: Let $P(c, \sigma, \sigma') =_{def} (\sigma, \sigma') \in \mathcal{C}[\![c]\!]$. Use rule induction for commands given in Section 4.3.3. cf. Page 64. $\square$

5.3 Equivalence of the semantics

Theorem 0.18 For all commands c, $\mathcal{C}[\![c]\!] = \{(\sigma, \sigma') \mid \langle c, \sigma \rangle \to \sigma'\}$.

Proof: Restate the theorem as: for all commands c,
$(\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Leftrightarrow \langle c, \sigma \rangle \to \sigma'$.

($\Leftarrow$): Shown in Lemma 0.17.

($\Rightarrow$): By structural induction on commands c. In the case $c \equiv \textbf{while } b \textbf{ do } c_0$, show by mathematical induction on n that $\forall \sigma, \sigma' \in \Sigma.\ (\sigma, \sigma') \in \Gamma^n(\emptyset) \Rightarrow \langle c, \sigma \rangle \to \sigma'$. The base base $\Gamma^0(\emptyset) = \emptyset$ is trivial. For the induction step, assume $(\sigma, \sigma') \in \Gamma^{n+1}(\emptyset)$. Then (i) either $\mathcal{B}[\![b]\!]\sigma = \textbf{true}$ and $(\sigma, \sigma'') \in \mathcal{C}[\![c_0]\!]$, $(\sigma'', \sigma') \in \Gamma^n(\emptyset)$ for some σ'', (ii) or $\mathcal{B}[\![b]\!]\sigma = \textbf{false}$ and $\sigma' = \sigma$. For (i), $\langle b, \sigma \rangle \to \textbf{true}$ by Lemma 0.16, $\langle c_0, \sigma \rangle \to \sigma''$ by structural induction hypothesis, and $\langle c, \sigma'' \rangle \to \sigma'$ by mathematical induction hypothesis. So $\langle c, \sigma \rangle \to \sigma'$. For (ii), $\langle b, \sigma \rangle \to \textbf{false}$ by Lemma 0.16, so $\langle c, \sigma \rangle \to \sigma$.

$\square$

5.4 Complete partial orders

A partial order (p.o.) is a set with a binary relation $\sqsubseteq$ which is reflexive, antisymmetric, transitive.

For a partial order $(P, \sqsubseteq)$ and subset $X \subseteq P$, say p is an upper bound of X iff $\forall q \in X.\ q \sqsubseteq p$. Say p is a least upper bound (lub) of X iff p is an upper bound and for all upper bounds q of X, $p \sqsubseteq q$. Write $\bigsqcup X$ as the lub of X.

An ω-chain of the partial order is an increasing chain $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$. The partial order is a complete partial order (cpo) if it has lubs for all ω-chains. $(P, \sqsubseteq)$ is a cpo with bottom if it's a cpo with a least element $\perp$.

5.4 Complete partial orders: examples

- Any set ordered by the identity relation forms a discrete or flat cpo without bottom.
- A powerset $\mathcal{P}ow(X)$ of any set X, ordered by $\subseteq$ or $\supseteq$ forms a cpo as indeed does any complete lattice.
- The two element cpo $\bot \sqsubseteq \top$ is called **O**. Such an order arises as the powerset of a singleton ordered by $\subseteq$.
- The set of partial functions $X \rightharpoonup Y$ ordered by inclusion, between sets X, Y, is a cpo.
- Extending the set of natural numbers ω by ∞ and then in a chain

$$0 \sqsubseteq 1 \sqsubseteq \cdots \sqsubseteq n \sqsubseteq \cdots \infty$$

  yields a cpo, called Ω.

5.4 An alternative definition of CPO

If $(P, \sqsubseteq)$ is a partial order, then a subset $X \subseteq P$ is directed if every finite $X_0 \subseteq X$ has an upper bound in X.

- Every directed set is nonempty, since the empty subset of a directed set X must have an upper bound in X.
- If $X \subseteq P$ is linearly ordered, i.e. $x \sqsubseteq y$ or $y \sqsubseteq x$ for all $x, y \in X$, then X is directed.
- Consider the partial order $(P, \sqsubseteq)$ with $P = \{a_0, b_0, a_1, b_1, ...\}$, $a_i \sqsubseteq a_j, b_j$ and $b_i \sqsubseteq a_j, b_j$ for all $i < j$. A directed set is P.

A cpo is a partial order $(P, \sqsubseteq)$ s.t. every directed subset of P has a least upper bound.

The two definitions of cpo are equivalent. A general proof involves the axiom of choice, but for countable cpo's the proof is much simpler.

5.4 Continuous functions

A function $f : D \to E$ between cpos D, E is monotonic iff
$\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$.

It's continuous iff for all ω-chains
$f(\bigsqcup_{n\in\omega} d_n) = \bigsqcup_{n\in\omega} f(d_n)$.

Proposition 0.19 The identity function Id_D on a cpo D is continuous.
Let $f : D \to E$ and $g : E \to F$ be continuous functions on cpo's D, E, F.
Then their composition $g \circ f : D \to F$ is continuous.

5.4 Continuous functions: examples

The parallel-or function $por : \mathbf{T}_\perp \times \mathbf{T}_\perp \to \mathbf{T}_\perp$ given by

$$por(x,y) = \begin{cases} \mathbf{true} & \text{if } x = \mathbf{true} \text{ or } y = \mathbf{true} \\ \mathbf{false} & \text{if } x = y = \mathbf{false} \\ \perp & \text{otherwise} \end{cases}$$

is continuous.

5.4 Continuous functions: examples

A solution to the "halting problem" would be a definable function $total? : (\mathbb{N}_\perp \to \mathbb{N}_\perp) \to \mathbf{T}_\perp$ with the property that for every $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$,

$$total?(f) = \begin{cases} \mathbf{true} & \text{if } \forall n \neq \perp_{\mathbb{N}}.\ f(n) \neq \perp_{\mathbb{N}} \\ \mathbf{false} & \text{otherwise} \end{cases}$$

There is no (PCF) expression defining $total?$ because this function is not continuous. In fact it is not even monotonic.

5.4 Fixed point theorem

Let $f : D \to D$ be a function. A fixed point of f is an element d with $f(d) = d$. A prefixed point of f is an element d with $f(d) \sqsubseteq d$.

Proposition 0.20 Let $f : D \to D$ be a continuous function on a cpo with bottom D. Define $fix(f) = \bigsqcup_{n\in\omega} f^n(\bot)$. Then $fix(f)$ is the fixed point of f and the least prefixed point f.

Proof:

- $f(\bigsqcup_{n\in\omega} f^n(\bot)) = \bigsqcup_{n\in\omega} f^{n+1}(\bot) = (\bigsqcup_{n\in\omega} f^{n+1}(\bot)) \sqcup \{\bot\}$
- If d is a prefixed point. By induction on n we have $f^n(\bot) \sqsubseteq d$. So $\bigsqcup_{n\in\omega} f^n(\bot) \sqsubseteq d$.

$\square$

5.5 The Knaster-Tarski theorem for minimum fixed points

Let $(P, \sqsubseteq)$ be a partial order and $X \subseteq P$. Similar to lub, we can define a greatest lower bound (glb) of X. A complete lattice is a partial order which has glbs of arbitrary subsets.

Proposition 0.21 Let $(L, \sqsubseteq)$ be a complete lattice and $f : L \to L$ a monotonic function. Define $m = \bigsqcap\{x \in L \mid f(x) \sqsubseteq x\}$. Then m is a fixed point of f and the least prefixed point f.

Proof: Let $X = \{x \in L \mid f(x) \sqsubseteq x\}$. For any $x \in X$, we have $m \sqsubseteq x$, thus $f(m) \sqsubseteq f(x)$ by monotonicity of f. But $f(x) \sqsubseteq x$ as $x \in X$. So $f(m) \sqsubseteq x$ for any $x \in X$. Thus $f(m) \sqsubseteq \bigsqcap X = m$, i.e. m is the least prefixed point.

By $f(m) \sqsubseteq m$ and monotonicity, $f(f(m)) \sqsubseteq f(m)$. So $f(m) \in X$ which entails $m \sqsubseteq f(m)$. Thus $f(m) = m$. □

5.5 The Knaster-Tarski theorem for maximum fixed points

Proposition 0.22 Let $(L, \sqsubseteq)$ be a complete lattice and $f : L \to L$ a monotonic function. Define $m = \bigsqcup\{x \in L \mid x \sqsubseteq f(x)\}$. Then m is a fixed point of f and the greatest postfixed point f (i.e. $x \sqsubseteq f(x)$).

Proof: A monotonic function on $(L, \sqsubseteq)$ is also monotonic on the complete lattice $(L, \sqsupseteq)$. Then the result follows from the minimum-fixed-point theorem. $\square$

Chapter 6. The axiomatic semantics of IMP

6.1 The idea

Assertions in programs.

$$S := 0; N := 1$$

$$\{S = 0 \ \& \ N = 1\}$$

$$\textbf{while } \neg(N = 101) \textbf{ do } S := S + N; N := N + 1$$

$$\{S = \sum_{1 \leq m \leq 100} m\}$$

6.1 Partial correctness

Let A, B be assertions like those in **Bexp**, and c a command. We write $\{A\}c\{B\}$ to mean: for all states σ which satisfy A (precondition) if the execution c from state σ terminates in state σ' then σ' satisfies B (postcondition).

NB: $\{\mathbf{true}\}\mathbf{while\ true\ do\ skip}\{\mathbf{false}\}$

In contrast to total correctness assertions $[A]c[B]$ — the execution of c from any state which satisfies A will terminate in a state which satisfies B.

6.1 Partial correctness

Consider $\mathcal{C}[\![c]\!]$ as a total function in $(\Sigma \rightarrow \Sigma_\perp)$ instead of partial function in $(\Sigma \rightharpoonup \Sigma)$.

Write $\sigma \models A$ to mean the state σ satisfies assertion A. Let $\perp \models A$ for any A. Then the meaning of $\{A\}c\{B\}$ will be

$$\forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow \mathcal{C}[\![c]\!]\sigma \models B.$$

6.2 The assertion language Assn

Let i range over integer variables, **Intvar**. Extending **Aexp** with integer variables to be **Aexpv**:

$$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

Extending **Bexp** to be **Assn**:

$$A ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \forall i.A \mid$$

6.2 Free integer variables

Define free integer variables in **Aexpv** or **Assn** expressions by structural induction.

$$FV(n) = FV(X) = \emptyset$$

$$FV(i) = \{i\}$$

$$FV(a_0 + a_1) = FV(a_0 - a_1) = FV(a_0 \times a_1) = FV(a_0) \cup FV(a_1)$$

$$FV(\mathbf{true}) = FV(\mathbf{false}) = \emptyset$$

$$FV(a_0 = a_1) = FV(a_0 \leq a_1) = FV(a_0) \cup FV(a_1)$$

$$FV(A_0 \wedge A_1) = FV(A_0 \vee A_1) = FV(A_0 \Rightarrow A_1) = FV(A_0) \cup FV(A_1)$$

$$FV(\neg A) = FV(A)$$

$$FV(\forall i.A) = FV(\exists i.A) = FV(A) \backslash \{i\}$$

6.2 Substitution

Define substitution for **Aexpv** or **Assn** expressions by structural induction.

$$n[a/i] \equiv n \qquad X[a/i] \equiv X$$

$$j[a/i] \equiv j \qquad i[a/i] \equiv a$$

$$(a_0 + a_1)[a/i] \equiv (a_0[a/i] + a_1[a/i])$$

$$\ldots$$

$$\mathbf{true}[a/i] \equiv \mathbf{true} \qquad \mathbf{false}[a/i] \equiv \mathbf{false}$$

$$(a_0 = a_1)[a/i] \equiv (a_0[a/i] = a_1[a/i])$$

$$(A_0 \wedge A_1)[a/i] \equiv (A_0[a/i] \wedge A_1[a/i])$$

$$(\neg A)[a/i] \equiv \neg(A[a/i])$$

$$(\forall j.A)[a/i] \equiv \forall j.(A[a/i]) \qquad (\forall i.A)[a/i] \equiv \forall i.A$$

$$(\exists j.A)[a/i] \equiv \exists j.(A[a/i]) \qquad (\exists i.A)[a/i] \equiv \exists i.A$$

6.3 The meaning of expressions, Aexpv

An interpretation is a function $I : \mathbf{Intvar} \to \mathbf{N}$ assigning an integer to each integer variable. The value of an expression $a \in \mathbf{Aexpv}$ in an interpretation I and state σ is written $\mathcal{A}v[\![a]\!]I\sigma$ or $(\mathcal{A}v[\![a]\!](I))(\sigma)$.

$$
\begin{aligned}
&\mathcal{A}v[\![n]\!]I\sigma = n\\
&\mathcal{A}v[\![X]\!]I\sigma = \sigma(X)\\
&\mathcal{A}v[\![i]\!]I\sigma = I(i)\\
&\mathcal{A}v[\![a_0 + a_1]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma + \mathcal{A}v[\![a_1]\!]I\sigma\\
&\mathcal{A}v[\![a_0 - a_1]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma - \mathcal{A}v[\![a_1]\!]I\sigma\\
&\mathcal{A}v[\![a_0 \times a_1]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma \times \mathcal{A}v[\![a_1]\!]I\sigma
\end{aligned}
$$

6.3 The meaning of assertions, Assn

Write $I[n/i]$ for the interpretation given by $I[n/i](j) = n$ if $j \equiv i$, and $I(j)$ otherwise.

For $A \in \mathbf{Assn}$, write $\sigma \models^I A$ to mean σ satisfies A in interpretation I.

$$\sigma \models^I \mathbf{true}$$

$$\sigma \models^I (a_0 = a_1) \text{ if } \mathcal{A}v[\![a_0]\!]I\sigma = \mathcal{A}v[\![a_1]\!]I\sigma$$

$$\sigma \models^I A \wedge B \text{ if } \sigma \models^I A \text{ and } \sigma \models^I B$$

$$\sigma \models^I A \Rightarrow B \text{ if } \sigma \not\models^I A \text{ or } \sigma \models^I B$$

$$\sigma \models^I \forall i.A \text{ if } \sigma \models^{I[n/i]} A \text{ for all } n \in \mathbf{N}$$

$$\sigma \models^I \exists i.A \text{ if } \sigma \models^{I[n/i]} A \text{ for some } n \in \mathbf{N}$$

$$\perp \models^I A$$

$$\ldots$$

6.3 Partial correctness assertions

Write $A^I = \{\sigma \in \Sigma_\perp \mid \sigma \models^I A\}$.

- $\sigma \models^I \{A\}c\{B\}$ iff $(\sigma \models^I A \Rightarrow \mathcal{C}[\![c]\!]\sigma \models^I B)$.
- $\models^I \{A\}c\{B\}$ iff $\forall \sigma \in \Sigma_\perp.\ \sigma \models^I \{A\}c\{B\}$
- Validity: $\models \{A\}c\{B\}$ iff
  $\sigma \models^I \{A\}c\{B\}$ for all interpretations I and states σ
- Similarly, A is valid, $\models A$, means $\sigma \models^I A$ for all interpretations I and states σ.

6.4 Proof rules for partial correctness

The proof rules are called Hoare rules and the proof system Hoare logic.

$$\{A\}\ \mathbf{skip}\ \{A\}$$

$$\{B[a/X]\}\ X := a\ \{B\}$$

$$\frac{\{A\}c_0\{C\} \quad \{C\}c_1\{B\}}{\{A\}\ c_0; c_1\ \{B\}}$$

$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\}\ \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\ \{B\}}$$

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{A \wedge \neg b\}}$$

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}\ c\ \{B\}}$$

6.5 Soundness of the proof system

A rule is sound in the sense that if the rule's premise is valid then so is its conclusion. The proof system is sound if every rule is sound. Then by rule induction, every theorem obtained from the proof system is a valid partial correctness assertion.

Lemma 0.23 Let I be an interpretation, σ a state, and $X \in \mathbf{Loc}$.

- Let $a, a_0 \in \mathbf{Aexpv}$. Then $\mathcal{A}v[\![a_0[a/X]]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma[\mathcal{A}v[\![a]\!]I\sigma/X]$
- Let $B \in \mathbf{Assn}$. Then $\sigma \models^I B[a/X]$ iff $\sigma[\mathcal{A}[\![a]\!]\sigma/X] \models^I B$

Proof: By structural induction on a_0 and B respectively. $\square$

6.5 Soundness of the proof system

Theorem 0.24 Let $\{A\}c\{B\}$ be a partial correctness assertion. If $\vdash \{A\}c\{B\}$ then $\models \{A\}c\{B\}$.

Proof: Show that each proof rule is sound. Consider the rule for while-loops. Let $w \equiv \textbf{while } b \textbf{ do } c$. Then $\mathcal{C}[\![w]\!] = \bigcup_{n\in\omega} \theta_n$ where

$$\theta_0 = \emptyset$$

$$\theta_{n+1} = \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textbf{true} \ \& \ (\sigma, \sigma') \in \theta_n \circ \mathcal{C}[\![c]\!]\} \cup \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textbf{false}\}$$

and $P(n) =_{def} \forall \sigma, \sigma' \in \Sigma.(\sigma, \sigma') \in \theta_n \ \& \ (\sigma \models^I A \Rightarrow \sigma' \models^I A \wedge \neg b)$. Show by induction that $P(n)$ holds for all $n \in \omega$. cf. Page 92. $\square$

6.6 Using the Hoare rules

Let $w \equiv (\textbf{while } X > 0 \textbf{ do } Y := X \times Y; X := X - 1)$, and show

$$\{X = n \ \& \ n \geq 0 \ \& \ Y = 1\}w\{Y = n!\}$$

Take $I \equiv (Y \times X! = n! \ \& \ X \geq 0)$, then

$$\{I \wedge X > 0\}Y := X \times Y; X := X - 1\{I\}$$

and so $\{I\}w\{I \wedge X \not> 0\}$.

Note $X = n \ \& \ n \geq 0 \ \& \ Y = 1 \Rightarrow I$ and $I \wedge X \not> 0 \Rightarrow Y = n!$

Chapter 7. Completeness of the Hoare rules

7.1 Gödel's incompleteness theorems

- The first incompleteness theorem states that no consistent system of axioms whose theorems can be listed by an "effective procedure" (essentially, a computer program) is capable of proving all facts about the natural numbers. For any such system, there will always be statements about the natural numbers that are true, but that are unprovable within the system.
- The second incompleteness theorem shows that if such a system is also capable of proving certain basic facts about the natural numbers, then one particular arithmetic truth the system cannot prove is the consistency of the system itself.

