Computing Science (CMPUT) 325

Nonprocedural Programming

Department of Computing Science
University of Alberta

Exercises for Evaluation using Contexts and Closures

- Notation:
- Context: CT = $\{n_1 \rightarrow v_1, ..., n_k \rightarrow v_k\}$
- Closure: `[fun, CT]`, where `fun` is a lambda function and CT represents a context
- We assume that the given expressions are evaluated in some context CT0
- We could encounter the expression “in the middle of an evaluation”
- Of course, CT0 could be empty, but we want to solve the evaluation in the general case, for *any* context we might be in

Exercise 1

- `((lambda (x y z) (f x y z)) 2 3 4)`
- Show the context when `(f x y z)` is being evaluated
- Answer: {x→2, y→3, z→4} ∪ CT0

Exercise 1 Detailed Steps

- The expr. is an application of the form `(fun 2 3 4)`
- eval arguments 2, 3, 4 in CT0 - they are constants which evaluate to themselves
- eval `fun = (lambda (x y z) (f x y z))` in CT0 - evaluate to closure `[fun, CT0]`
- Extend context CT0 with
  new bindings {x$\rightarrow$2, y$\rightarrow$3, z$\rightarrow$4}
- Evaluate body `(f x y z)`
  in extended context {x$\rightarrow$2, y$\rightarrow$3, z$\rightarrow$4} $\cup$ CT0

Exercise 2

```
(((lambda (x y) (lambda (x) (+ x y))) 2 3) 4)
```

- Show the context when `(+ x y)` is being evaluated
- Answer: eval `(+ x y)` in {x→4,x→2,y→3} ∪ CT0
- Why? Details on next slides

Exercise 2 Explained

eval

```
(((lambda (x y) (lambda (x) (+ x y))) 2 3) 4)
```

- An application of form `(e 4)` where

  `e = ((lambda (x y) (lambda (x) (+ x y))) 2 3)`
- `e` should (after some steps) evaluate to a closure containing a lambda function, and that function can then be applied to the evaluated argument 4.
- First eval arg. 4 in CT0, then eval `e` in CT0.
- eval 4 in CT0: returns 4.
- eval `e` in CT0: next two slides

Exercise 2 Explained

- eval `e = ((lambda (x y) (lambda (x) (+ x y))) 2 3)` in CT0
- `e` is an application of the form `(e1 2 3)` where `e1 = (lambda (x y) (lambda (x) (+ x y)))`
- We need to eval args 2 and 3, then `e1`, all in CT0
- 2 and 3 evaluate to 2 and 3
- `e1` is a lambda function, **(not an application)**, so it evaluates to a closure `[e1, CT0]`
- Now we can apply the `e1` in the closure to evaluated arguments 2, 3:

Exercise 2 Explained

- Apply `e1` from closure `[e1, CT0]`
  to evaluated arguments 2,3:
- Extend context: CT1 = {x→2, y→3} ∪ CT0
- Note: the context-to-extend, CT0,
  the argument list `(x y)` and the function body all **come from the closure** `[e1, CT0]`
- Body of `e1` is: `e2 = (lambda (x) (+ x y))`
- Eval `e2` in CT1: it is a lambda function,
  evaluates to a closure `[e2, CT1]`
- Now we are done with eval `e` in CT0:
  the result is the closure `[e2, CT1]`
- So, we have reduced the initial call `(e 4)`
  to `([e2, CT1] 4)`
- Next, we go ahead with this function application
  (next slide)

Exercise 2 Explained

- Function application `([e2, CT1] 4)`
  where `[e2, CT1]` is the result of eval `e` in CT0
- `e2` has argument list `(x)` and body `(+ x y)`
- Bind argument: $x \rightarrow 4$
- Extend context CT1: CT2 = $\{x \rightarrow 4\} \cup$ CT1
  = $\{x \rightarrow 4, x \rightarrow 2, y \rightarrow 3\} \cup$ CT0
- Eval body `(+ x y)` in CT2
- `(+ 4 3)`, application of built-in function +, so 7
- Note: contexts are accessed from left to right.
  The more local variable binding $x \rightarrow 4$ is chosen,
  not the outer one $x \rightarrow 2$.
  No renaming by α-reduction is needed when using
  contexts and closures.

Exercise 3

```
((lambda (x) (x 2)) (lambda (x) (+ x 1)))
```

- Show the context when `(x 2)` is being evaluated
- Answer: eval `(x 2)` in {x→ `[f2, CT0]`} ∪ CT0, where `f2 = (lambda (x) (+ x 1))`
- Application is of form `(fun arg)`, with `fun = (lambda (x) (x 2))` and `arg = (lambda (x) (+ x 1))`
- First, eval `arg`, then eval `fun`, both in CT0

Exercise 3 Explained

- Both `arg` and `fun` are functions (**not applications**), so both eval to a closure
- eval `(lambda (x) (+ x 1))` in CT0 = `[f2, CT0]` where `f2 = (lambda (x) (+ x 1))`
- eval `(lambda (x) (x 2))` in CT0 = `[f1, CT0]` where `f1 = (lambda (x) (x 2))`
- After evaluating both `arg` and `fun` in `(fun arg)` we have application `([f1, CT0] [f2, CT0])`

Exercise 3 Explained

- Evaluate the application `([f1, CT0] [f2, CT0])`
- Arg list `(x)` and body `(x 2)` from `[f1, CT0]`
- Evaluated argument `[f2, CT0]`
- New variable binding: x$\rightarrow$ `[f2, CT0]`
- Extend context: CT1 = {x$\rightarrow$ `[f2, CT0]`} $\cup$ CT0
- Answer: eval `(x 2)` in CT1 = {x$\rightarrow$ `[f2, CT0]`} $\cup$ CT0

Exercise 3 Explained

- `((lambda (x) (x 2)) (lambda (x) (+ x 1)))`
- Now, show the context when `(+ x 1)` is being eval'd
- Evaluate application `(x 2)`
  in context CT1 = {x→ `[f2, CT0]`} ∪ CT0
- Eval argument 2 in CT1: 2
- Eval `x` in CT1: `[f2, CT0]`
- For application, get parameter list `(x)`, body `(+ x 1)`,
  and context-to-extend CT0 from closure `[f2, CT0]`
- New variable binding: x→2
- Extend context CT0:
- CT2 = {x→2} ∪ CT0
- Answer: eval `(+ x 1)` in CT2 = {x→2} ∪ CT0

Exercise 4 (less detail given)

- `((lambda (x y) (+ x y))`
  `((lambda (y z) (+ y z)) 2 3) 4)`
- Show the context when `(+ y z)` is evaluated.
- Answer: {y→2, z→3} ∪ CT0
- We have `(fun arg1 arg2)` with
- `fun = (lambda (x y) (+ x y))`
  `arg1 = ((lambda (y z) (+ y z)) 2 3)`
  `arg2 = 4`
- Show the context when `(+ x y)` is evaluated
- Answer: {x→5, y→4} ∪ CT0
- `arg1` is an application, fully evaluated we get 5.