Lambda Calculus Part 2

Adam Wyner
CS3518, Spring 2017
University of Aberdeen

Plan of Lecture

1. Conversion
2. Normal Forms
3. The Church-Rosser Theorem
4. Normal Order Conversion
5. Applicative Order Conversion
6. Normal vs. Applicative Order Conversion
7. Normal Graph Conversion

Lambda Conversion

- λ-Calculus: conversion (rewrite) rules to manipulate λ-terms.
- $M \to N$: application of one or more conversion rules.
- A key part is the substitution of N for the free occurrences of x in M, that is, $M[N/x]$. Examples:

$$
\begin{array}{ll}
x[N/x] & \equiv N \\
y[N/x] & \equiv y, \text{ if } y \not\equiv x \\
M1\ M2[N/x] & \equiv M1[N/x]\ M2[N/x] \\
(\lambda x.M)[N/x] & \equiv (\lambda x.M), \text{ if no } x \text{ free in } M; \\
(\lambda y.M)[N/x] & \equiv \lambda y.(M\ [N/x]), \text{ if } y \not\equiv x
\end{array}
$$

Lambda Conversion - Example

- $((\lambda x.xy)y)[w/y] \equiv$
- $(\lambda x.xy)[w/y]\ y[w/y] \equiv$
- $(\lambda x.(xy[w/y]))\ w \equiv$
- $(\lambda x.xw)w$

- $((\lambda x.xy)y)[w/x] \equiv$
- $(\lambda x.xy)[w/x]\ y[w/x] \equiv$
- $(\lambda x.(xy[w/x]))\ y \equiv$
- $(wy)\ y$

α-Conversion

- The name of bound variables isn't important:

  $\lambda x.x2 \equiv \lambda y.y2 \equiv \lambda w.w2$

- Formally:

  $\lambda x.M \rightarrow_{\alpha} \lambda y.M[y/x]$

provided y does not occur in M so as not to bind a variable that is otherwise free or bound to some other operator.

This is the same conversion as alphabetic variation in predicate logic. Return to this a bit later.

β-Conversion

- β-Conversion is the rule for function application:

  $(\lambda x.x^2)2 \rightarrow 2^2 \rightarrow 4$

- Formally, if no free variable in N occurs bound in M then:

  $(\lambda x.M)N \rightarrow_\beta M[N/x]$

β-Conversion Example

- Function composition example:

$$(\lambda x.x^2)((\lambda y.2 * y)4) \rightarrow_\beta (\lambda x.x^2)(2 * 4)$$
$$\rightarrow_\beta (\lambda x.x^2\ )8$$
$$\rightarrow_\beta 8^2$$
$$\rightarrow_\beta 64$$

- A function is applied to an argument.
- Computation equals simplification (with β conversion).
- Can be done in a sequence of conversions.

η-Conversion

- η-Conversion is the rule for transforming a lambda expression:

  $\lambda x.Mx <\text{--->}_\eta M$

  provided that x does not occur as a free variable in M.

  The expressions are equivalent.
- η-reduction (left to right) is useful to eliminate redundant lambda abstractions, e.g. the lambda abstraction passes its argument (x) to another function M.
- η-abstraction (right to left) is useful to create an explicit predicate, e.g. a predicate P (the set of things of which P is true) to a predicate λx.P(x) (same set, but explicit).

Potential Confusion with Bound Variables

- Naive application of function application can cause problems due to clashes of variables.

- For instance:

$$(\lambda xy.x * y)y \to_\beta (\lambda y.x * y)[y/x] \equiv (\lambda y.y * y)$$

- To avoid this, apply α-conversion to rename problem variables, before applying the substitution operator:

$$\begin{aligned}(\lambda xy.x * y)y \quad &\to_\alpha (\lambda xz.x * z)y \\ &\to_\beta (\lambda z.x * z)[y/x] \\ &\equiv (\lambda z.y * z)\end{aligned}$$

Normal Forms

- A λ-term is in its normal form if we cannot apply any β- or η-conversions to it.
- Examples of λ-terms in normal form are:

1	z
xy	λx.y
λa.a (λb.b + 1)	λa.λb.a b a

- The following λ-terms are not in normal form:

  (λx.y) z => λx.y   λa.(λy.y^3) a => λy.y^3

- β-conversion can be applied to the left one, and η-conversion to the right one.

Finding Normal Forms

- A term in normal form is one which cannot be reduced any further.
- It can therefore be seen as the end of a sequence of computations — i.e. as the "result" of the function.
- The normal form of the expression (λx.x + x)(2 + 4) is 12:

  $(\lambda x.x+x)(2+4) \rightarrow (\lambda x.x+x)6$

  $\rightarrow_{\beta} 6+6$

  $\rightarrow 12$
- "Computation" in the λ-calculus is the process of applying the conversion rules until a normal form is found.

