CSC2021 Tutorials
Overture Tools
2019

2. Logic, Basic Data, Collections

1. Aims

- Introduce and practice fundamentals of predicate calculus
- Modify and evaluate a specification file
- Extend a specification to include state
- Build a set comprehension expression

This is the second of two tutorials. Completing them, and submitting evidence to Ness, gains 2 marks per tutorial (4 marks total = 4%). A larger piece of modelling coursework will follow, which will be worth 16%.

Submission:
This tutorial is to be completed, and evidence submitted to Ness, by **15:00 Friday 22 Feb**.

By working through all the steps in the tutorial, you will generate an updated specification file which is to be submitted to Ness. See the submission instructions at the end of the tutorial.

2. Introduction

This tutorial provides practice with forming and evaluating logical expressions, and using the VDM debug console. Go through the tasks listed, and submit to Ness your updated specification file.

The tutorial uses the Temperature Monitor example from the "Logic" section of the lecture notes. We assume that you have completed Tutorial 1, so revisit that if you have problems in completing this tutorial.

3. Temperature Monitor

Download **monitorSL.zip** from Blackboard (in the Tutorials section).

Import the monitor into Overture, as in Tutorial 1:

- Select **File**, **Import** from the Overture menu
- Choose **General,** then **Existing Projects into Workspace**
- Select **Archive File** and browse to the **monitorSL.zip** file you downloaded
- Ensure **monitorSL** is selected, and click on **Finish.**

You'll now add some test values to this project: at the beginning of the specification file (monitor.vdmsl) you'll see an empty **values** section: add the following data (on the next page) to that file.

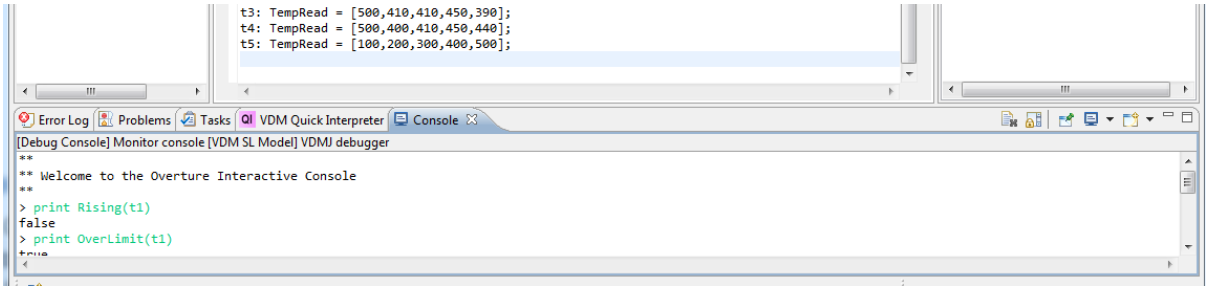
```
-- Monitor test data
-- enter test values from Tutorial 2 here
values

t1: TempRead = [200,300,450,300,200];
t2: TempRead = [200,100,200,250,210];
t3: TempRead = [500,410,410,450,390];
t4: TempRead = [500,480,460,450,440];
t5: TempRead = [100,200,300,400,500];
t6: TempRead = [400,300,450,300,500];
```

The first 2 lines are comments (prefixed by "-- "). The **values** keyword indicates the beginning of new section of the file, which is to contain test values. Ensure you fix any syntax or type errors indicated before moving onto the next exercise.

4. Debug Configurations

We'll now set a Console debug configuration in the same way as in tutorial 1. From the **Run** menu choose **Debug Configurations**, and double-click VDM-SL from the left-hand pane. Call the new configuration *Monitor Console* and use the Browse button to locate the MonitorSL project. Select the **Launch Mode** as *Console*, and click **Debug**. This should open a VDMJ debugger session in the console, as shown below:



a) Use the **print** command as shown above to evaluate expressions in the environment: e.g.

```
print Rising(t1)
```

This will use the variables defined in your values file to test the **Rising** function.

b) Now look at the values of t1, t2, t3, t4 and t5 and the definitions of the functions OverLimit and Safe. Predict the result of the following expressions. Use the Debug Console as in part (a), to check your answer. Remember to use the `print` command to evaluate each expression.