7.1 No proof system for Assn

Theorem 0.25 There is no effective proof system for **Assn** such that the theorems coincide with the valid assertions of **Assn**.

It follows that there is no effective proof system for partial correctness assertions. As $\models B$ iff $\models \{\mathbf{true}\}\mathbf{skip}\{B\}$, if we had an effective proof system for partial correctness it would reduce to an effective proof system for assertions in **Assn**, which is impossible by Theorem 0.25.

7.1 No proof system for partial correctness assertions

Proposition 0.26 There is no effective proof system for partial correctness assertions such that its theorems are precisely the valid partial correctness assertions.

Proof: An alternative and direct proof: Observe that $\models \{\mathbf{true}\}c\{\mathbf{false}\}$ iff the command c diverges on all states. If we had an effective proof system for partial correctness assertions it would yield a computational method of confirming that a command c diverges on all states. But this is known to be impossible. $\square$

Still we seek relative completeness of the Hoare rules for partial correctness — their completeness is relative to being able to draw from the set of valid assertions about arithmetic.

7.2 Weakest preconditions

Motivation: consider to prove $\{A\}c_0; c_1\{B\}$. In order to use the rule for composition one requires an assertion C so that $\{A\}c_0\{C\}$ and $\{C\}c_1\{B\}$ are provable. Why assertion C can be found?

Let $c \in \mathbf{Com}, B \in \mathbf{Assn}$ and I an interpretation. The weakest precondition $wp^I[\![c, B]\!]$ of B wrt c in I is
$wp^I[\![c, B]\!] = \{\sigma \in \Sigma_\perp \mid \mathcal{C}[\![c]\!]\sigma \models^I B\}$.

It's all those states from which the execution of c either diverges or ends up in a final state satisfying B.

7.2 Weakest preconditions and expressiveness

$\models^I \{A\}c\{B\}$ iff $A^I \subseteq wp^I[\![c, B]\!]$.

If there is an assertion A_0 s.t. in all interpretation I, $A_0^I = wp^I[\![c, B]\!]$, then $\models^I \{A\}c\{B\}$ iff $\models^I (A \Rightarrow A_0)$ for any interpretation I, i.e.
$\models \{A\}c\{B\}$ iff $\models (A \Rightarrow A_0)$.

So the weakest precondition is implied by any precondition that makes the partial correctness assertion valid.

Say **Assn** is expressive iff for every command c and assertion B there is an assertion A_0 s.t. $A_0^I = wp^I[\![c, B]\!]$ for any interpretation I.

7.2 Weakest preconditions and expressiveness

In showing expressiveness we use Gödel's β predicate, which involves the operation **mod** . For $x = a \ \mathbf{mod}\ b$ we write

$$a \geq 0 \wedge b \geq 0 \wedge$$
$$\exists k.((k \geq 0 \wedge k \times b \leq a) \wedge (k+1) \times b > a \wedge x = a - (k \times b)).$$

7.2 Chinese Remainder Theorem

Theorem 0.27 Suppose $m_1, ..., m_n$ are relatively prime. Then for any $a_1, ..., a_n$ there is an x such that $x = a_i \textbf{ mod } m_i$ for $i = 1, ..., n$.

Proof: Let

$$M_i = \Pi_{j \neq i} m_j.$$

Since M_i and m_i are relatively prime, we can find b_i such that $b_i M_i = 1 \textbf{ mod } m_i$. Let

$$x = \sum_{i=1}^{n} a_i b_i M_i.$$

Since $m_i | M_j$ for $j \neq i$, we get $x = a_i b_i M_i \textbf{ mod } m_i = a_i \textbf{ mod } m_i$ for $i = 1, ..., n$. $\square$

7.2 Gödel's β predicate

Lemma 0.28 Let $\beta(a, b, i, x)$ be the predicate over natural numbers defined by

$$\beta(a, b, i, x) \quad =_{def} \quad x = a \ \mathbf{mod}\ ((1 + i) \times b + 1).$$

For any sequence $n_0, ..., n_k$ of natural numbers there are natural numbers n, m such that for all j, $0 \leq j \leq k$, and all x we have

$$\beta(n, m, j, x) \Leftrightarrow x = n_j.$$

Proof: Let $m' = \max\{k + 1, n_0, ..., n_k\}$ and $m = m'!$. We claim that $m + 1,\ 2m + 1, ..., (k + 1)m + 1$ are relatively prime. Suppose $p|(im + 1)$ and $p|(jm + 1)$ where $j > i > 0$. Then $p|(j - i)m$, thus $p|(j - i)$ or $p|m$. Since $(j - i)|m$, we have $p|m$. But then $p \not|(im + 1)$, a contradiction.

By the Chinese remainder theorem there is a number n such that $n = n_j\ \mathbf{mod}\ ((j + 1)m + 1)$ for $j = 0, ..., k$. □

Lemma 0.29 Let $F(x, y)$ be the predicate over natural numbers x, and positive and negative numbers y given by

$$\begin{aligned} F(x, y) \quad =_{def} \quad & x \geq 0 \ \& \\ & \exists z \geq 0.((x = 2z \Rightarrow y = z) \ \& \ (x = 2z + 1 \Rightarrow y = -z)) \end{aligned}$$

Define $\beta^{\pm}(n, m, j, y) =_{def} \exists x.(\beta(n, m, j, x) \ \& \ F(x, y))$.
Then for any sequence $n_0, ..., n_k$ of positive or negative numbers there are natural numbers n, m s.t. for all j, $0 \leq j \leq k$, and all x we have
$\beta^{\pm}(n, m, j, x) \Leftrightarrow x = n_j$.

Proof: $F(n, m)$ expresses the 1-1 correspondence between $n \in \omega$ and $m \in \mathbf{N}$ in which even n stand for non-negative and odd n for negative numbers. Then apply Lemma 0.28. $\square$

7.2 Weakest preconditions and expressiveness

Theorem 0.30 **Assn** is expressive.

Proof: Show by structural induction on commands c that for all assertion B there is an assertion $w[\![c, B]\!]$ s.t. for all interpretation I,
$wp^I[\![c, B]\!] = w[\![c, B]\!]^I$, i.e.
$\sigma \models^I w[\![c, B]\!]$ iff $\mathcal{C}[\![c]\!]\sigma \models^I B$ for all states σ.

- $w[\![\mathbf{skip}, B]\!] \equiv B$
- $w[\![X := a, B]\!] \equiv B[a/X]$
- $w[\![c_0; c_1, B]\!] \equiv w[\![c_0, w[\![c_1, B]\!]]\!]$
- $w[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, B]\!] \equiv (b \wedge w[\![c_0, B]\!]) \vee (\neg b \wedge w[\![c_1, B]\!])$.
- $w[\![\mathbf{while}\ b\ \mathbf{do}\ c_0]\!]$ is complicated but can be defined. See Page 105.

$\square$

7.2 Weakest preconditions and expressiveness

Lemma 0.31 For $c \in \mathbf{Com}$, $B \in \mathbf{Assn}$, let $w[\![c, B]\!]$ be an assertion expressing the weakest precondition, i.e. $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$. Then $\vdash \{w[\![c, B]\!]\}c\{B\}$.

Proof: Show by structural induction on commands c. cf. Pag 107. $\square$

Theorem 0.32 The proof system for partial correctness is relatively complete, i.e. if $\models \{A\}c\{B\}$ then $\vdash \{A\}c\{B\}$.

Proof: Lemma 0.31 gives $\vdash \{w[\![c, B]\!]c\{B\}\}$. If $\models \{A\}c\{B\}$ then $\models A \Rightarrow w[\![c, B]\!]$, by the consequence rule we obtain $\vdash \{A\}c\{B\}$. $\square$

7.3 Proof of Gödel's Theorem

Theorem 0.33 The subset of assertions $\{A \in \mathbf{Assn} \mid \models A\}$ is not recursively enumerable.

Proof: For a command c, let A_c be the assertion $w[\![c, \mathbf{false}]\!][\widetilde{0}/\widetilde{X}]$ where $\widetilde{X}$ collects all locations mentioned in $w[\![c, \mathbf{false}]\!]$. If $\{A \in \mathbf{Assn} \mid \models A\}$ is recursively enumerable, there would be a computational method to check the validity of A_c, thus confirming the divergence of c on the zero-state. But it is known that the commands c which diverge on the zero-state do not form a recursively enumerable set. $\square$

7.3 Proof of Gödel's Theorem

Theorem 0.34 There is no effective proof system for **Assn** s.t. its theorems coincide with the valid assertions of **Assn**.

Proof: Suppose there were an effective proof system for **Assn** so that A is provable iff A is valid. Being effective means there is a computational method to confirm precisely when something is a proof. Searching through all proofs systematically till a proof of assertion A is found would provide a computational method of confirming precisely when A is valid, contradicting Theorem 0.33. $\square$

7.4 Verification conditions

$\models \{A\}c\{B\}$ Iff $\models A \Rightarrow w[\![c, B]\!]$. However, the previous method of obtaining $w[\![c, B]\!]$ is inefficient and not practical.

Define annotated commands:

$$\begin{aligned} c ::= \quad & \mathbf{skip} \mid X := a \mid c_0; (X := a) \mid c_0; \{D\}c_1 \mid \\ & \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ \{D\}c \end{aligned}$$

where D is an assertion, and in $c_0; \{D\}c_1$, the annotated command c_1 is NOT an assignment. The assertion D in a while-loop is intended to be an invariant, i.e. $\{D \wedge b\}c\{D\}$ is valid.

7.4 Verification conditions

$\models \{A\}c\{B\}$ Iff $\models A \Rightarrow w[\![c, B]\!]$. However, the previous method of obtaining $w[\![c, B]\!]$ is inefficient and not practical.

Define verification conditions:

$$
\begin{array}{rcl}
vc(\{A\}\mathbf{skip}\{B\}) & = & \{A \Rightarrow B\} \\
vc(\{A\}X := a\{B\}) & = & \{A \Rightarrow B[a/X]\} \\
vc(\{A\}c_0; X := a\{B\}) & = & vc(\{A\}c_0\{B[a/X]\}) \\
vc(\{A\}c_0; \{D\}c_1\{B\}) & = & vc(\{A\}c_0\{D\}) \cup vc(\{D\}c_1\{B\}) \\
vc(\{A\}\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\{B\}) & = & vc(\{A \wedge b\}c_0\{B\}) \cup vc(\{A \wedge \neg b\}c_1\{B\}) \\
vc(\{A\}\mathbf{while}\ b\ \mathbf{do}\ \{D\}c\{B\}) & = & vc(\{D \wedge b\}c\{D\}) \cup \{A \Rightarrow D\} \cup \{D \wedge \neg b \Rightarrow B\}
\end{array}
$$

7.4 Verification conditions

To show the validity of an annotated partial correctness assertion it is sufficient (but not necessary) to show its verification conditions are valid.

E.g. $\{\mathbf{true}\}\mathbf{while}\ \mathbf{false}\ \mathbf{do}\ \{\mathbf{false}\}\mathbf{skip}\{\mathbf{true}\}$ is certainly valid with **false** as an invariant, its verification condition contains

$$\mathbf{true} \Rightarrow \mathbf{false}$$

which is not a valid assertion.

7.5 Predicate transformers

Previously a command is a function $f : \Sigma \to \Sigma_\perp$, a state transformer.

Now consider the set of partial correctness predicates to be $Pred(\Sigma) = \{Q \mid Q \subseteq \Sigma_\perp \ \&\ \perp \in Q\}$. The cpo of predicates is $(Pred(\Sigma), \supseteq)$.

Let $f : \Sigma \to \Sigma_\perp$ be a partial function on states. Define

$$\begin{aligned} &W\ f : Pred(\Sigma) \to Pred(\Sigma);\\ &(W\ f)(Q) = \{\sigma \in \Sigma_\perp \mid f(\sigma) \in Q\} \cup \{\perp\} \end{aligned}$$

A command c is a predicate transformer with $(W(\mathcal{C}[\![c]\!]))(B^I) = wp^I[\![c, B]\!]$, which given a postcondition returns the weakest precondition.

Chapter 8. Introduction to domain theory

8.2 Streams — an example

Let S be the set of finite or infinite sequences of 0's and 1's which may end with a special symbol "\$". They admit the partial order: $s \sqsubseteq s'$ if s is a prefix of s'. This yields a cpo with bottom ϵ, the empty sequence.

Let's define a function $isone : S \to \{\mathbf{true}, \mathbf{false}\}$ to detect whether or not 1 appears in an input sequence. Certainly we have $isone(000\$) = \mathbf{false}$. How about $isone(000)$?

We introduce a "don't know" element standing for undefined. Then $isone : S \to \{\mathbf{true}, \mathbf{false}\}_\perp$ is a continuous function defined by

$$\begin{array}{rclrcl} isone(1s) & = & \mathbf{true} & isone(\$) & = & \mathbf{false} \\ isone(0s) & = & isone(s) & isone(\epsilon) & = & \perp \\ isone(0^\omega) & = & \perp & & & \end{array}$$

8.3 Constructions on cpo's

8.3.1 Discrete cpo's

- Discrete cpo's are simply sets where the partial order relation is the identity. Then an ω-chain has to be constant.
- Basic values, like truth values or the integers form discrete cpo's, as do syntactic sets.
- Any function from a discrete cpo to a cpo is always continuous. In particular, semantic functions from syntactic sets are continuous.

8.3.2 Finite products

Assume that $D_1, \cdots, D_k$ are cpo's. Their product $D_1 \times \cdots \times D_k$ is a cpo. The partial order is determined "coordinatewise", i.e.

$$(d_1, \cdots, d_k) \sqsubseteq (d'_1, \cdots, d'_k) \text{ iff } d_i \sqsubseteq d'_i \text{ for all } 1 \leq i \leq k$$

An ω-chain $(d_{1n}, \cdots, d_{kn})$ for $n \in \omega$, of the product has $(\bigsqcup_{n\in\omega} d_{1n}, \cdots, \bigsqcup_{n\in\omega} d_{kn})$ as an upper bound, and indeed the least upper bound. So

$$\bigsqcup_{n\in\omega} (d_{1n}, \cdots, d_{kn}) \;=\; (\bigsqcup_{n\in\omega} d_{1n}, \cdots, \bigsqcup_{n\in\omega} d_{kn})$$

8.3.2 Projection

The projection function $\pi_i : D_1 \times \cdots \times D_k \to D_i$, for $i = 1, \cdots, k$, selects the ith coordinate of a tuple: $\pi(d_1, \cdots, d_k) = d_i$.

Projection functions are continuous:

$$
\begin{aligned}
\pi(\bigsqcup_{n\in\omega}(d_{1n}, \cdots, d_{kn})) &= \pi(\bigsqcup_{n\in\omega} d_{1n}, \cdots, \bigsqcup_{n\in\omega} d_{kn}) \\
&= \bigsqcup_{n\in\omega} d_{in} \\
&= \bigsqcup_{n\in\omega} \pi(d_{1n}, \cdots, d_{kn})
\end{aligned}
$$

8.3.2 Tupling

Let $f_i : E \to D_i$, for $i = 1, \cdots, k$ be continuous functions. Define the tupling function $\langle f_1, \cdots, f_k \rangle : E \to D_1 \times \cdots \times D_k$ by taking $\langle f_1, \cdots, f_k \rangle(e) = (f_1(e), \cdots, f_k(e))$.

The tupling function satisfies the property

$$\pi \circ \langle f_1, \cdots, f_k \rangle = f_i \quad \mathrm{for} i = 1, \cdots, k$$

and is continuous:

$$\begin{array}{rcl} \langle f_1, \cdots, f_k \rangle(\bigsqcup_{n\in\omega} e_n) & = & (f_1(\bigsqcup_{n\in\omega} e_n), \cdots, f_k(\bigsqcup_{n\in\omega} e_n)) \\ & = & (\bigsqcup_{n\in\omega} f_1(e_n), \cdots, \bigsqcup_{n\in\omega} f_k(e_n)) \\ & = & \bigsqcup_{n\in\omega}(f_1(e_n), \cdots, f_k(e_n)) \\ & = & \bigsqcup_{n\in\omega}\langle f_1, \cdots, f_k \rangle(e_n) \end{array}$$

8.3.2 Product of functions

Let $f_i : D_i \to E_i$, for $i = 1, \cdots, k$, be continuous functions. Define $f_1 \times \cdots \times f_k : D_1 \times \cdots \times D_k \to E_1 \times \cdots \times E_k$ by taking $(f_1 \times \cdots \times f_k)(d_1, \cdots, d_k) = (f_1(d_1), \cdots, f_k(d_k))$

That is, $f_1 \times \cdots \times f_k \ = \ \langle f_1 \circ \pi_1, \cdots, f_k \circ \pi_k \rangle$.

Each component $f_i \circ \pi_i$ is continuous, being the composition of continuous functions, so is the tupling function $\langle f_1 \circ \pi_1, \cdots, f_k \circ \pi_k \rangle$.

8.3.2 Three important properties (1/3)

Lemma 0.35 Let $h : E \to D_1 \times \cdots \times D_k$ be a function from a cpo E to a product of cpo's. It is continuous iff for all i, $1 \leq i \leq k$, the functions $\pi_i \circ h : E \to D_i$ are continuous.

Proof: ($\Rightarrow$) The composition of continuous functions is continuous.

