Excercise 1.6

Josh H

January 4, 2024

Eva defines a new version of `if`:

```
(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
(else else-clause)))
```

We trace the demonstration.

```
(new-if (= 2 3) 0 5)
(new-if #f 0 5)
(cond (#f 0)
      (else 5))
5
```

5

```
(new-if (= 1 1) 0 5)
(new-if #t 0 5)
(cond (#t 0)
      (else 5))
0
```

0

Now we find what happens when Alyssa attempts to use this to compute square roots. We evaluate the expression `(sqrt-iter 1.0 2)`.

```
(new-if (good-enough? 1.0 2)
1.0
(sqrt-iter (improve 1.0 2) 2))
```

The first operand is evaluated and is false. Ideally, we would evaluate the alternative and return its value. However `new-if` is not a special form, and we proceed with evaluating the combination.

```
(new-if #f
1.0
(sqrt-iter (improve 1.0 2) 2))
```

We evaluate further and we get that the third operand itself has `sqrt-iter` as a subexpression.

```
(new-if #f
1.0
(sqrt-iter 1.5 2))
```

We cannot yet substitute the formal parameters of `new-if` for the arguments because `sqrt-iter` needs to be evaluated.

```
(new-if #f
1.0
(new-if (good-enough? 1.5 2)
1.5
(sqrt-iter (improve 1.5 2) x)))
```

Each time `sqrt-iter` is evaluated, it expands into another `new-if`. If `good-enough?` is true, `new-if` never evaluates to `guess` because the body of `new-if` never has its formal parameters replaced for the `cond` inside.

Had Alyssa used `if`, the interpreter would first evaluate the predicate then evaluate either the consequent or the alternative.