PCF

"Programming Computable Functions"
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \rightarrow \tau \]

E.g.

\[ \text{bool} \rightarrow (\text{bool} \rightarrow \text{bool}) \]

\[ (\text{nat} \rightarrow \text{bool}) \rightarrow \text{bool} \]
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \rightarrow \tau \]

E.g.

\[ \text{bool} \rightarrow (\text{bool} \rightarrow \text{bool}) \]

\[ (\text{nat} \rightarrow \text{bool}) \rightarrow \text{bool} \]

\[ \rightarrow \text{ is right associative:} \]

"\( \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \)" means \( \tau_1 \rightarrow (\tau_2 \rightarrow \tau_3) \)
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \to \tau \]

Expressions

\[ M ::= 0 \mid \text{succ}(M) \mid \text{pred}(M) \]
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \to \tau \]

Expressions

\[ M ::= 0 \mid \text{succ}(M) \mid \text{pred}(M) \]

\[ \quad \mid \text{true} \mid \text{false} \mid \text{zero}(M) \]
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \rightarrow \tau \]

Expressions

\[ M ::= 0 \mid \text{succ}(M) \mid \text{pred}(M) \mid \text{true} \mid \text{false} \mid \text{zero}(M) \mid x \mid \text{if } M \text{ then } M \text{ else } M \]
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \rightarrow \tau \]

Expressions

\[ M ::= 0 \mid \text{succ}(M) \mid \text{pred}(M) \]
\[ \mid \text{true} \mid \text{false} \mid \text{zero}(M) \]
\[ \mid x \mid \text{if } M \text{ then } M \text{ else } M \]
\[ \mid \text{fn } x : \tau . M \mid M \ M \mid \text{fix}(M) \]

where \( x \in \mathcal{V} \), an infinite set of variables.

Application is left associative:

"\( M_1 M_2 M_3 \)" means \( (M_1 M_2) M_3 \)
Whereas in OCaml one might write

\[
\text{let rec } f \ x = \text{ if } x=0 \text{ then } 1 \text{ else } x * f(x-1) \text{ in } f \ 42
\]

in PCF one has to write

\[
(\text{fix } (\text{fn } f : \text{nat} \to \text{nat}. \ \text{fn } x : \text{nat}. \ \text{if } \text{zero}(x) \text{ then } \text{Succ}(0) \text{ else } \text{times } x (f(\text{pred}(x))) ) ) \ \text{Succ}^4(0)
\]
Whereas in Ocaml one might write

```ocaml
let rec f x = if x=0 then 1 else x*f(x-1) in f 42
```

in PCF one has to write

\[
\text{fix (fn f : nat -> nat. fn x : nat.}
\]
\[
\begin{align*}
\text{if zero(x) then Succ(0) } \\
\text{else times x (f (pred(x)) )}
\end{align*}
\]
\[
\text{Succ}^{42}(0)
\]

Where \(\text{Succ}^{42}(0) \triangleq \text{Succ(Succ(... Succ(0) ...))} \) times is as on p67 of the notes.
PCF syntax

Types

\[ \tau ::= \text{nat} \mid \text{bool} \mid \tau \rightarrow \tau \]

Expressions

\[ M ::= 0 \mid \text{succ}(M) \mid \text{pred}(M) \]
\[ \mid \text{true} \mid \text{false} \mid \text{zero}(M) \]
\[ \mid x \mid \text{if } M \text{ then } M \text{ else } M \]
\[ \mid \text{fn } x : \tau . M \mid M \ M \mid \text{fix}(M) \]

where \( x \in V \), an infinite set of variables.

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

E.g. \( \text{fix}(\text{fn } x : \tau . x) = \text{fix}(\text{fn } y : \tau . y) \)
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 $\text{dom}(\Gamma)$).
- $M$ is a term.
- $\tau$ is a type.

