$(f \times id) : D \times D \to D \times D$ $\quad (f \times id)(d_1, d_2) = (f(d_1), d_2)$

Example (II)

Let D be a domain and let $f, g : D \to D$ be continuous functions such that $f \circ g \sqsubseteq g \circ f$. Then,

$$f(\perp) \sqsubseteq g(\perp) \implies \mathit{fix}(f) \sqsubseteq \mathit{fix}(g) .$$

WRONG

$x \in S \iff fx \sqsubseteq x$

$$\frac{fx \sqsubseteq x}{gfx \sqsubseteq gx}$$

$$\frac{fgx \sqsubseteq gfx}{}$$

$$\frac{gx \in S \iff f(gx) \sqsubseteq gx}{x \in S \implies gx \in S}$$

Chain closed

$(f \times id)^{-1}(\sqsubseteq)$

Idea: $S = \{x \mid fx \sqsubseteq x\}$

$\perp \in S$?

$$\frac{f(\mathit{fix}\, g) \sqsubseteq \mathit{fix}\, g \iff \mathit{fix}\, g \in S}{\mathit{fix}(f) \sqsubseteq \mathit{fix}(g)}$$

$\cancel{f\perp \sqsubseteq \perp}$ $\quad \cancel{f(\perp) = \perp}$

Example (II)

Let D be a domain and let $f, g : D \to D$ be continuous functions such that $f \circ g \sqsubseteq g \circ f$. Then,

$$f(\perp) \sqsubseteq g(\perp) \implies \mathit{fix}(f) \sqsubseteq \mathit{fix}(g) \ .$$

$f \times g : D \times D \to D \times D : (d_1, d_2) \longmapsto (f(d_1), g(d_2))$

$S = \{x \mid fx \sqsubseteq gx\} = (f \times g)^{-1}(\sqsubseteq)$ — chain closed $\boxed{?}$ $f\perp \sqsubseteq g\perp$?✓

proceed by Scott Induction

$f(\mathit{fix}\, g) \sqsubseteq \mathit{fix}(g) = g(\mathit{fix}\, g) \Leftrightarrow \mathit{fix}\, g \in S$

$\mathit{fix}\, f \sqsubseteq \mathit{fix}\, g$

Will not work, so strengthen it

Example (II)

NB: $fix\,g : D \times D \to D \times D$

$fix(f \times g) = (fix\,f, fix\,g)$

Let D be a domain and let $f, g : D \to D$ be continuous functions such that $f \circ g \sqsubseteq g \circ f$. Then,

EXERCISE

$$f(\bot) \sqsubseteq g(\bot) \implies fix(f) \sqsubseteq fix(g)\ .$$

$\Leftrightarrow (f \times g)(x, y) \in S$

$fx \sqsubseteq gy \Rightarrow ffx \sqsubseteq ggy$

Try Scott Induction for $f \times g$

admissible

$S = \{ (x, y) \mid fx \sqsubseteq gy \}$

$fix\,h = h(fix\,h)$

$$\frac{f(fix\,f) \sqsubseteq g(fix\,g)}{fix\,f \sqsubseteq fix\,g}$$

Topic 5

PCF

PCF syntax

here omitted

Types

$$\tau ::= nat \mid bool \mid \tau \to \tau \quad \mid \tau * \tau$$

PCF syntax

Types

$$\tau ::= nat \mid bool \mid \tau \to \tau$$

Expressions

$$\begin{array}{rcl} M & ::= & \mathbf{0} \mid \mathbf{succ}(M) \mid \mathbf{pred}(M) \\ & \mid & \mathbf{true} \mid \mathbf{false} \mid \mathbf{zero}(M) \end{array}$$

Test for equality with 0

Types

$$\tau ::= nat \mid bool \mid \tau \to \tau$$

Expressions

application

recursive definitions

anonymous functions

$$\begin{array}{rcl} M & ::= & \mathbf{0} \mid \mathbf{succ}(M) \mid \mathbf{pred}(M) \\ & \mid & \mathbf{true} \mid \mathbf{false} \mid \mathbf{zero}(M) \\ & \mid & x \mid \mathbf{if}\ M\ \mathbf{then}\ M\ \mathbf{else}\ M \\ & \mid & \mathbf{fn}\, x : \tau \,.\, M \mid M\, M \mid \mathbf{fix}(M) \end{array}$$

where $x \in \mathbb{V}$, an infinite set of variables.

Technicality: We identify expressions up to α-conversion of bound variables (created by the $\mathbf{fn}$ expression-former): by definition a PCF term is an α-equivalence class of expressions.

PCF typing relation, $\Gamma \vdash M : \tau$

- Γ is a type environment, *i.e.* a finite partial function mapping variables to types (whose domain of definition is denoted $dom(\Gamma)$)
- M is a term
- τ is a type.

Notation:

$M : \tau$ means M is closed and $\emptyset \vdash M : \tau$ holds.

$\mathrm{PCF}_\tau \stackrel{\text{def}}{=} \{M \mid M : \tau\}$.

PCF typing relation (sample rules)

$$(:_{\mathrm{fn}}) \quad \frac{\Gamma[x \mapsto \tau] \vdash M : \tau'}{\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M : \tau \to \tau'} \quad \text{if } x \notin dom(\Gamma)$$

$$(:_{\mathrm{app}}) \quad \frac{\Gamma \vdash M_1 : \tau \to \tau' \quad \Gamma \vdash M_2 : \tau}{\Gamma \vdash M_1\, M_2 : \tau'}$$

IDEA!

body of a recursive definition

$$(:_{\mathrm{fix}}) \quad \frac{\Gamma \vdash M : \tau \to \tau}{\Gamma \vdash \mathbf{fix}(M) : \tau}$$

the recursive def.

