

Type Systems for Programming Languages (15-814)

Lecture Notes, Fall 2006, Week 3

Olatunji Ruwase

Sept 26 and 28, 2006

1 λ definability for booleans

A function $f : \mathbb{B} \rightarrow \mathbb{B}$ is λ definable iff $\exists \lambda$ term t st:

$$\begin{aligned}t \overline{\text{tr}} &=_{\beta} \overline{f(\text{tr})} \\t \overline{\text{fa}} &=_{\beta} \overline{f(\text{fa})}\end{aligned}$$

This generalizes to functions with multiple arguments, i.e

$$f : \mathbb{B}^{(n)} \rightarrow \mathbb{B}.$$

2 Church Numerals (Natural numbers)

Before diving into the representation of natural numbers, we examine pairing.

2.1 Pairing

Pairs are specified by the following interface.

introduction form:

$$\overline{\text{pair}} := \lambda x. \lambda y. \lambda b. b x y$$

elimination forms:

$$\begin{aligned}\overline{\text{1st}} &:= \lambda x. x \overline{\text{tr}} \\ \overline{\text{2nd}} &:= \lambda x. x \overline{\text{fa}}\end{aligned}$$

specification:

$$\begin{aligned}\overline{\text{1st}} (\overline{\text{pair}} t u) &=_{\beta} t \\ \overline{\text{2nd}} (\overline{\text{pair}} t u) &=_{\beta} u\end{aligned}$$

Note that the given specification is a weak one.

Because of lack of types we have that:

$$\overline{\text{pair}} (\overline{\text{1st}} x)(\overline{\text{2nd}} x) \neq_{\beta} x.$$

This can be shown as follows

$$\begin{aligned}\overline{\text{pair}} (\overline{\text{1st}} x)(\overline{\text{2nd}} x) &=_{\beta} \lambda b. b(\overline{\text{1st}} x)(\overline{\text{2nd}} x) \\ &=_{\beta} \lambda b. b(x \overline{\text{tr}})(x \overline{\text{fa}}) \\ &\neq_{\beta} x\end{aligned}$$

So although we have that,

$$\begin{aligned}\overline{1st} x &=_{\beta} \overline{1st} (\lambda b. b(x \overline{tr})(x \overline{fa})) \\ \overline{2nd} x &=_{\beta} \overline{2nd} (\lambda b. b(x \overline{tr})(x \overline{fa})) \\ \text{yet, } x &\neq_{\beta} \lambda b. b(x \overline{tr})(x \overline{fa}) !\end{aligned}$$

2.2 Representation of Natural numbers(\mathbb{N})

introduction form:

$$\begin{aligned}\overline{0} &:= \lambda s. \lambda z. z \\ \overline{n+1} &:= \lambda s. \lambda z. s(\overline{n} s z)\end{aligned}$$

elimination form:

$$\overline{rec} t, t_s, t_z := t t_s t_z$$

specification:

$$\begin{aligned}\overline{0} t_s t_z &=_{\beta} t_z \\ \overline{n+1} t_s t_z &=_{\beta} t_s (\overline{n} t_s t_z)\end{aligned}$$

Note:

$$\overline{n} =_{\beta} \lambda s. \lambda z. \underbrace{s^n(z)}_{s(s(\dots s(z)\dots))}$$

i.e iterate the function s $n \geq 0$ times starting with z

2.3 Examples

2.3.1 \overline{succ} is λ definable

$$\overline{succ} := \lambda n. \lambda s. \lambda z. s (n s z)$$

2.3.2 \overline{plus} is λ definable

$$\overline{plus} := \lambda m. \lambda n. \lambda s. \lambda z. m s (n s z)$$

$$\begin{aligned}\text{specification: } \overline{plus} \overline{m} \overline{n} &=_{\beta} \lambda s. \lambda z. \overline{m} s (\overline{n} s z) \\ &=_{\beta} \lambda s. \lambda z. \overline{m} s (s^{(n)} z) \\ &=_{\beta} \lambda s. \lambda z. s^{(m)}(s^{(n)} z) \\ &= \lambda s. \lambda z. s^{(m+n)} z \\ &= \overline{m+n}\end{aligned}$$

