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

Due: 1630 Monday December 3, 2001

All of your PVS work for this assignment should be done in a single file called `a5_01.pvs`. Download this file from the web at the URL: `http://www.cas.mcmaster.ca/~lawford/2F04/Notes/a5_01.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_01.dmp`. Written work will be handed in ITC/201 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. **Partial Functions, PVS Typechecking & Predicate Subtypes in Logic** (20 Marks)

   a) (6 marks) 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.,

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

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

   b) (10 marks) In the theory `a5Q1` 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) For $f(x)$ defined as above:

   $$g(x) = \frac{1}{f(x) + 9}$$

   iii) For $f(x)$ defined as above:

   $$h(x, y) = \frac{1}{y - f(x) + 9}$$

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

   c) (4 marks) Let $\phi[t|x]$ be a valid substitution. Explain why

   $$\Gamma, (\forall x)\phi \vdash \psi \text{ iff } \Gamma, (\forall x)\phi, \phi[t|x] \vdash \psi$$

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

   a) (10 marks) By hand prove

   i) Huth+Ryan p.56# 3.

   ii) Huth+Ryan p.56# 5.

b) (15 marks) Prove both of the above in PVS by proving theorems `Q2bi` and `Q2bii` respectively in theory `a5Q2`. 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.

c) (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):

$$\vdash \phi[0/n] \wedge \forall m(\phi[m/n] \rightarrow \phi[m+1/n]) \rightarrow \forall n\phi$$

Here ϕ is a formula of Peano Arithmetic, $n \in FV(\phi)$ and m is a "new" variable that does not occur in ϕ.

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 $\forall n\phi$.

3. **Software Verification with PVS** (50 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 specification of the pressure sensor trip makes use of deadbands to eliminate "sensor chatter". The 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. The functions `Trip2bool` and `bool2Trip` convert back and forth between the specifications enumerated type `Trip` and the implementation's use of `bool` to represent the trip state.

The PVS theory `sentrip` that generated Figure 1 is at the end of the file `a5_01.pvs`. Use it as a starting point to complete the following questions.

a) (5 marks) At the bottom of the specification in Figure 1, the theorem `Sentrip1` is an example of a *block comparison theorem*. Typecheck the `sentrip` theory and prove `Sentrip1`. What does this tell you about the specification and implementation?

b) (5 marks) The module is also supposed to implement a trip status indicator that is used to flag when pressure sensor trip has occurred. Once every 3 seconds the Trip Computer transmits the status indicator flag to the operator's display computer. The transmitted indicator value depends upon the history of the pressure sensor trip in the previous 3 seconds. If there was a sensor trip at any time during the last 3 seconds, the transmitted indicator value is $TRUE$, otherwise, it is $FALSE$.

The specification of the trip status indicator function is given by the vertical condition table `f_PressStatus` shown in Figure 2. When the condition on the left is $TRUE$, the value on the right is returned. The interpretation of this table is that if the current value of `f_PressTrip` is

sentrip : THEORY
BEGIN

Trip : TYPE = {Tripped, NotTripped}

AItype : TYPE = {i : nat | 0 ≤ i ∧ i ≤ 5000}

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

Pressure < 2400	2400 ≤ Pressure ∧ Pressure < 2450	Pressure ≥ 2450
NotTripped	f_PressTripS1	Tripped

ENDTABLE

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

PRES < 2400	2400 ≤ PRES ∧ PRES < 2450	PRES ≥ 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
(∀ (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 the fixed pressure sensor trip example

tripped (i.e., there is a sensor trip) then the status indicator `f_PressStatus` is set to $TRUE$. When there is not a sensor trip, if it is time to transmit (i.e., variable `Transmit` is $TRUE$ when the current time is a multiple of 3 seconds) then `f_PressStatus` is "cleared" by setting it to $FALSE$. Otherwise `f_PressStatus` is left at its previous value `f_PressStatusS1`.