Partial recursive functions in PCF

- Primitive recursion. $h = \rho(f, g)$

$$\begin{cases} h(x,0) = f(x) \\ h(x, y+1) = g(x, y, h(x,y)) \end{cases}$$

$h : nat \to nat \to nat$

$h\ x\ y =$ if $y = 0$ then $f(x)$
else $g\ x\ (y-1)\ (h\ x\ (y-1))$

zero(y)

pred y

M

$$= \underline{fn}\ h.\ \underline{fn}\ x.\ \underline{fn}\ y.$$
$$\underline{\text{if}}\ (\text{zero}\ y)\ \underline{\text{then}}\ f\ x$$
$$\underline{\text{else}}\ \ g\ x\ (\underline{\text{pred}}\ y)\ (h\ x\ (\underline{\text{pred}}\ y))$$

$$\rho(f, g) = \underline{\text{fix}}(M)$$

$$F = \underline{\text{fix}}(\underline{fn}\ f.\ \sim\sim\ f \sim\sim)$$

IDEA: $F = \sim\sim\ F \sim\sim$

Partial recursive functions in PCF

- Primitive recursion.

$$\begin{cases} h(x,0) = f(x) \\ h(x,y+1) = g(x,y,h(x,y)) \end{cases}$$

- Minimisation.

Exercise

$$m(x) \;=\; \text{the least } y \geq 0 \text{ such that } k(x,y) = 0$$

Idea:

$m(x) = m'(x, 0)$

$m'(x,y) =$ if $k(x,y) = 0$ then y
else $m'(x, y+1)$

PCF evaluation relation

takes the form

$$M \Downarrow_\tau V$$

where

- τ is a PCF type
- $M, V \in \mathrm{PCF}_\tau$ are closed PCF terms of type τ
- V is a value,

$$V ::= \mathbf{0} \mid \mathbf{succ}(V) \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{fn}\, x : \tau \,.\, M.$$

Can implement lazy data types; eg: infinite streams can be encoded as functions $nat \to \tau$

PCF evaluation (sample rules)

$$(\Downarrow_{\mathrm{val}}) \quad V \Downarrow_\tau V \qquad (V \text{ a value of type } \tau)$$

$$(\Downarrow_{\mathrm{cbn}}) \quad \frac{M_1 \Downarrow_{\tau \to \tau'} \mathbf{fn}\, x : \tau \,.\, M_1' \qquad M_1'[M_2/x] \Downarrow_{\tau'} V}{M_1\, M_2 \Downarrow_{\tau'} V}$$

PCF evaluation (sample rules)

$$(\Downarrow_{\mathrm{val}}) \quad V \Downarrow_\tau V \qquad (V \text{ a value of type } \tau)$$

$$(\Downarrow_{\mathrm{cbn}}) \quad \frac{M_1 \Downarrow_{\tau \to \tau'} \mathbf{fn}\, x : \tau \,.\, M_1' \qquad M_1'[M_2/x] \Downarrow_{\tau'} V}{M_1\, M_2 \Downarrow_{\tau'} V}$$

RECALL

f(fix f) = fix f

$$(\Downarrow_{\mathrm{fix}}) \quad \frac{M\,(\mathbf{fix}(M)) \Downarrow_\tau V}{\mathbf{fix}(M) \Downarrow_\tau V}$$

Contextual equivalence

Two phrases of a programming language are contextually equivalent if any occurrences of the first phrase in a complete program can be replaced by the second phrase without affecting the observable results of executing the program.

Contextual equivalence of PCF terms

Given PCF terms M_1, M_2, PCF type τ, and a type environment Γ, the relation $\boxed{\Gamma \vdash M_1 \cong_{\mathrm{ctx}} M_2 : \tau}$ is defined to hold iff

- Both the typings $\Gamma \vdash M_1 : \tau$ and $\Gamma \vdash M_2 : \tau$ hold.
- For all PCF contexts $\mathcal{C}$ for which $\mathcal{C}[M_1]$ and $\mathcal{C}[M_2]$ are closed terms of type γ, *where* $\gamma = nat$ *or* $\gamma = bool$, and for all values $V : \gamma$,

$$\mathcal{C}[M_1] \Downarrow_\gamma V \;\Leftrightarrow\; \mathcal{C}[M_2] \Downarrow_\gamma V.$$

PCF denotational semantics — aims

- PCF types $\tau \;\mapsto\;$ domains $[\![\tau]\!]$.

- Closed PCF terms $M : \tau \;\mapsto\;$ elements $[\![M]\!] \in [\![\tau]\!]$.

  Denotations of open terms will be continuous functions.

- Compositionality.

  In particular: $[\![M]\!] = [\![M']\!] \;\Rightarrow\; [\![\mathcal{C}[M]]\!] = [\![\mathcal{C}[M']]\!]$.

- Soundness.

  For any type τ, $M \Downarrow_\tau V \;\Rightarrow\; [\![M]\!] = [\![V]\!]$.

- Adequacy.

  For $\tau = bool$ or nat, $[\![M]\!] = [\![V]\!] \in [\![\tau]\!] \implies M \Downarrow_\tau V$.