2.3.3 $\overline{\text{times}}$ is λ definable

$$\overline{\text{times}} := \lambda m. \lambda n. \lambda s. \lambda z. m (n s) z$$

$$\begin{aligned} \text{specification: } \overline{\text{times}} \overline{m} \overline{n} &=_{\beta} \lambda s. \lambda z. \overline{m} (\overline{n} s) z \\ &=_{\beta} \lambda s. \lambda z. \overline{m} (\lambda z'. s^{(n)} z') z \\ &=_{\beta} \lambda s. \lambda z. (\lambda z'. s^{(n)} z')^{(m)} z \\ &=_{\beta} \lambda s. \lambda z. (s^n)^{(m)} z \\ &= \lambda s. \lambda z. s^{(mn)} z \\ &= \overline{mn} \end{aligned}$$

2.3.4 $\overline{\text{pred}}$ is λ definable

$\overline{\text{pred}}$ is trickier to define than the previous examples. We assume the following specification.

$$\begin{aligned} \overline{\text{pred}} \overline{0} &=_{\beta} \overline{0} \\ \overline{\text{pred}} \overline{n+1} &=_{\beta} \overline{n} \end{aligned}$$

Although it seems that the definition should be similar in form to previous examples i.e

$$\overline{\text{pred}} := \lambda n. \lambda s. \lambda z. n (???) z$$

It is not clear what s should be and what role it plays in the recursion to generate the predecessor value. It turns out that we need an auxilliary function in order for defining $\overline{\text{pred}}$. The function $\overline{\text{p}}$ (which implements a shift register) has the following specification

$$\begin{aligned} \overline{\text{p}} \overline{0} &=_{\beta} \overline{\text{pair}} \overline{0} \overline{0} \\ \overline{\text{p}} \overline{m+1} &=_{\beta} \overline{\text{pair}} \overline{m+1} \overline{m} \end{aligned}$$

and the following definition

$$\overline{\text{p}} := \lambda a. a (\lambda x. \overline{\text{pair}} (\overline{\text{succ}} (\overline{\text{1st}} x)) (\overline{\text{1st}} x)) (\overline{\text{pair}} \overline{0} \overline{0})$$

We can observe closely what $\overline{\text{p}}$ actually does below

$$\begin{aligned} \overline{\text{p}} \overline{0} &=_{\beta} \overline{\text{pair}} \overline{0} \overline{0} \\ \overline{\text{p}} \overline{n+1} &=_{\beta} (\lambda x. \overline{\text{pair}} (\overline{\text{succ}} (\overline{\text{1st}} x)) (\overline{\text{1st}} x)) (\overline{\text{p}} \overline{n}) (\overline{\text{pair}} \overline{n} \overline{n-1}) \\ &=_{\beta} \overline{\text{pair}} (\overline{\text{succ}} (\overline{n})) \overline{n} \end{aligned}$$

Finally, $\overline{\text{pred}}$ can be easily defined in terms of $\overline{\text{p}}$ as follows

$$\overline{\text{pred}} := \lambda n. \overline{2nd} (\overline{\text{p}} n)$$

3 General recursion/Self reference

Using general recursion the factorial function ($\overline{\text{fact}}$) can be written as follows:

$$\overline{\text{fact}} := \lambda n. \overline{\text{ifz}} n 1 (\overline{\text{times}} n (\overline{\text{fact}} (\overline{\text{pred}} n)))$$

The definition in λ calculus can not be written in a similar way because of the self reference to $\overline{\text{fact}}$. Instead we would like to find a solution to the recursion equation (up to β equivalence). i.e

$$\overline{\text{fact}} =_{\beta} \lambda n. \overline{\text{ifz}} \ n \ 1 \ (\overline{\text{times}} \ n \ (\overline{\text{fact}} \ (\overline{\text{pred}} \ n)))$$

We proceed by introducing a function $FACT$ defined as follows

$$FACT := \lambda f. \lambda n. \overline{\text{ifz}} \ n \ 1 \ (\overline{\text{times}} \ n \ (f \ (\overline{\text{pred}} \ n)))$$

