

Reduction Strategies

COMP3610 – Principles of Programming Languages

Ben Lippmeier

Australian National University

Semester 2, 2009

Mechanising reduction

In the previous lecture we defined a reduction relation:

$$(\lambda x.M) N \rightarrow M[x := N]$$

This can be applied to *any redex* that appears in a term, for example:

$$\frac{(\lambda x.M) ((\lambda y.P) N)}{(\lambda x.M) ((\lambda y.P) N) \rightarrow M[x := (\lambda y.P) N]}$$
$$(\lambda x.M) ((\lambda y.P) N) \rightarrow \lambda x.M (P[y := N])$$

If we wish to *mechanise* reduction (e.g. write a reduction program) we must decide on a way of choosing *which redex* to reduce next.

(We don't consider parallel or non-deterministic approaches.)

Next redex

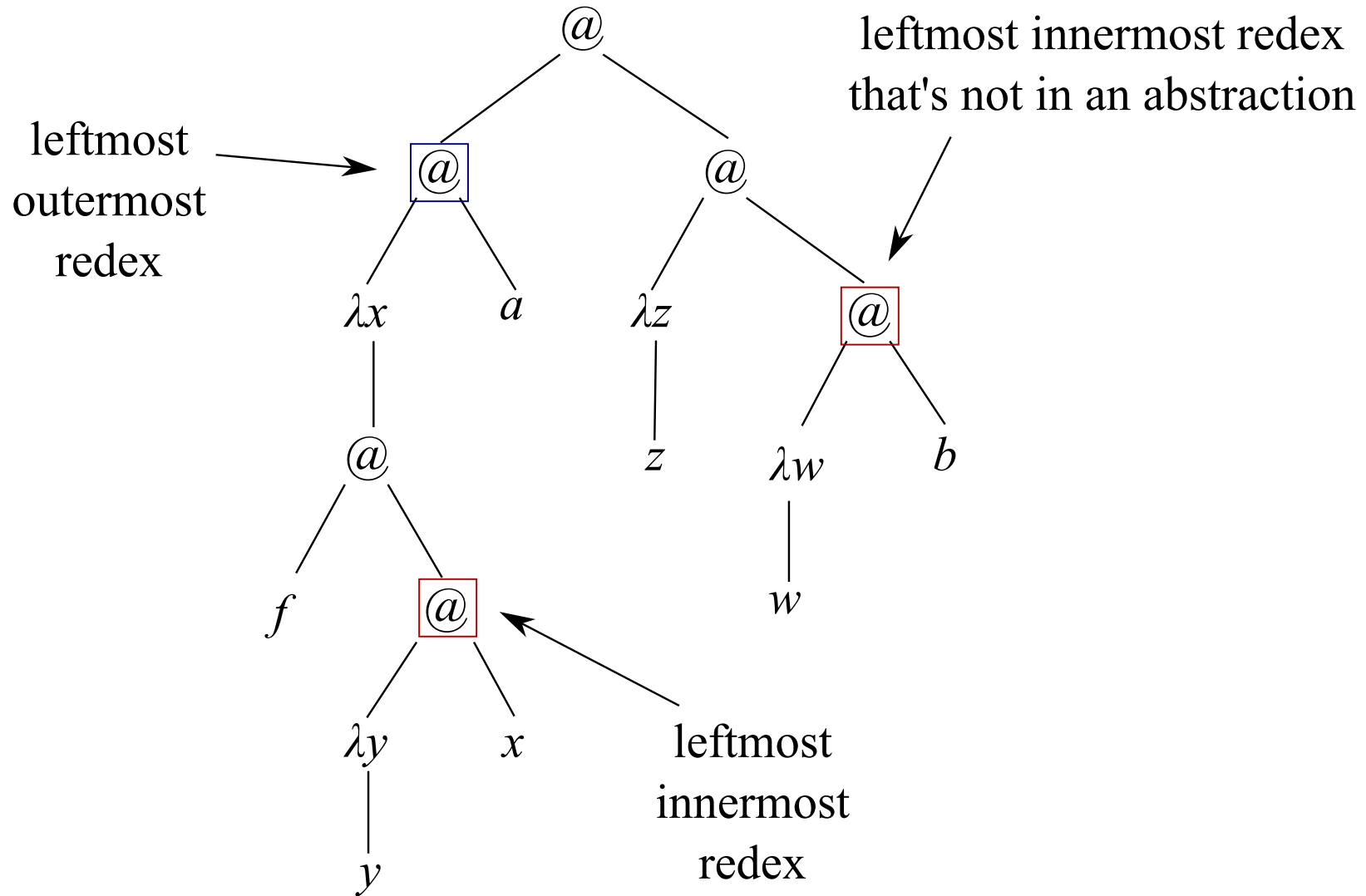
There are two important cases:

