Ellie Makin (erm67)

Semantics supervision work

1. **Write a program to compute the factorial of the integer initially in location l_1. Take care to ensure that your program really is an expression in L1.**

```
(*
    The syntax isn't well defined, so assume things mean
    what it seems like they are supposed to mean.
*)
l2 := !l1;
while l1 := !l1 + -1; !l1 >= 1 do {
    l3 := !l2;
    l4 := 1;
    while l4 := !l4 + 1; !l1 >= !l4 do {
        l3 := !l3 + !l2
    };
    l2 := !l3
};
!l2
```

2. **Give full derivations of all the reduction steps of**
   $\langle (l_0 := 7); (l_1 := (!l_0 + 2)), \{l_0 \mapsto 0, l_1 \mapsto 0\}\rangle$

$$
\begin{aligned}
&\langle (l_0 := 7); (l_1 := (!l_0 + 2)), \{l_0 \mapsto 0, l_1 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle \text{skip}\,; (l_1 := (!l_0 + 2)), \{l_0 \mapsto 7, l_1 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle (l_1 := (!l_0 + 2)), \{l_0 \mapsto 7, l_1 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle (l_1 := (7 + 2)), \{l_0 \mapsto 7, l_1 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle (l_1 := 9), \{l_0 \mapsto 7, l_1 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle \text{skip}\,, \{l_0 \mapsto 7, l_1 \mapsto 9\}\rangle \\
\not\longrightarrow\ &
\end{aligned}
$$

3. **Give full derivations of the first four reduction steps of the $\langle e, s\rangle$ of the first L1 example on Slide 22.**

$$
\begin{aligned}
&\langle l_2 := 0; \text{while}\, !l_1 \geq 1\, \text{do}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1), \{l_1 \mapsto 3, l_2 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle \text{skip}\,; \text{while}\, !l_1 \geq 1\, \text{do}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1), \{l_1 \mapsto 3, l_2 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle \text{while}\, !l_1 \geq 1\, \text{do}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1), \{l_1 \mapsto 3, l_2 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle \text{if}\, !l_1 \geq 1\, \text{then}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1); \text{while}\, !l_1 \geq 1\, \text{do}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1) \\
&\quad \text{else skip}\,, \{l_1 \mapsto 3, l_2 \mapsto 0\}\rangle \\
\longrightarrow\ &\langle \text{if}\, 3 \geq 1\, \text{then}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1); \text{while}\, !l_1 \geq 1\, \text{do}\, (l_2 := !l_2 + !l_1; l_1 := !l_1 + -1) \\
&\quad \text{else skip}\,, \{l_1 \mapsto 3, l_2 \mapsto 0\}\rangle
\end{aligned}
$$

4. **Adapt the implementation code to correspond to the two rules (op1b) and (op2b) on Slide 44. Give some test cases that distinguish between the original and the new semantics.**

```
| Op (e1,opr,e2) ->
    (match (e1,opr,e2) with
    | (Integer n1, Plus, Integer n2) -> Some(Integer (n1+n2), s)   (*op + *)
    | (Integer n1, GTEQ, Integer n2) -> Some(Boolean (n1 >= n2), s)(*op >=*)
    | (e1,opr,e2) -> (
        if (is_value e2) then
          (match reduce (e1,s) with
          | Some (e1',s') -> Some (Op(e1',opr,e2),s')     (* (op2b) *)
          | None -> None )
        else
          (match reduce (e2,s) with
          | Some (e2',s') -> Some(Op(e1,opr,e2'),s')      (* (op1b) *)
          | None -> None ) ) )
```

This would affect cases such as $(l_0 := 1; !l_1) + !l_0$. If both locations were initially 0, then using the original code this gives 1, and in the new code it gives 0.

7. **Give a type derivation for $(l_0 := 7); (l_1 := (!l_0 + 2))$ with $\Gamma = l_0 : \texttt{intref}, l_1 : \texttt{intref}$.**

For brevity, I've only done one half of this tree.

$$\dfrac{\dfrac{l_1 : \texttt{intref} \quad \{l_0 : \texttt{intref}, l_1 : \texttt{intref}\} \vdash 7 : \texttt{int}}{\{l_0 : \texttt{intref}, l_1 : \texttt{intref}\} \vdash (l_0 := 7) : \texttt{unit}} \quad \nabla}{\{l_0 : \texttt{intref}, l_1 : \texttt{intref}\} \vdash (l_0 := 7); (l_1 := (!l_0 + 2)) : \texttt{unit}}$$

9. **Does Type Preservation hold for the variant language with rules (assign1') and (seq1')? on Slide 45? If not, give an example, and show how the type rules could be adjusted to make it true.**

No, because the typing rule (assign) says that an assignment has a type `unit`. We can change this rule to say that the return type is `int`, but then we also have to add another (seq) rule where the left-hand expression is of type `int`. Then it will have type preservation.

10. **Adapt the type inference implementation to match your revised type system from Exercise 9.**

```
    | Assign (l,e) ->
        (match (lookup gamma l, infertype gamma e) with
        | (Some Ty_intref,Some Ty_int) -> Some Ty_int
        | _ -> None)
    ...
    | Seq (e1,e2) ->
       (match (infertype gamma e1, infertype gamma e2) with
       | (Some Ty_unit, Some t2) -> Some t2
       | (Some Ty_int, Some t2) -> Some t2
       | _ -> None )
```