Observe that the solution to the following equation is a solution to the factorial function.

$$\overline{\text{fact}} =_{\beta} FACT(\overline{\text{fact}})$$

Thus we now seek a fixed point of $FACT$. In fact all functions in pure λ calculus have fixed points, that is

$$\forall t \exists u \ t u =_{\beta} u$$

Note that this is true only because of lack of types in pure λ calculus. For example,

$$\exists u \ \overline{\text{succ}} \ u =_{\beta} u \ (\because u \text{ is not a church numeral !})$$

We now define $FACT'$ as

$$FACT' := \lambda f. \lambda n. \overline{\text{ifz}} \ n \ 1 \ (\overline{\text{times}} \ n \ (f \ f \ (\overline{\text{pred}} \ n)))$$

Note that f above passes itself as the first argument. So,

$$\begin{aligned} FACT' \ FACT' &=_{\beta} \lambda n. \overline{\text{ifz}} \ n \ 1 \ (\overline{\text{times}} \ n \ (FACT' \ FACT' \ (\overline{\text{pred}} \ n))) \\ \text{i.e } FACT' \ FACT' &=_{\beta} FACT(FACT' \ FACT') \end{aligned}$$

Thus $FACT' \ FACT'$ is a fixed point of $FACT$.

For general function F define

$$\omega_F := \lambda f. F(f \ f)$$

Observe that

$$\omega_F \ \omega_F =_{\beta} F(\omega_F \ \omega_F)$$

i.e $\omega_F \ \omega_F$ is a fixed point of F .

Now define Y as follows:

$$\begin{aligned} Y &:= \lambda F. \omega_F \ \omega_F \\ &=_{\beta} \lambda F. (\lambda f. F(f \ f)) (\lambda f. F(f \ f)) \\ &=_{\alpha} \lambda F. (\lambda x. F(x \ x)) (\lambda x. F(x \ x)) \end{aligned}$$

Observe that

$$\begin{aligned} Y \ F &=_{\beta} F(Y \ F) \\ &=_{\beta} \omega_F \ \omega_F \\ &=_{\beta} F(\omega_F \ \omega_F) \end{aligned}$$

Therefore Y solves all fixed points. For example,

$$\begin{aligned} Y \ FACT &=_{\beta} FACT(Y \ FACT) \\ Y \ \overline{\text{succ}} &=_{\beta} \overline{\text{succ}} (Y \ \overline{\text{succ}}) \end{aligned}$$

Theorem 3.1. The λ definable functions on \mathbb{N} are precisely the partial recursive functions.

The proof of this theorem requires the following 2 steps.

1. Show that $\overline{0}$, $\overline{\text{succ}}$, $\overline{\text{pred}}$ e.t.c are definable.
2. Show that λ calculus can be encoded as partial recursive functions.

4 Decidability of β equivalence

Theorem 4.1. β equivalence is undecidable

There are two versions of the above claim, these are

Extensional version: \nexists λ term ε such that

$$\begin{aligned} \varepsilon t u &=_{\beta} \bar{\text{tr}} \text{ iff } t =_{\beta} u \\ \varepsilon t u &=_{\beta} \bar{\text{fa}} \text{ iff } t \neq_{\beta} u \end{aligned}$$

Intensional version: \nexists λ term δ such that

$$\begin{aligned} \delta \ulcorner t \urcorner \ulcorner u \urcorner &=_{\beta} \bar{\text{tr}} \text{ iff } t =_{\beta} u \\ \delta \ulcorner t \urcorner \ulcorner u \urcorner &=_{\beta} \bar{\text{fa}} \text{ iff } t \neq_{\beta} u \end{aligned}$$

Where $\ulcorner t \urcorner := \overline{\#t}$

$$\# : t \in \Lambda \longleftrightarrow n \in \mathbb{N}$$

Λ is the set of all λ terms.

Theorem 4.2. (*Rice's Theorem*).

No non trivial property of the behavior of λ terms is decidable.

