PCF Axioms

Equality

(ref) $\quad M = M$

nat **and** *bool*

(add) $\quad 0 + 0 = 0,\ 0 + 1 = 1,\ldots,3 + 5 = 8,\ldots$

$(Eq?)$ $\quad Eq?\ n\ n = true$

$\quad Eq?\ n\ m = false (n, m$ distinct numerals$)$

$(cond)$ $\quad \texttt{it}\ true\ then\ M\ \texttt{else}\ N = M$

$\quad \texttt{it}\ false\ then\ M\ \texttt{else}\ N = N$

PCF Axioms

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) $fix_\sigma = \lambda f : \sigma \to \sigma.\ f(fix_\sigma f)$

The η Axiom Revisited

The η rule is:

$$(\eta) \qquad \lambda x : \sigma . M x = M, \text{ provided } x \text{ not free in } M.$$

Which should be read as:

$$(\eta) \qquad \lambda x : \sigma . (M x) = M, \text{ provided } x \text{ not free in } M.$$

in which case the β rule does not apply. Let's consider an example. Any application of the above function will look like:

$$(\lambda x : \sigma . M x) N$$

We can now apply the β axiom and get:

$$\begin{aligned} &(\lambda x : \sigma . M x) N \\ &= [N/x](M x) \\ &= M N, \text{ since } x \text{ is not free in } M \end{aligned}$$

Therefore, in any application, $\lambda x : \sigma . M x$ works just like M, so they should be equal.

The fix Axiom

The desugaring of `letrec` is:

$$\texttt{letrec}\ f : \sigma = M\ \texttt{in}\ N$$
$$=^{def}\ \texttt{let}\ f : \sigma = (fix_\sigma.M)\ \texttt{in}\ N.$$

And a further desugaring rule for function definitions:

$$\texttt{letrec}\ f(x : \tau) : \sigma = M\ \texttt{in}\ N$$
$$=^{def}\ \texttt{letrec}\ f : \tau \to \sigma = \lambda x : \tau.M\ \texttt{in}\ N.$$

The axioms for recursive functions is:

(fix) $\quad fix_\sigma = \lambda f : \sigma \to \sigma.\ f(fix_\sigma f)$

Recursion Example

To apply this, consider the following definition of factorial:

$$
\begin{array}{l}
\texttt{letrec}\ f(x : nat) : nat = \lambda x : nat. \\
\qquad\qquad\qquad\qquad\qquad \texttt{if}\quad Eq?\ x\ 0\ \texttt{then}\ 1 \\
\qquad\qquad\qquad\qquad\qquad \texttt{else}\quad x * f(x-1) \\
\ \ \texttt{in}\quad f\ 1
\end{array}
$$

Which desugars into (with one alpha renaming):

$$
\begin{array}{l}
\texttt{let}\ f : nat \rightarrow nat\ = (fix_{nat \rightarrow nat}\ \lambda g : nat \rightarrow nat\ .\lambda y : nat\ . \\
\qquad\qquad\qquad\qquad \texttt{if}\quad Eq?\ y\ 0\ \texttt{then}\ 1 \\
\qquad\qquad\qquad\qquad \texttt{else}\quad y * g(y-1)) \\
\ \ \texttt{in}\quad f\ 1
\end{array}
$$

Which by desugaring the `let` gives:

$$
\begin{array}{l}
(\lambda f : nat \rightarrow nat.\ f\ 1)\ (fix_{nat \rightarrow nat}\ \lambda g : nat \rightarrow nat\ .\lambda y : nat\ . \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{if}\quad Eq?\ y\ 0\ \texttt{then}\ 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{else}\quad y * g(y-1))
\end{array}
$$

Example continued

Applying beta we get:

$$
\begin{aligned}
(fix_{nat\to nat}\ \lambda g :nat \to nat\ .\lambda y : nat\ .\ \\
&\texttt{if}\ \ Eq?\ y\ 0\ \texttt{then}\ 1 \\
&\texttt{else}\ \ y * g(y-1))1
\end{aligned}
$$

Applying fix (the β-reduced form), we get:

$$
\begin{aligned}
&(\lambda g :nat \to nat\ .\lambda y : nat\ . \\
&\quad \texttt{if}\ \ Eq?\ y\ 0\ \texttt{then}\ 1 \\
&\quad \texttt{else}\ \ y * g(y-1)1) \\
&(fix_{nat\to nat}\ \lambda g :nat \to nat\ .\lambda y : nat\ . \\
&\qquad\qquad \texttt{if}\ \ Eq?\ y\ 0\ \texttt{then}\ 1 \\
&\qquad\qquad \texttt{else}\ \ y * g(y-1)1)
\end{aligned}
$$

Applying β again will finally produce the definition of factorial on the outside, but the definition of fix would have to be applied again to get the answer for 1.