The Church-Rosser Theorem (1)

- One of the most important theoretical results of the λ-calculus.
- It guarantees that different orders of evaluating subparts of a λ-term always yield the same normal formal.
- A λ-term M is convertible to N, denoted by M → N, by repeatedly applying any of the three conversion rules.

The Church-Rosser Theorem (2)

- The Church-Rosser theorem states that for any λ-terms M and N:

  if $M \rightarrow N$ then there is a λ-term L such that

  $M \rightarrow L$ and $N \rightarrow L$

- This theorem has an important corollary, which says that normal forms are unique up to α-conversion.
- In other words, each λ-expression has a unique normal form.
- There are several proofs in the literature.

The Church-Rosser Theorem (3)

- However, there are two complications...
- One complication is that not all λ-expressions possess a normal form. For instance,

  $(\lambda x.x\ x)\ (\lambda x.x\ x) \rightarrow (\lambda x.x\ x)\ (\lambda x.x\ x) \rightarrow \cdots$
- The Church-Rosser corollary, correctly stated, therefore is:

  normal forms (if they exist) are unique (up to α-conversion).
- Another complication is that there may be more than one applicable conversion rule to a given λ-expression, hence there may be more than one way in which to generate the normal form.

The Church-Rosser Theorem (4)

- $(\lambda x.\lambda y.x\ y)\ a\ ((\lambda x.\lambda y.y\ x)\ a\ b)$

may be reduced to normal form in (at least) two ways:

$$\begin{aligned}
(\lambda x.\lambda y.x\ y)\ a\ ((\lambda x.\lambda y.y\ x)\ a\ b) \quad &\rightarrow_\beta (\lambda x.\lambda y.x\ y)\ a\ ((\lambda y.y\ a)\ b) \\
&\rightarrow_\beta (\lambda x.\lambda y.x\ y)\ a\ (b\ a) \\
&\rightarrow_\beta (\lambda y.a\ y)\ (b\ a) \\
&\rightarrow_\beta a\ (b\ a)
\end{aligned}$$

and

$$\begin{aligned}
(\lambda x.\lambda y.x\ y)\ a\ ((\lambda x.\lambda y.y\ x)\ a\ b) \quad &\rightarrow_\beta (\lambda y.a\ y)\ ((\lambda x.\lambda y.y\ x)\ a\ b) \\
&\rightarrow_\beta a\ ((\lambda x.\lambda y.y\ x)\ a\ b) \\
&\rightarrow_\beta a\ (\lambda y.y\ a)\ b) \\
&\rightarrow_\beta a\ (b\ a)
\end{aligned}$$

The Church-Rosser Theorem (5)

- Computation does involve some element of choice.
- The choice of conversion sequence can be significant.
- Consider the expression (λx.λy.y) ((λx.x x) (λx.x x))
- By applying β-conversion to the outermost function application, this expression can be reduced to normal form in one step:

  $$(\lambda x.\lambda y.y)\ ((\lambda x.x\ x)\ (\lambda x.x\ x)) \rightarrow_{\beta} (\lambda y.y)$$

- If, instead, we choose to start with the argument term, we get:

  $$\begin{aligned}(\lambda x.\lambda y.y)\ ((\lambda x.x\ x)\ (\lambda x.x\ x)) &\rightarrow_{\beta} (\lambda x.\lambda y.y)\ ((\lambda x.x\ x)\ (\lambda x.x\ x)) \\ &\rightarrow_{\beta} \ldots\end{aligned}$$

The Church-Rosser Theorem (6)

- At each step, we must select
  1. which conversion rule to apply next, and
  2. which part of the λ-expression to convert next.
- Some conversion sequences may not find the normal form of some λ-expression, even though one may exist.
- In other words, some conversion sequences are infinitely long.
- Fortunately, there is one strategy which is guaranteed to find the normal form if it exists.

Normal Order Conversion

- The 2nd Church-Rosser Theorem states that:

  If M → N and N is in normal form, then there is a normal order conversion sequence from M to N.
- A normal order conversion sequence is one in which at each step, the leftmost, outermost element of the expression is the one which is converted.
- In MN all conversions applicable to M are performed before any conversion is done on N.
- In MN, if M is a function abstraction, then the β-conversion applying M to N must be performed before any other conversions are performed.

Applicative Order Conversion

- An alternative evaluation strategy is applicative order conversion:

  At each step, the arguments to a function application are evaluated fully, before the function application itself is evaluated (leftmost-innermost).

- In MN,
  1. M is reduced to its normal form, then
  2. N is reduced to its normal form, and finally
  3. M is applied to N.
- Normal order conversion ≡ lazy evaluation.
- Applicative order conversion ≡ eager evaluation.