($\Leftarrow$) Suppose $\pi_i \circ h$ is continuous for $i = 1, \cdots, k$. For any $x \in E$,

$$h(x) = (\pi_1(h(x)), \cdots, \pi_k(h(x))) = (\pi_1 \circ h(x), \cdots, \pi_k \circ h(x)) = \langle \pi_1 \circ h, \cdots, \pi_k \circ h \rangle (x$$

Therefore, $h = \langle \pi_1 \circ h, \cdots, \pi_k \circ h \rangle$ which is continuous as each $\pi_i \circ h$ is. $\square$

8.3.2 Three important properties (2/3)

Proposition 0.36 Suppose $e_{n,m}$ are elements of a cpo E for $n, m \in \omega$ with the property that $e_{n,m} \sqsubseteq e_{n',m'}$ when $n \leq n'$ and $m \leq m'$. Then the set $\{e_{n,m} \mid n, m \in \omega\}$ has a least upper bound

$$\bigsqcup_{n,m\in\omega} e_{n,m} = \bigsqcup_{n\in\omega}(\bigsqcup_{m\in\omega} e_{n,m}) = \bigsqcup_{m\in\omega}(\bigsqcup_{n\in\omega} e_{n,m}) = \bigsqcup_{n\in\omega} e_{n,n}$$

Proof: We show that all of the sets

$$\{e_{n,m} \mid n, m \in \omega\}, \quad \{\bigsqcup_{m\in\omega} e_{n,m} \mid n \in \omega\}, \quad \{\bigsqcup_{n\in\omega} e_{n,m} \mid m \in \omega\}, \quad \{e_{n,n} \mid n \in \omega\}$$

have the same upper bounds, hence the same lubs. Easy to see that $\{e_{n,m} \mid n, m \in \omega\}$ and $\{e_{n,n} \mid n \in \omega\}$ have the same upper bounds because the former includes the latter and any $e_{n,m}$ can be dominated by one $e_{n,n}$. As the lub of an ω-chain $\bigsqcup_n e_{n,n}$ exists, hence the lub $\bigsqcup_{n,m\in\omega} e_{n,m}$ exists and is equal to it. Any upper bound of $\{\bigsqcup_m e_{n,m} \mid n \in \omega\}$ must be an upper bound of $\{e_{n,m} \mid n, m \in \omega\}$. Conversely any upper bound of $\{e_{n,m} \mid n, m \in \omega\}$ dominates any lub $\bigsqcup_{m\in\omega} e_{n,m}$ for any $m \in \omega$. Thus $\bigsqcup_m e_{n,m} \mid n \in \omega$ and $\{e_{n,m} \mid n, m \in \omega\}$ share the same upper bounds, so have equal lubs. Similarly, $\bigsqcup_{n,m\in\omega} e_{n,m} = \bigsqcup_{n\in\omega}(\bigsqcup_{m\in\omega} e_{n,m})$. $\square$

8.3.2 Three important properties (3/3)

Lemma 0.37 Let $f : D_1 \times \cdots \times D_k \to E$ be a function. Then f is continuous iff f is "continuous in each argument separately", i.e. for all i with $1 \leq i \leq k$, and any $d_1, ..., d_{i-1}, d_{i+1}, ..., d_k$ the function $D_i \to E$ given by $d_i \mapsto f(d_1, ..., d_i, ..., d_k)$ is continuous.

Proof: $(\Rightarrow)$Trivial.

$(\Leftarrow)$ Let $k = 2$; the general case is similar. Let $(x_0, y_0) \sqsubseteq (x_1, y_1) \sqsubseteq \cdots$ be a chain in $D_1 \times D_2$.

$$\begin{array}{rcll} f(\bigsqcup_n (x_n, y_n)) & = & f(\bigsqcup_p x_p, \bigsqcup_q y_q) & \\ & = & \bigsqcup_p f(x_p, \bigsqcup_q y_q) & \\ & = & \bigsqcup_p \bigsqcup_q f(x_p, y_q) & \\ & = & \bigsqcup_n f(x_n, y_n) & \text{by Prop. 0.36} \end{array}$$

$\square$

8.3.3 Function space

Let D, E be cpo's. The function space $[D \to E]$ consists of elements $\{f \mid f : D \to E \text{ is continuous}\}$ ordered pointwise by $f \sqsubseteq g$ iff $\forall d \in D.\ f(d) \sqsubseteq g(d)$. If E has a bottom element $\bot_E$, then the function space has a bottom s.t. $\bot_{[D \to E]}(d) = \bot_E$ for all $d \in D$. Lubs of chains of functions are given pointwise: a chain $f_0 \sqsubseteq f_1 \sqsubseteq \cdots$ has lub $\bigsqcup_{n \in \omega} f_n$ with $(\bigsqcup_n f_n)(d) = \bigsqcup_n f_n(d)$. The lub is continuous: let $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ be a chain in D, then

$$
\begin{array}{rcl}
(\bigsqcup_n f_n)(\bigsqcup_m d_m) & = & \bigsqcup_n f_n(\bigsqcup_m d_m) \\
& = & \bigsqcup_n (\bigsqcup_m f_n(d_m)) \\
& = & \bigsqcup_m (\bigsqcup_n f_n(d_m)) \\
& = & \bigsqcup_m ((\bigsqcup_n f_n)(d_m))
\end{array}
$$

So the function space $[D \to E]$ is also a cpo.

8.3.3 Function space

Let I be a discrete cpo and D a cpo. The special function space $[I \to D]$ is called power, often written as D^I. Its elements can be thought of as tuples $(d_i)_{i \in I}$ ordered coordinatewise.

When I is the finite set $\{1, 2, \cdots, k\}$, the cpo D^I is isomorphic to the product $D \times \cdots \times D$, written D^k.

8.3.3 Application

Let D, E be cpo's. Define $apply : [D \to E] \times D \to E$ to act as $apply(f, d) = f(d)$. Then *apply* is continuous as it's continuous in each argument separately:

- Let $f_0 \sqsubseteq f_1 \sqsubseteq \cdots$ be a chain of functions.

$$apply(\bigsqcup_n f_n, d) = (\bigsqcup_n f_n)(d) = \bigsqcup_n f_n(d) = \bigsqcup_n apply(f_n, d)$$

- Let $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ be a chain in D. Then

$$apply(f, \bigsqcup_n d_n) = f(\bigsqcup_n d_n) = \bigsqcup_n f(d_n) = \bigsqcup_n apply(f, d_n)$$

8.3.3 Curring[a]

Let D, E, F be cpo's and $g \in [F \times D \to E]$. Define $curry(g) : F \to [D \to E]$ to act as $curry(g) = \lambda v \in F.\lambda d \in D.g(v, d)$. Write h for $curry(g)$. Check that $h(v)$ for each $v \in F$ is continuous and that h is continuous.

- Let $v \in F$ and $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ be a chain in D.

$$h(v)(\bigsqcup_n d_n) = g(v, \bigsqcup_n d_n) = \bigsqcup_n g(v, d_n) = \bigsqcup_n h(v)(d_n)$$

- Let $v_0 \sqsubseteq v_1 \sqsubseteq \cdots$ be a chain in F and $d \in D$. Then

$$h(\bigsqcup_n v_n)(d) = g(\bigsqcup_n v_n, d) = \bigsqcup_n g(v_n, d) = \bigsqcup_n h(v_n)(d) = (\bigsqcup_n h(v_n))(d)$$

[a]Named after the US logician Haskell Curry

8.3.4 Lifting

Let D be a cpo. Define an injective (lifting) function $\lfloor - \rfloor$ on D with $\bot \neq \lfloor d \rfloor$ for any $d \in D$.

The lifted cpo $D_\bot$ has underlying set

$$D_\bot = \{\lfloor d \rfloor \mid d \in D\} \cup \{\bot\}$$

and partial order

$$d'_0 \sqsubseteq d'_1 \text{ iff either } d'_0 = \bot \text{ or } (\exists d_0, d_1. d'_0 = \lfloor d_0 \rfloor \ \& \ d'_1 = \lfloor d_1 \rfloor \ \& \ d_0 \sqsubseteq d_1).$$

So $\lfloor d_0 \rfloor \sqsubseteq \lfloor d_1 \rfloor$ in $D_\bot$ iff $d_0 \sqsubseteq d_1$ in D. Clearly the function $\lfloor - \rfloor : D \to D_\bot$ is continuous.

8.3.4 Lifting

A continuous function $f \in D \to E$ from a cpo D to a cpo E with a bottom, can be extended to a continuous function $f^* : D_\perp \to E$ by defining

$$f^*(d') = \begin{cases} f(d) & \text{if } d' = \lfloor d \rfloor \text{ for some } d \in D \\ \perp_E & \text{otherwise} \end{cases}$$

The operation $(-)^*$ is continuous. Let $f_0 \sqsubseteq f_1 \sqsubseteq \cdots$ be a chain in $[D \to E]$ and $d' \in D_\perp$.

- If $d' = \perp$ then $(\bigsqcup_n f_n)^*(d') = \perp_E = (\bigsqcup_n f_n^*)(d')$
- If $d' = \lfloor d \rfloor$ then

$$(\bigsqcup_n f_n)^*(d') = (\bigsqcup_n f_n)(d) = \bigsqcup_n f_n(d) = \bigsqcup_n f_n^*(d') = (\bigsqcup_n f_n^*)(d')$$

If function f is described by $\lambda x.e$ then write $let\ x \Leftarrow d'.e$ for $(\lambda x.e)^*(d')$.

8.3.5 Sums

Let $D_1, \cdots, D_k$ be cpo's. A sum $D_1 + \cdots + D_k$ has underlying set

$$\{in_1(d_1) \mid d_1 \in D_1\} \cup \cdots \cup \{in_k(d_k) \mid d_k \in D_k\}$$

and partial order

$$\begin{array}{lll} d \sqsubseteq d' & \text{iff} & (\exists d_1, d_1' \in D_1.\ d = in_1(d_1)\ \&\ d' = in_1(d_1')\ \&\ d_1 \sqsubseteq d_1')\ \| \\ & & \vdots \\ & & (\exists d_k, d_k' \in D_1.\ d = in_k(d_k)\ \&\ d' = in_k(d_k')\ \&\ d_k \sqsubseteq d_k') \end{array}$$

where $in_i(d) \neq in_j(d')$ for all $d \in D_i, d' \in D_j$ with $i \neq j$.

Easy to see that $D_1 + \cdots + D_k$ is a cpo and the injection functions $in_i : D_i \to D_1 + \cdots + D_k$ are continuous.

8.3.5 Sums

Let $f_i : D_i \to E$ are continuous functions, for $i = 1, ..., k$. They can be combined to be a function

$$[f_1, \cdots, f_k] : D_1 + \cdots + D_k \to E$$

given by

$$[f_1, \cdots, f_k](in_i(d_i)) = f_i(d_i) \text{ for all } d_i \in D_i,$$

for all $i = 1, ..., k$. That is, $[f_1, \cdots, f_k] \circ in_i = f_i$.

By Lemma 0.37 it can be shown that $[f_1, \cdots, f_k]$ is continuous.

8.3.5 Conditional

The truth values $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$ can be regarded as the sum of two cpo's: $\{\mathbf{true}\} + \{\mathbf{false}\}$, with $in_1(\mathbf{true}) = \mathbf{true}$ and $in_2(\mathbf{false}) = \mathbf{false}$. Let $\lambda x_1.e_1 : \{\mathbf{true}\} \to E$ and $\lambda x_2.e_2 : \{\mathbf{false}\} \to E$ be two obviously continuous functions to a cpo E.

Then $cond(t, e_1, e_2) =_{def} [\lambda x_1.e_1, \lambda x_2.e_2](t)$ behaves as a conditional:

$$cond(t, e_1, e_2) = \begin{cases} e_1 & \text{if } t = \mathbf{true} \\ e_2 & \text{if } t = \mathbf{false} \end{cases}$$

The conditional $(b \to e_1|e_2) =_{def} let\ t \Leftarrow b.\ cond(t, e_1, e_2)$ acts as

$$(b \to e_1|e_2) = \begin{cases} e_1 & \text{if } b = \lfloor \mathbf{true} \rfloor \\ e_2 & \text{if } b = \lfloor \mathbf{false} \rfloor \\ \bot & \text{if } b = \bot \end{cases}$$

8.3.5 Case construction

Let E be a cpo, and $D_1 + \cdots + D_k$ be a sum of cpo's with an element d. Suppose $\lambda x_i.e_i : D_i \to E$ are continuous functions for $1 \leq i \leq k$ Then

$$[\lambda x_1.e_1, ..., \lambda x_k.e_k](d)$$

describes the case-construction

$$\begin{array}{ll} case\ d\ of & in_1(x_1).e_1| \\ & \vdots \\ & in_k(x_k).e_k \end{array}$$

8.4 A metalanguage

Let expression e be an element of a cpo E. Say e is continuous in the variable $x \in D$ iff the function $\lambda x \in D.e : D \to E$ is continuous. Say e is continuous in its variables iff e is continuous in all variables.

Variables: A variable x ranging over elements of a cpo is continuous in its variables.

Constants: $\perp_D$; **true**; **false**; π_i; *apply*; *curry*; $(-)^*$; in_i; $[f_1, ..., f_k]$ etc.

Tupling: Let $e_i \in E_i$ for $i = 1, ..., k$. The tuple $(e_1, ..., e_k)$ is continuous in its variables provided its components are.

$$
\begin{array}{ll}
 & \lambda x.(e_1, \cdots, e_k) \text{ is continuous} \\
\Leftrightarrow & \pi_i \circ (\lambda x.(e_1, \cdots, e_k)) \text{ is continuous for } 1 \le i \le k \quad \text{by Lem. 0.35} \\
\Leftrightarrow & \lambda x.e_i \text{ is continuous for } 1 \le i \le k \\
\Leftrightarrow & e_i \text{ is continuous in } x \text{ for } 1 \le i \le k
\end{array}
$$

8.4 A metalanguage

Application: Let K be a continuous function (in **Constants**), and e is an argument.

$$\begin{array}{rl} & \lambda x.K(e) \text{ is continuous} \\ \Leftrightarrow & K \circ (\lambda x.e) \text{ is continuous} \\ \Leftarrow & \lambda x.e \text{ is continuous} \\ \Leftrightarrow & e \text{ is continuous in } x \end{array}$$

The application is continuous in its variables provided its argument is.

E.g. the general form of application $e_1(e_2)$ are continuous in variables if e_1, e_2 are, since $e_1(e_2) = apply(e_1, e_2)$, i.e. applying the constant *apply* to the tuple (e_1, e_2).

8.4 A metalanguage

λ-abstraction: Let $e \in E$ be continuous function in its variables. Form the abstraction $\lambda y.e : D \to E$. It is continuous in x iff

$$\begin{array}{ll} & \lambda x.\lambda y.e \text{ is continuous} \\ \Leftrightarrow & curry(\lambda x, y.e) \text{ is continuous} \\ \Leftarrow & \lambda x, y.e \text{ is continuous as } curry \text{ preserves continuity} \\ \Leftrightarrow & e \text{ is continuous in } x \text{ and } y \end{array}$$

The application is continuous in its variables provided its body is.

E.g. function composition preserves the property of being continuous in variables as $e_1 \circ e_2 = \lambda x.e_1(e_2(x))$.

8.4 A metalanguage

let-**construction:** Let D be be a cpo and E a cpo with bottom. If $e_1 \in D_\perp$ and $e_2 \in E$ are continuous in variables then so is the expression $let\ x \Leftarrow e_1.e_2$ since

$$(let\ x \Leftarrow e_1.e_2) = (\lambda x.e_2)^*(e_1)$$

which is built up by the methods admitted above.

case-**construction** Assume E is a cpo and $D_1 + \cdots + D_k$ a sum of cpo's with an element e continuous in variables. Suppose $e_i \in E$ are continuous in variables, then so is the case construction

$$\begin{array}{ll} case\ e\ of & in_1(x_1).e_1| \\ & \ldots \\ & in_k(x_k).e_k \end{array}$$

because it is just $[\lambda x_1.e_1, ..., \lambda x_k.e_k](e)$.

8.4 A metalanguage

Fixed-point operators: Each cpo D with bottom is associated with a fixed-point operator $fix : [D \to D] \to D$, which is continuous because

$$fix \;=\; \bigsqcup_{n \in \omega} (\lambda f. f^n(\perp)),$$

i.e. fix is the lub of the chain of functions

$$\lambda f.\perp \sqsubseteq \lambda f.f(\perp) \sqsubseteq \lambda f.f(f(\perp)) \sqsubseteq \cdots$$

where each of these is continuous and so an element of the cpo $[[D \to D] \to D]$. Thus their lub fix exists in the cpo.

Notation: we use $\mu x.e$ to abbreviate $fix(\lambda x.e)$.

Chapter 9. Recursion equations

9.1 The language REC

A simple programming language for recursive definition of functions. It has syntactic sets:

- numbers $n \in \mathbf{N}$
- variables over numbers $x \in \mathbf{Var}$
- function variables $f_1, f_2, ... \in \mathbf{Fvar}$

Terms $t, t_0, ...$ of **REC** have the following syntax:

$$t ::= n \mid x \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \mid \mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \mid f_i(t_1, ..., t_{a_i})$$

Evaluating t_0 to 0 means **true** and to nonzero numbers means **false**.

A term is closed when it contains no variables from **Var**.

9.1 The language REC

Function symbols f are given meaning by a declaration, consisting of equations:

$$\begin{array}{rcl} f_1(x_1, \cdots, x_{a_1}) & = & d_1 \\ & \vdots & \\ f_k(x_1, \cdots, x_{a_k}) & = & d_k \end{array}$$

where the variables of d_i are included in $x_1, \cdots, x_{a_i}$. Term d_i is the definition of f_i.

9.1 Two methods of evaluation

To evaluate a term $f(t)$, there are two methods:

- call-by-value: evaluate t first and once an integer n is obtained then evaluate $f(n)$
- call-by-name: pass to the definition of f, replacing all occurrences of x by t.

Consider the equations

$$\begin{aligned} f_1(x) &= f_1(x) + 1 \\ f_2(x) &= 1 \end{aligned}$$

How to evaluate the term $f_2(f_1(3))$?

9.2 Operational semantics of call-by-value

$$\frac{}{n \rightarrow_{va}^{d} n}$$

$$\frac{t_1 \rightarrow_{va}^{d} n_1 \quad t_2 \rightarrow_{va}^{d} n_2}{t_1 \ \mathbf{op} \ t_2 \rightarrow_{va}^{d} n_1 \ op \ n_2}$$

$$\frac{t_0 \rightarrow_{va}^{d} 0 \quad t_1 \rightarrow_{va}^{d} n_1}{\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \rightarrow_{va}^{d} n_1}$$

$$\frac{t_0 \rightarrow_{va}^{d} n_0 \quad t_2 \rightarrow_{va}^{d} n_2 \qquad n_0 \not\equiv 0}{\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \rightarrow_{va}^{d} n_2}$$

$$\frac{t_1 \rightarrow_{va}^{d} n_1 \ \cdots \ t_{a_i} \rightarrow_{va}^{d} n_{a_i} \quad d_i[n_1/x_1, ..., n_{a_i}/x_{a_i}] \rightarrow_{va}^{d} n}{f_i(t_1, ..., t_{a_i}) \rightarrow_{va}^{d} n}$$

Proposition 0.38 If $t \rightarrow_{va}^{d} n_1$ and $t \rightarrow_{va}^{d} n_2$, then $n_1 \equiv n_2$. $\square$

9.3 Denotational semantics of call-by-value

Terms will be assigned meanings in the presence of environments for variables and function variables.