Figure 2 also contains the formatted PVS for this version of the implementation. To efficiently

f_PressStatus(f_PressTrip : Trip, f_PressStatusS1, Transmit : bool) : bool = TABLE

f_PressTrip = Tripped	TRUE
¬(f_PressTrip = Tripped) ∧ Transmit	FALSE
¬(f_PressTrip = Tripped) ∧ ¬Transmit	f_PressStatusS1

ENDTABLE

STATUS(PRES : AItype, PREV : bool) : bool = TABLE

PRES < 2400	2400 ≤ PRES ∧ PRES < 2450	PRES ≥ 2450
PREV	PREV	TRUE

ENDTABLE

```
Status1 : THEOREM
(∀ (Pressure : real, f_PressTripS1 : Trip, f_PressStatusS1, Transmit : bool) :
  f_PressStatus(f_PressTrip(Pressure, f_PressTripS1), f_PressStatusS1, Transmit) =
   IF¬(Transmit) THEN STATUS(real2AIType(Pressure), f_PressStatusS1)
    ELSE FALSE
    ENDIF)
```

Figure 2: Formatted PVS input for the trip status indicator block comparison

meet all the specifications for the pressure trip module, the developers have decided to partially compute the status output at the same time that `PTRIP` is computed. This is done by using a table of the same structure as `PTRIP` in the implementation function `STATUS`. Here `PREV` is the previous value of `STATUS`.

In addition to the tabular function, the implementation contains the main program thread that provides the function call sequence for the main program loop. The computation of the implementation status output is finished in in the main program thread by a conditional statement that checks the `Transmit` value to determine when it is time to transmit the status, in which case it resets the indicator value to $FALSE$ (i.e., it sets the value the `STATUS` to to $FALSE$ once it is transmitted). This part of the status indicator computation is modeled by the IF-THEN-ELSE statement that is part of the block comparison theorem.

The definitions from Figure 2 are appended to the specification in Figure 1. Attempting to prove the block comparison theorem `Status1` results in several unprovable sequents, including the one below:

```
{-1}    Transmit!1
{-2}    f_PressTripS1!1 = Tripped
  |-------
{1}    Pressure!1 < 2400
```

c) (5 marks) Write down the characteristic equation for the unprovable sequent.

d) (5 marks) Find all the values of `Transmit!1`, `Pressure!1` and `f_PressTripS1!1` that provide counter examples for the characteristic equation you found in (c).

e) (5 marks) Pick specific values for `Transmit!1`, `Pressure!1`, `f_PressStatusS1!1` and `f_PressTripS1!1` that provide a counter example and confirm that it provides a counter example to theorem `Status1` by evaluating

`f_PressStatus(f_PressTrip(Pressure!1,f_PressTripS1!1),f_PressStatusS1!1,Transmit!1)`

and

`IF NOT(Transmit) THEN STATUS(real2AItype(Pressure),f_PressStatusS1) ELSE FALSE`

and comparing the results.

f) (5 marks) Using PVS state and prove a new theorem called `Status2` that could be used to prove that the implementation does not meet the specification. This would provide confirmation that the unprovable sequent for theorem `Status1` 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.

g) (5 marks) Assume that initially `f_PressTripS1= NotTripped` and that the other previous state values are initially *FALSE*. The update functions `f_PressTrip, PTRIP, f_PressStatus` and `STATUS` are called once every second and the value of transmit is *TRUE* every 3 seconds. Sketch a sequence of inputs for `Pressure` and the resulting sequence of outputs produced by the specification and implementation that results in specification and implementation having different status outputs for an extended period of time.

h) (15 marks) How could the implementation be altered to fix this problem? (HINT: Think about how you could use the current value of `PTRIP` in the main program loop.) Add any additional definitions that you require to fix the implementation in the `sentrip` theory and then state and prove a theorem called `Status3` that proves that if `f_PressTripS1=Trip2bool(PREV)` and `PREV=bool2Trip(f_PressTripS1)` (i.e., previous values agreed) then the implementation is equivalent to the specification (current values agree).