Contents

- Review : rules for eval
- Review : represent
- β-reduction, nam
- Reduction strategies and the Church Rosser theorem
- Different kinds of Normal Form

Review : rules for evaluation

- α-reduction
- β-reduction

- η-reduction

$\lambda x.(E\ x) \rightarrow E$ (*if x is*

- δ-rules — there is a separate δ-rule for each operator (suc -rule for $+$ says that $3 + 4$ evaluates to 7

Review : η-reduction example

-

tion)
- In context (left hand

$$\lambda x.((\lambda x.(x + 1))\ x)\ 45 \rightarrow (\lambda x.(x \quad on)$$

Review : representing numbers

- In the pure type-free λ-calculus there are no constants.
- In the previous lectu $\lambda f.\lambda x.(f\ x)$.
- This extends to all wh $\lambda x.(f\ (f\ (f\ x)))$

- What makes this an s functions to perform arithmetic on thsese representation. For example, it is possible to wr $\lambda x.\lambda y.E$ that will take a representation of the number 1 and a representation of the num to return a representation of the number 3.

- Challenge — can you write the λ-calculus function for ad f numbers ?

Name Clashes

- Example "name clash" with embedded function definitions using the same variable name :

$$(\lambda x.((\lambda x.(x+3))\ (x+4)))\ 5$$

- Using a normal orde the rule
  $(\lambda x.E)\ z\ \rightarrow$ h occurrences of x in
  E should be replac e x in "λx")

$$\rightarrow\ ((\lambda x.(5+3))$$

$$\rightarrow\ ((\lambda x.(x+3))$$

$$\rightarrow\ ((\lambda x.(5+3))\ (\ \ +\ \ ))\ \ ?$$

- NB : $E[z/x]$ means "for each *free* occurrence of x in E replace that x with z". It can help to annotate each occurrence of x according to whether it is bound or free, as follows : $E = ((\lambda x.(x_{bound}+3))\ (x_{free}+4))$. Thus, the correct reduction result is $((\lambda x.(x+3))\ (5+4))$

Free Variable Capture

- For β-reduction, identifying free variables in E is not enough !
- Consider the "free variable capture" problem as demonstrated by the following example subexpression wh … ee :[1]

- This β-reduces to the following, where *both* copies of t … e second a, previously free, has been "captured" and is now bound by the … the behaviour we want) :

$$(\lambda a.((\lambda f.(f\ a$$

- So β-reduction needs to be more sophisticated in the way that it operates.

1. Assume this subexpression is part of a larger enclosing expression which contains a lambda binding for the second a.

Avoiding Free Variable Capture

- During β-reduc at are bound inside E, t **before** performing the β-reduction subst

- Thus :

$$(\lambda f.(\lambda a.(f\ a)))\ \ (\lambda f.(f\ a))$$

$$(\lambda f.(\lambda b.(f\ b)))\ \ (\lambda f.(f\ a))$$

$$(\lambda b.((\lambda f.(f\ a$$

Reduction Strategies

- Any expression that matches the left-hand-side of a reduction rule is called a "reducible expression" or "redex"
- An expression cont

- **Execution is the su** β**-reduction) until Normal Form is reached.**
- Whether an arbitrary expression E has a NF is undecideabl Problem).

- Many different sequences of reductions are possible — how does thi

Church-Rosser Theorem

- The Church-Ross *that terminate* will converge on the sam

- Corollary : the Normal Form, for a given expression is unique (if it exists

- So β-reductions can be performed in any order (even in parallel!).

Normalising orders

- Not all reduction str

- So which should we c

- Normal Order Reduction ("leftmost-outermost-first") is guar n is possible
  - Strategies that are guaranteed to terminate are called "normalising" reduction orders

Comparing strategies

- Normal Order Reduction
  - “leftmost-outermost first”
  - Safe, but can be slow
  - Similar to “call-by-reference” passing of function arguments (though simple implementations can suffer from du

$$(3 + 5))$$

$$(\lambda x.3)\ \ ((\lambda x.(x\ x))\ (\lambda x.(x\ x))) \qquad \textit{by } \beta \textit{ reduction} \qquad 3$$

- Applicative Order Reduction
  - “leftmost-innermost first”
  - Fast, but unsafe (may not terminate)
  - Similar to “call by value” passing of function arguments

$$(\lambda x.(x + x))\ \ (3 + 5) \quad \rightarrow \textit{by } \delta \textit{ reduction} \quad \rightarrow \quad (\lambda x.(x + x))\ \ 8$$

$$(\lambda x.3)\ \ ((\lambda x.(x\ x))\ (\lambda x.(x\ x))) \quad \rightarrow \textit{by } \beta \textit{ reduction} \quad \rightarrow \quad (\lambda x.3)\ \ ((\lambda x.(x\ x))\ (\lambda x.(x\ x)))$$

Different kinds of Normal Form

- Practical implem ation)

- Weak Head Norm nts on the journey to full Normal Form
- The definitions consider all possible syntactic variants of an expres terms of the simple untyped λ-calculus only, where an exp application, or a lambda abstraction (a function definition).

2. If we were to add data constructors to the lambda calculus (which is not strictly necessary), we would extend the definitions appropriately.

Definition

An expression is in Normal

- Variable : x is in N
- Application : M , t a lambda abstraction
- Abstraction : $(\lambda x.E)$ is in Normal Form if E is in Norm

Normal Form is unique.

Definition

An expression M is in Head Normal Form if it is of the form
$M \equiv \lambda x_1 \ldots x_n \,.\, x\ N$

Note that in the above defin[illegible]

- Variable : x is in H[illegible] $= 0$)
- Application : $xN_1 \ldots N_m$ is in Head Normal Form (consider
- Abstraction : $\lambda x.E$ is in Head Normal Form if E is in H

Head Normal Form is not unique.

3. We assume that the variable x will be bound to a lambda abstraction by some enclosing expression — here we just consider whether this subexpression is in HNF.

Definition

An expression M is in Weak Head Normal Form if it is of one of the following two forms :

$M \equiv \lambda x_1 \ldots x_n \,.\, x \; N$

or

$M \equiv \lambda x.N$

- Variable : x is in Weak Head Normal Form
- Application : xN is in Weak Head Normal Form
- Abstraction : $\lambda x.E$ is in Weak Head Normal Form

Weak Head Normal Form is not unique.

Examples

- WHNF

- HNF

$$\lambda x.(+\ (5$$

- NF

$$\lambda x.(+\ 11)$$

Summary

- Review : rules for eva
- β-reduction, nam
- Reduction strategies and the Church Rosser theorem
- Different kinds of Normal Form