An environment for variables is a function $\rho : \mathbf{Var} \to \mathbf{N}$. Write $\mathbf{Env}_{va} = [\mathbf{Var} \to \mathbf{N}]$ for the cpo of all such environments.

An environment for the function variables $f_1, ..., f_k$ is a tuple $\varphi = (\varphi_1, ..., \varphi_k)$ where $\varphi_i : \mathbf{N}^{a_i} \to \mathbf{N}_\perp$. Write $\mathbf{Fenv}_{va} = [\mathbf{N}^{a_1} \to \mathbf{N}_\perp] \times \cdots \times [\mathbf{N}^{a_k} \to \mathbf{N}_\perp]$ for the cpo of environments for function variables.

9.3 Denotational semantics of call-by-value

A term t denotes a function $[\![t]\!]_{va} \in [\mathbf{Fenv}_{va} \to [\mathbf{Env}_{va} \to \mathbf{N}_{\perp}]]$

$$
\begin{array}{rcl}
[\![n]\!]_{va} & = & \lambda\varphi.\lambda\rho.\lfloor n \rfloor \\
[\![x]\!]_{va} & = & \lambda\varphi.\lambda\rho.\lfloor \rho(x) \rfloor \\
[\![t_1 \ \mathbf{op}\ t_2]\!]_{va} & = & \lambda\varphi.\lambda\rho.[\![t_1]\!]_{va}\varphi\rho \ op_{\perp}\ [\![t_2]\!]_{va}\varphi\rho \qquad \mathbf{op} \ = +, -, \times \\
[\![\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]\!]_{va} & = & \lambda\varphi.\lambda\rho.Cond([\![t_0]\!]_{va}\varphi\rho, [\![t_1]\!]_{va}\varphi\rho, [\![t_2]\!]_{va}\varphi\rho) \\
[\![f_i(t_1, \cdots, t_{a_i})]\!]_{va} & = & \lambda\varphi.\lambda\rho. \\
 & & (let\ v_1 \Leftarrow [\![t_1]\!]_{va}\varphi\rho, ..., v_{a_i} \Leftarrow [\![t_{a_i}]\!]_{va}\varphi\rho.\ \varphi_i(v_1, \cdots, v_{a_i}
\end{array}
$$

9.3 Denotational semantics of call-by-value

Let $iszero : \mathbf{N} \to \mathbf{T}$ be defined as

$$iszero = \lambda n \in \mathbf{N}.\mathbf{if}\ n\ \mathbf{then\ true\ else\ false}$$

Its strict extension $iszero_\perp : \mathbf{N}_\perp \to \mathbf{T}_\perp$ is

$$iszero_\perp = \lambda z \in \mathbf{N}_\perp.let\ n \Leftarrow z.\lfloor iszero(n) \rfloor$$

which acts so

$$iszero_\perp(z) = \begin{cases} \lfloor \mathbf{true} \rfloor & \text{if } z = \lfloor 0 \rfloor \\ \lfloor \mathbf{false} \rfloor & \text{if } z = \lfloor n \rfloor \ \&\ n \neq 0 \\ \perp & \text{otherwise} \end{cases}$$

Then $Cond(z_0, z_1, z_2) = (iszero_\perp(z_0) \to z_1 | z_2)$ is continuous.

9.3 Denotational semantics of call-by-value

Lemma 0.39 For all terms t of **REC**, the denotation $[\![t]\!]_{va}$ is a continuous function in $[\mathbf{Fenv}_{va} \to [\mathbf{Env}_{va} \to \mathbf{N}_\perp]]$.

Proof: By structural induction on terms t. $\square$

Lemma 0.40 For all terms t of **REC**, if environments $\rho, \rho' \in \mathbf{Env}_{va}$ yield the same result on all variables which appear in t then for any $\varphi \in \mathbf{Fenv}_{va}$,

$$[\![t]\!]_{va}\varphi\rho = [\![t]\!]_{va}\varphi\rho'.$$

In particular, the denotation of a closed term $[\![t]\!]_{va}\varphi\rho$ is independent of the environment ρ.

Proof: By structural induction on terms t. $\square$

9.3 Denotational semantics of call-by-value

A declaration

$$\begin{array}{rcl} f_1(x_1,\cdots,x_{a_1}) & = & d_1 \\ & \vdots & \\ f_k(x_1,\cdots,x_{a_k}) & = & d_k \end{array}$$

determines a function environment $\delta = (\delta_1, ..., \delta_k)$ such that

$$\begin{array}{rcl} \delta_1(n_1,\cdots,n_{a_1}) & = & [\![d_1]\!]_{va}\delta\rho[n_1/x_1,...,n_{a_1}/x_{a_1}], \text{ for all } n_1,...,n_{a_1} \in \mathbf{N} \\ & \vdots & \\ \delta_k(n_1,\cdots,n_{a_k}) & = & [\![d_k]\!]_{va}\delta\rho[n_1/x_1,...,n_{a_k}/x_{a_k}], \text{ for all } n_1,...,n_{a_1} \in \mathbf{N} \end{array}$$

The updated environment $\rho[n/x]$ is continuous. View the discrete cpo **Var** as a sum of the singleton $\{x\}$ and $\mathbf{Var}\backslash\{x\}$, with the injection functions $in_1 : \{x\} \to \mathbf{Var}$ and $in_2 : (\mathbf{Var}\backslash\{x\}) \to \mathbf{Var}$ being the inclusion functions. Then $\rho[n/x]$ is equal to $\lambda y \in \mathbf{Var}.case\ y\ of\ in_1(x).n \mid in_2(w).\rho(w)$.

9.3 Denotational semantics of call-by-value

The equations of a declaration d will not in general determine a unique solution. We are interested in the least one, which is the least fixed point of the continuous function $F : \mathbf{Fenv}_{va} \to \mathbf{Fenv}_{va}$ given by

$$\begin{aligned} F(\varphi) \quad = \quad & (\lambda n_1, ..., n_{a_1} \in \mathbf{N}.[\![d_1]\!]_{va}\varphi\rho[n_1/x_1, ..., n_{a_1}/x_{a_1}], \\ & \cdots, \\ & \lambda n_1, ..., n_{a_k} \in \mathbf{N}.[\![d_k]\!]_{va}\varphi\rho[n_1/x_1, ..., n_{a_k}/x_{a_k}]) \end{aligned}$$

The function environment determined by the declaration d is $\delta = fix(F)$. A closed term t denotes a result $[\![t]\!]_{va}\delta\rho$ in $\mathbf{N}_\perp$ wrt this function environment δ, independent of what environment ρ is.

9.3 Denotational semantics of call-by-value: example 1

Consider the declaration

$$\begin{aligned} f_1 &= f_1 + 1 \\ f_2(x) &= 1 \end{aligned}$$

which determines the denotation of f_1, f_2 as $\delta = (\delta_1, \delta_2) \in \mathbf{N}_\perp \times [\mathbf{N} \to \mathbf{N}_\perp]$ where

$$\begin{aligned} (\delta_1, \delta_2) &= \mu\varphi.(\llbracket f_1 + 1 \rrbracket_{va}\varphi\rho,\ \lambda m \in \mathbf{N}.\llbracket 1 \rrbracket_{va}\varphi\rho[m/x]) \\ &= \mu\varphi.(\varphi_1 +_\perp \lfloor 1 \rfloor,\ \lambda m \in \mathbf{N}.\lfloor 1 \rfloor) \\ &= (\perp,\ \lambda m \in \mathbf{N}.\lfloor 1 \rfloor) \end{aligned}$$

From which, $\llbracket f_2(f_1) \rrbracket_{va}\delta\rho \ = \ let\ n_1 \Leftarrow \delta_1.\ \delta_2(n_1) \ = \ \perp$

9.3 Denotational semantics of call-by-value: example 2

Consider the declaration $f(x) = \textbf{if } x \textbf{ then } 1 \textbf{ else } x \times f(x-1)$. Let f denote the function $\delta \in [\mathbf{N} \to \mathbf{N}_\perp]$, t be the definition and ρ an arbitrary environment for variables.

$$\begin{aligned} \delta &= fix(\lambda\varphi.(\lambda m.[\![t]\!]_{va}\varphi\rho[m/x])) \\ &= \textstyle\bigsqcup_{r\in\omega} \delta^r. \end{aligned}$$

For an arbitrary $m \in \mathbf{N}$, $\delta^0(m) = \perp$ and

$$\delta^1(m) = cond(iszero(m), \lfloor 1 \rfloor, \lfloor m \rfloor \times_\perp \delta^0(m-1)) = \begin{cases} \lfloor 1 \rfloor & \text{if } m = 0 \\ \perp & \text{otherwise} \end{cases}$$

By mathematical induction, we obtain $\delta^r(m) = \begin{cases} \lfloor m! \rfloor & \text{if } 0 \le m < r \\ \perp & \text{otherwise} \end{cases}$. The least upper bound δ is $\delta(m) = \begin{cases} \lfloor m! \rfloor & \text{if } 0 \le m \\ \perp & \text{otherwise} \end{cases}$.

9.4 Equivalence of semantics for call-by-value

Lemma 0.41 Let t be a term and n a number. Let $\varphi \in \mathbf{Fenv}_{va}, \rho \in \mathbf{Env}_{va}$. Then $[\![t]\!]_{va}\varphi\rho[n/x] = [\![t[n/x]]\!]_{va}\varphi\rho$.

Proof: By structural induction on t. $\square$

Lemma 0.42 Let t be a closed term and n a number. Let $\rho \in \mathbf{Env}_{va}$. Then $t \rightarrow^{d}_{va} n \Rightarrow [\![t]\!]_{va}\delta\rho = \lfloor n \rfloor$.

Proof: By rule induction. Consider the rule instance for the last rule.

Assume

$$t_1 \rightarrow^d_{va} n_1 \quad \text{and} \quad [\![t_1]\!]_{va}\delta\rho = \lfloor n_1 \rfloor,$$
$$\vdots$$
$$t_{a_i} \rightarrow^d_{va} n_{a_i} \quad \text{and} \quad [\![t_{a_i}]\!]_{va}\delta\rho = \lfloor n_{a_i} \rfloor,$$
$$d_i[n_1/x_1, ..., n_{a_i}/x_{a_i}] \rightarrow^d_{va} n \quad \text{and} \quad [\![d_i[n_1/x_1, ..., n_{a_i}/x_{a_i}]]\!]_{va}\delta\rho = \lfloor n \rfloor$$

Then

$$\begin{aligned}
[\![f_i(t_1, ..., t_{a_i})]\!]_{va}\delta\rho = & \quad let\ v_1 \Leftarrow [\![t_1]\!]_{va}\delta\rho, ..., v_{a_i} \Leftarrow [\![t_{a_i}]\!]_{va}\delta\rho.\ \delta_i(v_1, ..., v_{a_i}) \\
= & \quad \delta_i(n_1, ..., n_{a_i}) \\
= & \quad [\![d_i]\!]_{va}\delta\rho[n_1/x_1, ..., n_{a_i}/x_{a_i}] \text{ by } \delta\text{'s definition as a fixed p} \\
= & \quad [\![d_i[n_1/x_1, ..., n_{a_i}/x_{a_i}]]\!]_{va}\delta\rho \text{ by Lem. 0.41} \\
= & \quad \lfloor n \rfloor
\end{aligned}$$

$\square$

9.4 Equivalence of semantics for call-by-value

Lemma 0.43 Let t be a closed term and $\rho \in \mathbf{Env}_{va}$. For all $n \in \mathbf{N}$,
$[\![t]\!]_{va}\delta\rho = \lfloor n \rfloor \Rightarrow t \rightarrow^d_{va} n$.

Proof: Define the function $\varphi_i : \mathbf{N}^{a_i} \rightarrow \mathbf{N}_\perp$, for $i = 1, ..., k$ by taking

$$\varphi_i(n_1, ..., n_{a_i}) = \begin{cases} \lfloor n \rfloor & \text{if } d_i[n_1/x_1, ..., n_{a_i}/x_{a_i}] \rightarrow^d_{va} n \\ \perp & \text{otherwise} \end{cases}$$

and show $\varphi = (\varphi_1, ..., \varphi_k)$ is a prefixed point the function F, thus $\delta \sqsubseteq \varphi$.

To this end, show by structural induction on t that provided the variables in t are included in $x_1, ..., x_l$, then

$$[\![t]\!]_{va}\varphi\rho[n_1/x_1, ..., n_l/x_l] = \lfloor n \rfloor \Rightarrow t[n_1/x_1, ..., n_l/x_l] \rightarrow^d_{va} n \qquad (1)$$

for all $n, n_1, ..., n_l \in \mathbf{N}$.

For the case $t \equiv f_i(t_1, ..., t_{a_i})$, suppose
$[\![f_i(t_1, ..., t_{a_i})]\!]_{va}\varphi\rho[n_1/x_1, ..., n_l/x_l] = \lfloor n \rfloor$. Then there must be

$m_1, ..., m_{a_i} \in \mathbf{N}$ s.t. $[\![t_j]\!]_{va}\varphi\rho[n_1/x_1, ..., n_l/x_l] = \lfloor m_j \rfloor$ for $j = 1, ..., a_i$, with $\varphi_i(m_1, ..., m_{a_i}) = \lfloor n \rfloor$. By induction, $t_j[n_1/x_1, ..., n_l/x_l] \rightarrow^d_{va} m_j$ for all j and $d_i[m_1/x_1, ..., m_{a_i}/x_{a_i}] \rightarrow^d_{va} n$. It follows that $f_i(t_1, ..., t_{a_i})[n_1/x_1, ..., n_l/x_l] \rightarrow^d_{va} n$ as was to be proved.

As a special case of (1),

$$[\![d_i]\!]_{va}\varphi\rho[n_1/x_1, ..., n_{a_i}/x_{a_i}] = \lfloor n \rfloor \Rightarrow d_i[n_1/x_1, ..., n_{a_i}/x_{a_i}] \rightarrow^d_{va} n$$

for all $n, n_1, ..., n_{a_i} \in \mathbf{N}$. Thus by the definition of φ,

$$\begin{array}{rcl} \lambda n_1, ..., n_{a_1} \in \mathbf{N}.[\![d_i]\!]_{va}\varphi\rho[n_1/x_1, ..., n_{a_1}/x_{a_1}] & \sqsubseteq & \varphi_1 \\ & \vdots & \\ \lambda n_1, ..., n_{a_k} \in \mathbf{N}.[\![d_i]\!]_{va}\varphi\rho[n_1/x_1, ..., n_{a_k}/x_{a_k}] & \sqsubseteq & \varphi_k \end{array}$$

which makes φ a prefixed point of F. It follows that

$$[\![t]\!]_{va}\delta\rho = \lfloor n \rfloor \;\Rightarrow\; [\![t]\!]_{va}\varphi\rho = \lfloor n \rfloor \;\Rightarrow t \rightarrow^d_{va} n$$

□

9.5 Operational semantics of call-by-name

$$\frac{}{n \rightarrow_{na}^{d} n}$$

$$\frac{t_1 \rightarrow_{na}^{d} n_1 \quad t_2 \rightarrow_{na}^{d} n_2}{t_1 \ \mathbf{op} \ t_2 \rightarrow_{na}^{d} n_1 \ op \ n_2}$$

$$\frac{t_0 \rightarrow_{na}^{d} 0 \quad t_1 \rightarrow_{na}^{d} n_1}{\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \rightarrow_{na}^{d} n_1}$$

$$\frac{t_0 \rightarrow_{na}^{d} n_0 \quad t_2 \rightarrow_{na}^{d} n_2 \qquad n_0 \not\equiv 0}{\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \rightarrow_{na}^{d} n_2}$$

$$\frac{d_i[t_1/x_1, ..., t_{a_i}/x_{a_i}] \rightarrow_{na}^{d} n}{f_i(t_1, ..., t_{a_i}) \rightarrow_{na}^{d} n}$$

Proposition 0.44 If $t \rightarrow_{na}^{d} n_1$ and $t \rightarrow_{na}^{d} n_2$, then $n_1 \equiv n_2$. $\square$

9.6 Denotational semantics of call-by-name

A term will be assigned a meaning as a value in $\mathbf{N}_\perp$ wrt environments for variables and function variables.

An environment for variables is now a function $\rho : \mathbf{Var} \to \mathbf{N}_\perp$. Write $\mathbf{Env}_{na} = [\mathbf{Var} \to \mathbf{N}_\perp]$ for the cpo of all such environments.

An environment for the function variables $f_1, ..., f_k$ is a tuple $\varphi = (\varphi_1, ..., \varphi_k)$ where $\varphi_i : \mathbf{N}_\perp^{a_i} \to \mathbf{N}_\perp$. Write $\mathbf{Fenv}_{na} = [\mathbf{N}_\perp^{a_1} \to \mathbf{N}_\perp] \times \cdots \times [\mathbf{N}_\perp^{a_k} \to \mathbf{N}_\perp]$ for the cpo of environments for function variables.