If this contains distinct variables $x_1, x_2, \ldots, x_n$ and $\Gamma(x_i) = \tau_i$, we sometimes write $\Gamma$ as

\[ \{ x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n \} \]
PCF typing relation (sample rules)

\[(\cdot_{\text{fn}})\]
\[
\frac{\Gamma[x \mapsto \tau] \vdash M : \tau'}{\Gamma \vdash \text{fn } x : \tau \cdot M : \tau \rightarrow \tau'} \quad \text{if } x \notin \text{dom}(\Gamma)
\]

\[
\text{dom}(\Gamma[x \mapsto \tau]) = \text{dom}\Gamma \cup \{x\}
\]

\[\Gamma[x \mapsto \tau] \text{ maps } x \text{ to } \tau \text{ and otherwise acts like } \Gamma\]
PCF typing relation (sample rules)

(\text{:fn}) \quad \frac{\Gamma[x \mapsto \tau] \vdash M : \tau'}{\Gamma \vdash \text{fn} \, x : \tau \cdot M : \tau \rightarrow \tau'} \quad \text{if } x \notin \text{dom}(\Gamma)

(\text{:app}) \quad \frac{\Gamma \vdash M_1 : \tau \rightarrow \tau' \quad \Gamma \vdash M_2 : \tau}{\Gamma \vdash M_1 \, M_2 : \tau'}
PCF typing relation (sample rules)

(\texttt{\textbackslash::fn}) \quad \frac{\Gamma[x \mapsto \tau] \vdash M : \tau'}{\Gamma \vdash \text{fn } x : \tau. M : \tau \rightarrow \tau'} \quad \text{if } x \notin \text{dom}(\Gamma)

(\texttt{\textbackslash::app}) \quad \frac{\Gamma \vdash M_1 : \tau \rightarrow \tau' \quad \Gamma \vdash M_2 : \tau}{\Gamma \vdash M_1 M_2 : \tau'}

(\texttt{\textbackslash::fix}) \quad \frac{\Gamma \vdash M : \tau \rightarrow \tau}{\Gamma \vdash \text{fix}(M) : \tau}
Proposition 5.3.1 (i) \[ p57 \]

If $\Gamma \vdash M : \tau$ and $\Gamma \vdash M : \tau'$ are both derivable, then $\tau = \tau'$.

Proof: Use rule induction—show that

$$H \triangleq \{ (\Gamma, M, \tau) \mid \Gamma \vdash M : \tau \land \forall \tau'. \Gamma \vdash M : \tau' \Rightarrow \tau = \tau' \}$$

is closed under the typing rules.
\[
H \triangleq \{ (\Gamma, M, \tau) \mid \Gamma \vdash M : \tau \land \forall \tau'. \exists \tau'' \in \{2, 1, 2''\} \}
\]
is closed under the typing rules.

Crucial induction step is for the \((\text{fn})\) rule:
Want to show
\[
( [\lambda x : \tau_1], M, \tau_2 ) \in H \Rightarrow ( \Gamma, \text{fn} x : \tau_1 . M, \tau_2 \rightarrow \tau_2 ) \in H
\]
\[ H \triangleq \{ (\Gamma, M, \tau) \mid \Gamma \vdash M : \tau \wedge \forall \Delta. \Gamma \vdash M : \tau \implies \Delta = \tau = \Delta \} \]

is closed under the typing rules.

Crucial induction step is for the \((\text{: fn})\) rule:

Want to show

\[ (\Gamma, \lambda x : \tau_1, M, \tau_2) \in H \Rightarrow (\Gamma, \text{fn} x : \tau_1. M, \tau \Rightarrow \tau_2) \in H \]

Suppose \((\Gamma, \lambda x : \tau_1, M, \tau_2) \in H\) \& \(\Gamma \vdash \text{fn} x : \tau_1. M : \tau \Rightarrow \tau_2\)

Need to see that \(\tau = \tau_1 \Rightarrow \tau_2\).
\[ H \triangleq \{ (\Gamma, M, z) \mid \Gamma \vdash M : 2 \land \forall \ z'. \Gamma \vdash M : z' \rightarrow z' \} \]

is closed under the typing rules.

Crucial induction step is for the \((\text{fn} )\) rule:

Want to show

\[ (\frac{[]} x : t_1 \rightarrow t_2, M, z_2) \in H \Rightarrow (\Gamma, \text{fn}_x : t_1. M, z_2 \rightarrow z_2) \in H \]

Suppose \( (\frac{[]} x : t_1 \rightarrow t_2, M, z_2) \in H \) \& \( \Gamma \vdash \text{fn}_x : t_1. M : 2 \rightarrow 2' \)

Need to see that \( z_2 = z_2' \rightarrow z_2 \).

