Practice Problems - Lecture 17 - Wed Feb 22

Nano1 Theorems

Here are some additional practice theorems for the Nano1 language covered in lecture. As a reminder, our syntax is

```
e ::= x                            -- variables
    | v                            -- values
    | e1 + e2                      -- addition
    | let x = e1 in e2             -- let-bindings

v ::= n                            -- values
```

and our operational semantics rules are:

```
               e1 => e1'           -- premise
[Add-L]    --------------------
           e1 + e2 => e1' + e2     -- conclusion

              e2 => e2'
[Add-R]    --------------------
           n1 + e2 => n1 + e2'

[Add]      n1 + n2 => n           where n == n1 + n2

                        e1 => e1'
[Let-Def]  --------------------------------------
           let x = e1 in e2 => let x = e1' in e2

[Let]      let x = v in e2 => e2[x := v]
```

The theorems in these exercises may sound obvious but will give you practice with structuring an inductive argument Try proving the following theorem by induction on `e`.

1) If `e` is a closed term (no free variables) then for any `x` and any value `v` we must have `e[x := v] = e`.

Try proving the following theorem by induction on the reduction relation `e => e'`.

2) If `e` is a closed term and `e'` is a term such that `e => e'` then `e'` is also closed. In other words, the property of "being closed" is preserved under reduction.

NanoB Theorems

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
```

We can define a term `size` function for `NanoB` as follows

```
size true                    = 1
size false                   = 1
size (if e1 then e2 else e3) = size e1 + size e2 + size e3
```

We can prove termination for NanoB in the a similar manner as we did in lecture for Nano1. Prove the following two lemmas for NanoB:

3) **Lemma 1.** For any `e`, `size e > 0`. [*Hint:* Use induction on `e`.]

4) **Lemma 3.** For any `e, e'` such that `e => e'`, then we have `size e' < size e`. [*Hint:* Proceed by induction on `e => e'`]

With these two lemmas in hand we could now give the same proof of Termination because the proof in lecture combined Lemmas 1 and 3 above without using any specific properties of the `Nano1` syntax or semantics.

Theorem I [Termination]: For any expression `e` there exists `e'` such that `e =~> e'`.