9.6 Denotational semantics of call-by-name

A term t denotes a function $[\![t]\!]_{na} \in [\mathbf{Fenv}_{na} \rightarrow [\mathbf{Env}_{na} \rightarrow \mathbf{N}_{\perp}]]$

$$
\begin{array}{rcl}
[\![n]\!]_{na} & = & \lambda\varphi.\lambda\rho.\lfloor n \rfloor \\
[\![x]\!]_{na} & = & \lambda\varphi.\lambda\rho.\lfloor \rho(x) \rfloor \\
[\![t_1 \ \mathbf{op} \ t_2]\!]_{na} & = & \lambda\varphi.\lambda\rho.[\![t_1]\!]_{na}\varphi\rho \ op_{\perp} \ [\![t_2]\!]_{na}\varphi\rho \qquad \mathbf{op} \ = +, -, \times \\
[\![\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2]\!]_{na} & = & \lambda\varphi.\lambda\rho.Cond([\![t_0]\!]_{na}\varphi\rho, [\![t_1]\!]_{na}\varphi\rho, [\![t_2]\!]_{na}\varphi\rho) \\
[\![f_i(t_1, \cdots, t_{a_i})]\!]_{na} & = & \lambda\varphi.\lambda\rho. \ \varphi_i([\![t_1]\!]_{na}\varphi\rho, \cdots, [\![t_{a_i}]\!]_{na}\varphi\rho)
\end{array}
$$

9.6 Denotational semantics of call-by-name

Lemma 0.45 For all terms t of **REC**, the denotation $[\![t]\!]_{na}$ is a continuous function in $[\mathbf{Fenv}_{na} \to [\mathbf{Env}_{na} \to \mathbf{N}_{\perp}]]$.

Proof: By structural induction on terms t. $\square$

Lemma 0.46 For all terms t of **REC**, if environments $\rho, \rho' \in \mathbf{Env}_{na}$ yield the same result on all nariables which appear in t then for any $\varphi \in \mathbf{Fenv}_{na}$,

$$[\![t]\!]_{na}\varphi\rho = [\![t]\!]_{na}\varphi\rho'.$$

In particular, the denotation of a closed term $[\![t]\!]_{na}\varphi\rho$ is independent of the environment ρ.

Proof: By structural induction on terms t. $\square$

9.6 Denotational semantics of call-by-name

Let d be a declaration

$$
\begin{array}{rcl}
f_1(x_1, \cdots, x_{a_1}) & = & d_1 \\
 & \vdots & \\
f_k(x_1, \cdots, x_{a_k}) & = & d_k
\end{array}
$$

Define $F : \mathbf{Fenv}_{na} \to \mathbf{Fenv}_{na}$ by

$$
\begin{array}{rcl}
F(\varphi) & = & (\lambda z_1, ..., z_{a_1} \in \mathbf{N}.[\![d_1]\!]_{na}\varphi\rho[z_1/x_1, ..., z_{a_1}/x_{a_1}], \\
 & & \cdots, \\
 & & \lambda z_1, ..., z_{a_k} \in \mathbf{N}.[\![d_k]\!]_{na}\varphi\rho[z_1/x_1, ..., z_{a_k}/x_{a_k}])
\end{array}
$$

The function environment determined by the declaration d is $\delta = fix(F)$.

9.6 Denotational semantics of call-by-name: an example

Consider the declaration

$$\begin{aligned} f_1 &= f_1 + 1 \\ f_2(x) &= 1 \end{aligned}$$

which determines the denotation of f_1, f_2 as
$\delta = (\delta_1, \delta_2) \in \mathbf{N}_\perp \times [\mathbf{N}_\perp \to \mathbf{N}_\perp]$ where

$$\begin{aligned} (\delta_1, \delta_2) &= \mu\varphi.(\llbracket f_1 + 1 \rrbracket_{na}\varphi\rho,\ \lambda z \in \mathbf{N}_\perp.\llbracket 1 \rrbracket_{na}\varphi\rho[z/x]) \\ &= \mu\varphi.(\varphi_1 +_\perp \lfloor 1 \rfloor,\ \lambda z \in \mathbf{N}_\perp.\lfloor 1 \rfloor) \\ &= (\perp,\ \lambda z \in \mathbf{N}_\perp.\lfloor 1 \rfloor) \end{aligned}$$

From which, $\llbracket f_2(f_1) \rrbracket_{na}\delta\rho \ = \ \delta_2(\delta_1) \ = \ \lfloor 1 \rfloor$

9.7 Equivalence of semantics for call-by-name

Lemma 0.47 Let t be a term and n a number. Let $\varphi \in \mathbf{Fenv}_{na}, \rho \in \mathbf{Env}_{na}$. Then $[\![t]\!]_{na}\varphi\rho[[\![t']\!]_{na}\varphi\rho/x] = [\![t[t'/x]]\!]_{na}\varphi\rho$.

Proof: By structural induction on t. $\square$

9.7 Equinalence of semantics for call-by-name

Lemma 0.48 Let t be a closed term and n a number. Let $\rho \in \mathbf{Env}_{na}$. Then $t \rightarrow^{d}_{na} n \Rightarrow [\![t]\!]_{na}\delta\rho = \lfloor n \rfloor$.

Proof: By rule induction. Consider the rule instance for the last rule. Assume

$$d_i[t_1/x_1, ..., t_{a_i}/x_{a_i}] \rightarrow^{d}_{na} n \text{ and } [\![d_i[t_1/x_1, ..., t_{a_i}/x_{a_i}]]\!]_{na}\delta\rho = \lfloor n \rfloor$$

Then

$$\begin{aligned}
[\![f_i(t_1, ..., t_{a_i})]\!]_{na}\delta\rho = \quad & \delta_i([\![t_1]\!]_{na}\delta\rho, ..., [\![t_{a_i}]\!]_{na}\delta\rho) \\
= \quad & [\![d_i]\!]_{na}\delta\rho[[\![t_1]\!]_{na}\delta\rho/x_1, ..., [\![t_{a_i}]\!]_{na}\delta\rho/x_{a_i}] \text{ by } \delta\text{'s def as a fixed p} \\
= \quad & [\![d_i[t_1/x_1, ..., t_{a_i}/x_{a_i}]]\!]_{na}\delta\rho \text{ by Lem. 0.47} \\
= \quad & \lfloor n \rfloor
\end{aligned}$$

□

9.7 Equinalence of semantics for call-by-name

Lemma 0.49 Let t be a closed term and $\rho \in \mathbf{Env}_{na}$. For all $n \in \mathbf{N}$, $[\![t]\!]_{na}\delta\rho = \lfloor n \rfloor \Rightarrow t \rightarrow^{d}_{na} n$.

Proof: Define $res(t) = \begin{cases} \lfloor n \rfloor & \text{if } t \rightarrow^{d}_{na} n \\ \perp & \text{otherwise} \end{cases}$ Let δ^r be the rth approximant to δ. Show by induction on $r \in \omega$ that

$$[\![t]\!]_{na}\delta^r\rho[res(u_1)/y_1, ..., res(u_s)/y_s] = \lfloor n \rfloor \Rightarrow t[u_1/y_1, ..., u_s/y_s] \rightarrow^{d}_{na} n \qquad (2)$$

It is equivalent to

$$[\![t]\!]_{na}\delta^r\rho[res(u_1)/y_1, ..., res(u_s)/y_s] = \lfloor n \rfloor \sqsubseteq res(t[u_1/y_1, ..., u_s/y_s])$$

Consider the induction step. Suppose the induction hypothesis holds for $(r-1)$. We show (2) by structural induction on t. Only consider the case

$t \equiv f_i(t_1, ..., t_{a_i})$. Let $\rho' = \rho[res(u_1)/y_1, ..., res(u_s)/y_s]$.

$$\begin{array}{rcl} [\![f_i(t_1, ..., t_{a_i})]\!]_{na}\delta^r\rho' & = & \delta_i^r([\![t_1]\!]_{na}\delta^r\rho', ..., [\![t_{a_i}]\!]_{na}\delta^r\rho') \\ & = & [\![d_i]\!]_{na}\delta^{r-1}\rho'[[\![t_1]\!]_{na}\delta^r\rho'/x_1, ..., [\![t_{a_i}]\!]_{na}\delta^r\rho'/x_{a_i}] \end{array}$$

By structural induction

$$\begin{array}{rcl} [\![t_j]\!]_{na}\delta^r\rho' & = & [\![t_j]\!]_{na}\delta^r\rho[res(u_1)/y_1, ..., res(u_s)/y_s] \\ & \sqsubseteq & res(t_j[u_1/y_1, ..., u_s/y_s]) \end{array}$$

Then

$$\begin{array}{rcll} [\![f_i(t_1, ..., t_{a_i})]\!]_{na}\delta^r\rho' & \sqsubseteq & [\![d_i]\!]_{na}\delta^{r-1}\rho'[res(t_1')/x_1, ..., res(t_{a_i}')/x_{a_i}] & \text{monotonicity} \\ & \sqsubseteq & res(d_i[t_1'/x_1, ..., t_{a_i}'/x_{a_i}]) & \text{by mathematical induction} \\ & = & res(f_i(t_1', ..., t_{a_i}')) & \text{by operational semantics} \end{array}$$

where $t_j' = t_j[u_1/y_1, ..., u_s/y_s]$, thus establishes the induction hypothesis.

Therefore, for closed term t, $[\![t]\!]_{na}\delta^r\rho = \lfloor n \rfloor \Rightarrow t \rightarrow_{na}^d n$ for all $r \in \omega$. Since $[\![t]\!]_{na}\delta\rho = [\![t]\!]_{na} \bigsqcup_r \delta^r\rho = \bigsqcup_r [\![t]\!]_{na}\delta^r\rho$ by the continuity of semantic function, $[\![t]\!]_{na}\delta\rho = \lfloor n \rfloor$ implies $[\![t]\!]_{na}\delta^r\rho = \lfloor n \rfloor$ for some r, and hence $t \rightarrow_{na}^d n$. $\square$

9.8 Local declarations

Let $S \equiv \textbf{let rec } A \Leftarrow t \textbf{ and } B \Leftarrow u \textbf{ in } v$. The denotation of S can be taken to be

$$[\![S]\!]\varphi\rho = [\![v]\!]\varphi[\alpha_0/A, \beta_0/B]\rho$$

where (α_0, β_0) is the least fixed point of the continuous function

$$(\alpha, \beta) \mapsto ([\![t]\!]\varphi[\alpha/A, \beta/B]\rho, [\![u]\!]\varphi[\alpha/A, \beta/B]\rho)$$

In general the language allows:

$$\begin{array}{llrcl}
\textbf{let} & \textbf{rec} & f_1(x_1, ..., x_{a_1}) & = & d_1 \textbf{ and} \\
 & & \vdots & & \\
 & & f_k(x_1, ..., x_{a_k}) & = & d_k \\
\textbf{in} & t & & &
\end{array}$$

Chapter 10. Techniques for recursion

10.1 Bekić's theorem

Theorem 0.50 Let $F : D \times E \to D$ and $G : D \times E \to E$ be continuous functions where D, E are cpo's. The least fixed point of $\langle F, G\rangle : D \times E \to D \times E$ is the pair with coordinates

$$\begin{array}{rcl}\hat{f} & = & \mu f.F(f, \mu g.G(\mu f.F(f, g), g)) \\ \hat{g} & = & \mu g.G(\mu f.F(f, g), g)\end{array}$$

Proof: First show $(\hat{f}, \hat{g})$ is a fixed point of $\langle F, G\rangle$. By definition $\hat{f} = \mu f.F(f, \hat{g})$, the least fixed point of $\lambda f.F(f, \hat{g})$. Also the definition of $\hat{g}$ says

$$\hat{g} = G(\mu f.F(f, \hat{g}), \hat{g}) = G(\hat{f}, \hat{g})$$

Thus $(\hat{f}, \hat{g}) = \langle F, G\rangle(\hat{f}, \hat{g})$.

Let (f_0, g_0) be the least fixed point of $\langle F, G\rangle$, then $(f_0, g_0) \sqsubseteq (\hat{f}, \hat{g})$. For the converse ordering, as $f_0 = F(f_0, g_0)$, we have $\mu f.F(f, g_0) \sqsubseteq f_0$. The monotonicity of G yields $G(\mu f.F(f, g_0), g_0) \sqsubseteq G(f_0, g_0) = g_0$ Thus $\hat{g} \sqsubseteq g_0$. The monotonicity of F yields $F(f_0, \hat{g}) \sqsubseteq F(f_0, g_0) = f_0$, thus $\hat{f} \sqsubseteq f_0$. $\square$

10.1 Bekić's theorem

- The proof only relies on the monotonicity and the properties of least fixed point, so it works for monotonic functions on lattices.
- From Bekić's theorem we can deduce a symmetric form of simultaneous least fixed point.

$$\begin{aligned}\hat{f} &= \mu f.F(f, \mu g.G(f, g)) \\ \hat{g} &= \mu g.G(\mu f.F(f, g), g)\end{aligned}$$

The second equation is the same as in Bekić's theorem. The first follows by the symmetry between f and g.

10.1 Bekić's theorem: an example

Consider the term
$$\begin{array}{rcl} T & \equiv & \textbf{let}\ \ \textbf{rec}\ B \Leftarrow (\textbf{let rec}\ A \Leftarrow t\ \textbf{in}\ u) \\ & & \textbf{in}\ (\textbf{let rec}\ A \Leftarrow t\ \textbf{in}\ v) \end{array}$$

Abbreviate $F(f,g) = [\![t]\!]\varphi[f/A, g/B]\rho$ and $G(f,g) = [\![u]\!]\varphi[f/A, g/B]\rho$. Then $[\![T]\!]\varphi\rho = [\![v]\!]\varphi[\hat{f}/A, \hat{g}/B]\rho$ where

$$\begin{array}{rcl} \hat{g} & = & \mu g.[\![\textbf{let rec}\ A \Leftarrow t\ \textbf{in}\ u]\!]\varphi[g/B]\rho \\ & = & \mu g.[\![u]\!]\varphi[g/B, \mu f.[\![t]\!]\varphi[f/A, g/B]\rho/A]\rho \\ & = & \mu g.G(\mu f.F(f,g), g) \end{array}$$

and $\hat{f} = \mu f.[\![t]\!]\varphi[f/A, \hat{g}/B]\rho = \mu f.F(f, \hat{g})$. By Bekić's theorem, $(\hat{f}, \hat{g})$ is the (simultaneous) least fixed point of $\langle F, G\rangle$. So

$$[\![T]\!] = [\![\textbf{let rec}\ A \Leftarrow t\ \textbf{and}\ B \Leftarrow u\ \textbf{in}\ v]\!]$$

10.2 Fixed point induction

Let D be a cpo. A subset P of D is inclusive iff for all ω-chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$, if $d_n \in P$ for all $n \in \omega$ then $\bigsqcup_{n\in\omega} d_n \in P$.

Proposition 0.51 Let D be a cpo with bottom $\perp$, and $F : D \to D$ be continuous. Let P be an inclusive subset of D. If $\perp \in P$ and $\forall x \in D.\ x \in P \Rightarrow F(x) \in P$ then $fix(F) \in P$.

Proof: Note $fix(F) = \bigsqcup_n F^n(\perp)$. If P satisfies the condition above, then $F^n(\perp) \in P$ for all n by mathematical induction. By the inclusiveness of P, we obtain $fix(F) \in P$. $\square$

10.2 Fixed point induction

Fixed point induction implies Park induction.

Proposition 0.52 Let D be a cpo with bottom, and $F : D \to D$ be continuous. Let $d \in D$. If $F(d) \sqsubseteq d$ then $fix(F) \sqsubseteq d$.

Proof: The set $P = \{x \in D \mid x \sqsubseteq d\}$ is inclusive. If every element in a chain is below d, then so is the lub $\bigsqcup_n d_n$. Clearly, $\perp \in P$. If $x \in P$, i.e. $x \sqsubseteq d$, then the monotonicity of F yields $F(x) \sqsubseteq F(d) \sqsubseteq d$, thus $F(x) \in d$. By fixed point induction, $fix(F) \in P$, i.e. $fix(F) \sqsubseteq d$. $\square$

10.2 Fixed point induction

A predicate $Q(x_1, ..., x_k)$ with free variables $x_1, ..., x_k$ ranging over the cpo's $D_1, \cdots, D_k$ respectively, determines a set

$$P = \{(x_1, ..., x_k) \in D_1 \times \cdots \times D_k \mid Q(x_1, \cdots, x_k)\}$$

We say the predicate $Q(x_1, ..., x_k)$ is inclusive if its extension as a set is inclusive.

We rephrase fixed point induction as follows. Let $F : D_1 \times \cdots \times D_k \to D_1 \times \cdots \times D_k$ be a continuous function on a product cpo $D_1 \times \cdots \times D_k$ with bottom $(\bot_1, ..., \bot_k)$. Assuming $Q(x_1, ..., x_k)$ is an inclusive predicate, if $Q(\bot_1, ..., \bot_k)$ and

$$\forall x_1 \in D_1, ..., x_k \in D_k.\ Q(x_1, ..., x_k) \Rightarrow Q(F(x_1, ..., x_k))$$

then $Q(fix(F))$.

10.2 Inclusive sets and predicates

Basic relations: Let D be a cpo. The binary relations

$$\{(x,y) \in D \times D \mid x \sqsubseteq y\} \text{ and } \{(x,y) \in D \times D \mid x = y\}$$

are inclusive subsets of $D \times D$. So the predicates $x \sqsubseteq y,\ x = y$ are inclusive.

Inverse image: Let $f : D \to E$ be a continuous function between cpo's D and E. If P is an inclusive subset of E then the inverse image

$$f^{-1}P = \{x \in D \mid f(x) \in P\}$$

is an inclusive subset of D.

10.2 Inclusive sets and predicates

Substitution: Inclusive predicates are closed under the substitution of terms for their variables, provided the terms are continuous in their variables. Let $Q(y_1, \cdots, y_l)$ be an inclusive predicate of $E_1 \times \cdots \times E_l$, i.e.

$$P =_{def} \{(y_1, ..., y_l) \in E_1 \times \cdots \times E_l \mid Q(y_1, \cdots, y_l)\}$$

is an inclusive set. Suppose $e_1, \cdots, e_l$ are expressions for elements of $E_1, \cdots, E_l$, continuous in their variables $x_1, \cdots, x_k$ over $D_1, \cdots, D_k$. Then function $f =_{def} \lambda x_1, \cdots, x_k.\ (e_1, ..., e_l)$ is continuous. Thus

$$f^{-1}P =_{def} \{(x_1, ..., x_k) \in D_1 \times \cdots \times D_k \mid Q(e_1, \cdots, e_l)\}$$

is inclusive, and thus $Q(e_1, \cdots, e_l)$ is an inclusive predicate of $D_1 \times \cdots \times D_k$.

E.g. Take $f = \lambda x \in D.\ (x, c)$. If $R(x, y)$ is an inclusive predicate of $D \times E$, then $R(x, c)$, obtained by fixing y to a constant c is an inclusive predicate of D.