This must have been proved by applying \((\text{fn} )\) to \( \frac{[]} x : t_1 \rightarrow t_2 \vdash M : 2 \rightarrow 2' \) for some \( z_2' \) with \( 2' = z_2' \rightarrow z_2 \).
\[ H \triangleq \{ (\Gamma, M, \tau) \mid \Gamma \vdash M : \tau \land \forall \tau'. \Gamma \vdash M : \tau' \Rightarrow \text{\texttt{c2} = c1} \} \]

is closed under the typing rules.

Crucial induction step is for the \((\text{\texttt{fn}})\) rule:

Want to show

\[ (\Gamma \vdash x : \tau_i, M, \tau_2) \in H \Rightarrow (\Gamma, \text{\texttt{fn}}x:\tau_i.M, \tau_2 \rightarrow \tau_2) \in H \]

Suppose \((\Gamma \vdash x : \tau_i, M, \tau_2) \in H \land \Gamma \vdash \text{\texttt{fn}}x:\tau_i.M : \tau_2'\)

Need to see that \(\tau_2 = \text{\texttt{c2} = c1}^{-1} \rightarrow \tau_2 \).

Have \(\Gamma \vdash x : \tau_i, M : \tau \text{\texttt{c2} = c1}^{-1} \rightarrow \tau\) with \(\tau_2 = \text{\texttt{c2} = c1}^{-1} \rightarrow \tau_2'\).
\[ \mathcal{H} \triangleq \{ (\Gamma, M, \tau) \mid \Gamma \vdash M : \tau \text{ and } \forall \tau'. \Gamma \vdash \tau' \Rightarrow \tau = \tau' \} \]

is closed under the typing rules.

Crucial induction step is for the \((\text{fn })\) rule:

Want to show

\[ (\Gamma[x \mapsto \tau_1], M, \tau_2) \in \mathcal{H} \Rightarrow (\Gamma, \text{fn} x : \tau_1. M, \tau_2 \tau_2) \in \mathcal{H} \]

Suppose \((\Gamma[x \mapsto \tau_1], M, \tau_2) \in \mathcal{H} \) and \(\Gamma \vdash \text{fn} x : \tau_1. M : \tau_2'\)

Need to see that \(\tau_2' = \tau_1 \rightarrow \tau_2\).

Have \(\Gamma[x \mapsto \tau_1] \vdash M : \tau_2''\) with \(\tau_2'' = \tau_2' = \tau_1 \rightarrow \tau_2\).

So \(\tau_2 = \tau_2''\) and hence \(\tau_2' = \tau_2'' = \tau_1 \rightarrow \tau_2\).
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.

$\text{PCF}_\tau \overset{\text{def}}{=} \{ M \mid M : \tau \}$. i.e. $\text{fv}(M) = \emptyset$

where...
\text{fv}(M) - \text{set of free variables of } M \\
\text{is defined by:} \\
\text{fv}(0) = \text{fv}(\text{true}) = \text{fv}(\text{false}) = \emptyset \\
\text{fv}(\text{succ}(M)) = \text{fv}(\text{pred}(M)) = \text{fv}(\text{zero}(M)) \\
\quad = \text{fv}(\text{fix}(M) = \text{fv}(M)) \\
\text{fv}(\text{if } M \text{ then } M' \text{ else } M") = \text{fv}(M) \cup \text{fv}(M') \cup \text{fv}(M") \\
\text{fv}(M \ M') = \text{fv}(M) \cup \text{fv}(M') \\
\text{fv}(x) = \{x\} \\
\text{fv}(\text{fun } x : \tau. \ M) = \{x' \in \text{fv}(M) \mid x' \neq x\}
PCF evaluation relation

takes the form

\[ M \downarrow_{\tau} V \]

where

- \( \tau \) is a PCF type
- \( M, V \in \text{PCF}_\tau \) are closed PCF terms of type \( \tau \)
- \( V \) is a value,

