CSCI 3155: Homework Assignment 5

Due Thursday, October 22, 2009

For this assignment, each student will turn in a write-up. **You need to have your own write-up.** As always, you are welcome and encouraged to collaborate and discuss these questions in groups. Just be sure to acknowledge those with which you discussed.

**Bookkeeping**

**Exercise 1:** Indicate in a sentence or two how much time you spent on this homework, how difficult you found it subjectively, and what you found to be the hardest part. Any non-empty answer will receive full credit.

If you would like share something about yourself that I do not already know, please do so. And if your opinions have changed since the last submission, indicate one thing you like about the class so far and one thing you would change about it.

**Type Equality**

The following two questions use the PL-Detective. You will need to refer to the MYSTERY grammar to answer these questions.

[http://www-plan.cs.colorado.edu/diwan/pldetective/pldsyntax.htm](http://www-plan.cs.colorado.edu/diwan/pldetective/pldsyntax.htm)

The most important part of these questions is not the final answer (e.g., the type equality mechanism) but the reasoning that led to the answer. Thus, don’t leave the write up to the last minute! Also since the number of attempts you can submit to the PL-Detective is limited, it is worth thinking carefully before each submission. The limit is set high enough that you can waste about half of the attempts and still get the full score. When you submit a program to the PL-Detective, it tells you how many programs you have submitted without syntax errors so far. Note that type errors are not syntax errors.
We have studied two type equality mechanisms: name type equality and structural type equality. In this exercise, you will figure out the type equality mechanism used by MYSTERY. As with previous exercises, you will accomplish this by submitting programs to the PL-Detective and observing the results.

For the purpose of this assignment, you should assume that an assignment

\[
\langle Expr_1 \rangle := \langle Expr_2 \rangle
\]

is legal only if the types of \(\langle Expr_1 \rangle\) and \(\langle Expr_2 \rangle\) are equal. Also, assume that the type of a constant number (e.g., 5) is INTEGER.

**Exercise 2:** In this question, ignore TYPE declarations in MYSTERY. TYPE declarations are similar to typedef in C and C++. For example,

\[
\text{TYPE T = INTEGER}
\]

in MYSTERY creates a new type name T for the type INTEGER. This declaration is similar to

\[
\text{typedef int T}
\]

in C/C++.

1. **Skill 10.4.** Determine and describe how MYSTERY defines type equality for ARRAY, PROCEDURE, and subrange types. Your description should be precise and complete enough that a reader can take any two types (where both are array types, procedure types, or subrange types) in MYSTERY and determine whether or not they are equal. You may submit up to 8 programs to the PL-Detective. You will lose 5% of the points for this question for each additional program that you use. Submitted programs that fail due to syntax errors do not count in your total. Note that type errors are not syntax errors. Use the following link for this exercise:

http://www-plan.cs.colorado.edu/diwan/pldteq.htm

Hint: The issue of whether two types are name or structurally equal comes up only when they are not obviously different. For example, the subrange types \([0 \ TO \ 10]\) and \([5 \ TO \ 20]\) are obviously different and thus you do not need to submit a program to the PL-Detective to determine if they are equal or not.

Present an argument for your answer. Your argument should include the evidence (e.g., programs you submitted to the PL-Detective and the output) along with a carefully reasoned argument explaining how your answer is justified by your evidence.
2. **Skills 10.2 and 1.1.** Give one point in favor of and one point against MYSTERY’s type equality mechanism (which you just discovered). Explain your point referring to the characteristics in Table 1.1 in the text.

**Exercise 3:** In this question you will determine how MYSTERY’s TYPE declarations affect type equality. In other words, after

\[
\text{TYPE } T = \langle\text{some type}\rangle
\]

are \(T\) and \(\langle\text{some type}\rangle\) equal?