Semantics Overview

The *observable values* in a language are the set of values that can be printed or are final answers when running a complete program.
In PCF:

- observable types are: nat and $bool$
- *expressible* values include functions, which are not observable
- a *program* is a closed (variable free) well-formed expression of observable type
- *results* are $true, false, 0, 1, 2, 3, ...$

Semantics Overview

The three types of semantics we will study and their applications:

- Axiomatic semantics: a proof system for deducing properties of programs and their parts.
- Operational semantics: defines an abstract machine model for evaluating complete programs.
- Denotational semantics: gives a mathemtical meaning (a model) for programs and their parts.

We have seen most of the axioms for PCF. In addition to these axioms (each is an equation), there are a set of inference rules.

Inference Rules

Equivalence
(sym), $(trans)$

$$\frac{M = N}{N = M} \qquad \frac{M = N, N = P}{M = P}$$

Congruence
Functions

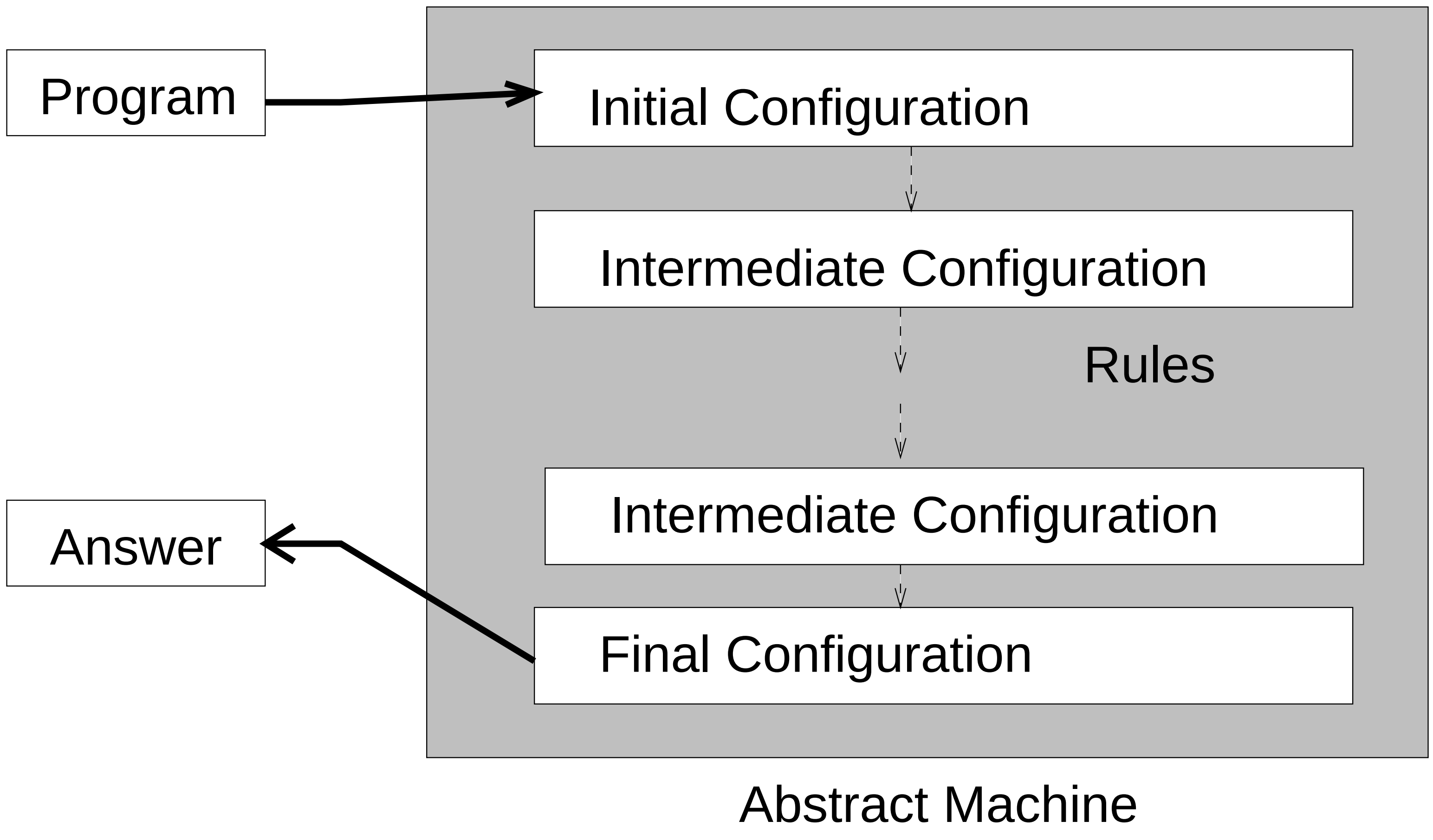
$$\frac{M = N}{\lambda x : \sigma.M = \lambda x : \sigma.N} \qquad \frac{M = N, P = Q}{MP = NQ}$$

