# *Topic 5*

PCF

# PCF syntax

Types

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

# PCF syntax

Types

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

Expressions

$$M \quad ::= \quad \mathbf{0} \mid \mathbf{succ}(M) \mid \mathbf{pred}(M)$$

# PCF syntax

Types

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

Expressions

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

# PCF syntax

## Types

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

## Expressions

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

# PCF syntax

Types

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

Expressions

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

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

# PCF syntax

Types

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

Expressions

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

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

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

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

- $\Gamma$ 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
- $\tau$ is a type.

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

- $\Gamma$ 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
- $\tau$ is a type.

**Notation:**

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

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

## PCF typing relation (sample rules)

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

## PCF typing relation (sample rules)

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

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

## PCF typing relation (sample rules)

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

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

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

# Partial recursive functions in PCF

- Primitive recursion.

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

## 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.

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

## PCF evaluation relation

takes the form

$$M \Downarrow_{\tau} V$$

where

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

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

## PCF evaluation (sample rules)

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

## 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)

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

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

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

## 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 $\tau$, and a type environment $\Gamma$, 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 $\gamma$, *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 denotational semantics — aims

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

# 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.

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

# 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 $\tau$, $M \Downarrow_\tau V \;\Rightarrow\; [\![M]\!] = [\![V]\!]$.

## 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 $\tau$, $M \Downarrow_\tau V \ \Rightarrow\ [\![M]\!] = [\![V]\!]$.
- Adequacy.

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

**Theorem.** *For all types* $\tau$ *and closed terms* $M_1, M_2 \in \mathrm{PCF}_\tau$, *if* $[\![M_1]\!]$ *and* $[\![M_2]\!]$ *are equal elements of the domain* $[\![\tau]\!]$, *then* $M_1 \cong_{\mathrm{ctx}} M_2 : \tau$.

**Theorem.** *For all types* $\tau$ *and closed terms* $M_1, M_2 \in \mathrm{PCF}_\tau$*, if* $[\![M_1]\!]$ *and* $[\![M_2]\!]$ *are equal elements of the domain* $[\![\tau]\!]$*, then* $M_1 \cong_{\mathrm{ctx}} M_2 : \tau$*.*

*Proof.*

$$\mathcal{C}[M_1] \Downarrow_{nat} V \Rightarrow [\![\mathcal{C}[M_1]]\!] = [\![V]\!] \quad \text{(soundness)}$$

$$\Rightarrow [\![\mathcal{C}[M_2]]\!] = [\![V]\!] \quad \text{(compositionality on } [\![M_1]\!] = [\![M_2]\!]\text{)}$$

$$\Rightarrow \mathcal{C}[M_2] \Downarrow_{nat} V \quad \text{(adequacy)}$$

and symmetrically. □

# Proof principle

To prove

$$M_1 \cong_{\mathrm{ctx}} M_2 : \tau$$

it suffices to establish

$$[\![M_1]\!] = [\![M_2]\!] \text{ in } [\![\tau]\!]$$

## Proof principle

To prove

$$M_1 \cong_{\text{ctx}} M_2 : \tau$$

it suffices to establish

$$[\![M_1]\!] = [\![M_2]\!] \text{ in } [\![\tau]\!]$$

$\boxed{?}$ The proof principle is sound, but is it complete? That is, is equality in the denotational model also a necessary condition for contextual equivalence?