LAMBDA CALCULUS: PART II

Chandreyee Chowdhury

$$
\begin{aligned}
1 &\equiv \lambda sz.s(z)\\
2 &\equiv \lambda sz.s(s(z))\\
3 &\equiv \lambda sz.s(s(s(z)))
\end{aligned}
$$

COMPUTING WITH NATURAL NUMBERS

The successor function

successor n $=_{def}$ $\lambda n.\lambda f.\lambda x.f(nfx)$

Successor of 0 (S0) is $_{def}$ $(\lambda nfx.f(nfx))\ (\lambda sz.z)$

$$\lambda yx.y((\lambda sz.z)yx) = \lambda yx.y((\lambda z.z)x) = \lambda yx.y(x) \equiv 1$$

Successor of 1 (S1) is $_{def}$ $(\lambda wyx.y(wyx))\ (\lambda sz.s(z))$

Addition

$_{def}$ $\lambda m.\lambda n.\lambda f.\lambda x.\ m(f)(n(f)(x))$

2S3 is $_{def}$ $(\lambda sz.s(s(z)))(\lambda wyx.y(wyx))(\lambda uv.u(u(u(v))))$

Multiplication $_{def}$ $\lambda m.\lambda n.\lambda f.\lambda x.m(n(f))(x)$

SOLVING A LAMBDA EXPRESSION

The main goal of manipulating a lambda expression is to reduce it to a "simplest form" and consider that as the value of the lambda expression.

Definition : A lambda expression is in **normal form** if it contains no β-redexes (and no δ-rules in an applied lambda calculus), so that it cannot be further reduced using the β-rule or the δ-rule. An expression in normal form has no more function applications to evaluate. ▮

Expressions e ::= $\lambda x.\ x$ | e | $e_1\ e_2$

The only reduction possible for an expression in normal form is an a-reduction.

LAMBDA CALCULUS TO PROGRAMMING

Data Types

- Booleans, numbers
- Collections

Conditional expressions

Arithmetic expressions

Recursions

a ***combinator*** is a λ-term with no free variables

ENCODING BOOLEANS IN LAMBDA CALCULUS

What can we do with a boolean?

- we can make a binary choice
- *ConditionFunction (condition, then_do, else_do) {*
- *If (condition)*
- *retun then_do*
- *Else*
- *return else_do*
- }

- def **λcond.λthen_do. λelse_do.??**

ENCODING BOOLEANS IN LAMBDA CALCULUS

What can we do with a boolean?

- we can make a binary choice
- *ConditionFunction (condition, then_do, else_do) {*
- *If (true)*
- *return then_do*
- *Else*
- *return else_do*
- *}*
- True= $_{def}$ λ*then_do*. λ*else_do*.**then_do**
- **False=** $_{def}$ λ*then_do*. λ*else_do*.**else_do**

$_{def}$ λ*cond*.λ*then_do*. λ*else_do*.**?? then_do else_do**

BOOLEAN DATA TYPE

A boolean is a function that given two choices selects one of them

- true $=_{def}$ λ*then_do*. λ*else_do*. *then_do*
- false $=_{def}$ λ*then_do*. λ*else_do*. *else_do*
- if_then_else$=_{def}$ λ*cond*.λ*then_do*. λ*else_do*. Cond (*then_do*) (*else_do*)

Example: semester_time

SleepHours=if_then_else(semester_time) (six) (ten)

$If\_then\_else = (_{def}\ \lambda cond.\lambda then\_do.\ \lambda else\_do.\ \underline{Cond}\ (then\_do)\ (else\_do)\ )$

HANDLING BOOLEANS

NOT $= (_{def}\ \lambda boolean.\lambda then\_do.\ \lambda else\_do.\ \underline{boolean}\ (else\_do)\ (then\_do)\ )$

$(_{def}\ \lambda boolean.\lambda then\_do.\ \lambda else\_do.\ \underline{boolean}\ (else\_do)\ (then\_do)\ )\ (\mathbf{true})$

$= \lambda then\_do.\ \lambda else\_do.\ (\mathbf{true})\ (else\_do)\ (then\_do)$

$= \lambda then\_do.\ \lambda else\_do.\ (\lambda td.\ \lambda ed.\ td)\ (else\_do)\ (then\_do)$

$= \lambda then\_do.\ \lambda else\_do.\ else\_do$

Boolean	Outcome of the expression
true	false
false	true

Red_Green=tru

NOT(Red_Green)

NOT(NOT(Red_Green))

NOT(NOT(NOT(Red_Green)))

(five)(NOT)(Red_Green)

(four)(NOT)(Red_Green)

Is_even=

λn. n(NOT)(true)

MORE PREDICATES

(zero) $(\lambda x.\ (\text{decorated}))(\text{plain\_tree})$

$=(\lambda f.\lambda s.(s))\ (\lambda x.\ (\text{decorated}))(\text{plain\_tree})$

$=(\text{plain\_tree})$

$\lambda n.n(\lambda x.\ \text{decorated})(\text{plain\_tree})$

$\lambda n.n\ (\lambda x.\ \text{false})(\text{true})$

$\text{Is\_Zero}=(\lambda n.n\ (\lambda x.\ \text{false})\text{true})$

PREDICATES

is_zero=λn.n(λx. false)(true)

is_zero(zero)=zero(λx. false)(true)

=(λf.λs.(s)) (λx. false)(true)

=true

For all other cases the result is false

BOOLEAN DATA TYPE

A boolean is a function that given two choices selects one of them

- true $=_{def}$ λ*then_do*. λ*else_do*. *then_do*
- false $=_{def}$ λ*then_do*. λ*else_do*. *else_do*
- if_then_else= $_{def}$ λ*cond*.λ*then_do*. λ*else_do*. Cond (*then_do*) (*else_do*)

Example: Any_Assignment_Deadlines

SleepHours=if_then_else(Any_Assignment_Deadlines) (six) (ten)

NOT =($_{def}$ ***λboolean.λthen_do. λelse_do. boolean (else_do) (then_do)***)

Is_even=λn. n(NOT)(true)

Is_Zero= (λn.n (λx. false)true)

AND == λa.λb.a b FALSE
OR == λa.λb.a TRUE b

Boolean	Outcome of the expression
true	false
false	true