There are two parts to the above theorem,

1. $\exists t_0 \in P, \exists t_1 \notin P$
2. $t \in P$ and $t =_{\beta} u \Rightarrow u \in P$

Proof: Suppose P is extensionally decidable

1. $\varepsilon t =_{\beta} \bar{\text{tr}} \text{ iff } t \in P$
2. $\varepsilon t =_{\beta} \bar{\text{fa}} \text{ iff } t \notin P$
3. $\varepsilon t =_{\beta} \bar{\text{tr}} \text{ or } \varepsilon t =_{\beta} \bar{\text{fa}}$

Crux:

$$d := \Upsilon (\lambda x. \varepsilon x t_1 t_0)$$

By fixed point property $f(\Upsilon f) =_{\beta} \Upsilon f$

So if, $f = (\lambda x. \varepsilon x t_1 t_0)$

$$\begin{aligned} d &:= \Upsilon (\lambda x. \varepsilon x t_1 t_0) =_{\beta} f(\Upsilon f) \\ &=_{\beta} fd \\ d &=_{\beta} \varepsilon d t_1 t_0 \end{aligned}$$

Cases:

1. $\varepsilon d =_{\beta} \bar{\text{tr}}$, so $d \in P$

$$\begin{aligned} d &=_{\beta} (\varepsilon d) t_1 t_0 \\ &=_{\beta} \bar{\text{tr}} t_1 t_0 \\ &=_{\beta} t_1, \quad t_1 \notin P \quad \equiv \text{contradiction} \end{aligned}$$

2. $\varepsilon d =_{\beta} \overline{fa}$, so $d \notin P$

$$\begin{aligned} d &=_{\beta} \varepsilon d t_1 t_0 \\ &=_{\beta} \overline{fa} t_1 t_0 \\ &=_{\beta} t_0, \quad t_0 \in P \quad \equiv \text{contradiction} \end{aligned}$$

Intensional version: Suppose P is intensionally decidable

$$\begin{aligned} \delta^{\ulcorner} t^{\urcorner} &=_{\beta} \overline{tr} \quad \text{iff } t \in P \\ \delta^{\ulcorner} t^{\urcorner} &=_{\beta} \overline{fa} \quad \text{iff } t \notin P \\ \delta^{\ulcorner} t^{\urcorner} &=_{\beta} \overline{tr} \quad \text{or} \quad \delta^{\ulcorner} t^{\urcorner} =_{\beta} \overline{fa} \quad \text{but not both} \end{aligned}$$

Fact: $\exists d \ s.t \ \varphi^{\ulcorner} d^{\urcorner} =_{\beta} d$, where $\varphi = \lambda x. \delta x t_1 t_0$
(Then conclude proof as previous).

Fact: $\forall \varphi \ \exists d \ \varphi^{\ulcorner} d^{\urcorner} =_{\beta} d$

Proof :

$$\begin{aligned} w &:= \lambda x. \varphi(Ap x (Num x)) \\ Z &:= w^{\ulcorner} w^{\urcorner} \\ &=_{\beta} \varphi(Ap^{\ulcorner} w^{\urcorner} (Num^{\ulcorner} w^{\urcorner})) \\ &=_{\beta} \varphi(\ulcorner w^{\urcorner} w^{\urcorner}) \\ &=_{\beta} \varphi(\ulcorner w^{\urcorner} w^{\urcorner}) \\ &=_{\beta} \varphi(\ulcorner Z^{\urcorner}) \end{aligned}$$

Where

$$\begin{aligned} Ap^{\ulcorner} t^{\urcorner} \ulcorner u^{\urcorner} &=_{\beta} \ulcorner t u^{\urcorner} \\ Num^{\ulcorner} w^{\urcorner} &=_{\beta} \ulcorner w^{\urcorner} \end{aligned}$$

5 Reduction

$t \longrightarrow_{\beta} u$ means β reduction.

