Formal Method in Software Engineering (SE-313)

Course Teacher
Assistant Professor
Engr. Mustafa Latif

Introduction to Vienna Development Method (VDM)

Vienna Development Method (VDM)

- VDM dates from work done by the IBM research laboratory in Vienna in the late 1960s.
- VDM is a "model-oriented approach", and this means that an explicit model of the state of an abstract machine is given, and operations are defined in terms of this state.
- Operations may act on the system state, taking inputs and producing outputs and a new system state.
- Operations are defined in a precondition and postcondition style.
- Each operation has an associated proof obligation to ensure that if the precondition is true, then the operation preserves the system invariant.
- VDM specifications are structured into modules, with a module containing the module name, parameters, types, operations etc.

Vienna Development Method (VDM)

- Example:The following is a very simple example of a VDM specification. It is a simple library system that allows borrowing and returning of books. The data types for the library system are first defined, and the operation to borrow a book is then defined. It is assumed that the state is made up of three sets, and these are the set of books on the shelf, the set of books which are borrowed, and the set of missing books. These sets are mutually disjoint.

Vienna Development Method (VDM)

```
types

        Bks = Bkd-id set

state Library of

        On-shelf : Bks            inv mk-Library (os, mb, bb) Δ is-disj({os,mb,bb})
        Missing : Bks             end
        Borrowed : Bks            borrow (b:Bkd-id)
                                  ext wr on-shelf, borrowed : Bks
                                  pre b ∈ on-shelf
                                  post on-shelf = on-shelf↼ - {b} ∧

                                          borrowed = borrowed↼ ∪ {b}
```

Vienna Development Method (VDM)

- A VDM specification consists of
  - Type definitions
  - State Definitions
  - Invariant for the system
  - Definition of the operations of the system
- The notation Bkd-id set specifies that Bks is a set of Bkd-ids, e.g. Bks = {b1, b2,… bn}.
- The invariant specifies the property that must remain true for the library system.
- The borrow operation is defined using preconditions and postconditions.
- The notation "ext wr" indicates that the borrow operation affects the state, whereas
- The notation "ext rd" indicates an operation that does not affect the state.

Vienna Development Method (VDM)

Sets

- Sets are a key building block of VDM specifications. A set is a collection of objects that contain no duplicates. The set of all even natural numbers less than or equal to 10 is given by:
  - S { 2, 4, 6, 8, 10}
- Sets may be specified by enumeration (as in S = {2, 4, 6, 8, 10}). And set comprehension form: {set membership | predicate}
- Cardinality: card {7, 2, 12} = 3

Vienna Development Method (VDM)

Sets

There are a number of in-built sets that are part of VDM including

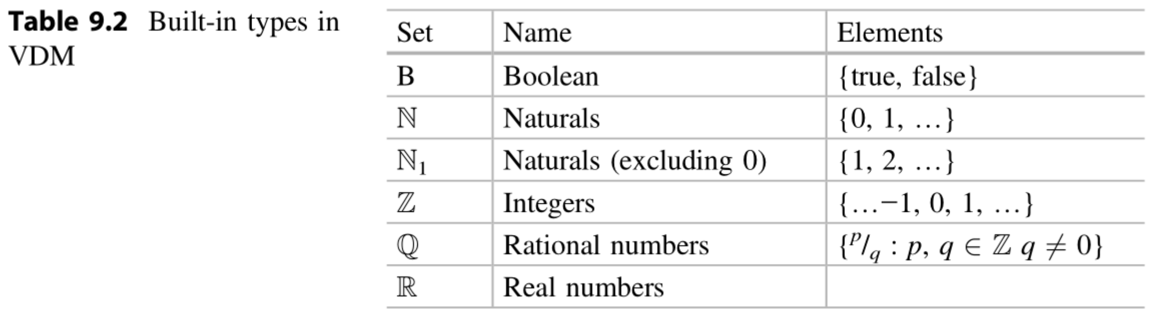


Table 9.2 Built-in types in VDM

Set	Name	Elements
B	Boolean	{true, false}
$\mathbb{N}$	Naturals	{0, 1, …}
$\mathbb{N}_1$	Naturals (excluding 0)	{1, 2, …}
$\mathbb{Z}$	Integers	{…−1, 0, 1, …}
$\mathbb{Q}$	Rational numbers	$\{^p/_q : p, q \in \mathbb{Z}\ q \neq 0\}$
$\mathbb{R}$	Real numbers	