\[ V ::= 0 \mid \text{succ}(V) \mid \text{true} \mid \text{false} \mid \text{fn } x : \tau . M. \]
PCF evaluation (sample rules)

\[ (\updownarrow_{\text{val}}) \quad V \downarrow_{\tau} V \quad (V \text{ a value of type } \tau) \]
PCF evaluation (sample rules)

\[ (\downarrow_{\text{val}}) \quad V \downarrow_{\tau} V \quad (V \text{ a value of type } \tau) \]

\[ (\downarrow_{\text{cbn}}) \quad \frac{M_1 \downarrow_{\tau \rightarrow \tau'} \text{ fn } x : \tau . M'_1 \quad M'_1[M_2/x] \downarrow_{\tau'} V}{M_1 \quad M_2 \downarrow_{\tau'} V} \]
PCF evaluation (sample rules)

\[(\downarrow_{\text{val}}) \quad V \downarrow_\tau V \quad (V \text{ a value of type } \tau)\]

\[(\downarrow_{\text{cbn}}) \quad \frac{M_1 \downarrow_{\tau \rightarrow \tau'} \text{ fn } x : \tau . M'_1 \quad M'_1[M_2/x] \downarrow_{\tau'} V}{M_1 \ M_2 \downarrow_{\tau'} V} \]

**Substitution** (capture-avoiding - but since \(M_2\) is closed there can be no capture)

\[\text{NB} \quad \text{if } \Gamma[x : \tau] \vdash M_1 : \tau' \\Rightarrow \Gamma \vdash M'_1[M_2/x] : \tau' \quad \text{(see p S7).}\]
PCF evaluation (sample rules)

\[\downarrow_{\text{val}} \quad V \downarrow_{\tau} V \quad (V \text{ a value of type } \tau)\]

\[\downarrow_{\text{cbn}} \quad \frac{M_1 \downarrow_{\tau \rightarrow \tau'} \textbf{fn} \ x : \tau \ . \ M'_1 \quad M'_1[M_2/x] \downarrow_{\tau'} V}{M_1 \ M_2 \downarrow_{\tau'} V}\]

\[\downarrow_{\text{fix}} \quad \frac{M \ \texttt{fix}(M) \downarrow_{\tau} V}{\texttt{fix}(M) \downarrow_{\tau} V}\]
PCF evaluation (sample rules)

\[
(\text{pred}) \quad \begin{align*}
M \Downarrow \text{nat} & \quad \text{Succ}(V) \\
\text{pred}(M) \Downarrow \text{nat} & \quad V
\end{align*}
\]

is the only rule for \text{pred}.

Since \( 0 \Downarrow \text{nat} V \) only holds for \( V = 0 \),
we conclude that \( \text{pred}(0) \nDownarrow \text{nat} V \)

\((\text{Making } \text{pred}(0) \text{ not evaluate to anything is a somewhat arbitrary choice.})\)
Defining $\Omega_2 \triangleq \text{fix}(\text{fix } x : 2. x)$

we get $\Omega_2 : 2 \quad (\text{proof - easy})$

$\exists \forall. \Omega_2 \downarrow \forall \quad (\text{proof ...})$
If \( \text{fix}(\text{fn} x: \tau. x) \Downarrow_T V \) had any proof, then we could find one of smallest height, \( n \) say, and it must look like

\[
\begin{align*}
\text{fn} x: \tau. x \Downarrow_T V & \quad \text{(\( \Downarrow_{\text{val}} \))} \\
\text{fix}(\text{fn} x: \tau. x)/x & \Downarrow_T V \\
(\text{fn} x: \tau. x)(\text{fix}(\text{fn} x: \tau. x)) & \Downarrow_T V \\
\text{fix}(\text{fn} x: \tau. x) & \Downarrow_T V \\
\text{fix}(\text{fn} x: \tau. x) & \Downarrow_T V \quad \text{(\( \Downarrow_{\text{fix}} \))}
\end{align*}
\]
If \( \text{fix}(\text{fn}\ x:\tau.\ x) \Downarrow_{\tau} V \) had any proof, then we could find one of smallest height, \( n \) say, and it must look like