- leftmost-innermost
- leftmost-outermost

An *innermost redex* is a redex that *contains* no other redex.

An *outermost redex* is a redex that *is contained by* no other redex.

	<i>advantage</i>	<i>disadvantage</i>
<i>innermost</i>	reduce argument exactly once	argument may not be needed
<i>outermost</i>	don't reduce argument if not needed	may generate several occurrences of argument redex to reduce



Variations on normal form

- Depending on the application, we may not be interested in reducing all the way to normal form.
- For example in Haskell, if the result is a function there is no need to do evaluation inside the function body.
- Four important variations are on the next slide.

V is the particular normal form. (read: V = value).

M is an arbitrary term.

Normal form (no remaining redexes):

$$V ::= \lambda x.V \mid x V_1 \dots V_n$$

Head normal form :

$$V ::= \lambda x.V \mid x M_1 \dots M_n$$

Weak head normal form (the term whole is not a redex):

$$V ::= \lambda x.M \mid x M_1 \dots M_n$$

Weak normal form (don't reduce inside abstractions):

$$V ::= \lambda x.M \mid x V_1 \dots V_n$$

Reduction strategies

When we combine a mechanism for choosing the next redex (innermost or outermost) with a particular notion of normal form, we get a *reduction strategy*.

Call by name: *outermost* reduction to *weak head normal form*

Normal order: *outermost* reduction to *normal form*

Call by value: *innermost* reduction to *weak normal form*

Applicative order: *innermost* reduction to *normal form*

Strict languages (e.g. ML, O’Caml) correspond to *call by value*.

They evaluate the function argument before substituting.

- All function arguments are evaluated, irrespective of whether their values will be used or not.

```
choose b x y = if b then x else y
```

Non-strict languages (e.g. Haskell) correspond to *call by name*.

They substitute the argument before evaluating it.

- If a function’s argument is never used it is never evaluated.

```
from n      = n : from (n + 1)
fac n      = product (take n (from 1))
```

Normalising strategies

A strategy is **normalising** if it is guaranteed to reach normal form, if it exists.

Normal order and *call by name* are normalising.

Applicative order and *call by value* are not normalising.

(In Haskell, a non-terminating argument may never be evaluated. In ML it will always be evaluated.)

Exercise:

Try reducing $(K\ I\ \Omega)$ on the lambda workbench using different strategies. $K\ I\ \Omega = (\lambda x\ y.\ x)\ (\lambda y.\ y)\ ((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))$

The following slides give definitions of the four reduction strategies using big-step operational semantics.

