Computer Science 131, Fall 2000
Assignment 2: Type Safety

Out: Thursday, September 14

Due: Friday, September 22, 5:00pm

This assignment involves no programming. Your written answers must be given directly to the professor (or put under his office door, 1253 Olin). *Your work must be clearly legible.* If you cannot write neatly, use LaTeX or otherwise typeset the proofs.

If you submit your solution late, be sure to mark it with the date and time that it was handed in.

1 Warm-up (10%)

Show the steps required to evaluate the following program to a value. You need not justify each step (as long as you are correct). For example, if the program were `1 + (2 + 3)` then the answer would be `1 + (2 + 3)` $\rightarrow$ `1 + 5` $\rightarrow$ `6`.

1.
```
let x be 3 in
  let f be (fun g(y:Int):Int is x+y) in
    let x be 4 in
      f(x)
```

2. For the following code, assume that we have rules in the dynamic semantics for subtraction and multiplication exactly analogous to the three existing rules for addition.

```
let fact be (fun g(y:Int):Int is
                if y<1 then 1 else y*g(y-1))
in
    fact 2
```

2 Adding pairs to NQSML. (65%)

Consider the following extension of NQSML of adding pairs. The abstract syntax is extended as follows:

$$
\begin{array}{rcl}
v & ::= & \cdots \\
& | & \langle v_1, v_2 \rangle \\
\\
e & ::= & \cdots \\
& | & \langle e_1, e_2 \rangle \\
& | & e.1 \\
& | & e.2 \\
\\
t, u & ::= & \cdots \\
& | & t_1 \times t_2
\end{array}
$$

A pair of values is considered a value. The new syntax allows creation of a pair, and operations to project out the first or second component of a pair. Unlike SML, there is no pattern-matching for pairs, so in this abstract syntax a function to raise an integer to an positive integer power would look like:

```
fun p(arg:Int×Int):Int is
  let x be arg.1 in
     let n be arg.2 in
       if (n < 1) then 1 else p(⟨x, n − 1⟩)
```

The new typing rules are:

$$\frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash \langle e_1, e_2 \rangle : t_1 \times t_2} \tag{26}$$

$$\frac{\Gamma \vdash e : t_1 \times t_2}{\Gamma \vdash e.1 : t_1} \tag{27}$$

$$\frac{\Gamma \vdash e : t_1 \times t_2}{\Gamma \vdash e.2 : t_2} \tag{28}$$

and the new evaluation rules are:

$$\frac{e_1 \rightarrow e_1'}{\langle e_1, e_2 \rangle \rightarrow \langle e_1', e_2 \rangle} \tag{29}$$

$$\frac{e_2 \rightarrow e_2'}{\langle v_1, e_2 \rangle \rightarrow \langle v_1, e_2' \rangle} \tag{30}$$

$$\frac{e \rightarrow e'}{e.1 \rightarrow e'.1} \tag{31}$$

$$\frac{}{\langle v_1, v_2 \rangle.1 \rightarrow v_1} \tag{32}$$

$$\frac{e \;\to\; e'}{e.2 \;\to\; e'.2} \tag{33}$$

$$\frac{}{\langle v_1, v_2\rangle.2 \;\to\; v_2} \tag{34}$$

One can prove the Type Preservation and Progress properties for this extended language by taking the proof for the original system and simply adding new cases corresponding to the new rules. (There will be new cases in the Type Preservation proof corresponding to the new dynamic semantics rules, and new cases in the Progress proof corresponding to the new static semantics rules.) State the new cases required and give the proofs for these cases.

You will need to add new Inversion properties you and to extend the Canonical Forms lemma. You should state these extensions, but need not give the proofs.

3 Natural Semantics (25%)

So far you have seen what is called a "small-step" operational semantics, where execution is defined by a relation $\to$ that defines a single step of evaluation; from this we can define the multi-step relation $\to^*$. Another common presentation is a "big-step" operational semantics, where the notion of "evaluating to a value" is defined directly. This is done by defining a new relation $\Downarrow$ between programs and values, where $e \Downarrow v$ means that evaluating the program e yields the value v.

The natural semantics for the NQSML language (without pairs or lists) is given by the following collection of inference rules. As usual, stating that "$e \Downarrow v$ holds" or that "$e \Downarrow v$ is true" is equivalent to saying there exists a finite proof that $e \Downarrow v$.

$$\frac{}{v \;\Downarrow\; v} \tag{35}$$

$$\frac{e_1 \;\Downarrow\; \overline{\mathtt{m}_1} \qquad e_2 \;\Downarrow\; \overline{\mathtt{m}_2}}{e_1 + e_2 \;\Downarrow\; \overline{\mathtt{m}_1 + \mathtt{m}_2}} \tag{36}$$

$$\frac{e_1 \;\Downarrow\; \overline{\mathtt{m}_1} \qquad e_2 \;\Downarrow\; \overline{\mathtt{m}_2}}{e_1 < e_2 \;\Downarrow\; \overline{\mathtt{m}_1 < \mathtt{m}_2}} \tag{37}$$

$$\frac{e_1 \;\Downarrow\; \overline{\mathtt{tt}} \qquad e_2 \;\Downarrow\; v_2}{\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \;\Downarrow\; v_2} \tag{38}$$

$$\frac{e_1 \;\Downarrow\; \overline{\mathtt{ff}} \qquad e_3 \;\Downarrow\; v_3}{\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \;\Downarrow\; v_3} \tag{39}$$

$$\frac{e_1 \;\Downarrow\; v_1 \qquad e_2[x \mapsto v_1] \;\Downarrow\; v_2}{\mathtt{let}\ x\ \mathtt{be}\ e_1\ \mathtt{in}\ e_2 \;\Downarrow\; v_2} \tag{40}$$

$$\frac{\begin{array}{c} e_1 \;\Downarrow\; (\texttt{fun}\; f(x{:}t_1){:}t_2\; \texttt{is}\; e_1') \qquad e_2 \;\Downarrow\; v_2 \\ (e_1'[x{\mapsto}v_2])[f{\mapsto}(\texttt{fun}\; f(x{:}t_1){:}t_2\; \texttt{is}\; e_1')] \;\Downarrow\; v \end{array}}{e_1\, e_2 \;\Downarrow\; v} \tag{41}$$

The two dynamic semantics for NQSML were designed to be equivalent in the following sense: we expect $e \Downarrow v$ if and only if $e \rightarrow^* v$. The goal of this problem is to prove this fact.

1. Prove that if $e \Downarrow v$ then $e \rightarrow^* v$. Your proof should be by induction on *the proof* of $e \Downarrow v$. (You can't prove this by induction on e itself.)

   It is obvious from the definition of the small-step semantics that if $e_1 \rightarrow^* v_1$ then $e_1 + e_2 \rightarrow^* v_1 + e_2$. You are not required to explicitly prove this or the other similar properties that you will need. You will also need to use the fact that (by definition) the relation $\rightarrow^*$ is reflexive and transitive.

2. Prove that if $e \rightarrow e'$ and $e' \Downarrow v$ then $e \Downarrow v$. Your proof should use induction; state explicitly what you are inducting over.

3. Prove that if $e \rightarrow^* v$ then $e \Downarrow v$. (Hint: use induction on the number of steps needed to get from e to v.)