DD2552 Seminar 2: Untyped Lambda Calculus

Karl Palmskog

KTH

Thursday August 31, 2023

Course material

- https://www.cs.cmu.edu/~rwh/pfpl/supplements/ulc.pdf
- https://github.com/ott-lang/ott/blob/master/tests/test10.7.ott

Why Untyped Lambda Calculus?

- Formalism for describing any(?) computation from 1930s
- Minimalistic syntax and semantics
- Lambda notation used in my functional languages

Syntax

t	$::=$			term
	$\mid$	x		variable
	$\mid$	$\lambda x.t$	bind x in t	lambda
	$\mid$	$t\,t'$		app
	$\mid$	(t)	S	
	$\mid$	$[t/x]t'$	M	
v	$::=$			value
	$\mid$	$\lambda x.t$		lambda

Semantics

$$\frac{}{(\lambda x.t_{12})\ v_2 \longrightarrow [v_2/x]t_{12}}\ \textsc{ax\_\_app}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t \longrightarrow t_1'\ t}\ \textsc{ctx\_\_app\_\_fun}$$

$$\frac{t_1 \longrightarrow t_1'}{v\,t_1 \longrightarrow v\,t_1'}\ \textsc{ctx\_\_app\_\_arg}$$

Free variables

$$\frac{}{x \in \mathrm{FV}(x)} \quad \textsc{var}$$

$$\frac{x \in \mathrm{FV}(t_1)}{x \in \mathrm{FV}(t_1\ t_2)} \quad \textsc{app\_l}$$

$$\frac{x \in \mathrm{FV}(t_2)}{x \in \mathrm{FV}(t_1\ t_2)} \quad \textsc{app\_r}$$

$$\frac{\begin{array}{l} x \in \mathrm{FV}(t) \\ x \neq y \end{array}}{x \in \mathrm{FV}(\lambda y.t)} \quad \textsc{lam}$$

α-equivalence

- we can equate terms that differ only in bound variable names
- such terms are called α-**equivalent**

$$\frac{}{t \equiv_\alpha t}\ \text{AEQ\_\_ID} \qquad \frac{t \equiv_\alpha t'}{t' \equiv_\alpha t}\ \text{AEQ\_\_SYM}$$

$$\frac{t \equiv_\alpha t' \quad t' \equiv_\alpha t''}{t \equiv_\alpha t''}\ \text{AEQ\_\_TRANS} \qquad \frac{t_1 \equiv_\alpha t_1' \quad t_2 \equiv_\alpha t_2'}{t_1\ t_2 \equiv_\alpha t_1'\ t_2'}\ \text{AEQ\_\_APP}$$

$$\frac{t \equiv_\alpha t'}{\lambda x.t \equiv_\alpha \lambda x.t'}\ \text{AEQ\_\_LAM} \qquad \frac{x' \notin FV(t)}{\lambda x.t \equiv_\alpha \lambda x'.[x'/x]t}\ \text{AEQ\_\_SUBST}$$

Properties of reduction

- order of applying reduction rules could vary
- do we still get “same” results?
- is there a point where no reduction rule applies?

β-equivalence

$$\frac{}{t \equiv_\beta t}\ \textsc{beq\_\_id} \qquad \frac{t \equiv_\beta t'}{t' \equiv_\beta t}\ \textsc{beq\_\_sym}$$

$$\frac{t \equiv_\beta t' \quad t' \equiv_\beta t''}{t \equiv_\beta t''}\ \textsc{beq\_\_trans} \qquad \frac{t_1 \equiv_\beta t_1' \quad t_2 \equiv_\beta t_2'}{t_1\ t_2 \equiv_\beta t_1'\ t_2'}\ \textsc{beq\_\_app}$$

$$\frac{t \equiv_\beta t'}{\lambda x.t \equiv_\beta \lambda x.t'}\ \textsc{beq\_\_lam} \qquad \frac{}{(\lambda x.t)\ t' \equiv_\beta [t'/x]t}\ \textsc{beq\_\_subst}$$

β-equivalence as a rewriting system

- does order of β rewriting matter?
  - no, by Church-Rosser theorem / confluence
- do we always reach "normal form"?
  - no, and no way to decide when we do

Encoding arithmetic

- Church numerals: $\lambda b.\lambda s.s^{(n)}b$
  - n-fold application of s
  - b is the base case
- recursive functions captured via Y combinator

$$\lambda proto.(\lambda this.proto\,(this\,this))\,(\lambda this.proto\,(this\,this))$$

Harper writes:

> *to define a recursive function, give it an "extra" first argument with which it may refer to itself, then ensure that whenever that function is called, it is implicitly applied to the function itself everywhere it is used, including the internal call sites at which a function calls itself.*