VDM-SL Toolbox
Tutorials

1. Toolbox Introduction and Guided Tour

Aims

- To introduce the VDM-SL toolbox, creating project files and using the interpreter
- To introduce the general format of a specification file
- How to configure a VDM-SL project in the toolbox
- Syntax checking and editing a model
- Introduce abstractions: record types, sets and mappings
- Datatype invariants

1 Introduction

The VDM-SL toolbox is an industrial strength support tool for the formal modelling language VDM-SL. This first tutorial is an introduction to the toolbox: subsequent tutorials will assume familiarity with the material here, so it is essential that you complete this material.

Work through the numbered tasks at your own pace. When you have completed the tasks you should discuss them with a demonstrator and obtain a signature on your tutorial record sheet.

1.1 Assumptions

- All the tutorials will assume you have a linux shell running with Exceed. From a Windows PC, look in *Departmental Software* → *Computing Science* → *Utilities* and run the shortcut *linux.cs*, which will start a ssh connection to linux.cs and ensure Exceed is running.

2 Getting started

In these tutorials you will be creating a number of files, so organise your work by creating a work directory called (for example) `csc264` and keep all csc264 files there.

1. Start the toolbox with the command `vdmgde &`. The `&` is used to run the command in the background. You should be presented with the main Toolbox window.
2. Click on Windows, then Interpreter.
3. We'll use the Interpreter later to query a model: for now, we can use it to evaluate straightforward mathematical and logical expressions. Type the following at the `vdm>` prompt, noting the answers returned after each line:

```
print 3*12
print 3 < 4
print {1,...,10}
```

Most output should be self-explanatory: notice that the interpreter can evaluate logical expressions such as $3 < 4$ as well as mathematical ones. The last expression returns a *set* of integers.

Other commands are available in the interpreter dialog window: the `help` command lists them and gives more detailed help on specific commands.

4. Exit the toolbox, by selecting Project and Exit from the main window.

3 Organising Projects, and Reading Files

So far you've looked at basic use of the interpreter, to evaluate expressions. None of these expressions has involved any variables.

To introduce variables you need to load in a specification file to the toolbox. Such files normally have the extension `.vdm`, and have a specific format.

In the following instructions you'll use the menus in the main window to set up a project and load specification files into the project. There are also toolbar buttons for quick access to most commands.

1. Go to the csc264 practicals page (*http://www.cs.ncl.ac.uk/modules/2005-06/csc264/practicals/*) and right click on `example1.vdm` to save it in your csc264 work directory.

2. Start the toolbox from the same directory, and open the interpreter window.

3. Start a new project by clicking on the Project menu, then New Project. Add the file you have just created to your new project: click on Project then Add File to Project. Find the file you've just copied. The file "example1.vdm" should now appear in the Files in the project window.

   Select the specification file by clicking on the file name.

4. With the specification file highlighted, select Actions, Syntax Check. A message should appear in the Log Window.

5. Select Actions, Type Check. In this example there are no types defined, but it is a good habit to get into to syntax check and type check your model before trying to execute it.

6. Switch to the interpreter, then type the command

   ```
   init
   ```

   Now investigate the effect of the following commands:

   ```
   values
   print x
   print y
   print xset
   print yset
   ```

7. Now you know the values of $x, y, xset$ and $yset$, try to predict what effect the following commands will have. Then run them and see if you're right.

Command	Expected	Correct ?

`print x = 80`
`print x in set xset`
`print xset union yset`		

8. In the interpreter, you can only query the model: you can't modify it. In order to do that, you have to modify the specification file.

9. Use your favourite text editor to open the example1.vdm file you saved in your work directory. You should find it contains the following lines:

```
values

-- Test values for use with toolbox interpreter
x = 45;
xset = {1,...,50};
y = 32;
yset = {i ** 2 | i in set xset & i <= 10};
```

   This is the shortest specification file you're likely to come across. It features:

   - A keyword, `values`, which denotes the beginning of the values section of the file.
   - A comment, indicated by `--`. Anything after `--` anywhere on a line is ignored by the interpreter.
   - Four lines, terminated by semicolons, binding values to the variables `x`, `y`, `xset` and `yset`.

   The value for `yset` uses a *set comprehension* to build a set of the squares of all numbers in the set `xset` that are less than or equal to 10. Note that the comprehension consists of an *expression*, `i ** 2`, a *binding*, `i in set xset`, and a *predicate*, `i <= 10`.

10. Modify the example1.vdm file to remove the semicolon from the line `y = 32;`, and save the file.

11. Syntax check the file again. An error window should appear, telling you where the error occurred and what may have gone wrong. Be sure you understand the error message before returning to your file and fixing the error. Save the file, and re-run the Syntax Check. This should now pass - if not, check that you have saved the file.

12. Save the project by selecting Save Project from the main window. Give the project a name such as `tutorial1`. Then exit the toolbox.

The rest of the tutorial uses the material presented in the lectures on the Guided Tour, using the model of the Chemical Plant Alarm System.

4 Specification template

You've already seen a simple specification file which only had a "values" section. A standard specification will also include types and functions.

