COMP0020 Functional Programming

Contents

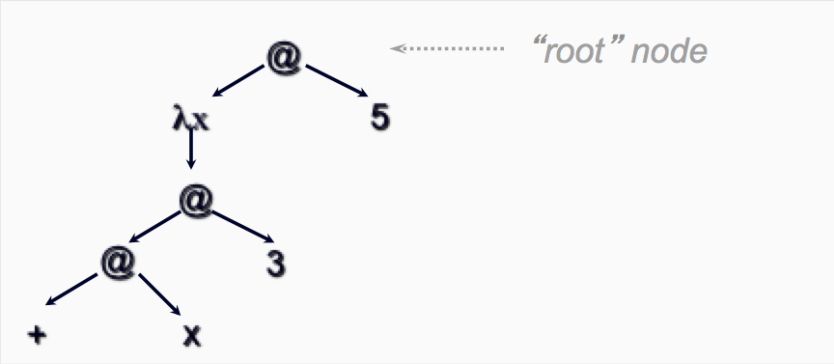
- Representation of lambda expressions:
  - Abstract
  - Physical
- Performing a beta reduction
- Reduction orders
- Lazy and strict evaluation
- Parallel evaluation

Abstract representation (1)

$$f\ x\ =\ x\ +\ 3$$

$$main\ =\ f\ 5$$

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



Abstract representation (2)

$$z = 3$$
$$f\ x\ y\ =\ x\ +\ y$$
$$main\ =\ f\ z\ z$$
$$(\lambda z\ .\ ((\lambda x\ .\ (\lambda y\ .\ (x\ +\ y)))\ z\ z\ ))\ 3 \qquad (sharing)$$

Syntax tree after first beta reduction :

Abstract representation (3)

Now consider a recursive function definition :

$f\ x$

$main = f\ 5$

$$(Y\ (\lambda\ f\ .\ (\lambda\ x\ .\ (if\ (x = 0)\ 3\ (1 + (f\ (x - 1))))))\ )\ 5$$

(Y (λ f . (λ x . (if (x=0) 3 (1+(f (x-1))))))) 5

Alternative abstract representation

Physical representation

- Each node (and leaf) of the graph is held in
- Position in memory is irrelevant
  - Just because two nodes occupy ad hey are related
  - Related nodes are connected using pointers (addresses)
- Set aside a chunk of memory just to hold these nodes
  - The “HEAP"

Physical representation (2)

- Each node requires sufficient memo
  - A tag (@, *I*, "*op*", *IND*, .
  - A pointer to a left subtree (alternati r or a variable)
  - A pointer to a right subtree (alternatively, a variable or a constan
- We call each such small chunk of memory a "cell"
- The heap therefore consists of many cells

Physical representation (3)

- The heap :

Performing a beta reduction

- Make a COPY of the function body
- Substitute the actual parameters fo
- OVERWRITE the root node of the expression with an indirection to the root node of the copy (*preserves sharing*)

Beta reduction (2)

Beta reduction (2)

- A “redex" is a reducible expression, i.e. an expression that can be reduced by α, β, η or δ-rule reduction
- A program may contain many redexe
- Evaluation of a program is a process of successive transformatio e graph until the final value is found :
  1. Decide which redex to reduce next
  2. Reduce it
  3. Loop to 1.

Redexes

Reduction orders

- There will often be many redexes that ce
- The evaluator must choose an EVALUATION ORDER : this m n the performance of the system (but not on its (partial) correctness)

Reduction orders

- NORMAL ORDER EVALUATION
  - starts at the top node and follows the left pointers until the first redex is found
  - leftmost outermost
  - lazy evaluation.
- APPLICATIVE ORDER EVALU
  - leftmost innermost redex
  - strict evaluation
- PARALLEL EVALUATION
  - referential transparency
  - redexes can be reduced in any order
  - may be reduced concurrently !

Summary

- Physical representation of lambda e
- Performing a beta reduction
- Reduction orders
- Lazy and strict evaluation
- Parallel evaluation