VDM Tools Introduction

Aims

- Syntax checking, type checking and editing a model
- Revise previous lab questions about Types, Instance variables, Values and Invariants
- Introduce operations (Explicit/Implicit)
  - Able to differentiate the syntax for both Explicit and Implicit operation
- Using the interpreter to test your model
- Check the correct data type required for each operator.

1. Case Study: Student

Write a formal specification about student. This formal specification will capture general attributes such as name, date of birth, year of study and course taken. It also has a simple behavior which is to calculate the age of student and also total credit hour taken.

1. Define a class of student.
2. Define the following required data types:
   2.1. Month
   (Tips: Define using union type. Example of value – January, February,…, December.)
   2.2. Date
   (Tips: Define using product type. Example of value – 01 September 2013, can use the month data type defined in 2.1.)
   2.3. Course
   (Tips: Define using composite data type. This course data type should have 3 elements, which are the course code, course name, and also course credit hours. Example of value – {course code, course name, course credits} = {TME4013, Formal Method, 3})
3. Define a constant hour required for 1 credit hour.
   (Tips: 1 credit = 1 hour)
4. Define the following attributes for a student:
   4.1 Student's Name (string)
   4.2 Date of birth (date)
   4.3 Year of study (number)
   4.4 Course registered (a set of course)
5. Define an invariant for Year of Study with a condition that the value of year of study should be greater than or equal to 1.

6. Write explicit operations for the following requirements:

```
explicit operation definition =
operation name, ':', discretionary type, '==>',
  discretionary type,
operation name, '(', parameters, ')', '==',
  operation body,
['pre', expression],
['post', expression]
```

Figure 1.0 Syntax for explicit operation

The general form of Explicit Operation definition is:

```
op: A * B * ... ==> R
  op(a,b,...) ==
      statements
  pre expression
  post expression
```

```
op: TypeA * TypeB * ... ==> TypeR
  op(varName-a,varName-b,...) ==
      statements
  pre expression
  post expression
```

Figure 2.0 General form of explicit operation

6.1 Assign a parameter (currentYear) value to yearOfStudy variable that defined in the class.

6.2 Assign parameters (day, month, year) value to dateOfBirth variable that define in class.

(Tips: dateOfBirth is productType, assign using token. Mk_(x,…,x_n).

6.3 Return the year of birth from dateOfBirth variable.

6.4 Calculate student age.

(Tips: make use of operation from 6.3, and yearOfStudy variable)

6.5 Calculate total course credit hours based on total number of credit, with a precondition that total credit should be greater than 0.

(Tips: parameter : totalCredit; write a pre condition)

7. Create a new file implicit.vpp, model implicit operations for the following requirements, and compare what are the differences between the two operations from Question 6.

```
implicit operations definition =
operation name, '(', [parameter types], ')', [result
type],
['ext', instance variable information],
['pre', expression],
'post', expression,
[exceptions]

instance variable information = mode, instance
variable name, [':', type]
{mode, instance variable name, [':', type] }

mode = 'rd' | 'wr';
exceptions = 'errs', error list;
```

Figure 3.0 Syntax for Implicit operations

The general form of Implicit operations definitions is:

```
op(a:A, b:B,...) r:R
  ext rd ...
      wr ...
  pre expression
  post expression
```

Figure 4.0 General form for implicit operations

7.1 Year of Study of student. (Post condition Year of study = current year)

7.2 Date of birth of students.

7.3 Calculation of student's age.

2. Test Model

1. By using the specification you create in part 1, Open Interpreter > Init.
2. At the command line, in the interpreter window, create a new object for class Student.
   - To create a new object:

     >> create student := new Student()
3. Set year of study equal to 2013 by calling the setYearOfStudy() operation.
   - To set the year of study which is equal to 2013:

     >> print student.setYearOfStudy(2013)
4. Set date of birth of student to 1, January 1995 by calling the setDateOfBirth() operation
   - To set the date of birth of student to 1, January 1995:

     >> print student.setDateOfBirth(1, <January>, 1995)
5. Call the operation calculateAge() to generate student's age.
6. Call the operation calculateTotalHour() to calculate the total hour of credit.

3. Exercise: Error Checking for Data Type required for each operator

Operator	Name	Type Signature
not b	Negation	**bool** -> **bool**
a **and** b	Conjunction	**bool** * **bool** -> **bool**
a **or** b	Disjunction	**bool** * **bool** -> **bool**
a => b	Implication	**bool** * **bool** -> **bool**
a <=> b	Biimplication	**bool** * **bool** -> **bool**
a = b	Equality	**bool** * **bool** -> **bool**
a <> b	Inequality	**bool** * **bool** -> **bool**

Figure 5.0: Boolean Type

Take an example of Conjunction: a **and** b, has type signature of bool * bool -> bool, this is an "and" Boolean operator, thus it requires 2 Boolean types, and return a Boolean type.

7. Find and correct the following errors by using VDM tool:

```
class TypeOperator
        instance variable
                private chrMale:char:='m';
                private chrFemale:char:="f";
                private strBool:seq of char := "false";
                private strTotal:seq of char := "100";
                private strMale:seq of char := "male";
                private boolTrue:bool := true;
                private boolFalse:bool := "false";
                private natValue:nat:=12;
                private intValue:int := 2;
                private boolResult:bool;
                private strResult:seq of char;
                private natResult:nat;
                private realResult:real;

end TypeOperator
```

8. Add the following operation into the class created in Question 1, try to spot and correct errors occurred.

```
public testOperatorSignature:()==>()
        testOperatorSignature()==
        (
                boolResult := ( chrMale = strMale);
                boolResult:=strBool and boolTrue;
                boolResult:=strTotal>natValue;
                strBool := not boolTrue;
                natResult:=strTotal + intValue;

        )
```

End of Lab 2