1. Go to the csc264 practicals page (*http://www.cs.ncl.ac.uk/modules/2005-06/csc264/practicals/*) and copy the file `template.vdm` into your csc264 work directory.

2. Use an editor to open this file. It should look like Figure 1.

   The file contains a header comment and three main headings: `types`, `functions` and `values`. Edit the header comment by adding relevant details and then save the file under a new name (such as `tour.vdm`).

3. Start the VDM-SL tool (`vdmgde &`) and begin a new project. Add this file to your project.

4. Syntax check and type check the file. It's empty apart from the headings so the check should pass.

```
--------------------------------------------
-- Author:
-- Created:
-- Updated:
-- Description:
--------------------------------------------

--
-- Types definition section
--
types


--
-- Functions definition section
--
functions


--
-- Values definition section
--
values
```

Figure 1: VDM-SL specification template

5 Types

Now you'll define the types which make up the alarm system.

1. Go to the csc264 practicals page and save the file `alarmtypes.vdm` in your csc264 work directory. Open it in an editor, copy and paste the datatypes (everything after the `types`) line into the types section of your specification file.

2. Save the specification file, then syntax check it. You should have one error, which is deliberate: use the error tool to locate and fix the error. Ask a demonstrator if you're not sure (or if you have several errors!). After fixing the error, save the file, re-run the syntax check and the type check.

6 Adding test values

We haven't defined any functions yet, but we can still test the model by defining some test values.

1. Enable dynamic type checking in the interpreter. This ensures that a type check is run on any values defined in the model. Also enable checking of pre- and post-conditions. Choose Project, Project Options, then Interpreter, then enable all the options listed for the interpreter. **Always remember to do this each time you begin using the toolbox, as the options aren't saved between sessions.**

2. Add the following lines to your specification file, under the `values` heading:

```
e1 : ExpertId = mk_token("e1");
e2 : ExpertId = mk_token("e2");
wilma : Expert = mk_Expert(e1,{<Elec>,<Chem>});
fred : Expert = mk_Expert(e2,{});
```

This demonstrates the use of the **record** type constructor, `mk_`. First `mk_token` is used to create two new **tokens**, for new `ExpertId` elements `e1` and `e2`. Then `mk_Expert` creates two new values of type `Expert`. These are then given the variable names `wilma` and `fred`. Compare the arguments to `mk_Expert` with the type definition for `Expert`.

3. Save, syntax check and type check your file. Assuming no problems, open the interpreter and initialise the specification (press Init). Examine your model using the commands `types`, `values` and `print`.

4. Add a new value to your model for the expert to be identified by the variable name `barney`, using a token `e3`. The qualifications will be `<Bio>` and `<Elec>`. Syntax check, type check and re-initialise your model.

How can you check that the new value has been set correctly?

5. The `Alarm` datatype is another record type. Define the following three alarms as test values, by adding them to your model[1]. Use the record constructor `mk_Alarm`.

Variable name	*Text*	*Qualification*
`alarm1`	"Power supply missing"	`<Elec>`
`alarm2`	"Tank overflow"	`<Mech>`
`alarm3`	"CO2 detected"	`<Chem>`

Remember to syntax check, type check and re-initialise your model, then use the interpreter to check these new values.

We'll now add a few more test values.

6. Save the file `alarmval.vdm` from the csc264 practicals page to your work directory.

7. Paste the new values (everything below the `values` lines) at the end of your specification file. Syntax and type check your specification.

   The `alarmval.vdm` file defines 8 experts (`ex1, ex2, ..., ex8`), 4 time periods (`p1, ..., p4`), 3 alarms (`a1, a2, a3`), a schedule (`sch`) and the overall plant (`plant`).

8. To access the fields of a record element, a "." notation is used. For example: evaluate the following expression in the interpreter:

```
print wilma.expertid
```

***Using the interpreter**, query the model to find out:*

(a) the set of qualifications held by the expert `ex3`

(b) the text for the alarm `a2`

[1] A text string is represented in VDM-SL as a sequence of characters (`seq of char`). However, you can use double quotes (") to denote a string value, like this: `"this is a string"`.

7 Datatype invariants

1. After discussions with the client, it is decided that experts must have at least one qualification. This can be modelled using a **data type invariant** in the `Expert` datatype. Edit your specification to include this invariant by changing the `Expert` datatype as follows:

   ```
   Expert :: expertid : ExpertId
             quali    : set of Qualification
   inv ex == ex.quali <> {};
   ```

   This defines an invariant over a typical element, `ex`, of the type `Expert`. The invariant will be satisfied as long as the set of qualifications held by the expert is not empty (not equal to the empty set).

2. Syntax check and type check your model. If you have problems here, check that the invariant has been added to the correct type (`Expert`), that it has been defined with a double equals sign (`==`), and that it has been placed *before* the semicolon. Discuss with a demonstrator if you have a problem.

3. Now return to the interpreter, and re-initialise the specification.

Explain why there is a problem with your test values. (Hint: if there isn't a problem, it'll be because you didn't enable dynamic type checking previously)

4. Modify your test values to satisfy the invariant.

This completes the first tutorial. It has introduced a number of key features of modelling and the use of the toolbox to investigate a model. Remember to get a signature from a demonstrator when you have finished this tutorial.