Homework 3: LF and Twelf

COS 598E: Foundations of Language-Based Security
Due: Monday March 25th, 2002

1. Read Frank Pfenning's introduction to logical frameworks, accessible from the course web page:

   `http://www.cs.princeton.edu/~dpw/courses/spring02/mdorf01.pdf`

2. Download and walk through Andrew Appel's tutorial on using Twelf.

   `http://www.cs.princeton.edu/~appel/twelf-tutorial/`

3. Give an *intrinsically typed* encoding of the call-by-value simply-typed lambda calculus static semantics. In an intrinsically typed encoding, only well-typed terms are encoded. There is no encoding of ill-typed terms. Phrased differently, we say that the typing rules act as a more elaborate definition of syntactically correct terms.

   In contrast, an *extrinsically typed* encoding will encode the syntax of the lambda calculus (both well-typed and ill-typed terms) and then afterwards, encode a relation on lambda calculus terms that defines well-typedness.

   The syntax and static semantics of the variant of the simply typed lambda calculus that you are to encode can be found in Figures 1 and 2. There are two typing judgments. One is for values, and one is for terms. An explicit rule allows a value to be typed as a term.

   For this question and following questions, all answers must be verified in Twelf. To start the intrinsically typed encoding, define the set of types in the calculus and then, for each type, define the set of terms and values of that type.

$$
\begin{array}{llll}
\text{(types)} & \tau & ::= & \mathit{int} \mid \tau_1 \to \tau_2 \\
\text{(values)} & v & ::= & x \mid 0 \mid \mathtt{s}\ v \mid \lambda x{:}\tau.e \\
\text{(terms)} & e & ::= & v \mid \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \mid e_1\ e_2 \mid e - 1
\end{array}
$$

Figure 1: Syntax of the Simply Typed Lambda Calculus

```
tp     :  type.
value  :  tp → type.
term   :  tp → type.
```

Remember, in the intrinsic encoding we do not define the representation of the syntax of values and terms explicitly. We start by representing the well-typed terms right away. To get you started, here is the representation of types, 0, successor, decrement and the coercion from value typing to term typing.

```
int       :  tp.
arrow     :  tp → tp → tp.

zero      :  value int.
succ      :  value int → value int.

minus     :  term int → term int.
val2term  :  ΠT:tp.  value T → term T.
```

4. Once you have represented the static semantics, attempt to represent the dynamic semantics. I have laid out the dynamic semantics for some of the constructs in figure 3. Spell out the rest of the dynamic semantics.

   The single step dynamic semantics is a relation between two well typed terms. Therefore, to begin with we define a new type family for representing this relation.

   ```
   step  :  ΠT:tp.term T → term T → type.
   ```

   The rule for values follows. Fill in the rest of the rules for the single step relation. Define a new relation for multi-step evaluation.

   ```
   value-step  :  ΠT:tp.ΠV:value T.
                      step T (val2term T V) (val2term T V).
   ```

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash_v x : \tau}$$

$$\frac{}{\Gamma \vdash_v 0 : \mathit{int}}$$

$$\frac{\Gamma \vdash_v v : \mathit{int}}{\Gamma \vdash_v \mathtt{s}\ v : \mathit{int}}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash_v \lambda x{:}\tau_1.e : \tau_1 \to \tau_2}\ (x \notin \mathsf{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash_v v : \tau}{\Gamma \vdash v : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x{:}\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \tau_2}\ (x \notin \mathsf{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

$$\frac{\Gamma \vdash e : \mathit{int}}{\Gamma \vdash e - 1 : \mathit{int}}$$

Figure 2: Static Semantics for the Simply Typed Lambda Calculus

$$\frac{}{v \longmapsto v}\ (value)$$

$$\frac{}{(\lambda x{:}\tau.e)\ v \longmapsto e[v/x]}\ (\beta \longmapsto)$$

$$\frac{e_1 \longmapsto e_1'}{e_1 e_2 \longmapsto e_1' e_2}\ (cong1 \longmapsto)$$

$$\frac{e_2 \longmapsto e_2'}{v_1 e_2 \longmapsto v_1 e_2'}\ (cong2 \longmapsto)$$

$$\frac{}{e \longmapsto^* e}\ (reflex \longmapsto^*)$$

$$\frac{e \longmapsto e'' \quad e'' \longmapsto^* e'}{e \longmapsto^* e'}\ (trans \longmapsto^*)$$

Figure 3: Operational Semantics of the Simply Typed Lambda Calculus (incomplete)