OverLimit(t1)
OverLimit(t2)
Safe(t3)
Safe(t4)
Safe(t5)

If you are surprised by any of the results (in particular the last three "Safe" expressions), check your understanding of *implication* by revisiting the lecture notes, or asking a demonstrator.

5. Add new functions

You'll now extend the model to add two new functions.

a) A monitor reading in which **one** of the temperatures is over the limit, but **not all** readings are over the limit, is in a "Danger Zone". In the **functions** section of the model, add a function with the following signature:

```
DangerZone: TempRead -> bool
DangerZone(temp) ==
```

Complete the function definition.
HINT: a reading is in the Danger Zone if it is OverLimit but not ContOverLimit. You can make calls to these functions in the body of your DangerZone function.

b) Next, write a function `Falling` with the following signature:

```
Falling: TempRead -> bool
Falling(temp) ==
```

The function should return true if each reading in the sample is strictly less than the previous. This will require a quantified expression of the form "forall i in set ...", similar to the examples in lectures.

6. Adding state to the model

We'll now extend the monitor specification to include state. The state will hold a collection of monitors, represented as a mapping from monitorId to TempRead. At the same time, we'll remove the "magic number" of 400 that is currently used in the model, and replace it with a state variable which holds the current maximum temperature value.

a) Add the following definitions to the monitor model. They can be placed anywhere in the file.

```
types

monitorId = token;

MonitorMap = map monitorId to TempRead;

values
maxValue: nat = 400;

m1: monitorId = mk_token("m1");
m2: monitorId = mk_token("m2");
m3: monitorId = mk_token("m3");
m4: monitorId = mk_token("m4");
m5: monitorId = mk_token("m5");
m6: monitorId = mk_token("m6");

state Monitors of
 mmap : MonitorMap
 init m ==
   m = mk_Monitors({m1 |-> t1, m2 |-> t2, m3 |-> t3,
                    m4 |-> t4, m5 |-> t5, m6 |-> t6})
end;
```

The state defines a state variable `mmap`, which is the map of monitors. The initial value maps monitor ids m1 to m6 to temperature readings t1 to t6 respectively.

The values section also defines `maxValue`, the maximum temperature for a monitor which is used by the OverLimit, ContOverLimit and Safe functions. This is initialised to 400.

b) Remove the 'magic number' (400) from the model by changing all the 400s to `maxValue`.
(e.g. `temp(i) > maxValue`)

c) Add the following function to the functions section of your model. It uses a **set comprehension** expression to build a set of all monitor ids whose readings are rising (using the Rising function in the specification).

```
AllRising: MonitorMap -> set of monitorId
AllRising(monitors) == { m | m in set dom monitors & Rising(monitors(m))};
```

The **set comprehension** expression can be read as "Build a set of all monitorIds from the domain of the monitors mapping, where the temperature for each id is Rising".

d) The function can be called on the state variable mmap by running AllRising(mmap). Execute

print AllRising(mmap)

and add a comment to your specification file, after the AllRising function definition, giving the result of calling this function. For example, if the output is {mk_token("m1"), mk_token("m2")} then add a comment to your specification:

```
-- result of AllRising is {mk_token("m1"), mk_token("m2")}
```

This completes tutorial 2. Make sure you save the updated specification file before submitting it.

The evidence for completion of this tutorial is the updated specification file monitor.vdmsl. **This can be found in your Overture workspace: if you cannot find it, go to file properties in Overture: right click on the file name in the VDM Explorer pane, and in the Properties window look at the Location field.**

Submit your updated specification file monitor.vdmsl **to Ness by the deadline (15:00, 22 Feb)**