Computer Science 131, Spring 2001
Sample Solution for Assignment 3

Out: Wednesday, February 7

Due: Wednesday, February 14, 11:00am

1 Mini-ML (50%)

1. **Dynamic Semantics**

   Show all of the state transitions required to evaluate the given programs to value. For each step, cite the number of the rule or rules needed to justify that step; the rules are shown in Appendix A. You may use "..." to avoid recopying chunks of code that do not change from one step to another, as long as your intent is completely clear.

   (a)
   ```
   let x be 3 in
     let f be (fun g(y:Int):Int is x+y) in
       let x be 4 in
         f(x)
   ```

   → by Rule 18
   ```
   let f be (fun g(y:Int):Int is 3+y) in
     let x be 4 in
       f(x)
   ```

   → by Rule 18
   ```
   let x be 4 in
     (fun g(y:Int):Int is 3+y)(x)
   ```

   → by Rule 18 `(fun g(y:Int):Int is 3+y)(4)`
   → by Rule 21 `3+4`
   → by Rule 7 `7`

   (b)
   ```
   let fact be (fix g(y) is if y<=0 then 1 else y*g(y+(-1)))
   in
       fact 2
   ```

→ by Rule 18

```
(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(2)
```

→ by Rule 21

```
if 2<=0 then 1
   else 2*(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(2 + -1)
```

→ by Rules 14 and 13

```
if ff then 1
   else 2*(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(2 + -1)
```

→ by Rule 16

```
2*(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(2 + -1)
```

→ by Rules 9, 20, and 7

```
2*(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(1)
```

→ by Rules 9 and 21

```
2*(if 1<=0 then 1
     else (fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(1 + (-1)))
```

→ by Rules 9, 14, and 13

```
2*(if ff then 1
     else (fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(1 + (-1)))
```

→ by Rules 9 and 16

```
2*(1*(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(1 + (-1)))
```

→ by Rules 9, 20, and 7

```
2*(1*(fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(0))
```

→ by Rules 9 and 21

```
2*(1*(if 0<=0 then 1
     else (fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(0 - 1)))
```

→ by Rules 9, 14, and 13

```
2*(1*(if tt then 1
     else (fix g(y) is if y<=0 then 1 else y*g(y+(-1)))(0 - 1)))
```

$\rightarrow$ by Rules 9 and 15

```
2*(1*1)
```

$\rightarrow$ by Rules 9 and 10

```
2*1
```

$\rightarrow$ by Rule 10

```
2
```

2. **Static Semantics**

   (a) `fix f(x) is x+1 : int->int`

   (b) `fix f(x) is x :` t `->` t for any type t.

   (c) `fix f(x) is f(x+1) : int ->` t for any type t.

   (d) `fix f(x) is f(x) :` t_1 `->` t_2 for any types t_1 and t_2.

   (e) The code

   ```
   let id be (fix f(x) is x) in
       if id(true) then id(3) else id(4)
   ```

   cannot be well-typed because the application `id(true)` of the let requires the type of `id` to be `bool->bool` while the body of the let requires the type of `id` to be of the form `int->`t for some type t. In this type system, a single variable cannot be shown to have both of these types.

1. **Evaluation**

	$\langle$`x := 0; while x<=0 do (skip; x := x+1)`$, \emptyset\rangle$	
$\rightarrow$	$\langle$`while x<=0 do (skip; x := x+1)`$, \mathtt{x} = 0\rangle$	(by Rules 38, 37)
$\rightarrow$	$\langle$`(skip;x:=x+1);while x<=0 do (skip;x:=x+1)`$, \mathtt{x} = 0\rangle$	(by Rules 42, 35, 33, 32)
$\rightarrow$	$\langle$`x:=x+1; while x<=0 do (skip; x := x+1)`$, \mathtt{x} = 0\rangle$	(by Rules 36, 38)
$\rightarrow$	$\langle$`while x<=0 do (skip; x := x+1)`$, \mathtt{x} = 1\rangle$	(by Rules 38, 37)
$\rightarrow$	$(\mathtt{x} = 1)$	(by Rules 43, 35, 33, 32)

2. **Concurrency**

   The possible interleaving of the commands means that the program

   `(x:=0; x:=x+1)` $\|$ `(x:=3; x:=x+5)`

executes the assignments in the same order as one of the following six programs:

```
x:=0; x:=x+1; x:=3; x:=x+5
x:=0; x:=3; x:=x+1; x:=x+5
x:=0; x:=3; x:=x+5; x:=x+1
x:=3; x:=x+5; x:=0; x:=x+1
x:=3; x:=0; x:=x+5; x:=x+1
x:=3; x:=0; x:=x+1; x:=x+5
```

So the final value for `x` can be any of $\{8, 9, 1, 6\}$.

3. **loop-while-repeat**

The easiest solution is to expand the `loop-while-repeat` to equivalent code as soon as you see it, e.g.,

$$\frac{}{\begin{array}{c}\langle \texttt{loop}\ c_1\ \texttt{while}\ e\ c_2\ \texttt{repeat}, M\rangle \quad \rightarrow \\ \langle (c_1\texttt{;}\ \texttt{if}\ e\ \texttt{then}\ (c_2\texttt{;}\ \texttt{loop}\ c_1\ \texttt{while}\ e\ c_2\ \texttt{repeat})\ \texttt{else skip}), M\rangle\end{array}} \tag{1}$$

or

$$\frac{}{\langle \texttt{loop}\ c_1\ \texttt{while}\ e\ c_2\ \texttt{repeat}, M\rangle \rightarrow \langle (c_1\texttt{;}\ \texttt{while}\ e\ \texttt{do}\ (c_1\texttt{;}\ c_2)), M\rangle} \tag{2}$$

if you still have `while` in the language.

Alternatively, if you really want to evaluate e in these inference rules (as is done for the `while` loop), then it has to be done *after* the command c_1 has terminated (because the execution of c_1 may change the memory, which can affect the value of e). This results in a pair of rules such as:

$$\frac{\langle c_1, M\rangle \rightarrow^* M' \qquad \langle e, M'\rangle \Downarrow \texttt{tt}}{\langle \texttt{loop}\ c_1\ \texttt{while}\ e\ c_2\ \texttt{repeat}, M\rangle \rightarrow \langle (c_2\texttt{;}\ \texttt{loop}\ c_1\ \texttt{while}\ e\ c_2\ \texttt{repeat}), M'\rangle} \tag{3}$$

$$\frac{\langle c_1, M\rangle \rightarrow^* M' \qquad \langle e, M'\rangle \Downarrow \texttt{ff}}{\langle \texttt{loop}\ c_1\ \texttt{while}\ e\ c_2\ \texttt{repeat}, M\rangle \rightarrow M'} \tag{4}$$

Note the use of the $\rightarrow^*$ relation here, which as before is the reflexive transitive closure of the $\rightarrow$ relation on program states.