Practice Problems - Lecture 16 - Solutions

Nano1 Semantics

What is the next step in the reduction of each of these terms? Prove it by building the derivation tree using our rules for the operational semantics of Nano1.

1) `0 + (2 + (1 + 3))`

Solution: The next step is `0 + (2 + 4)`. We can derive this by:

```
[Add]                 --------------
                       1 + 3   => 4
[Add-R]          ------------------------
                  2 + (1 + 3)  => 2 + 4
[Add-R] ------------------------------------
         0 + (2 + (1 + 3)) => 0 + (2 + 4)
```

2) `let x = 5 in x + let x = 3 in x`

Solution: The next step is `5 + let x = 3 in x` because

`(x + let x = 3 in x)[x := 5]  =  5 + let x = 3 in x`.

We can derive this by:

```
[Let] ------------------------------------------------------
       let x = 5 in x + let x = 3 in x => 5 + let x = 3 in x
```

NanoB Semantics

Let's consider another very simple language (call it NanoB) where we just have true, false, and a conditional expression (but no integers). Here is the syntax:

```
e ::= v                          -- values
    | if e1 then e2 else e3      -- conditionals

v ::= true
    | false
```

Here are the operational semantics rules:

```
                                  e1 => e1'
[If]          ---------------------------------------------
               if e1 then e2 else e3 => if e1' then e2 else e3

[If-True]   if true then e2 else e3 => e2

[If-False]  if false then e2 else e3 => e3
```

3) What is the next step in the reduction of each of this term? Write down the derivation tree.

```
if (if false then true else false) then false else true
```

Solution: The next step is `if false then false else true`. We can derive this by:

```
[If-False]                    ------------------------------------------
                               if false then true else false    =>   false
[If] ------------------------------------------------------------------------
      if (if false then true else false) then false else true
                                          => if false then false else true
```

4) Notice that in our operational semantics above the guard is evaluated first and at most one of the branches is ever evaluated. Suppose we wanted to change our operational semantics so that the then and else branches are evaluated *before* the guard is evaluated. Write a set of reduction rules that give an implementation of this semantics. What kind of design decisions do you have to make if you want these semantics to be deterministic?

Solution: To make the semantics deterministic we have to choose whether to evaluate the then-branch or the else-branch first. Another consideration is if the guard is already a value (true or false) do we immediately reduce to the corresponding branch? Or would we still evaluate the branches first?

In the following rules, for example, let's choose to evaluate the then-branch first, then the else-branch, and only look at the guard after both branches have been fully reduced. Note that only one rule can ever apply to reduce any particular term:

```
                                e2 => e2'
[If-Then]    ------------------------------------------------
              if e1 then e2 else e3 => if e1 then e2' else e3

                                e3 => e3'
[If-Else]    ------------------------------------------------
              if e1 then v2 else e3 => if e1 then v3 else e3'

                                e1 => e1'
[If-Guard]   ------------------------------------------------
              if e1 then v2 else v3 => if e1' then v2 else v3

[If-True]    if true then v2 else v3 => v2

[If-False]   if false then v2 else v3 => v3
```