$$\frac{}{(\lambda x. t)u \longrightarrow_{\beta} [u/x]t} \quad \frac{t_1 \longrightarrow_{\beta} u_1}{t_1 t_2 \longrightarrow_{\beta} u_1 t_2} \quad \frac{t_2 \longrightarrow_{\beta} u_2}{t_1 t_2 \longrightarrow_{\beta} t_1 u_2} \quad \frac{t \longrightarrow_{\beta} u}{\lambda x. t \longrightarrow_{\beta} \lambda x. u}$$

$t \longrightarrow_{\beta}^* u$ means zero or more steps of \longrightarrow_{β}

$$\frac{}{t \longrightarrow_{\beta}^* t} \quad \frac{t \longrightarrow_{\beta}^* v \quad v \longrightarrow_{\beta}^* u}{t \longrightarrow_{\beta}^* u} \quad \frac{t \longrightarrow_{\beta} u}{t \longrightarrow_{\beta}^* u}$$

5.1 Normal Form

$$nf_{\beta}(t) = \begin{cases} u, & \text{if } t \longrightarrow_{\beta}^* u \\ \uparrow & \text{otherwise} \end{cases}$$

Computation \equiv reduction to normal form (if any).

Note that $Y(\lambda x. x)$ has no normal form.

$$\begin{aligned} Y(\lambda x. x) &\longrightarrow_{\beta} (\lambda x. x)(Y(\lambda x. x)) \\ &\longrightarrow_{\beta} Y(\lambda x. x) \end{aligned}$$

5.2 Common reduct

Two λ terms t and u are said to have a common reduct, written $t \downarrow_{\beta} u$

$$\text{iff } \exists v. \begin{array}{l} t \longrightarrow_{\beta}^* v \\ u \longrightarrow_{\beta}^* v \end{array}$$

The common reduct v need not be a normal form.

Theorem 5.1. (*Church-Rosser Theorem*).

$$t \downarrow_{\beta} u \Leftrightarrow t =_{\beta} u$$

Proof:

Show $t \downarrow_{\beta} u \Rightarrow t =_{\beta} u$: This is easy !.

$$\begin{array}{l} \exists v. t \longrightarrow_{\beta}^* v, u \longrightarrow_{\beta}^* v \\ t \longrightarrow_{\beta}^* v \Rightarrow t =_{\beta} v \\ u \longrightarrow_{\beta}^* v \Rightarrow u =_{\beta} v \end{array}$$

Since $=_{\beta}$ is bidirectional, $t =_{\beta} u$

Show $t =_{\beta} u \Rightarrow t \downarrow_{\beta} u$: Try rule induction on definition of $=_{\beta}$ by showing the following

1. Reflexivity.
2. Symmetry.
3. Compatibility.
4. β axiom i.e $(\lambda x. t)u =_{\beta} [u/x]t$.
5. Transitivity, using $t \downarrow_{\beta} u$ and $u \downarrow_{\beta} v$ to show $t \downarrow_{\beta} v$. This is really hard.

Corollary:

$$(\lambda x. \lambda y. x \neq_{\beta} \lambda x. \lambda y. y)$$

Both are irreducible and different and thus have no common reduct.

6 Operational Semantics/Reduction Strategies

We will examine the following reduction strategies:

1. Call-by-value (CBV).
2. Call-by-name (CBN).

Properties of these reduction strategies:

1. Define a specific ordering of reduction.
2. Do not reduce inside λ abstractions and are therefore non normalizing. Defined only for closed terms.

$t \mapsto_{CBN} u$:

$$\frac{}{\lambda x. t \text{ value}} \qquad \frac{}{(\lambda x. t)u \mapsto_{CBN} [u/x]t} \qquad \frac{t \mapsto_{CBN} t'}{tu \mapsto_{CBN} t'u}$$

$t \mapsto_{CBV} u$:

$$\frac{}{\lambda x. t \text{ value}} \quad \frac{}{x \text{ value}} \quad \frac{u \text{ value}}{(\lambda x. t)u \mapsto_{CBV} [u/x]t} \quad \frac{t \mapsto_{CBV} t'}{tu \mapsto_{CBV} t'u} \quad \frac{t \text{ value} \quad u \mapsto_{CBV} u'}{tu \mapsto_{CBV} tu'}$$

Note the differences in what a *value* is, in both strategies.