10.2 Inclusive sets and predicates

Logical operation: Let D be a cpo. Then D (predicate "true") and $\emptyset$ ("false") are inclusive. Let $P, Q \subseteq D$ be inclusive, then $P \cup Q$ and $P \cap Q$ are inclusive. That is, if $P(x_1, ..., x_k)$ and $Q(x_1, ..., x_k)$ are inclusive predicates then so are $P(x_1, ..., x_k)$ or $Q(x_1, ..., x_k)$ and $P(x_1, ..., x_k) \;\;\&\;\; Q(x_1, ..., x_k)$.

If $P_i, i \in I$ is an indexed family of inclusive subsets of E then so is $\bigcap_{i \in I} P_i$.

Note that infinite unions of inclusive subsets need not be inclusive, and thus inclusive predicates are not generally closed under $\exists$-quantification.

10.2 Inclusive sets and predicates

Direct image under order-monics: Let D, E be cpo's. A continuous function $f : D \to E$ is an order-monic iff $f(d) \sqsubseteq f(d') \Rightarrow d \sqsubseteq d'$ for all $d, d' \in D$. E.g. the "lifting" function $\lfloor - \rfloor$ and injection functions in_i associated with a sum are order-monics.

If P is inclusive then so is its direct image fP where f is an order-monic. Thus, if $Q(x)$ is an inclusive predicate of D then $\exists x \in D.\ y = f(x)\ \&\ Q(x)$ with free variable $y \in E$, is an inclusive predicate of E.

10.2 Inclusive sets and predicates

Discrete cpo's: Any subset of a discrete cpo, and any predicate on a discrete cpo, are inclusive.

Product cpo's: Let $P_i \subseteq D_i$ be inclusive subsets. Then

$$P_1 \times \cdots \times P_k = \{(x_1, \cdots, x_k) \mid x_1 \in P_1 \;\&\; \cdots \;\&\; x_k \in P_k\}$$

is an inclusive subset of the product $D_1 \times \cdots \times D_k$ as

$$P_1 \times \cdots \times P_k = \pi_1^{-1} P_1 \cap \cdots \cap \pi_k^{-1} P_k.$$

Each inverse image $\pi_i^{-1} P_i$ is inclusive, and is their intersection. $P(x_1, ..., x_k)$ is inclusive in each argument separately, if for each i, the predicate $P(d_1, ..., d_{i-1}, x_i, d_{i+1}, ..., d_k)$ got by fixing all but the ith argument, is an inclusive predicate of D_i. If $P(x_1, ..., x_k)$ is inclusive then it is inclusive in each argument separately. The converse does not hold in general. Consider the product cpo $\Omega \times \Omega$, and the predicate $P(x, y) =_{def} (x = y \;\&\; x \neq \infty)$.

10.2 Inclusive sets and predicates

Function space: Let D, E be cpo's, and $P \subseteq D, Q \subseteq E$ be inclusive subsets. Then

$$P \to Q =_{def} \{f \in [D \to E] \mid \forall x \in P. f(x) \in Q\}$$

is an inclusive subset of the function space $[D \to E]$. Thus, the predicate $\forall x \in D.\ P(x) \Rightarrow Q(f(x))$, with free variable $f \in [D \to E]$, is inclusive when $P(x), Q(y)$ are inclusive predicates of D, E respectively.

Lifting: Let P be an inclusive subset of a cpo D. As $\lfloor - \rfloor$ is an order-monic, the direct image $\{\lfloor d \rfloor \mid d \in P\}$ is an inclusive subset of $D_\perp$. If $Q(x)$ is an inclusive predicate of D, then $\exists x \in D.\ y = \lfloor x \rfloor\ \&\ Q(x)$ with free variable $y \in D_\perp$, is an predicate of $D_\perp$.

10.2 Inclusive sets and predicates

Sum: Let P_i be an inclusive subset of the cpo D_i. Then

$$P_1 + \cdots + P_k = in_1 P_1 \cup \cdots \cup in_k P_k$$

is an inclusive subset of the sum $D_1 + \cdots + D_k$, because each injection is an order-monic. Thus the predicate

$$(\exists x_1 \in D_1.\, y = in_1(x_1)\ \&\ Q_1(x_1)) \text{ or } \cdots \text{ or } (\exists x_k \in D_k.\, y = in_k(x_k)\ \&\ Q_k(x_k))$$

with free variables $y \in D_1 + \cdots + D_k$ is an inclusive predicate of the sum if each $Q_i(x_i)$ is inclusive in D_i.

Proposition 0.53 Any predicate of the form $\forall x_1, ..., x_n.\ P$ is inclusive where $x_1, ..., x_n$ are variables ranging over specific cpo's, and P is built up by conjunctions and disjunctions of basic predicates of the form $e_0 \sqsubseteq e_1$ or $e_0 = e_1$, where e_0, e_1 are expressions in the metalanguage of expressions from Section 8.4.

10.3 Well-founded induction

Let $\prec$ be a well founded relation on a set A. Let P be a property. Then $\forall a \in A.P(a)$ iff $\forall a \in A.\ ((\forall b \prec a.\ P(b)) \Rightarrow P(a))$

Well founded relations:

- Product: If $\prec_1$ and $\prec_2$ are well-founded, taking $(a_1, a_2) \preceq (a_1', a_2') \ \Leftrightarrow\ a_1 \preceq_1 a_1'$ and $a_2 \preceq_2 a_2'$ determines a well founded relation $\prec = (\preceq \setminus Id_{A_1 \times A_2})$
- Lexicographic products: $(a_1, a_2) \prec_{lex} (a_1', a_2') \ \Leftrightarrow\ a_1 \prec_1 a_1'$ or $(a_1 = a_1'\ \&\ a_2 \prec_2 a_2')$
- Inverse image: Let $f : A \to B$ be a function and $\prec_B$ is well-founded on B, then so is $\prec_A$ on A, where $a \prec_A a' \ \Leftrightarrow\ f(a) \prec_B f(a')$

10.3 Well-founded induction: an example

Ackermann's function.

$$
\begin{array}{llll}
A(x,y) = & \textbf{if } x \textbf{ then} & y+1 \textbf{ else} & \\
 & & \textbf{if } y \textbf{ then} & A(x-1,1) \textbf{ else} \\
 & & & A(x-1, A(x,y-1))
\end{array}
$$

For call-by-value, this declaration denotes the least function $a \in [\mathbf{N}^2, \mathbf{N}_\perp]$ s.t.

$$
a(m,n) = \begin{cases} \lfloor n+1 \rfloor & \text{if } m = 0 \\ a(m-1,1) & \text{if } m \neq 0 \;\&\; n = 0 \\ \mathit{let}\ l \Leftarrow a(m,n-1) \;.\; a(m-1,l) & \text{otherwise} \end{cases}
$$

for all $m, n \in \mathbf{N}$. The fact that $a(m,n)$ terminates is shown by well-founded induction on (m,n) ordered lexicographically.

10.3 Well-founded recursion

Notation: Each element $b \in B$ has a set of predecessors $\prec^{-1}\{b\} = \{b' \in B \mid b' \prec b\}$. The restriction of function $f : B \to C$ to $B' \subseteq B$ is $f \restriction B' = \{(b, f(b)) \mid b \in B'\}$

Theorem 0.54 Let $\prec$ be a well-founded relation on set B. Suppose $F(b, h) \in C$, for all $b \in B$ and functions $h : \prec^{-1}\{b\} \to C$. There is a unique $f : B \to C$ s.t. $\forall b \in B.\ f(b) = F(b, f \restriction \prec^{-1}\{b\})$

Proof: First show by well-founded induction a uniqueness property $P(x)$:

$$\begin{aligned} \forall y \prec^* x. \quad & f(y) = F(y, f \restriction \prec^{-1}\{y\}) \ \ \& \ \ g(y) = F(y, g \restriction \prec^{-1}\{y\}) \\ & \Rightarrow f(x) = g(x) \end{aligned}$$

for any $x \in B$. For any $x \in B$, assume $P(z)$ for every $z \prec x$. Then $f(z) = g(z)$. Thus $f \restriction \prec^{-1}\{x\} = g \restriction \prec^{-1}\{x\}$. It follows that $f(x) = F(x, f \restriction \prec^{-1}\{x\}) = F(x, g \restriction \prec^{-1}\{x\}) = g(x)$, thus $P(x)$.

Then show the existence of that function f. We need to prove a property $Q(x)$, for all $x \in B$, by well-founded induction,

$$\begin{aligned} \exists f_x \quad & : \prec^{*-1}\{x\} \to C. \\ & \forall y \prec^* x. f_x(y) = F(y, f_x \upharpoonright \prec^{-1} \{y\}). \end{aligned}$$

Suppose $\forall z \prec x. Q(z)$. Then $h = \bigcup\{f_z \mid z \prec x\}$ is a function because the uniqueness property ensures that the functions f_z agree on values assigned to common arguments y. Taking $f_x = h \cup \{(x, F(x, h))\}$ gives a function $f_x : \prec^{*-1}\{x\} \to C$ witnesses $Q(x)$.

Now take $f = \bigcup_{x \in B} f_x$. The uniqueness property yields $f : B \to C$, and f is the unique function we required. $\square$

10.3 Well-founded recursion: an example

By the well founded recursion theorem, there is a unique total function such that

$$ack(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ ack(m-1,1) & \text{if } m \neq 0 \ \& \ n = 0 \\ ack(m-1, ack(m,n-1)) & \text{otherwise} \end{cases}$$

for all $m, n \geq 0$. Observe that the value of ack at (m,n) is defined in terms of its value at the lexicographically smaller pairs $(m-1, l)$ and $(m, n-1)$.

Chapter 11. Languages with higher types

11.1 An eager language

Types are introduced in the language to classify different kinds of values terms can evaluate to.

Type expressions:

$$\tau ::= \mathbf{int} \mid \tau_1 * \tau_2 \mid \tau_1 \to \tau_2$$

Variables $x, y, \ldots$ in **Var** are associated with a unique type $\mathbf{type}(x)$.

11.1 Syntax of terms

$$
\begin{array}{lll}
t & ::= & x \mid \\
 & & n \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \mid \textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \mid \\
 & & (t_1, t_2) \mid \textbf{fst}(t) \mid \textbf{Snd}(t) \mid \\
 & & \lambda x.t \mid (t_1\ t_2) \mid \\
 & & \textbf{let } x \Leftarrow t_1 \textbf{ in } t_2 \mid \\
 & & \textbf{rec } y.(\lambda x.t)
\end{array}
$$

Here $\textbf{rec } y.(\lambda x.t)$ defines a function y to be $\lambda x.t$; the term t can involve y.
E.g. $fact \equiv \textbf{rec } f.(\lambda x.\textbf{if } x \textbf{ then } 1 \textbf{ else } x \times f(x-1))$.

11.1 Typing rules

$x : \tau \quad \text{if } \mathbf{type}(x) = \tau \qquad n : \mathbf{int}$

$$\frac{t_1 : \mathbf{int} \quad t_2 : \mathbf{int}}{t_1 \ \mathbf{op}\ t_2 : \mathbf{int}} \text{ where } \mathbf{op} \text{ is } +, - \text{ or } \times$$

$$\frac{t_0 : \mathbf{int} \quad t_1 : \tau \quad t_2 : \tau}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 : \tau}$$

$$\frac{t_1 : \tau_1 \quad t_2 : \tau_2}{(t_1, t_2) : \tau_1 * \tau_2} \qquad \frac{t : \tau_1 * \tau_2}{\mathbf{fst}(t) : \tau_1} \qquad \frac{t : \tau_1 * \tau_2}{\mathbf{Snd}(t) : \tau_2}$$

$$\frac{x : \tau_1 \quad t : \tau_2}{\lambda x.t : \tau_1 \to \tau_2} \qquad \frac{t_1 : \tau_1 \to \tau_2 \quad t_2 : \tau_1}{(t_1\ t_2) : \tau_2}$$

$$\frac{x : \tau_1 \quad t_1 : \tau_1 \quad t_2 : \tau_2}{\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2 : \tau_2} \qquad \frac{y : \tau \quad \lambda x.t : \tau}{\mathbf{rec}\ y.(\lambda x.t) : \tau}$$

11.1 Free variables

$FV(t)$ of a term t is defined by structural induction on t.

$$\begin{array}{rcl} FV(n) & = & \emptyset \\ FV(x) & = & \{x\} \\ & \vdots & \\ FV(\lambda x.t) & = & FV(t)\backslash\{x\} \\ FV(\ \mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2) & = & FV(t_1) \cup (FV(t_2)\backslash\{x\}) \\ FV(\ \mathbf{rec}\ y.(\lambda x.t)) & = & FV(\lambda x.t)\backslash\{y\} \end{array}$$

A term is closed iff $FV(t) = \emptyset$.

11.2 Eager operational semantics

Canonical forms of a type represent the values of the type.

- Ground type: numerals are canonical forms, i.e. $n \in C^e_{\mathbf{int}}$.
- Product type: if $c_1 \in C^e_{\tau_1}$ & $c_2 \in C^e_{\tau_2}$ then $(c_1, c_2) \in C^e_{\tau_1 * \tau_2}$.
- Function type: $\lambda x.t \in C^e_{\tau_1 \to \tau_2}$ if $\lambda x.t : \tau_1 \to \tau_2$ and $\lambda x.t$ is closed.

NB: Canonical forms are special kinds of closed terms.

11.2 Evaluation rules

$$c \rightarrow^e c \quad \text{where } c \in C^e_\tau$$

$$\frac{t_1 \rightarrow^e n_1 \quad t_2 \rightarrow^e n_2}{(t_1 \ \mathbf{op}\ t_2) \rightarrow^e n_1 \ op \ n_2} \text{ where } \mathbf{op} \text{ is } +, -, \times$$

$$\frac{t_0 \rightarrow^e 0 \quad t_1 \rightarrow^e c_1}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \rightarrow^e c_1} \qquad \frac{t_0 \rightarrow^e n \quad t_2 \rightarrow^e c_2 \quad n \not\equiv 0}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \rightarrow^e c_2}$$

$$\frac{t_1 \rightarrow^e c_1 \quad t_2 \rightarrow^e c_2}{(t_1, t_2) \rightarrow^e (c_1, c_2)} \qquad \frac{t \rightarrow^e (c_1, c_2)}{\mathbf{fst}(t) \rightarrow^e c_1} \qquad \frac{t \rightarrow^e (c_1, c_2)}{\mathbf{Snd}(t) \rightarrow^e c_2}$$

$$\frac{t_1 \rightarrow^e \lambda x.t_1' \quad t_2 \rightarrow^e c_2 \quad t_1'[c_2/x] \rightarrow^e c}{(t_1\ t_2) \rightarrow^e c}$$

$$\frac{t_1 \rightarrow^e c_1 \quad t_2[c_1/x] \rightarrow^e c_2}{\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2 \rightarrow^e c_2} \qquad \mathbf{rec}\ y.(\lambda x.t) \rightarrow^e \lambda x.(t[\ \mathbf{rec}\ y.(\lambda x.t)/y])$$

11.2 Eager operational semantics

Evaluation is deterministic and respects types.

Proposition 0.55 If $t \to^e c$ and $t \to^e c'$ then $c \equiv c'$.

If $t \to^e c$ and $t : \tau$ then $c : \tau$.

11.3 Eager denotational semantics

Guiding idea: denote t as an element of $(V^e_\tau)_\perp$ where V^e_τ is a cpo of *values* of type τ.

$$
\begin{array}{rcl}
V^e_{\mathbf{int}} & = & \mathbb{N} \\
V^e_{\tau_1 * \tau_2} & = & V^e_{\tau_1} \times V^e_{\tau_2} \\
V^e_{\tau_1 \to \tau_2} & = & [V^e_{\tau_1} \to (V^e_{\tau_2})_\perp]
\end{array}
$$

An environment is a function $\rho : \mathbf{Var} \to \bigcup\{V^e_\tau \mid t \text{ a type}\}$ which respects types: $x : \tau \Rightarrow \rho(x) \in V^e_\tau$ for any $x \in \mathbf{Var}$ and type τ.

11.3 Eager denotational semantics

$$
\begin{array}{rcl}
[\![n]\!]^e & = & \lambda\rho.\lfloor n\rfloor \\
[\![x]\!]^e & = & \lambda\rho.\lfloor \rho(x)\rfloor \\
[\![t_1 \ \mathbf{op}\ t_2]\!]^e & = & \lambda\rho.([\![t_1]\!]^e\rho \ op_\perp \ [\![t_2]\!]^e\rho) \quad \text{where } \ \mathbf{op} \text{ is } +, -, \times \\
[\![\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]\!]^e & = & \lambda\rho.Cond([\![t_0]\!]^e\rho, [\![t_1]\!]^e\rho, [\![t_2]\!]^e\rho) \\
[\![(t_1, t_2)]\!]^e & = & \lambda\rho.\ let\ \ v_1 \Leftarrow [\![t_1]\!]^e\rho, v_2 \Leftarrow [\![t_2]\!]^e\rho.\ \lfloor (v_1, v_2)\rfloor \\
[\![\mathbf{fst}(t)]\!]^e & = & \lambda\rho.\ let\ \ v \Leftarrow [\![t]\!]^e\rho.\ \lfloor \pi_1(v)\rfloor \\
[\![\mathbf{Snd}(t)]\!]^e & = & \lambda\rho.\ let\ \ v \Leftarrow [\![t]\!]^e\rho.\ \lfloor \pi_2(v)\rfloor \\
[\![\lambda x.t]\!]^e & = & \lambda\rho.\ \lfloor \lambda v \in V^e_{\tau_1}.\ [\![t]\!]^e\rho[v/x]\rfloor \qquad \text{where } \lambda x.t : \tau_1 \to \\
[\![(t_1\ t_2)]\!]^e & = & \lambda\rho.\ let\ \varphi \Leftarrow [\![t_1]\!]^e\rho,\ v \Leftarrow [\![t_2]\!]^e\rho.\ \varphi(v) \\
[\![\ \mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2]\!]^e & = & \lambda\rho.\ let\ v \Leftarrow [\![t_1]\!]^e\rho.\ [\![t_2]\!]^e\rho[v/x] \\
[\![\ \mathbf{rec}\ y.(\lambda x.t)]\!]^e & = & \lambda\rho.\lfloor \mu\varphi.(\lambda v.[\![t]\!]^e\rho[v/x, \varphi/y])\rfloor
\end{array}
$$

11.3 Eager denotational semantics

The function $Cond : \mathbb{N}_\perp \times D \times D \to D$ satisfies

$$Cond(z_0, z_1, z_2) = \begin{cases} z_1 & \text{if } z_0 = \lfloor 0 \rfloor, \\ z_2 & \text{if } z_0 = \lfloor n \rfloor \text{ for some } n \in \mathbb{N} \text{ with } n \neq 0, \\ \perp & \text{otherwise} \end{cases}$$

Lemma 0.56 Let t be a typable term. Let ρ, ρ' be environments which agree on the free variables of t. Then $[\![t]\!]^e \rho = [\![t]\!]^e \rho'$.

