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

Due: 1130 Friday December 1, 2000

All of your PVS work for this assignment should be done in a single file called `A5.pvs`. Download this file from the web at the URL: `http://www.cas.mcmaster.ca/~lawford/2F04/Notes/A5_00.pvs`.

It contains some of the PVS you will need to do this assignment. You will have to add to this file as detailed in the questions below. When you are done the PVS part, you will submit it electronically as a PVS dump file called `A5_00.dmp`. Written work will be handed in separately at the *start* of class on the due date.

NOTE: For up to date information on how all you voracious little PVS piranhas can submit your work to the sacrificial cow, please check out the URL:

`http://www.cas.mcmaster.ca/~lawford/2F04/e-submissions.html`

1. **Tabular Specification I: Weakening conditions on tabular definitions** (25 Marks Total)

   Consider the tabular definition:

$$f(x,y) = \begin{array}{|c|c|c|}\hline C_1xy & C_2xy & C_3xy \\ \hline f_1(x,y) & f_2(x,y) & f_3(x,y) \\ \hline \end{array}$$

   **a)** (7 marks) 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.

   The PVS example in theory `A5Q1` of the file `A5.pvs` provides some insight as to why the Disjointness conditions generated by PVS are overly restrictive.

   **b)** (2 marks) Prove the theorem `same` in file `A5_00.pvs`.

   **c)** (6 marks) Typecheck the theory and have PVS try to prove the resulting TCCs with the PVS/Parsing and Typechecking/typecheck-prove command. Take a look at the TCCs for the file with the PVS/Viewing TCCs/show-tccs command. As you can see, the disjointness condition `h_TCC1` fails for the table defining `h`. Rerun the proof PVS tried by placing you cursor on this TCC and invoking the prover. Write down the characteristic equation for the resulting unprovable sequent and use it to obtain a counter example to the disjointness condition. This is a case where the table still defines a total function even though PVS' disjointness condition is violated. Why?

   **d)** (7 marks) 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.

   **e)** (3 marks) 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?

2. **Partial Functions, PVS Typechecking & Predicate Subtypes in Logic** (15 Marks)

   **a)** (4 marks) Consider the following PVS declaration contained in theory A5Q2:

   ```
   U:TYPE
   a:U
   ```

   **i)** What Type Correctness Condition (TCC) does this generate? Why?
   **ii)** Assuming PVS doesn't have any inconsistencies, why is it unprovable?
   **iii)** Why would it be provable if PVS had a soundness bug (i.e. it was possible to prove $\perp$)?

   **b)** (4 marks) Consider the function:

   $$f(x, y) = 2 + \sqrt{x + y}$$

   Write down the best PVS definition for $f$.

   **c)** (2 marks) Assume that $x \neq y$ is an abbreviation for $\neg(x = y)$. In the Parnas/IMPS traditional analysis approach to logic, what is the truth value of the formula $(\exists x)((\sqrt{x})^2 \neq |x|)$? Justify your answer.

   **d)** (5 marks) 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.

3. **Proof by Induction** (30 Marks Total)

   **a)** (5 marks) The following rule, call Rule MI, is an *axiom* of Peano Arithmetic (i.e. it can be used on any line of a proof for a formula of Peano Arithmetic):

   $$\vdash \phi[0|n] \land (\forall m)(\phi[m|n] \rightarrow \phi[m+1|n]) \rightarrow (\forall n)\phi$$

   Here $\phi$ is a formula of Peano Arithmetic, $n \in FV(\phi)$ and $m$ is a "new" variable that does not occur in $\phi$.
   Informal mathematical induction involves showing (i) $\phi[0|n]$ is true and (ii) assuming that $\phi[m|n]$ is true, you show that $\phi[m+1|n]$ is true. Use formal proof rules to show why this is sufficient to prove $(\forall n)\phi$.

   **b)** (10 marks) Use mathematical induction to informally prove (by hand) that every odd power of 2 + 1 is divisible by 3. That is, show that for every $n$, there exists some $i$ such that $2^{(2n+1)} + 1 = 3i$. (HINT: $2^{2n+1} = 2 * 4^n$)

   **c)** (5 marks) Prove the same thing by Induction in PVS by proving theorem `Q3d` of theory `A5Q3`. As the first proof step use the `(INDUCT "n")` command to let PVS know you are attempting a proof by induction on the variable `n`. Next, use the command `(EXPAND "^")` to rewrite the exponent shorthand in terms of the recursive function `expt(r,n)`. Expand the definition of `expt` in the bottom part of the sequent but not the top, then use the `(BOTH-SIDES ...)` command to obtain something that you can use with the `(REPLACE ...)` command. Use the PVS/Getting Help/help-pvs-prover and help-pvs-prover-command menu options for more information on these commands.

   **d)** (10 marks) Do Rubin p. 302 A 1,23 by hand.

4. **Software Verification with PVS** (30 marks)

   In this problem we study the verification of a simplified pressure sensor trip that monitors a pressure sensor and is "tripped" when the sensor value exceeds a normal operating setpoint. The proposed specification and the actual implementation for the sensor trip are give in Figure 1 by `f_PressTrip` and `PTRIP`, respectively. In the function definitions, `f_PressTripS1` and `PREV` play corresponding roles as the arguments for the previous value of the state variable computed by the function.