Other congruence rules are omitted. In general a *congruence* rule is something of the form:

$$\frac{M_1 = N_1, ...M_n = N_n}{F(M_1, ..., M_n) = F(N_1, ..., N_n)}$$

In this case the Fs are the compound language constructs in PCF.

Operational Semantics



PCF Operational Semantics

The PCF operational semantics defines a binary relation, which is a subset of the equality relation given by the axiomatic semantics.

I.e., the rules are directed equations – nothing new is added.

nat **and** *bool*

(add) $\quad 0 + 0 \to 0, 0 + 1 \to 1, ..., 3 + 5 \to 8, ...$

$(Eq?)$ $\quad Eq?\ n\ n \to true$

$\quad Eq?\ n\ m \to false (n, m \text{ distinct numerals})$

$(cond)$ $\quad \texttt{it}\ true\ then\ M\ \texttt{else}\ N \ \to\ M$

$\quad \texttt{it}\ false\ then\ M\ \texttt{else}\ N \ \to\ N$

PCF Operational Semantics

Binding

(α) $\quad \lambda x : \sigma . M = \lambda y : \sigma : [y/x]M,$
provided y not free in M.

Functions

(β) $\quad (\lambda x : \sigma . M) N \rightarrow [N/x]M$

Recursion

(fix) $\quad fix_\sigma \;\rightarrow\; \lambda f : \sigma \rightarrow \sigma .\ f(fix_\sigma f)$

Reduction Examples

The transitive closure of $\rightarrow$ and α equality is defined as:

$$\begin{aligned} M \twoheadrightarrow N \text{ if } & M =_\alpha N \text{ or} \\ & M \rightarrow M' \text{ and } M' \twoheadrightarrow N \end{aligned}$$

Equation example:

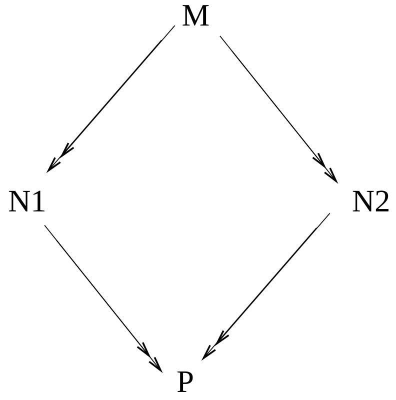
$$(\texttt{if } M \texttt{ then } 3+4 \texttt{ else } 7) = (\texttt{if } M \texttt{ then } 7 \texttt{ else } 3+4)$$

but we cannot reduce the LHS to the RHS using the $\rightarrow$ relation.

$$\begin{aligned} &5+2 \twoheadrightarrow 5+2 \\ &5+2 \not\rightarrow 5+2 \end{aligned}$$

Confluence

The rules so far are non-deterministic: multiple rules apply to single terms. One desirable property of a set of reduction rules is *confluence* or the *Church-Rosser* property.



The PCF reduction rules given above are confluent.

A *normal form* is a term that cannot be reduced by any rule.
Confluence implies that:

- If two terms reduce to the same normal form, then they are they are equal in the axiomatic semantics.
- If two terms reduce to distinct normal forms then they are not equal.
- Note: we haven't said anything about termination.

Structured Operational Semantics

In *structured* operational semantics, each rule applies to exactly one form of term.

left-first

$$\frac{M \rightarrow^{left} M'}{MN \rightarrow^{left} M'N}$$

$$\frac{N \rightarrow^{left} N'}{MN \rightarrow^{left} MN'}, M \text{ a normal form}$$

$$(\lambda x : \sigma . M)N \rightarrow^{left} [N/x]M$$