Normal vs. Applicative Order Conversion

- Applicative order conversion can sometimes be more efficient than normal order conversion, that is, it can provide a shorter conversion sequence.
- However, applicative order conversion is unsafe: it may fail to terminate.
- Sometimes, normal order conversion is more efficient.
- Normal order conversion is safe — it is guaranteed to find the normal form if one exists.

Normal Graph Conversion

- Normal graph conversion is a variant of normal order conversion, which retains its safe properties but avoids some of the redundant computation.
- Duplicate terms in a λ-expression are replaced by pointers to a single copy of the term.
- Recall the lecture in Knowledge Based Systems about Jess Efficiency using a RETE Network.... Similar idea here.

Two Additions

- Add:
  - Examples of constructing complex expressions, handling compositionality:
    - λx.P(x)
    - λy.R(y)
    - λx(λx.P(x)(z) and λy.R(y)(x)) =>
      - λx(P(z) and R(x))
  - Higher order types, e.g. generalised quantifiers:
    - λP.λx.P(x) λv.R(y,v) => λx.λv.R(y,v)(x) => λx.R(y,x)

Translation from Syntax to Semantics

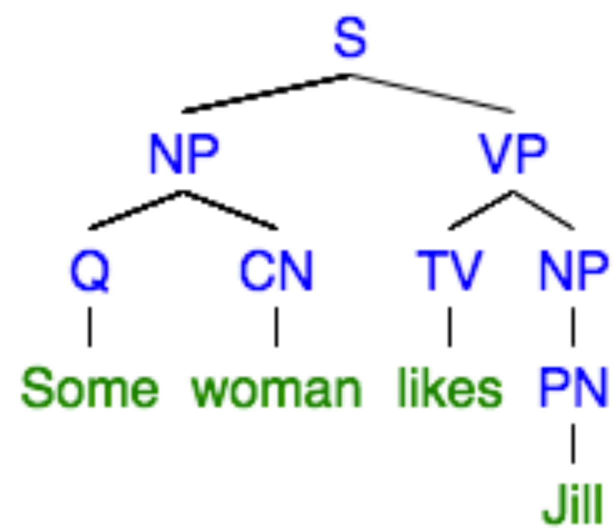
- The core idea is to provide a syntactic parse of a sentence, then to translate each word and phrase in the sentence into a corresponding semantic representation.
- Richard Montague (1970) *English as a Formal Language*.

  "I reject the contention that an important theoretical difference exists between formal and natural languages."
- Step 1: provide a syntactic parse (a tree).
- Step 2: apply the semantic translation rules.
- Step 3: apply β-conversion.

Syntax/Phrase Structure Rules

- PN -> Bob, Jill, Phil
- CN -> dog, cat, man, woman
- Q -> every, some
- TV -> likes, pushes
- IV -> sings, runs
- NP -> PN
- NP -> Q CN
- VP -> IV
- VP -> TV NP
- S -> NP VP

- Some woman likes Jill.
- $\exists$ y (woman'(y) & likes'(y,jill'))



Semantic Rules

- I (PN) = I (Jill), I (Phil), ...
- I (CN) = I (cat),

I (woman), ...

- I (cat) = λx cat' (x),

I (woman) = λx woman' (x)

- I (Q) = I (some), ...
- I (some) =

$\lambda R\, \lambda P\, \exists y(R (y) \,\&\, P (y))$

- I (TV) = I (likes), ...
- I (likes) = $\lambda y\, \lambda x$ likes' (x,y)

- I (NP) = I (PN)
- I (NP) = I (Q) (I (CN))
- I (VP) = I (IV)
- I (VP) = I (TV) (I (NP))
- I (S) = I (NP) (I (VP))

Translation on the Tree

I(S) =

I(NP)(I(VP)) =

$\lambda P \exists y\, (woman'(y)\ \&\ P(y))\ \lambda v(likes'(v,jill'))$ =

$\exists y\, (woman'(y)\ \&\ \lambda v(likes'(v,jill'))(y)$ =

$\exists y\, (woman'(y)\ \&\ likes'(y,jill'))$

I(NP) =

I(Q)(I(CN)) =

$\lambda R \lambda P \exists y\, (R(y)\ \&\ P(y))\ (woman')$ =

$\lambda P \exists y\, (woman'(y)\ \&\ P(y))$

I(Q) =

I(some) =

$\lambda R \lambda P \exists y\, (R(y)\ \&\ P(y))$

I(CN) =

I(woman) =

woman'

I(VP) =

I(TV)(I(NP)) =

$\lambda u \lambda v\, (likes'(v,u))\ (jill')$ =

$\lambda v\, (likes'(v,jill))$

I(TV) =

I(likes) =

$\lambda u \lambda v\, (likes'(v,u))$

I(NP) =

I(PN) =

I(Jill) =

jill'