Call by name

Outermost redex, to whnf:

$$x \downarrow_{bn} x$$

$$\lambda x.M \downarrow_{bn} \lambda x.M$$

$$\frac{M \downarrow_{bn} \lambda x.P \quad P[x := N] \downarrow_{bn} Q}{M N \downarrow_{bn} Q}$$

$$\frac{M \downarrow_{bn} P \not\equiv \lambda x.Q}{M N \downarrow_{bn} P N}$$

Example implementation:

```
reduceCBN e
= case e of
  XVar _      -> e
  XAbs _ _    -> e
  XApp m n
    -> let m' = reduceCBN m
        in case m' of
          XAbs x p -> reduceCBN (subNoCap x n p)
          _       -> XApp m' n
```

Normal order reduction

Outermost redex, to nf:

$$x \downarrow_{no} x$$

$$\frac{M \downarrow_{no} N}{\lambda x.M \downarrow_{no} \lambda x.N}$$

$$\frac{M \downarrow_{bn} \lambda x.P \quad P[x := N] \downarrow_{no} Q}{M N \downarrow_{no} Q}$$

$$\frac{M \downarrow_{bn} P \neq \lambda x.Q \quad P \downarrow_{no} R \quad N \downarrow_{no} T}{M N \downarrow_{no} R T}$$

The blue parts emphasise reductions beyond whnf. Compare with call by name. Note the use of (\downarrow_{bn}) to limit reduction before substitution.

Call by value

Innermost redex, to wnf:

$$x \downarrow_{bv} x$$

$$\lambda x.M \downarrow_{bv} \lambda x.M$$

$$\frac{M \downarrow_{bv} \lambda x.P \quad N \downarrow_{bv} Q \quad P[x := Q] \downarrow_{bv} R}{M N \downarrow_{bv} R}$$

$$\frac{M \downarrow_{bv} P \not\equiv \lambda x.Q \quad N \downarrow_{bv} R}{M N \downarrow_{bv} P R}$$

The green parts emphasise evaluation of arguments before substitution.
Compare with call by name.

Applicative order

Innermost redex, to nf:

$$x \downarrow_{ao} x$$

$$\frac{M \downarrow_{ao} N}{\lambda x.M \downarrow_{ao} \lambda x.N}$$

$$\frac{M \downarrow_{ao} \lambda x.P \quad N \downarrow_{ao} Q \quad P[x := Q] \downarrow_{ao} R}{M N \downarrow_{ao} R}$$

$$\frac{M \downarrow_{ao} P \not\equiv \lambda x.Q \quad N \downarrow_{ao} R}{M N \downarrow_{ao} P R}$$

The red part emphasises only difference with call by value: reductions under lambda.

Example comparison

$$K I \Omega = (\lambda x y. x) (\lambda z. z) ((\lambda x. x x) (\lambda x. x x))$$

Call by name:

$$\frac{\lambda x y. x \downarrow_{bn} \lambda x y. x \quad (\lambda y. x)[x := I] \downarrow_{bn} \lambda y. \lambda z. z}{\frac{K I \downarrow_{bn} \lambda y. \lambda z. z \quad (\lambda z. z)[y := \Omega] \downarrow_{bn} \lambda z. z}{(K I) \Omega \downarrow_{bn} \lambda z. z}}$$

Call by value:

$$\frac{\lambda x y. x \downarrow_{bv} \lambda x y. x \quad I \downarrow_{bv} I \quad (\lambda y. x)[x := I] \downarrow_{bv} \lambda y. \lambda z. z}{\frac{K I \downarrow_{bv} \lambda y. \lambda z. z \quad \Omega \uparrow_{bv}}{(K I)\Omega \uparrow_{bv}}}$$

Exercises

- Both the lambda calculus workbench, and the lambda interpreter support various reduction strategies. Step through a few examples to make sure you understand the differences.
- Invent a single example expression whereby a different redex will be chosen as the first to be reduced for call-by-name, call-by-value, and applicative order reduction. Interpret “first to be reduced” as the redex who’s substitution is encountered first in a depth first, left to right walk of the proof tree. Can you do the same for normal-order reduction as well?
- (bonus) Extend the lambda interpreter with the hybrid applicative order reduction described in Sestoft’s paper “Demonstrating Lambda Calculus Reduction” available on the course website.