Proof: By structural induction. □

11.3 Eager denotational semantics

Lemma 0.57 [Substitution Lemma] Let s be a closed term with $s : \tau$ and $[\![s]\!]^e\rho = \lfloor v \rfloor$. Let x be a variable with $x : \tau$. Assume $t : \tau'$. Then $t[s/x] : \tau'$ and $[\![t[s/x]]\!]^e\rho = [\![t]\!]^e\rho[v/x]$.

Proof: By structural induction. $\square$

Lemma 0.58

1. If $t : \tau$ then $[\![t]\!]^e\rho \in (V_\tau^e)_\perp$, for any ρ.
2. If $c \in C_\tau^e$ then $[\![c]\!]^e\rho \neq \perp$, the bottom element of $(V_\tau^e)_\perp$, for any ρ

Proof: By structural induction. $\square$

11.4 Agreement of eager semantics

Lemma 0.59 If $t \to^e c$ then $[\![t]\!]^e\rho = [\![c]\!]^e\rho$, for any environment ρ.

Proof: By rule induction on the rules for evaluation. E.g., consider the rule $\dfrac{t_1 \to^e \lambda x.t_1' \quad t_2 \to^e c_2 \qquad t_1'[c_2/x] \to^e c}{(t_1\ t_2) \to^e c}$. Assume $[\![t_1]\!]^e\rho = [\![\lambda x.t_1']\!]^e\rho$, $[\![t_2]\!]^e\rho = [\![c_2]\!]^e\rho$ and $[\![t_1'[c_2/x]]\!]^e\rho = [\![c]\!]^e\rho$. Then

$$\begin{array}{rcl}
[\![t_1\ t_2]\!]^e\rho & = & \mathit{let}\ \varphi \Leftarrow [\![t_1]\!]^e\rho,\ v \Leftarrow [\![t_2]\!]^e\rho.\varphi(v) \\
& = & \mathit{let}\ \varphi \Leftarrow [\![\lambda x.t_1']\!]^e\rho,\ v \Leftarrow [\![c_2]\!]^e\rho.\varphi(v) \\
& = & \mathit{let}\ \varphi \Leftarrow \lfloor \lambda v.[\![t_1']\!]^e\rho[v/x] \rfloor,\ v \Leftarrow [\![c_2]\!]^e\rho.\varphi(v) \\
& = & [\![t_1']\!]^e\rho[v/x] \text{ where } [\![c_2]\!]^e\rho = \lfloor v \rfloor \\
& = & [\![t_1'[c_2/x]]\!]^e\rho \text{ by the substitution lemma} \\
& = & [\![c]\!]^e\rho
\end{array}$$

□

11.4 Convergence

- Operational convergence: $t \downarrow^e$ iff $t \rightarrow^e c$ for some canonical form c.
- Denotational convergence: $t \Downarrow^e$ iff $\exists v \in V^e_\tau . [\![t]\!]^e \rho = \lfloor v \rfloor$.

  It follows from Lemma 0.59 that $t \downarrow^e$ implies $t \Downarrow^e$. But the converse implication is more difficult.

11.4 Convergence

A tentative proof of $t \Downarrow^e \quad \Rightarrow \quad t \downarrow^e$ would be by structural induction.

Consider the critical case $t \equiv (t_1\ t_2)$. Assume $t_1 \Downarrow^e \quad \Rightarrow \quad t_1 \downarrow^e$ and $t_2 \Downarrow^e \quad \Rightarrow \quad t_2 \downarrow^e$. Suppose $t \Downarrow^e$. Because $[\![t]\!]^e\rho = let\ \varphi \Leftarrow [\![t_1]\!]^e\rho,\ v \Leftarrow [\![t_2]\!]^e\rho.\ \varphi(v)$, this ensures $t_1 \Downarrow^e$ and $t_2 \Downarrow^e$, and so by induction $t_1 \rightarrow^e \lambda x.t_1'$ and $t_2 \rightarrow^e c_2$ for some canonical forms. Thus $[\![t]\!]^e\rho = \varphi(v)$ where $\varphi = [\![t_1]\!]^e\rho = \lambda u.[\![t_1']\!]^e\rho[u/x]$ and $\lfloor v \rfloor = [\![c_2]\!]^e\rho$. Hence, $[\![t]\!]^e\rho = [\![t_1']\!]^e\rho[v/x] = [\![t_1'[c_2/x]]\!]^e\rho$ by the substitution lemma. Since $t \Downarrow^e$ we have $t_1'[c_2/x] \Downarrow^e$. Now we'd like to conclude $t_1'[c_2/x] \downarrow^e$ so $t_1'[c_2/x] \rightarrow^e c$ and from the operational semantics that $t \rightarrow^e c$. But we can't use the structural induction hypothesis here as $t_1'[c_2/x]$ is not structurally smaller than t.

11.4 Logical relations

Define a relation $\lesssim^\circ \subseteq V^e_\tau \times C^e_\tau$ on types τ, and then extend it to a relation between element d of $(V^e_\tau)_\perp$ and closed term t by letting

$$d \lesssim_\tau t \;\text{ iff }\; \forall v \in V^e_\tau .\, d = \lfloor v \rfloor \;\Rightarrow\; \exists c.\, t \to^e c \;\&\; v \lesssim^\circ_\tau c$$

The relations $\lesssim^\circ_\tau$ are defined by structural induction on types τ:

- Ground type: $n \lesssim^\circ_{\mathbf{int}} n$, for all numbers n.
- Product types: $(v_1, v_2) \lesssim^\circ_{\tau_1 * \tau_2} (c_1, c_2)$ iff $v_1 \lesssim^\circ_{\tau_1} c_1 \;\&\; v_2 \lesssim^\circ_{\tau_2} c_2$.
- Function types: $\varphi \lesssim^\circ_{\tau_1 \to \tau_2} \lambda x.t$ iff
  $\forall v \in V^e_{\tau_1}, c \in C^e_{\tau_1} .\, v \lesssim^\circ_{\tau_1} c \;\Rightarrow\; \varphi(v) \lesssim_{\tau_2} t[c/x]$.

Lemma 0.60 Let $t : \tau$. Then

1. $\perp_{(V^e_\tau)_\perp} \lesssim_\tau t$
2. If $d \sqsubseteq d'$ and $d' \lesssim_\tau t$ then $d \lesssim_\tau t$.
3. If $d_0 \sqsubseteq d_1 \sqsubseteq ... \sqsubseteq d_n \sqsubseteq ...$ is an ω-chain in $(V^e_\tau)_\perp$ such that $d_n \lesssim_\tau t$ for all $n \in \omega$ then $\bigsqcup_{n\in\omega} d_n \lesssim_\tau t$.

Proof: Property 1 follows by definition. Properties 2 and 3 are shown by structural induction on types. For ground type **int** they certainly hold. Consider a function type. Let $d_0 \sqsubseteq d_1 \sqsubseteq ...$ be an ω-chain in $(V^e_{\tau_1\to\tau_2})_\perp$ with $d_n \lesssim_{\tau_1\to\tau_2} t$ for all n. Either $d_n = \perp$ for all $n \in \omega$ (easy case) or for some n and all $m \geq n$ we have $d_m = \lfloor\varphi_m\rfloor$, $t \to^e \lambda x.t'$ and $\varphi_m \lesssim^\circ_{\tau_1\to\tau_2} \lambda x.t'$. Assuming $v \lesssim^e_{\tau_1} c$ we obtain $\varphi_m(v) \lesssim_{\tau_2} t'[c/x]$ for $m \geq n$. By induction, $\bigsqcup_m(\varphi_m(v)) \lesssim_{\tau_2} t'[c/x]$, and so $(\bigsqcup_m \varphi_m)(v) \lesssim_{\tau_2} t'[c/x]$ whenever $v \lesssim^e_{\tau_1} c$. In other words $\bigsqcup_m \varphi_m \lesssim^\circ_{\tau_1\to\tau_2} \lambda x.t'$ whence $\bigsqcup_m d_m = \lfloor\bigsqcup_m \varphi_m\rfloor \lesssim_{\tau_1\to\tau_2} t$. $\square$

11.4 Agreement of eager semantics

Lemma 0.61 Let t be a typable closed term. Then $t \Downarrow^e$ implies $t \downarrow^e$.

Proof: Show by structural induction on terms that for terms $t : \tau$ with free variables $x_1 : \tau_1, ..., x_k : \tau_k$ that if $\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1, ..., \lfloor v_k \rfloor \lesssim_{\tau_1} s_k$ then

$$[\![t]\!]^e \rho[v_1/x_1, ..., v_k/x_k] \lesssim_\tau t[s_1/x_1, ..., s_k/x_k].$$

cf. pages 195-200. $\square$

Corollary 0.62 If t is a closed term with $t : \mathbf{int}$. Then

$$t \to^e n \text{ iff } [\![t]\!]^e \rho = \lfloor n \rfloor$$

for any $n \in \mathbf{int}$.

11.5 A lazy language

The syntax is the same as that for the early language except for recursion.

$$\textbf{rec } x.t$$

The typing rule $\dfrac{x : t \quad t : \tau}{\textbf{rec } x.t \ : \tau}$

Free variables $FV(\textbf{ rec } x.t) = FV(t) \backslash \{x\}$

11.6 Lazy operational semantics

Lazy canonical forms C^l_τ:

- Ground type: $n \in C^l_{\mathbf{int}}$.
- Product type: $(t_1, t_2) \in C^l_{\tau_1 * \tau_2}$ if $t_1 : \tau_1$ & $t_2 : \tau_2$ with t_1 and t_2 closed.
- Function type: $\lambda x.t \in C^l_{\tau_1 \to \tau_2}$ if $\lambda x.t : \tau_1 \to \tau_2$ and $\lambda x.t$ is closed.

11.6 Evaluation rules

$$c \to^l c \quad \text{where } c \in C^l_\tau$$

$$\frac{t_1 \to^l n_1 \quad t_2 \to^l n_2}{(t_1 \ \mathbf{op} \ t_2) \to^l n_1 \ op \ n_2} \ \text{where } \ \mathbf{op} \ \text{ is } +, -, \times$$

$$\frac{t_0 \to^l 0 \quad t_1 \to^l c_1}{\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \to^l c_1} \qquad \frac{t_0 \to^l n \quad t_1 \to^l c_2 \quad n \not\equiv 0}{\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \to^l c_2}$$

$$\frac{t \to^l (t_1, t_2) \quad t_1 \to^l c_1}{\mathbf{fst}(t) \to^l c_1} \qquad \frac{t \to^l (t_1, t_2) \quad t_2 \to^l c_2}{\mathbf{Snd}(t) \to^l c_2}$$

$$\frac{t_1 \to^l \lambda x.t_1' \quad t_1'[c_2/x] \to^l c}{(t_1 \ t_2) \to^l c}$$

$$\frac{t_2[t_1/x] \to^l c}{\mathbf{let} \ x \Leftarrow t_1 \ \mathbf{in} \ t_2 \to^l c} \qquad \frac{t[\ \mathbf{rec} \ x.t/x] \to^l c}{\mathbf{rec} \ x.t \to^l c}$$

11.6 Lazy operational semantics

Evaluation is deterministic and respects types.

Proposition 0.63 If $t \rightarrow^l c$ and $t \rightarrow^l c'$ then $c \equiv c'$.

If $t \rightarrow^l c$ and $t : \tau$ then $c : \tau$.

11.7 Lazy denotational semantics

Guiding idea: denote t as an element of $(V^l_\tau)_\perp$ where V^l_τ is a cpo of *values* of type τ.

$$
\begin{array}{rcl}
V^l_{\mathbf{int}} & = & \mathbb{N} \\
V^l_{\tau_1 * \tau_2} & = & (V^l_{\tau_1})_\perp \times (V^l_{\tau_2})_\perp \\
V^l_{\tau_1 \to \tau_2} & = & [(V^l_{\tau_1})_\perp \to (V^l_{\tau_2})_\perp]
\end{array}
$$

An environment is a function $\rho : \mathbf{Var} \to \bigcup\{(V^l_\tau)_\perp \mid t \text{ a type}\}$ which respects types: $x : \tau \Rightarrow \rho(x) \in V^l_\tau$ for any $x \in \mathbf{Var}$ and type τ.

11.7 Lazy denotational semantics

$$
\begin{array}{rcl}
[\![n]\!]^l & = & \lambda\rho.\lfloor n \rfloor \\
[\![x]\!]^l & = & \lambda\rho.\lfloor \rho(x) \rfloor \\
[\![t_1 \ \mathbf{op} \ t_2]\!]^l & = & \lambda\rho.([\![t_1]\!]^l \rho op_\perp [\![t_2]\!]^l \rho) \quad \text{where } \ \mathbf{op} \text{ is } +, -, \times \\
[\![\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2]\!]^l & = & \lambda\rho.Cond([\![t_0]\!]^l \rho, [\![t_1]\!]^l \rho, [\![t_2]\!]^l \rho) \\
[\![(t_1, t_2)]\!]^l & = & \lambda\rho. \lfloor ([\![t_1]\!]^l \rho, [\![t_2]\!]^l \rho) \rfloor \\
[\![\mathbf{fst}(t)]\!]^l & = & \lambda\rho. \, let \ \ v \Leftarrow [\![t]\!]^l \rho. \lfloor \pi_1(v) \rfloor \\
[\![\mathbf{Snd}(t)]\!]^l & = & \lambda\rho. \, let \ \ v \Leftarrow [\![t]\!]^l \rho. \lfloor \pi_2(v) \rfloor \\
[\![\lambda x.t]\!]^l & = & \lambda\rho. \lfloor \lambda v \in (V^l_{\tau_1})_\perp . [\![t]\!]^l \rho[v/x] \rfloor \qquad \text{where } \lambda x.t : \tau_1 \\
[\![(t_1 \ t_2)]\!]^l & = & \lambda\rho. \, let \ \varphi \Leftarrow [\![t_1]\!]^l \rho. \ \varphi([\![t_2]\!]^l \rho) \\
[\![\ \mathbf{let} \ x \Leftarrow t_1 \ \mathbf{in} \ t_2]\!]^l & = & \lambda\rho. [\![t_2]\!]^l \rho[[\![t_1]\!]^l \rho/x] \\
[\![\ \mathbf{rec} \ x.t]\!]^l & = & \lambda\rho.(\mu v.[\![t]\!]^l \rho[v/x])
\end{array}
$$

11.7 Lazy denotational semantics

The function $Cond : \mathbb{N}_\perp \times D \times D \to D$ satisfies

$$Cond(z_0, z_1, z_2) = \begin{cases} z_1 & \text{if } z_0 = \lfloor 0 \rfloor, \\ z_2 & \text{if } z_0 = \lfloor n \rfloor \text{ for some } n \in \mathbb{N} \text{ with } n \neq 0, \\ \perp & \text{otherwise} \end{cases}$$

Lemma 0.64 Let t be a typable term. Let ρ, ρ' be environments which agree on $FV(t)$. Then $[\![t]\!]^l \rho = [\![t]\!]^l \rho'$.

Proof: By structural induction. □

11.7 Lazy denotational semantics

Lemma 0.65 [Substitution Lemma] Let s be a closed term with $s : \tau$. Let x be a variable with $x : \tau$. Assume $t : \tau'$. Then $t[s/x] : \tau'$ and $[\![t[s/x]]\!]^l\rho = [\![t]\!]^l\rho[[\![s]\!]^l\rho/x]$.

Proof: By structural induction. $\square$

Lemma 0.66

1. If $t : \tau$ then $[\![t]\!]^l\rho \in (V^l_\tau)_\perp$, for any ρ.
2. If $c \in C^l_\tau$ then $[\![c]\!]^l\rho \neq \perp$, the bottom element of $(V^l_\tau)_\perp$, for any ρ.

Proof: By structural induction. $\square$

11.8 Agreement of lazy semantics

Lemma 0.67 If $t \to^l c$ then $[\![t]\!]^l\rho = [\![c]\!]^l\rho$, for any environment ρ.

Proof: By rule induction on the rules for evaluation. E.g., consider the rule $\dfrac{t_1 \to^l \lambda x.t_1' \quad t_1'[t_2/x] \to^l c}{(t_1\ t_2) \to^l c}$. Assume $[\![t_1]\!]^l\rho = [\![\lambda x.t_1']\!]^l\rho$ and $[\![t_1'[t_2/x]]\!]^l\rho = [\![c]\!]^l\rho$. Then

$$
\begin{aligned}
[\![t_1\ t_2]\!]^l\rho &= \mathit{let}\ \varphi \Leftarrow [\![t_1]\!]^l\rho.\varphi([\![t_2]\!]^l\rho) \\
&= \mathit{let}\ \varphi \Leftarrow [\![\lambda x.t_1']\!]^l\rho.\varphi([\![t_2]\!]^l\rho) \\
&= \mathit{let}\ \varphi \Leftarrow \lfloor \lambda v.[\![t_1']\!]^l\rho[v/x]\rfloor.\varphi([\![t_2]\!]^l\rho) \\
&= [\![t_1']\!]^l\rho[[\![t_2]\!]^l\rho/x] \\
&= [\![t_1'[t_2/x]]\!]^e\rho \text{ by the substitution lemma} \\
&= [\![c]\!]^e\rho
\end{aligned}
$$

$\square$

11.8 Convergence

- Operational convergence: $t \downarrow^l$ iff $t \rightarrow^l c$ for some canonical form c.
- Denotational convergence: $t \Downarrow^l$ iff $\exists v \in V_\tau^l . [\![t]\!]^l \rho = \lfloor v \rfloor$.

  It follows from Lemma 0.67 that $t \downarrow^l$ implies $t \Downarrow^l$. But the converse implication is more difficult.