Vienna Development Method (VDM)

- Sequences
- A sequence is a collection of items that are ordered in a particular way.
- Duplicate items are allowed for sequences, whereas duplicate elements are not meaningful for sets (unless we are dealing with multi-sets or bags).
- A set may be refined to a sequence of unique elements.
- A sequence of elements x_1, x_2, … x_n is denoted by [x_1, x_2, … x_n], and the empty sequence is denoted by [ ].
- The length of a sequence is given by the len operator and len [] =0 ; len [1,2,6]=3
- The hd operation gives the first element of the sequence. It is applied to non-empty sequences only: hd [x,y,z]= x

Vienna Development Method (VDM)

- Sequences
- The tl operation gives the remainder of a sequence after the first element of the sequence has been removed. It is applied to non-empty sequences only: tl [x,y,z] = [y,z]
- The elems operation gives the elements of a sequence. It is applied to both empty and non-empty sequences: elems[x,y,z]={x,y,z}.
- The idx operation is applied to both empty and non-empty sequences. It returns the set {1, 2, … n} where n is the number of elements in the sequence: inds[x,y,z]={1,2,3}
- A subsequence operator is defined to allow us to extract a part of a sequence between two indices. If s=[x,y,z] then subseq(s,2,3) = [y,z]
- Two sequences may be joined together by the concatenation operator: $[x, y, z] \frown [a, b] = [x, y, z, a, b]$

Vienna Development Method (VDM)

- Maps
- Maps (also called partial functions) are frequently employed for modelling and specifications in VDM.
- A map is used to relate the members of two sets X and Y such that an element from the first set X is associated with (at most) one element in the second set Y.
- The map (partial function) from X to Y is denoted by:

$$f : X \to^{m} Y$$

- Total maps are termed functions, and a total function f from a set X to a set Y is denoted by: $f : X \to Y$

Vienna Development Method (VDM)

- Maps
- The domain of the map f is a subset of X and the range is a subset of Y.
- An example of a map declaration is: $f : \{Names \to^{m} AccountNmr\}$
- The map f may take on the values:

$$f = \{\}$$
$$f = \{eithne \mapsto 231, fred \mapsto 315\}$$

- The domain and range of f are given by:
- $$\operatorname{dom} f = \{eithne, fred\}$$
$$\operatorname{rng} f = \{231, 315\}$$

Vienna Development Method (VDM)

- **Maps**
- **The map overwrites operator f † g gives a map that contains all the maplets in the second operand together with the maplets in the first operand that are not in the domain of the second operand.**

$$f = \{eithne \mapsto 231, fred \mapsto 315\}$$

For $g = \{eithne \mapsto 412, aisling \mapsto 294\}$ then
$f \dagger g = \{eithne \mapsto 412, aisling \mapsto 294, fred \mapsto 315\}$

Vienna Development Method (VDM)

- **The map restriction operator has two operands: the first operator is a set, whereas the second operand is a map. It forms the map by extracting those maplets that have the first element equal to a member of the set. For example,**

$$\{eithne\} \lhd \{eithne \mapsto 412, aisling \mapsto 294, fred \mapsto 315\} = \{eithne \mapsto 412\}$$

- **The map deletion operator has two operands: the first operator is a set, whereas the second operand is a map. It forms the map by deleting those maplets that have the first element equal to a member of the set. For example,**

$$\{eithne, fred\} \ntriangleleft \{eithne \mapsto 412, aisling \mapsto 294, fred \mapsto 315\} = \{aisling \mapsto 294\}$$

Vienna Development Method (VDM)

Declaring Constants

- the declaration of constants come immediately before the state definition.

values

$MAX : \mathbb{Z} = 10$

$MIN : \mathbb{Z} = -10$

Specifying Functions Explicitly:

$add : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$
$add(x, y) \underline{\Delta}\, x + y$

Specifying Functions Implicitly:

$add(x : \mathbb{R}, y : \mathbb{R})\ z : \mathbb{R}$

pre TRUE

post $z = x + y$

* a function, does not access the state variables.

