SFWR ENG 2F04 Assignment 5: Typechecking, Induction, Software Verification, and much, much more!

Due: 1630 Monday December 2, 2002

To help you with the PVS related questions (2, 4, and 5) for this assignment, download the following file from the web at the URL: `http://www.cas.mcmaster.ca/~lawford/2F04/Notes/a5_02.pvs`.

1. **Partial Functions in Logic**

   a) Consider the following formula know as the "axiom of reflection" in traditional logical systems:

   $$\forall x \forall y (x = y \rightarrow f(x) = f(y))$$

   i) Show that this formal is a logical theorem of our (Huth+Ryan) traditional logic system by formally proving:

   $$\vdash \forall x \forall y (x = y \rightarrow f(x) = f(y))$$

   ii) When considered in a traditional analysis ("Parnas") style logic system, this formula is not a logical theorem. i.e.,

   $$\not\vdash (\forall x)(\forall y)(x = y \rightarrow f(x) = f(y))$$

   Supply an concrete interpretation for f using the universe of real numbers and explain why this is the case.

   b) Write down the most concise formulas in both the IMPS/Parnas (analysis style) logic and bounded quantification (PVS style) logic that could be used to specify that A, an N element array of integers, has the property:

   The array does not contain a strictly increasing sequence of elements.

2. **PVS Typechecking & Predicate Subtypes in Logic**

   a) In the theory `a5Q2` create the "best" PVS definitions for the following functions:

   i) $f(x) = x^2 + 4x - 5$ [Hint: use a little calculus to figure out the minimum of $f(x)$.]

   ii) Create the "best" PVS definitions for the function: $g(x, y) = \ln(x - y^2)$

   Try to prove all resulting TCCs to make sure that your functions are always defined.

3. **Proof by Induction**

   a) Huth+Ryan p.56# 3.

   b) Huth+Ryan p.56# 5.

   c) Prove using mathematical induction that for all $n \geq 0$, the value of $n^2 + 5n + 1$ is odd.

   d) Let f be a unary function symbol that we will interpret as$f^{\mathcal{M}}(x) = x^2 + 5x + 1$. Assuming that our universe of interpretation is the natural numbers, write down a formal predicate logic formula for the statement that we proved in part (a) above.

4. **Tabular Specifications** Consider the tabular definition:

$$f(x, y) = \begin{array}{|c|c|c|}\hline C_1(x, y) & C_2(x, y) & C_3(x, y) \\ \hline f_1(x, y) & f_2(x, y) & f_3(x, y) \\ \hline\end{array}$$

a) Assume functions f_1, f_2 and f_3 are defined when C_1, C_2 and C_3 are respectively true. Using our standard notation for predicate calculus, write down the two formulas, one for Disjointness and one for Completeness, that PVS would require a user to prove for the above table. In this case the Disjointness and Completeness proof obligations (TCCs) are sufficient to guarantee that the table defines a (total) function.

b) The PVS example in Figure 1 provides some insight as to why the Disjointness conditions generated by PVS are overly restrictive. The theorem "same" can be easily proved using the (GRIND) command but when a file containing the definitions is type checked, the disjointness condition fails for `f2`. For the table used to define f in part (a) above, create a weaker "disjointness" condition that together with the completeness condition provides necessary and sufficient conditions for the table defining f to be a total function.

```
x: VAR real
                     %-------------%
f1(x): real = TABLE  |[ x<0 | x>=0 ]|
                     %-------------%
                     |  x   | 2*x  ||
            ENDTABLE %-------------%

                     %--------------%
f2(x): real = TABLE  |[ x<=0 | x>=0 ]|
                     %--------------%
                     |   x   | 2*x  ||
            ENDTABLE %--------------%

same: THEOREM FORALL (x:real): f1(x)=f2(x)
```

Figure 1: Disjointness condition counter example

c) Although the weakened "disjointness" condition together with the usual completeness condition provides necessary and sufficient conditions for a table to define a function. Why is it preferable for software engineers to use the more strict disjointness condition when using tables to specify the functional requirements of software?

5. **Software Verification**

Consider the code fragment shown in Figure 2 that is supposed to implement a linear search for an element `key` in an n element array that runs from 0 to $n-1$.

In order to check that this code fragment is correct, a diligent program has formulated the PVS input shown in Figure 3 to verify the correctness of the loop. (NOTE: `(A(i)/=key)` is short for `NOT(A(i)=key)`.) The programmer tries to prove all of the TCCs generated by the file and the `THEOREM`s `C1`, `C2` and `C3`. It is possible to prove `C1` and `C3` but not `C2`. Also, there is one TCC the programmer is unable to prove.

The TCC that the programmer is unable to prove is shown in Figure 4 together with the sequent that results when trying to prove it.

a) If the programmer had been able to prove all of the theorems, could the programmer ignore the unproven TCC? Explain your answer briefly with reference to the purpose of TCCs in PVS.

$i := 0;$
while $0 \le i \le n \wedge A(i) \ne key$ **do**
    $i := i + 1;$
end;

Figure 2: Program fragment for Question 5

```
prog_ver : THEORY
  BEGIN

  n: int
  i: VAR nat
  AType: TYPE+
  key: VAR AType

  A(i:{k:nat|k<=n-1}):AType

  V(i,key):bool = (0<= n)
  B(i,key):bool = (0<=i) & (i<=n) & (A(i)/=key)
  I(i,key):bool = (0<=i) & (i<=n) & (forall (j:{k:nat|k<=i-1}): A(j)/=key)
  P(i,key):bool = (0<=i) & (i<=n) &  (forall (j:{k:nat|k<=i-1}): A(j)/=key)
                  & (i=n OR A(i)=key)

  C1: THEOREM V(i,key) => I(0,key)
  C2: THEOREM I(i,key)& B(i,key) => I(i+1,key)
  C3: THEOREM I(i,key)& NOT B(i,key) => P(i,key)

  END prog_ver
```

Figure 3: PVS code for Question 5

```
% Subtype TCC generated by B(i,key) for i
  % unfinished
B_TCC1: OBLIGATION
  FORALL (i): 0 <= i AND i <= n IMPLIES i >= 0 AND i <= n - 1


[-1]  0 <= i!1
[-2]  i!1 <= n
  |-------
[1]   i!1 <= n - 1

Rule?
```

Figure 4: Unproven TCC for Question 5

b) Write down the characteristic equation for the above sequent and find a counter example to the equation.

c) Is your counter example for the sequent a counter example for `B_TCC1` proof obligation?

d) Why does the definition of predicate `B` generate `B_TCC1` and why is it unprovable?

e) The predicate `B(i,key)` is taken from the condition of the **while**. What is this unprovable TCC telling you is wrong with the program fragment? How could the program fragment be fixed?

f) The predicates `I(i,key)` and `P(i,key)` both make use of the expression:

```
(forall (j:{k:nat|k<=i-1}): A(j)/=key)
```

What is the value of this expression when $i = 0$? Does it depend upon the interpretation of the array `A` and the value of `key`?

(HINT: Consider the negation of the formula.)