Figure 1 also contains the supporting type and abstraction function definitions for verifying that the implementation meets the specification. The abstraction function `real2AItype` models the A/D (analog to digital) conversion of the pressure sensor value by taking the integer part of its input using the `floor` function from the PVS prelude file. It is used to map the real valued specification input `Pressure` to the discrete implementation input `PRES` which has type `AIType`. `AIType` consists of the subrange of natural numbers between 0 and 5000.

At the bottom of the specification in Figure 1, the theorem `Sentrip1` is an example of a *block comparison theorem* that could be used to prove that the implementation `PTRIP` will produce the same output as the specification `f_PressTrip` for all possible inputs. For you convenience the PVS theory `sentrip` corresponding to Figure 1 is included in the PVS file `A5_00.pvs`.

**a)** (5 marks) Write down the characteristic equation for the unprovable sequent that results from trying to prove `Sentrip1`.

**b)** (5 marks) Find all the values of `Pressure!1` and `f_PressTripS1!1` that provide counter examples for the equation.

**c)** (2 marks) Pick specific values for `Pressure!1` and `f_PressTripS1!1` that provide a counter example to the equation in (a) and confirm that it provides a counter example to theorem `Sentrip1`.

**d)** (2 marks) State and prove a general theorem `Sentrip2` that shows that the implementation does *not* meet the specification. This provides confirmation that the unprovable sequent for theorem `Sentrip1` results from inconsistencies between the specification and implementation and not from a poor choice of PVS prover commands by the verifier. This is an example of refutation theorem proving where a software engineer tries to prove that the implementation is *NOT* equivalent to the specification.

**e)** (3 marks) In fact it is currently impossible to change the definition of `PTRIP` so that it will satisfy the specification `f_PressTrip`. To see this, draw the commutative diagram for Theorem `Sentrip1` and explain this situation in terms of the Theorem in Assignment 4 Question 5(c).

**f)** (5 marks) How could the original specification `f_PressTrip` be modified so that it is possible to find a new implementation of `PTRIP` that would make `Sentrip1` true? (HINT: You only need to make minor modifications to the inequalities of `f_PressTrip`.) Call this new specification `f_PressTrip2` and call the new implementation `PTRIP2`. State and prove a theorem called `Sentrip3` that is identical to `Sentrip1` except that `f_PressTrip` and `PTRIP` are replaced by `f_PressTrip2` and `PTRIP2` respectively

**g)** (8 marks) Suppose the A/D conversion hardware has a tolerance of $\pm 5$ associated with it. The interpretation is that an input `Pressure` value of say 17.3 could produce the same results as any value in the range $12.3 \leq \text{Pressure} \leq 22.3$. This being the case, strict functional equivalence of an implementation with a specification might be considered too restrictive. Suppose that it is now acceptable for the original implementation of `PTRIP` to merely produce an output that is within tolerance. This means that for every value of `Pressure` there is some value, say `P`, such that

$$\texttt{Pressure} - 5 \leq \mathrm{P} \leq \texttt{Pressure} + 5$$

and when `PTRIP` is evaluated at the discretized value of `Pressure`, applying f_PressTrip to `P` produces the same result.

Restate theorem `Sentrip1` as a new theorem `Sentrip 4` to take this tolerance into account. Prove the theorem. (HINT: One way to do it involves the use of existential quantification over a dependent type.)

sentrip : THEORY
BEGIN

Trip : TYPE $=$ {Tripped, NotTripped}

AItype : TYPE $= \{\mathrm{i} : \mathrm{nat} \mid 0 \leq \mathrm{i} \wedge \mathrm{i} \leq 5000\}$

f_PressTrip(Pressure : real, f_PressTripS1 : Trip) : Trip $=$ TABLE

| Pressure $\leq 2400$ | $2400 <$ Pressure $\wedge$ Pressure $< 2450$ | Pressure $\geq 2450$ |
|---|---|---|
| NotTripped | f_PressTripS1 | Tripped |

ENDTABLE

PTRIP(PRES : AItype, PREV : bool) : bool $=$ TABLE

| PRES $\leq 2400$ | $2400 <$ PRES $\wedge$ PRES $< 2450$ | PRES $\geq 2450$ |
|---|---|---|
| FALSE | PREV | TRUE |

ENDTABLE

Trip2bool(TripVal : Trip) : bool $=$ TABLE

| TripVal = Tripped | TripVal = NotTripped |
|---|---|
| TRUE | FALSE |

ENDTABLE

bool2Trip(BoolVal : bool) : Trip $=$ TABLE

| BoolVal = TRUE | BoolVal = FALSE |
|---|---|
| Tripped | NotTripped |

ENDTABLE

real2AItype($x$ : real) : AItype $=$ TABLE

| $x \leq 0$ | $0 < x \wedge x < 5000$ | $x \geq 5000$ |
|---|---|---|
| 0 | floor($x$) | 5000 |

ENDTABLE

Sentrip1 : THEOREM
($\forall$ (Pressure : real, f_PressTripS1 : Trip) :
f_PressTrip(Pressure, f_PressTripS1) =
bool2Trip(PTRIP(real2AItype(Pressure), Trip2bool(f_PressTripS1))))

END sentrip

Figure 1: Formatted PVS specification for pressure sensor trip example