11.8 Logical relations

Define a relation $\lesssim^\circ \subseteq V^l_\tau \times C^l_\tau$ on types τ, and then extend it to a relation between element d of $(V^l_\tau)_\perp$ and closed term t by letting

$$d \lesssim_\tau t \ \text{ iff } \ \forall v \in V^l_\tau.\, d = \lfloor v \rfloor \ \Rightarrow \ \exists c.\, t \to^l c \;\&\; v \lesssim^\circ_\tau c$$

The relations $\lesssim^\circ_\tau$ are defined by structural induction on types τ:

- Ground type: $n \lesssim^\circ_{\mathbf{int}} n$, for all numbers n.
- Product types: $(v_1, v_2) \lesssim^\circ_{\tau_1 * \tau_2} (t_1, t_2)$ iff $v_1 \lesssim_{\tau_1} t_1 \;\&\; v_2 \lesssim_{\tau_2} t_2$.
- Function types: $\varphi \lesssim^\circ_{\tau_1 \to \tau_2} \lambda x.t$ iff $\forall v \in (V^l_{\tau_1})_\perp$, closed $u : \tau_1.\ v \lesssim_{\tau_1} u \ \Rightarrow \ \varphi(v) \lesssim_{\tau_2} t[u/x]$.

11.8 Logical relations

Lemma 0.68 Let $t : \tau$. Then

1. $\bot_{(V^l_\tau)_\bot} \lesssim_\tau t$
2. If $d \sqsubseteq d'$ and $d' \lesssim_\tau t$ then $d \lesssim_\tau t$.
3. If $d_0 \sqsubseteq d_1 \sqsubseteq ... \sqsubseteq d_n \sqsubseteq ...$ is an ω-chain in $(V^l_\tau)_\bot$ such that $d_n \lesssim_\tau t$ for all $n \in \omega$ then $\bigsqcup_{n\in\omega} d_n \lesssim_\tau t$.

Proof: Similar to the proof of Lemma 0.60. Property 1 follows by definition. Properties 2 and 3 are shown by structural induction on types.
$\square$

11.8 Agreement of lazy semantics

Lemma 0.69 Let t be a typable closed term. Then $t \Downarrow^l$ implies $t \downarrow^l$.

Proof: Similar to the proof of Lemma 0.61. Show by structural induction on terms that for terms $t : \tau$ with free variables $x_1 : \tau_1, ..., x_k : \tau_k$ that if $\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1, ..., \lfloor v_k \rfloor \lesssim_{\tau_1} s_k$ then

$$[\![t]\!]^l \rho[v_1/x_1, ..., v_k/x_k] \lesssim_\tau t[s_1/x_1, ..., s_k/x_k].$$

cf. pages 206-209. $\square$

Corollary 0.70 If t is a closed term with $t : \mathbf{int}$. Then

$$t \to^l n \text{ iff } [\![t]\!]^l \rho = \lfloor n \rfloor$$

for any $n \in \mathbf{int}$.

11.9 Fixed-point operators

Let $R^l \equiv$ **rec** $Y.(\lambda f.(f(Y\,f)))$ then $[\![R^l(\lambda x.t)]\!]^l\rho = [\![$ **rec** $x.t]\!]^l\rho$. However, $[\![R^l(\lambda x.t)]\!]^e\rho = \perp$.

Let $R^e \equiv$ **rec** $Y.\ (\lambda f.\lambda x.((f(Y\,f))x))$. Then $[\![R^e(\lambda y.\lambda x.t)]\!]^e\rho = [\![$ **rec** $y.(\lambda x.t)]\!]^e\rho$.

11.10 Observations

The operational and denotational semantics agree on the "observations of interest", which expresses the adequacy of the denotational with respect to the operational semantics.

The adequacy wrt convergence will ensure that the two semantics also agree on how terms of type **int** evaluate. Consider the context $C \equiv \textbf{if}\ \_\ \textbf{then}\ 0\ \textbf{else}\ \Omega$, where $\Omega : \tau$ is a closed term which diverges. Then for both the eager and lazy semantics,

$$\begin{array}{rcll} t \to n & \Leftrightarrow & C[t] \downarrow & \\ & \Leftrightarrow & C[t] \Downarrow & \text{by adequacy} \\ & \Leftrightarrow & [\![t]\!]\rho = n. & \end{array}$$

11.10 Full abstraction

Suppose the observations of interest concern just the convergence behaviour of terms, then

$$t_1 \sim t_2 \text{ iff } (C[t_1]\downarrow \Leftrightarrow C[t_2]\downarrow)$$

for all contexts $C[\ ]$ for which $C[t_1], C[t_2]$ are closed and typable. A denotational semantics is fully abstract wrt the observations, if

$$[\![t_1]\!] = [\![t_2]\!] \text{ iff } t_1 \sim t_2$$

The "only if" direction follows provided the denotational semantics is adequate, the "if" direction is hard because in our cpo's of denotations there are elements like parallel or which cannot be defined by terms. *por* is a continuous function on $\mathbf{T}_\perp$ extending the usual disjunction with the property that $por(\mathbf{true}, \perp) = por(\perp, \mathbf{true}) = \mathbf{true}$.

11.11 Sums

Extend our language with the constructions:

$\mathbf{inl}(t)$; $\mathbf{inr}(t)$; $\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1,\ \mathbf{inr}(x_2).t_2$.

Free variables

$$FV(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1,\ \mathbf{inr}(x_2).t_2) = FV(t) \cup (FV(t_1) \setminus \{x_1\}) \cup (FV(t_2) \setminus \{x_2\})$$

Typing rules

$$\frac{t : \tau_1}{\mathbf{inl}(t) : \tau_1 + \tau_2} \qquad \frac{t : \tau_2}{\mathbf{inr}(t) : \tau_1 + \tau_2}$$

$$\frac{t : \tau_1 + \tau_2 \quad x_1 : \tau_1 \quad x_2 : \tau_2 \quad t_1 : \tau \quad t_2 : \tau}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1,\ \mathbf{inr}(x_2).t_2 : \tau}$$

11.11 Sums in eager semantics

Adding two canonical forms

$$\mathbf{inl}(c) \in C^e_{\tau_1+\tau_2} \text{ if } c \in C^e_{\tau_1}, \qquad \mathbf{inr}(c) \in C^e_{\tau_1+\tau_2} \text{ if } c \in C^e_{\tau_2}$$

The operational rules:

$$\frac{t \to^e \mathbf{inl}(c_1) \quad t_1[c_1/x_1] \to^e c}{(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1, \mathbf{inr}(x_2).t_2) \to^e c} \qquad \frac{t \to^e \mathbf{inr}(c_2) \quad t_2[c_2/x_2] \to^e c}{(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1, \mathbf{inr}(x_2).t_2)}$$

For denotational semantics, the cpo of values of a sum type:
$V^e_{\tau_1+\tau_2} = V^e_{\tau_1} + V^e_{\tau_2}$.

11.11 Sums in lazy semantics

Adding two canonical forms

$$\mathbf{inl}(t) \in C^l_{\tau_1+\tau_2} \text{ if } t : \tau_1 \text{ and } t \text{ is closed}$$
$$\mathbf{inr}(t) \in C^l_{\tau_1+\tau_2} \text{ if } t : \tau_2 \text{ and } t \text{ is closed}$$

The operational rules:

$$\frac{t \to^l \mathbf{inl}(t') \quad t_1[t'/x_1] \to^l c}{(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1, \mathbf{inr}(x_2).t_2) \to^l c} \qquad \frac{t \to^l \mathbf{inr}(t') \quad t_2[t'/x_2] \to^l c}{(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1, \mathbf{inr}(x_2).t_2)}$$

For denotational semantics, the cpo of values of a sum type:
$V^l_{\tau_1+\tau_2} = (V^l_{\tau_1})_\perp + (V^l_{\tau_2})_\perp$.

PCF

The syntax of pure PCF

The syntax of pure PCF, without extension by syntactic sugar, is summarized below by a BNF-like grammar. The first set of productions describe the expressions of an arbitrary type σ. These include variables, conditional expressions, and the results of function application, projection functions, and fixed-point application.

$$
\begin{array}{lcl}
\langle \sigma\_exp \rangle & ::= & \langle \sigma\_var \rangle \mid \texttt{if } \langle bool\_exp \rangle \texttt{ then } \langle \sigma\_exp \rangle \texttt{ else } \langle \sigma\_exp \rangle \mid \\
 & & \langle \sigma\_application \rangle \mid \langle \sigma\_projection \rangle \mid \langle \sigma\_fixed\_point \rangle \\
\langle \sigma\_application \rangle & ::= & \langle \tau \to \sigma\_exp \rangle \langle \tau\_exp \rangle \\
\langle \sigma\_projection \rangle & ::= & \mathbf{Proj}_1 \langle \sigma \times \tau\_exp \rangle \mid \mathbf{Proj}_2 \langle \tau \times \sigma\_exp \rangle \\
\langle \sigma\_fixed\_point \rangle & ::= & fix_\sigma \, \langle \sigma \to \sigma\_exp \rangle
\end{array}
$$

For function and product types, we also have lambda abstraction and explicit pairing.

$$
\begin{array}{lcl}
\langle \sigma \to \tau\_exp \rangle & ::= & \lambda x\colon \sigma . \langle \tau\_exp \rangle \\
\langle \sigma \times \tau\_exp \rangle & ::= & \langle\, \langle \sigma\_exp \rangle,\ \langle \tau\_exp \rangle \,\rangle
\end{array}
$$

The constants and functions for natural numbers and booleans are covered by the following productions.

$$
\begin{array}{lcl}
\langle bool\_exp \rangle & ::= & true \mid false \mid Eq?\, \langle nat\_exp \rangle \, \langle nat\_exp \rangle \\
\langle nat\_exp \rangle & ::= & 0 \mid 1 \mid 2 \mid \ldots \mid \langle nat\_exp \rangle + \langle nat\_exp \rangle
\end{array}
$$

Axiomatic semantics

Equational Proof System for PCF.

Axioms

Equality

(*ref*)	$M = M$

Types *nat* and *bool*

(*add*)	$0 + 0 = 0,\ 0 + 1 = 1, \ldots, 3 + 5 = 8, \ldots$
(*Eq?*)	$Eq?\, n\, n = true$, $Eq?\, n\, m = false$ (n, m distinct numerals)
(*cond*)	$\texttt{if}\ true\ \texttt{then}\ M\ \texttt{else}\ N = M$, $\texttt{if}\ false\ \texttt{then}\ M\ \texttt{else}\ N = N$

Pairs

(*proj*)	$\mathbf{Proj}_1 \langle M, N \rangle = M \quad \mathbf{Proj}_2 \langle M, N \rangle = N$
(*sp*)	$\langle \mathbf{Proj}_1 P, \mathbf{Proj}_2 P \rangle = P$

Binding

(α)	$\lambda x\!:\!\sigma.M = \lambda y\!:\!\sigma.[y/x]M$, provided y not free in M.

Functions

(β)	$(\lambda x\!:\!\sigma.M)N = [N/x]M$
(η)	$\lambda x\!:\!\sigma.Mx = M$, provided x not free in M

Recursion

(*fix*)	$fix_\sigma = \lambda f\!:\!\sigma \to \sigma.f\,(fix_\sigma\ f)$

Inference Rules

Equivalence

(*sym*), (*trans*)

$$\frac{M = N}{N = M} \qquad \frac{M = N,\ N = P}{M = P}$$

Congruence

Types *nat* and *bool*

$$\frac{M = N,\ P = Q}{M + P = N + Q} \qquad \frac{M = N,\ P = Q}{Eq?\ M\ P = Eq?\ N\ Q}$$

$$\frac{M_1 = M_2,\ N_1 = N_2,\ P_1 = P_2}{\texttt{if}\ M_1\ \texttt{then}\ N_1\ \texttt{else}\ P_1 = \texttt{if}\ M_2\ \texttt{then}\ N_2\ \texttt{else}\ P_2}$$

Pairs

$$\frac{M = N}{\mathbf{Proj}_i M = \mathbf{Proj}_i N} \qquad \frac{M = N,\ P = Q}{\langle M, P \rangle = \langle N, Q \rangle}$$

Functions

$$\frac{M = N}{\lambda x\!:\!\sigma.\ M = \lambda x\!:\!\sigma.\ N} \qquad \frac{M = N,\ P = Q}{M\,P = N\,Q}$$

Operational semantics

Reduction axioms for PCF.

Types *nat* and *bool*	
(*add*)	$0+0 \to 0, 0+1 \to 1, \ldots, 3+5 \to 8, \ldots$
(*Eq?*)	$Eq?\, n\, n \to true$, $Eq?\, n\, m \to false$ (n, m distinct numerals)
(*cond*)	$\texttt{if } true \texttt{ then } M \texttt{ else } N \to M$, $\texttt{if } false \texttt{ then } M \texttt{ else } N \to M$
Pairs ($\sigma \times \tau$)	
(*proj*)	$\mathbf{Proj}_1\langle M, N\rangle \to M \quad \mathbf{Proj}_2\langle M, N\rangle \to N$
Rename bound variables	
(α)	$\lambda x{:}\sigma.M = \lambda y{:}\sigma.[y/x]M$, provided y not free in M.
Functions ($\sigma \to \tau$)	
(β)	$(\lambda x{:}\sigma.M)N \to [N/x]M$
Recursion	
(*fix*)	$fix_\sigma \to \lambda f{:}\sigma \to \sigma.f(fix_\sigma\, f)$

$M =_{op} N$ if, for every context $C[\,]$ s.t. both $C[M]$ and $C[N]$ are programs, we have $eval(C[M]) \simeq eval(C[N])$. Here $eval$ is an evaluation partial function with $eval(M) = N$ iff M may be reduced to normal form N.

Denotational semantics

An environment ρ is a mapping from variables to $\bigcup_\sigma V^\sigma$ with $\rho(x) \in V^\sigma$ if $x : \sigma$.

- A type σ is denoted as a cpo V^σ, with $\mathbb{N}_\perp$ and $\mathbf{T}_\perp$ as bases and $V^{\sigma\times\tau} = V^\sigma \times V^\tau$ and $V^{\sigma\to\tau} = [V^\sigma \to V^\tau]$.
- Constants $0, 1, 2, ...$ and $\mathbf{true}, \mathbf{false}$ are interpreted as the standard natural number and boolean elements of $\mathbb{N}_\perp$ and $\mathbf{T}_\perp$.
- $+$ and $Eq?$ are interpreted as the lifted versions, $+_\perp$ and $Eq?_\perp$, of the standard functions that are strict in both arguments. E.g. $\perp_\mathbb{N} \ +_\perp \ x \ = \ \perp_\mathbb{N}$ and $Eq?_\perp \perp_\mathbf{T} x \ = \ \perp_\mathbf{T}$.

Denotational semantics

$$
\begin{array}{rcl}
[\![c]\!]\rho & = & Const(c) \\
[\![x]\!]\rho & = & \rho(x) \\
[\![\textbf{if } P \textbf{ then } M \textbf{ else } N]\!]\rho & = & \begin{cases} [\![M]\!]\rho & \text{if } [\![P]\!]\rho = \textbf{true} \\ [\![N]\!]\rho & \text{if } [\![P]\!]\rho = \textbf{false} \\ \bot & \text{otherwise} \end{cases} \\
[\![MN]\!]\rho & = & apply([\![M]\!]\rho,\ [\![N]\!]\rho) \\
[\![\lambda x : \tau.\ M]\!]\rho & = & \lambda v \in V^{\tau}.\ [\![M]\!]\rho[v/x] \\
[\![\mathbf{Proj}_1 M]\!]\rho & = & \mathbf{Proj}_1 [\![M]\!]\rho \\
[\![\mathbf{Proj}_2 M]\!]\rho & = & \mathbf{Proj}_2 [\![M]\!]\rho \\
[\![\langle M, N\rangle]\!]\rho & = & \langle [\![M]\!]\rho, [\![N]\!]\rho\rangle \\
[\![fix_{\sigma}\ M]\!]\rho & = & \bigsqcup_{n\geq 0}([\![M]\!]\rho)^n(\bot_{\sigma})
\end{array}
$$

Soundness

Theorem 0.71 Let M and N be expressions of PCF over typed variables from Γ. If $\Gamma \triangleright M = N : \sigma$ is provable from the axioms for PCF, then the CPO model satisfies that equation.

Corollary 0.72 If $\Gamma \triangleright M : \sigma$ is well-typed term of PCF, and $M \twoheadrightarrow N$, then the CPO model satisfies the equation $\Gamma \triangleright M = N : \sigma$.

For PCF terms, $=_{ax} \quad \subseteq \quad =_{den} \quad \subseteq \quad =_{op}$ and
$(\forall \text{ programs } M)(\forall \text{ results } N)\ M =_{ax} N \text{ iff } M =_{den} N \text{ iff } M =_{op} N$

Full abstract

The extension of PCF with parallel-or, PCF+por, is obtained by adding the constant $por : \mathbf{T} \to \mathbf{T} \to \mathbf{T}$ with the following reduction axioms.

$$\begin{array}{rcl} por\ \mathbf{true}\ M & \to & \mathbf{true} \\ por\ M\ \mathbf{true} & \to & \mathbf{true} \\ por\ \mathbf{false}\ \mathbf{false} & \to & \mathbf{false} \end{array}$$

Theorem 0.73 For PCF+por, the relations $=_{den}$, determined by the CPO model and $=_{op}$, determined by the reduction system, are identical.

The proof of $=_{den} \subseteq =_{op}$ involves an approximation theorem, the other direction makes use of algebraic PCPOs.

Algebraic PCPO

An element x of a cpo P is compact if, for every directed set $X \subseteq P$ with $x \sqsubseteq \bigsqcup X$, we have $x \sqsubseteq x'$ for some $x' \in X$. Let $K(P)$ be the set of compact elements of P. The cpo P is algebraic if every $p \in P$ is the limit of its compact approximants, i.e. $p = \bigsqcup\{x \sqsubseteq p \mid x \in K(P)\}$. Two elements p, p' of a cpo are consistent if there is some $p'' \in P$ with $p, p' \sqsubseteq p''$. A subset $X \subseteq P$ is pairwise consistent if every pair of elements from X is consistent. A pcpo (pairwise-consistent complete cpo) is a partial order with the property that every subset that is either directed or pairwise consistent has a least upper bound.