Lambda calculus

- Abstract syntax of lambda calculus
  - \( E ::= \text{constant} | \text{id} | l \text{id}. E | E E \)
- Beta reduction
  - \( (l \text{x}.e)f \mapsto e[f/x] \)
  - \( e[f/x] \) means replace all free occurrences of \( x \) with \( f \)
- Alpha conversion
  - \( (l \text{x}.e) \mapsto (l \text{y}.e[y/x]) \)
- Eta reduction
  - \( l \text{x}.(e \text{x}) \mapsto e \) (if \( x \) is not free in \( e \))

Normal form

- An expression is in normal form if it has no beta reductions
- Some expressions have no normal form
  - \( (\text{\textit{x}} \text{x} \text{x}) \) \( (\text{\textit{x}} \text{x} \text{x}) \)
- Expressions can sometimes be reduced multiple ways

Reduction choices

- Reduction strategy matters
  - \( (\text{\textit{y}}. (\text{\textit{z}}. \text{x} \text{y}) \mapsto (\text{\textit{z}}. \text{x} \text{y}) \) \)
- Normal order terminates if any do
  - outer most, left most redux first
- If reducing \( E \) by different strategies leads to two different normal forms, \( E_1 \) and \( E_2 \)
  - can alpha convert \( E_1 \) to \( E_2 \)

Boolean in \( \lambda \) calculus

- \text{True} = \( l \text{x}. l \text{y}. x \)
- \text{False} = \( l \text{x}. l \text{y}. y \)
- \text{Not} = \( l \text{b} \mapsto l \text{f}. l \text{x}. l \text{y}. \text{b} \text{y} \text{x} \)
- \text{And} = \( l \text{x} . l \text{y}. \text{x} \text{y} \) \text{False} \)
- \text{Or} = \( l \text{x} . l \text{y}. \text{x} \text{y} \) \text{True y} \)

Integers in \( \lambda \) calculus

- \text{Zero} = \( l \text{f}. l \text{x}. x \)
- \text{One} = \( l \text{f}. l \text{x}. f \text{x} \)
- \text{Two} = \( l \text{f}. l \text{x}. f \text{(f x)} \)
- \text{i} = \( l \text{f}. l \text{x}. \text{f} \text{x} \)
- \text{Plus} = \( l \text{m}. l \text{n}. l \text{f}. l \text{x}. (m \text{f}) \text{(n f x)} \)
Recursion

- Let rec fact = \[ \text{\(n\)}, \]
  if \(n < 2\) then 1 else \(n \times \text{fact}(n-1)\)
- We can define fact without letrec
- Use fixed point function \(Y\)
  - Let \(Y = \text{\(G. (\text{\(G(g g)\)})(\text{\(g. G(g g)\)})\)}\)
  - \(Y\) has no normal form
- let fact = \(Y f \cdot \text{\(n\). if \(n < 2\) then 1 else \(n \times \text{\(f(n-1)\)}\)}\)

beta expansion of fact

- \(\text{\(G. (\text{\(G(g g)\)})(\text{\(g. G(g g)\)})\)}\)
- \(\text{\(G(f. \text{\(n\). if \(n < 2\) then 1 else \(n \times f(n-1)\)}\)}\)}
- \(\text{\(G(g. (\text{\(f. \text{\(n\). f(n)\)})(g g))\)}\)}
- \(\text{\(g. G(f. \text{\(n\). f(n)\)})(g g))\)}
- \(\text{\(f. \text{\(n\). f(n)\)}(\text{\(g. (\text{\(f. \text{\(n\). f(n)\)})(g g))\)}\)}\)
- \(\text{\(g. (\text{\(f. \text{\(n\). f(n)\)})(g g))\)}\)

Table 18

- recursive types allow everything allowed in untyped lambda calculus
- \(\text{\(x\)}\) has no normal form
  - \(\text{\(x. (unfold_x x) x\)}\)
  - \(\text{\(fold_x (\text{\(x. (unfold_x x) x\)})\)}\)
- \(Y_x\) is the fixed point function
  - \(\text{\(E(A) A. (\text{\(x. (\text{\(f. (unfold_x x))\)} x))\)}\)
  - \(\text{\(fold_x (\text{\(x. (\text{\(f. (unfold_x x))\)} x))\)}\)

Table 19

- All untyped lambda calculus expressions can be given types
- All expressions have the type \(\text{\(\Box X\)}\)

Chapter 4, imperative types

- Any questions?

Second order types

- Allows for type parameters and abstraction
- Expression \(\text{\(X.M\)}\) indicates a function that takes a type, rather than a value, as a parameter
- Corresponding type \(\text{\(\Box X: A\)}\)
Universally quantified types

\[ \forall X \forall M : A \exists X.M : \forall X : A \]
\[ \exists M : \forall X : A \exists X.B : \exists M B : [B/X] A \]

Existentially quantified types

- used for defining abstract types
- There exists some representation of this abstract type, and you don’t need to know what it is

Defining a module

- boolModule : BoolInterface
  pack BoolInterface Bool = Unit+Unit
  with record(
      true = inLeftBool(unit)
      false = inRightBool(unit)
      cond = [A. True : Bool. [y_1 : A. [y_2 : A. case_x of x : Unit then y_1 | x_2 : Unit then y_2
  )

Using a module

open Real boolModule as
bool
boolOp:Record(
      true: Bool,
      false: Bool,
      cond: [A. Bool A A A]
)
in boolOp.cond Nat boolOp.true 1 0

Existential type rules

(Val Pack)
\[ (pack_{X,A} X-B with M): [X-A \]

(Val Open)
\[ [X,A] M : [X,A] N : B \]
\[ (open_{X,A} M as X, x:A in N):B \]
Basic rules

- Top is the set of all values
- Functions as discussed before
- Pairs and Unions:
  - elementwise covariant

Records and Variants

- Records
  - Can add new fields
  - For existing fields, covariant
- Variants
  - Can remove options
  - for retained options, covariant

F2<:

- $\forall X : A. M$
  - A function that takes any type $X$ s.t. $X <: A$ and returns $M$
- $\forall X : A. B$
  - corresponding type
- Note: $\forall X. M$ is the same as $\forall X : Top. M$

Universally quantified types

(Val Fun2<)

\[
\begin{array}{c}
\forall X : A. M : B \\
\forall X : A. M : \forall X : A. B
\end{array}
\]

(Val Appl2<)

\[
\begin{array}{c}
\forall X : A. B \\
\forall X : A. B \Rightarrow M : [A/X] B
\end{array}
\]

Existentially quantified types

- $\exists X : A. B$
  - $X$ is not completely known
  - but is known to be a subtype of $A$