

CMSC 631

Lambda calculus

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

9/12/02 CMSC 631

Normal form

- An expression is in normal form if it has no beta reductions
- Some expressions have no normal form
 - $(\lambda x.x x) (\lambda x.x x)$
- Expressions can sometimes be reduced multiple ways

9/12/02 CMSC 631

Reduction choices

- Reduction strategy matters
 - $(\lambda y. (\lambda z.z)) ((\lambda x.x x) (\lambda x.x x))$
- 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

9/12/02 CMSC 631

Boolean in λ calculus

- True = $\lambda x. \lambda y. x$
- False = $\lambda x. \lambda y. y$
- Not = $\lambda b. b \text{ False True}$
 - also equal to $\lambda b. \lambda x. \lambda y. (b y x)$
- And = $\lambda x. \lambda y. x y \text{ False}$
- Or = $\lambda x. \lambda y. x \text{ True } y$

9/12/02 CMSC 631

Integers in λ calculus

- Zero = $\lambda f. \lambda x. x$
- One = $\lambda f. \lambda x. f x$
- Two = $\lambda f. \lambda x. f (f x)$
- i = $\lambda f. \lambda x. f^i x$
- Plus = $\lambda m. \lambda n. \lambda f. \lambda x. (m f) ((n f) x)$

9/12/02 CMSC 631

Recursion

- Let $\text{rec fact} = \lambda n .$
 $\text{if } (n < 2) \text{ then } 1 \text{ else } n^*(\text{fact } (n-1))$
- We can define fact without letrec
- Use fixed point function Y
- Let $Y = \lambda G. (\lambda g. G(g g)) (\lambda g. G(g g))$
 – Y has no normal form
- let $\text{fact} = Y \lambda f .$
 $\lambda n . \text{if } (n < 2) \text{ then } 1 \text{ else } n^*(f (n-1))$

9/12/02 CMSC 631

beta expansion of fact

- $(\lambda G. (\lambda g. G(g g)) (\lambda g. G(g g)))$
 $(\lambda f . \lambda n . \text{if } (n < 2) \text{ then } 1 \text{ else } n^*(f (n-1)))$
- $(\lambda G. (\lambda g. G(g g)) (\lambda g. G(g g)))$
 $(\lambda f . \lambda n . F)$
- $(\lambda g. (\lambda f . \lambda n . F)(g g))$
 $(\lambda g. (\lambda f . \lambda n . F)(g g))$
- $(\lambda f . \lambda n . F)($
 $(\lambda g. (\lambda f . \lambda n . F)(g g))$
 $(\lambda g. (\lambda f . \lambda n . F)(g g))$

9/12/02 CMSC 631

Table 18

- recursive types allow everything allowed in untyped lambda calculus
- λ_A has no normal form
 – $(\lambda x:B . (\text{unfold}_B x) x)$
 $(\text{fold}_B (\lambda x:B . (\text{unfold}_B x) x))$
- Y_A is the fixed point function
 – $\lambda f:A \rightarrow A . (\lambda x:B . f ((\text{unfold}_B x) x))$
 $(\text{fold}_B (\lambda x:B . f((\text{unfold}_B x) x)))$

9/12/02 CMSC 631

Table 19

- All untyped lambda calculus expressions can be given types
- All expressions have the type $\lambda X.X \rightarrow X$

9/12/02 CMSC 631

Chapter 4, imperative types

- Any questions?

9/12/02 CMSC 631

Second order types

- Allows for type parameters and abstraction
- Expression $\lambda X.M$ indicates a function that takes a type, rather than a value, as a parameter
- Corresponding type $\lambda X:A$

9/12/02 CMSC 631

Universally quantified types

$$\frac{\Gamma, X \mid \Gamma \vdash M : A}{\Gamma \mid \Gamma \vdash \lambda X.M : \lambda X.A}$$

$$\frac{\Gamma \mid \Gamma \vdash M : \lambda X.A \quad \Gamma \mid \Gamma \vdash B}{\Gamma \mid \Gamma \vdash M B : [B/X] A}$$

9/12/02

CMSC 631

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

9/12/02

CMSC 631

Defining a module

- `boolModule : BoolInterface =`
`pack BoolInterface Bool = Unit+Unit`
with record(
`true = inLeftBool(unit)`
`false = inRightBool(unit)`
`cond = $\lambda A. \lambda x:\text{Bool}. \lambda y_1:A. \lambda y_2:A.$`
`caseA of $x_1:\text{Unit}$ then y_1`
`| $x_2:\text{Unit}$ then y_2`
`)`

9/12/02

CMSC 631

Using a module

```
openNat boolModule as
  Bool,
  boolOp:Record(
    true: Bool,
    false: Bool,
    cond:  $\lambda A. \text{Bool} \rightarrow A \rightarrow A \rightarrow A$ )

in
  boolOp.cond Nat boolOp.true 1 0
```

9/12/02

CMSC 631

Existential type rules

(Val Pack)

$$\frac{\Gamma \mid \Gamma \vdash [B/X] M : [B/X] A}{\Gamma \mid \Gamma \vdash (\text{pack}_{\Gamma X.A} X=B \text{ with } M) : \lambda X.A}$$

(Val Open)

$$\frac{\Gamma \mid \Gamma \vdash M : \lambda X.A \quad \Gamma, X, x:A \mid \Gamma \vdash N : B}{\Gamma \mid \Gamma \vdash (\text{open}_B M \text{ as } X, x:A \text{ in } N) : B}$$

9/12/02

CMSC 631

Subtypes

Basic rules

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

9/12/02 CMSC 631

Records and Variants

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

9/12/02 CMSC 631

F2<:

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

9/12/02 CMSC 631

Universally quantified types

(Val Fun2<:)

$$\lambda X.<:A \mid \lambda M : B$$

(Val Appl2<:)

$$\lambda M : \lambda X.<:A.B \mid \lambda A' <: A$$

$\lambda M A' : [A'/X] B$

9/12/02 CMSC 631

Existentially quantified types

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

9/12/02 CMSC 631

9/12/02 CMSC 631