\[
\begin{align*}
\text{fn}\ x:\tau.\ x \Downarrow_{\tau} & \quad \text{\text{(\text{val})}} \\
\text{fix}(\text{fn}\ x:\tau.\ x) \Downarrow_{\tau} & \quad V \\
(\text{fn}\ x:\tau.\ x)(\text{fix}(\text{fn}\ x:\tau.\ x)) \Downarrow_{\tau} & \quad V \\
\text{fix}(\text{fn}\ x:\tau.\ x) \Downarrow_{\tau} & \quad V \\
\end{align*}
\]
If \( \text{fix}(\text{fn} \ x : \tau . \ x) \downarrow_{\tau} V \) had any proof, then we could find one of smallest \underline{height}, \( n \), say, and it must look like:

\[
\text{fix}(\text{fn} \ x : \tau . \ x) \downarrow_{\tau} V
\]

\( (\downarrow_{\text{val}}) \)

\[
\vdash (\text{fn} \ x : \tau . \ x)(\text{fix}(\text{fn} \ x : \tau . \ x)) \downarrow_{\tau} V
\]

\( (\downarrow_{\text{fix}}) \)

\[
\text{fix}(\text{fn} \ x : \tau . \ x) \downarrow_{\tau} V
\]

So no such proof can exist.
PCF

"Programming Computable Functions"
We represent numbers $n \in \mathbb{N} = \{0, 1, 2, \ldots\}$ by closed values $\text{suc}^n(0) : \text{nat}$ in PCF

\[
\begin{align*}
\text{suc}^0(0) &= 0 \\
\text{suc}^{n+1}(0) &= \text{suc}(\text{suc}^n(0))
\end{align*}
\]

**FACT** For any computable partial function $f : \mathbb{N} \rightarrow \mathbb{N}$ there is a closed PCF term $F : \text{nat} \rightarrow \text{nat}$ such that for all $n, m \geq 0$

\[F(\text{suc}^m(0)) \downarrow_{\text{nat}} \text{suc}^n(0)\]

if & only if

$f$ is defined at $m$ & $f(m) = n$
Partial recursive functions in PCF

- Primitive recursion.

\[
\begin{align*}
h(x, 0) &= f(x) \\
h(x, y + 1) &= g(x, y, h(x, y))
\end{align*}
\]
Partial recursive functions in PCF

- Primitive recursion.

\[
\begin{align*}
  h(x, 0) &= f(x) \\
  h(x, y + 1) &= g(x, y, h(x, y))
\end{align*}
\]

If \( f \) is programmed in PCF by \( F : \text{nat} \to \text{nat} \) and \( g \) is programmed in PCF by \( G : \text{nat} \to \text{nat} \to \text{nat} \to \text{nat} \) then \( h \) can be programmed by:

\[
\text{fix } (\text{fn } h : \text{nat} \to \text{nat} \to \text{nat}. \text{fn } x : \text{nat}. \text{fn } y : \text{nat}. \text{ if zero } (y) \text{ then } F x \text{ else } G x (\text{pred } y) (h x (\text{pred } y)))
\]
Partial recursive functions in PCF

- Primitive recursion.

\[
\begin{align*}
  h(x, 0) &= f(x) \\
  h(x, y + 1) &= g(x, y, h(x, y))
\end{align*}
\]

- Minimisation.

\[m(x) = \text{the least } y \geq 0 \text{ such that } k(x, y) = 0\]
Partial recursive functions in PCF

- Primitive recursion.

\[
\begin{aligned}
    h(x, 0) &= f(x) \\
    h(x, y + 1) &= g(x, y, h(x, y))
\end{aligned}
\]

- Minimisation.

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

If \( k \) is programmed in PCF by \( k : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \), then \( m \) can be programmed by

\[
\text{fn } x : \text{nat}. \ M' \ x \ 0
\]

where

\[
M' = \text{fix (fn } m' : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}. \ \text{fn } x : \text{nat}. \ \text{fn } y : \text{nat}. \ \text{if } \text{zero}(k x y) \text{ then } y \text{ else } m' x (\text{suc } y))
\]