Lecture 26: Normal Forms and Reduction

MATH230

Te Kura Pāngarau | School of Mathematics and Statistics
Te Whare Wānanga o Waitaha | University of Canterbury

Outline

1. Examples
2. β-reduction Strategies
3. Encoding Booleans

Example

Perform a step of β-reduction on the following β-redex

$$(\lambda x.\ x\ (\lambda y.\ x\ y))\ z$$

Example

Use α-reduction to relabel this λ-expression so that there are no clashes between bound and free variables.

$$(\lambda x.\ x\ (\lambda y.\ x\ y))\ y$$

Now reduce the expression using β-reduction.

Reduction Strategies

One λ-expression can consist of multiple β-redex. If this is the case, then different strategies may give different outcomes.

$$(\lambda y.\ \lambda z.\ z)\ ((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))$$

If we are to define an automatic model for computation, then we can't have any ambiguity in the process. One needs to decide ahead of time how to deal with these choices.

Two primary strategies are known as call-by-name and call-by-value.

Call-by-name

Once the leftmost β-redex is identified, this strategy calls the leftmost abstraction leaving the input (potentially) unevaluated.

$$(\lambda y.\ \lambda z.\ z)\ ((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))$$

This strategy is also known as either *normal order* or *lazy evaluation*.

Call-by-value

Once the leftmost β-redex is identified, this strategy calls the innermost abstraction. Calls the value of the outermost β-redex.

$$(\lambda y.\ \lambda z.\ z)\ ((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))$$

This strategy is also known as either *strict evaluation* or *eager evaluation*.

Normal Form

We say a λ-expression is in normal form if it does not contain any β-redex. If a λ-expression is β-equivalent to a λ-expression in normal form, then it is unique upto α-reduction.

If a λ-expression has a normal form, then call-by-name evaluation will result in that normal form.

Observation: As a consequence of the above, if you can show that call-by-name evaluation does not terminate, then you may conclude that the λ-expression does not have a normal form.

Example: Call-by-Value

Perform a step of β-reduction on the following β-redex

$$(\lambda y.\ y\ a)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda u.\ u)\ z))$$

Example: Call-by-Name

Perform a step of β-reduction on the following β-redex

$$(\lambda y.\ y\ a)\ ((\lambda x.\ x)\ (\lambda z.\ (\lambda u.\ u)\ z))$$

Reduction Graphs

You may come across graphs of λ-expressions for which the different paths through the graph represent different evaluation strategies.

$$(\lambda x.\ 3 \cdot x)\ ((\lambda x.\ x + 1)\ 2)$$

Common Expressions

Some expressions that are common enough, or illustrative of some point, have been granted names. As we saw in the previous example, when writing λ-expressions it can be enough to know the name - until we need to evaluate that expression on some input.

$$I = \lambda x.\ x$$
$$\omega = \lambda x.\ x\ x$$
$$\Omega = \omega\omega$$
$$\text{Apply} = \lambda f.\ \lambda a.\ (f\ a)$$

We will now talk about how we can encode ideas from logic and arithmetic into the λ-calculus. This requires giving λ-expression encoding for Booleans (True/False), natural numbers, and arithmetic functions.

Conditional Execution

Remember: everything (!) is a λ-expression.

If we want to implement logic in the λ-calculus, then we need λ-expressions that *behave like* True and False, propositions, propositional connectives, quantifiers etc.

How do True and False *behave*? One use is conditional branching.

$$\mathsf{COND} = \lambda f.\ \lambda g.\ \lambda c.\ ((c\ f)\ g)$$

TRUE

COND is an expression with three arguments: a, b, c. Which should be equivalent to the first argument a if $c = \text{TRUE}$ and the second argument b if $c = \text{FALSE}$.

$$\text{COND } a\ b =_\beta \lambda c.\ ((c\ a)\ b)$$

If we just put in the first two arguments, we get a function waiting for a Boolean to tell it what to do.

Our λ-expression for TRUE needs to ignore the second input b

$$\text{TRUE} = \lambda x.\ \lambda y.\ x$$

TRUE

Perform β-reduction on the following λ-expression until the λ-expression is in normal form.

$$(((\text{COND } a)\ b)\ \text{TRUE}) = ((((\lambda f.\ \lambda g.\ \lambda c.\ ((c\ f)\ g))\ a)\ b)\ (\lambda x.\ \lambda y.\ x))$$

FALSE

If the λ-expression for TRUE ignores the second function, then λ-expression for FALSE needs to ignore the first argument.

$$\text{FALSE} =$$

FALSE

Perform β-reduction on the following λ-expression until the λ-expression is in normal form.

$$((\text{COND } a)\ b)(\text{FALSE}) =_\beta \lambda c.\ ((c\ a)\ b)(\lambda x.\ \lambda y.\ y)$$

Propositional Connectives

TRUE and FALSE should behave appropriately with some implementation in λ-expressions of the propositional connectives

NOT

AND

OR

IMP(LICATION)

NAND

The trick is to think about these in terms of selecting the first or second argument.

Example: If the first input to AND is TRUE, then which input should the AND expression return?

For some of these, it may just be easier to build them once you have NOT, OR, and AND.

NOT

A	NOT A
T	F
F	T

$\mathsf{TRUE} = \lambda x.\ \lambda y.\ x$

$\mathsf{FALSE} = \lambda x.\ \lambda y.\ y$

EXAMPLE

Perform β-reduction on the following λ-expression until the λ-expression is in normal form.

NOT FALSE

Further Reading

Here are some recommended reading to follow up on the lecture content. Some are freely available online.

Type Theory and Functional Programming, *Simon Thompson*

- Stanford Encyclopedia Articles:
  1. The Lambda Calculus
  2. Type Theory
  3. Church's Type Theory

Church's original papers are worth visiting, although more work than Turing's paper.
An Unsolvable Problem of Elementary Number Theory, *Alonso Church*

A Note on the Entscheidungsproblem, *Alonso Church*