1. **Skill 10.4.** Determine and describe how MYSTERY’s TYPE declarations affect type equality. Write this as if you are writing a language definition, that is, your answer to this question combined with your answer to question 2 should allow a reader to determine the equality of any two types in MYSTERY’s including ones that use type names declared using TYPE. You may submit up to 2 programs to the PL-Detective in your investigation. Each additional program is charged 5% of the points for this question. Submitted programs that fail due to syntax errors do not count in your total. Note that type errors are not syntax errors. Use the following link for this exercise:

   [http://www-plan.cs.colorado.edu/diwan/pldteq2.htm](http://www-plan.cs.colorado.edu/diwan/pldteq2.htm)

Present an argument for your answer. Your argument should include the evidence (e.g., programs you submitted to the PL-Detective and the output) along with a carefully reasoned argument explaining how your answer is justified by your evidence.

2. **Skills 10.2 and 1.1.** Give one point in favor of and one point against the semantics that you just discovered. Explain your point referring to the characteristics in Table 1.1 in text.

**Induction**

**Exercise 4:** **Skill 8.4.** Consider the following SML **datatype** for the abstract syntax of a simple language of arithmetic expressions, including integers, unary negation, and addition.

```
datatype expr =
   Int of int
| Neg of expr
| Plus of expr * expr
```
We can define an SML function \texttt{eval} that evaluates such arithmetic expressions directly by recursion over \texttt{expr}.

\[
(*) \quad \texttt{fun \ eval : expr \rightarrow int} *
\]

\[
\begin{aligned}
\texttt{eval\ (Int\ i)} & = i \\
| \texttt{eval\ (Neg\ e1)} & = \sim(\texttt{eval\ e1}) \\
| \texttt{eval\ (Plus\ (e1,\ e2))} & = (\texttt{eval\ e1}) + (\texttt{eval\ e2})
\end{aligned}
\]

We can also define a function \texttt{redex} that performs only local simplification (i.e., local reduction) and a \texttt{visit} function that applies a transformation to an expression bottom-up (like in Project 1).

\[
(*) \quad \texttt{fun \ redex : expr \rightarrow expr} *
\]

\[
\begin{aligned}
\texttt{redex\ (Neg\ (Int\ i))} & = \texttt{Int\ (\sim\ i)} \\
| \texttt{redex\ (Plus\ (Int\ i1,\ Int\ i2))} & = \texttt{Int\ (i1 + i2)} \\
| \texttt{redex\ e} & = e
\end{aligned}
\]

\[
(*) \quad \texttt{fun \ visit : (expr \rightarrow expr) \rightarrow expr \rightarrow expr} *
\]

\[
\begin{aligned}
\texttt{visit\ f\ (Int\ i)} & = f\ (Int\ i) \\
| \texttt{visit\ f\ (Neg\ e1)} & = f\ (Neg\ (visit\ f\ e1)) \\
| \texttt{visit\ f\ (Plus\ (e1,\ e2))} & = f\ (Plus\ (visit\ f\ e1,\ visit\ f\ e2))
\end{aligned}
\]

We can see that \texttt{visit\ redex} (i.e., the function of type \texttt{expr \rightarrow expr} that results from applying \texttt{redex} function to \texttt{visit}) is quite similar to \texttt{eval}. In particular, \texttt{visit\ redex} also evaluates an expression to an integer. To make this precise, we can prove the following theorem.

**Theorem 1.** For all \texttt{expr} values \(e\), if \texttt{eval\ e} evaluates to an \texttt{int} value \(i\), then \texttt{visit\ redex\ e} evaluates to an \texttt{expr} value \texttt{Int} \(i\).

For this exercise, prove Theorem 1. Be sure to state clearly on what you perform induction and to what you apply the induction hypothesis. Use the following notation to indicate evaluation in SML:

\[
\begin{aligned}
&e \rightarrow e' & \text{for one step of SML evaluation} \\
&e \rightarrow^* e' & \text{for some number of steps of SML evaluation}
\end{aligned}
\]

You may assume that \texttt{eval\ e} terminates for any \texttt{expr} value \(e\). In other words, you may assume the following lemma:

**Lemma 2** (Termination of \texttt{eval}). For all \texttt{expr} values \(e\), \texttt{eval\ e} evaluates to an \texttt{int} value \(i\) (for some \(i\)).

Note that the other direction of Theorem 1 also holds, though you need not show it for this exercise.