COMP0020 Functional Programming

Lecture 3

Further Lambda Calculus

Contents

- Review : rules for evaluation
- Review : representing numbers
- β-reduction, name clashes and Free Variable Capture
- Reduction strategies and the Church Rosser theorem
- Different kinds of Normal Form

Review : rules for evaluation

- α-reduction

$$\lambda x.E \;\rightarrow\; \lambda y.E[y/x]$$

- β-reduction

$$(\lambda x.E)\; z \;\rightarrow\; E[z/x]$$

- η-reduction

$$\lambda x.(E\; x) \;\rightarrow\; E \quad \textit{(if x is not free in E)}$$

- δ-rules — there is a separate δ-rule for each operator (such as $+, \times$); e.g. the δ-rule for $+$ says that $3 + 4$ evaluates to 7

Review : η-reduction example

- $$\lambda x.((\lambda x.(x+1))\ x) \rightarrow (\lambda x.(x+1)) \qquad (\textit{by } \eta \textit{ reduction})$$

- In context (left hand side) :

$$\lambda x.((\lambda x.(x+1))\ x)\ 45 \rightarrow (\lambda x.(x+1))\ 45 \qquad (\textit{by } \beta \textit{ reduction})$$

Review : representing numbers

- In the pure type-free λ-calculus there are no constants.
- In the previous lecture we said that we can represent 0 and 1 by λf.λx.x and λf.λx.(f x).
- This extends to all whole numbers – e.g. the number 3 is represented by λf.λx.(f (f (f x)))

- What makes this an *effective* representation is that we can write λ-calculus functions to perform arithmetic on thsese representation. For example, it is possible to write a function λx.λy.E that will take a representation of the number 1 and a representation of the number 2 and performs addition to return a representation of the number 3.

- Challenge — can you write the λ-calculus function for addition, given this representation of 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 order reduction strategy, the next step is a β-reduction using the rule $(\lambda x.E)\ z \quad \rightarrow \quad E[z/x]$ where $z = 5$ and $E = ((\lambda x.(x+3))\ (x+4))$, but which occurrences of x in E should be replaced with z during the β-reduction ? (NB we would never replace the x in "λx")

$$\rightarrow \quad ((\lambda x.(5+3))\ (x+4)) \quad ?$$

$$\rightarrow \quad ((\lambda x.(x+3))\ (5+4)) \quad ?$$

$$\rightarrow \quad ((\lambda x.(5+3))\ (5+4)) \quad ?$$

- 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 where the first a is bound and the second a (in the argument) is free : [1]

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

- This β-reduces to the following, where *both* copies of the name a are now bound (the second a, previously free, has been "captured" and is now bound by the λa, which is **not** the behaviour we want) :

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

- 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-reduction of $(\lambda x.E)\ z$, if z is an expression that contains any free variables that are bound inside E, then each such free variable must be α-converted inside E **before** performing the β-reduction substitution.

- Thus :

$$(\lambda f.(\lambda a.(f\ a)))\ \ (\lambda f.(f\ a)) \quad \rightarrow \alpha \textit{ reduce } \lambda a \textit{ to } \lambda b$$

$$(\lambda f.(\lambda b.(f\ b)))\ \ (\lambda f.(f\ a)) \quad \rightarrow \textit{by } \beta \textit{ reduction}$$

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

Reduction Strategies

- Any expression that matches the left-hand-side of a reduction rule is called a "reducible expression" or "redex"
- An expression containing no redexes is said to be in "Normal Form" (or "NF")
- **Execution is the successive application of reduction rules (primarily β-reduction) until Normal Form is reached.**
- Whether an arbitrary expression E has a NF is undecideable (it is equivalent to the Halting Problem).
- Many different sequences of reductions are possible — how does this affect the result ?

Church-Rosser Theorem

- The Church-Rosser theorem states that all reduction sequences (strategies) *that terminate* will converge on the same Normal Form.

- 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 strategies (orders of performing reductions) terminate

- So which should we choose?

- Normal Order Reduction ("leftmost-outermost-first") is guaranteed to terminate if termination 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 duplication)

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

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

- 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 implementations almost never reduce to full Normal Form (to avoid wasted computation)

- Weak Head Normal Form (WHNF) and Head Normal Form (HNF) are successive stopping points on the journey to full Normal Form
- The definitions consider all possible syntactic variants of an expression. Here we give definitions in terms of the simple untyped λ-calculus only, where an expression can either be a variable, an application, or a lambda abstraction (a function definition). [2]

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 Form if it does not contain a redex

- Variable : x is in Normal Form
- Application : MN is in Normal Form if M, N are in Normal Form and M is not a lambda abstraction
- Abstraction : $(\lambda x.E)$ is in Normal Form if E is in Normal Form

Normal Form is unique.

Definition

An expression M is in Head Normal Form if it is of the form
$M \equiv \lambda x_1 \ldots x_n \,.\, x\ N_1 \ldots N_m$ *where* $n, m \geq 0$ *and* x *is a variable*

Note that in the above definition x is the "head variable"

- Variable : x is in Head Normal Form (consider the definition where $n = m = 0$)
- Application : $xN_1 \ldots N_m$ is in Head Normal Form (consider $n = 0, m \geq 1$) [3]
- Abstraction : $\lambda x.E$ is in Head Normal Form if E is in Head Normal Form

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 :
$M \equiv \lambda x_1 \ldots x_n \,.\, x\ N_1 \ldots N_m$ *where* $n, m \geq 0$ *and* x *is a variable*
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

$$\lambda x.((\lambda y.(+))\ 4\ (5+6))$$

- HNF

$$\lambda x.(+\ (5+6))$$

- NF

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

Summary

- Review : rules for evaluation
- β-reduction, name clashes and Free Variable Capture
- Reduction strategies and the Church Rosser theorem
- Different kinds of Normal Form

END OF LECTURE