Vienna Development Method (VDM)

Specifying a State Invariant

- State Invariant incorporate restriction into the specification of the state. **inv**: $State \rightarrow \mathbb{B}$

inv $mk\text{-}Library\ (os,\ mb,\ bb)\ \underline{\Delta}\ is\text{-}disj(\{os, mb, bb\})$

Specifying an Initialization Function

- This function is specified after the declaration of the invariant, and prescribes the conditions that the system must satisfy when it is first brought into being. Init : State $\longrightarrow \mathbb{B}$

init mk-Library(os,mb,bb)$\underline{\Delta}$ os={} ^ mb={} ^ bb={}

The nil Value

VDM-SL allows the possibility of a term or expression having the value nil, meaning that it is undefined.

Vienna Development Method (VDM)

Specifying the Operations

- **In VDM-SL an operation consists of four sections, which we explain below. The four sections are:**

  - the operation header
  - the **external** clause
  - the **precondition**
  - the **postcondition.**

```
borrow (b:Bkd-id)
ext wr on-shelf, borrowed : Bks
pre b ∈ on-shelf
post on-shelf = on-shelf↼ - {b} ∧

      borrowed = borrowed↼ ∪ {b}
```

Vienna Development Method (VDM)

Data Types and Data Invariants

- **Larger specifications often require more complex data types.**
- **The VDM specification language allows composite data types to be created from their underlying component data types.**

```
TypeName :: fieldname1 : Type1
            fieldname2 : Type2
                       :
```

- **The composite type Time is composed of appropriate types for an hour value, a minute value and a second value.**
-

```
Time:: hour: ℕ
       minute: ℕ
       second: ℕ
```

Vienna Development Method (VDM)

- define a set of important times as follows:

$$\mathit{importantTimes} : \mathit{Time}\text{-}\mathbf{set}$$

- The most important composite object operator is the make function (mk-TypeName) that creates a new object of a given composite type.
- Here is an example of the use of a mk-Time function to create a Time object:

$$\mathit{someTime} = \mathbf{mk}\text{-}\mathit{Time}\ (16, 20, 44)$$

Vienna Development Method (VDM)

- This time should not be allowed as there are only 24 hours in a day! Just as states can have invariants defined on them, so can any composite object types you define. Here is an appropriate invariant added to the Time type definition:

$$\begin{aligned} &\mathit{Time}{::}\ \ \mathit{hour}: \mathbb{N} \\ &\qquad\quad \mathit{minute}: \mathbb{N} \\ &\qquad\quad \mathit{second}: \mathbb{N} \\ &\mathbf{inv\ mk}\text{-}\mathit{Time}\ (h, m, s) \ \underline{\Delta}\quad h < 24 \wedge m < 60 \wedge s < 60 \end{aligned}$$

- We can refer to a particular field of a composite object by using a selector operator.

$$\mathit{someTime.minute} = 20$$
$$\mathit{someTime.hour} = 16$$

Vienna Development Method (VDM)

- The mu function returns one composite object from another but with one or more fields changed.

  $newTime = \mu\ (someTime,\ hour \mapsto 15)$

Including Comments

- As with program code, the readability of a VDM-SL specification is greatly enhanced by the inclusion of comments. This is done by introducing the comment with the symbol – –. A new line ends the comment.

21

The standard template for VDM-SL specifications

types
  SomeType = ...

values
  constantName : ConstantType = someValue

state *SystemName* **of**
  attribute$_1$ *: Type*
      :
      :
  attribute$_n$ *: Type*

  inv mk-*SystemName*(i_1:*Type*, ..., i_n:*Type*) $\triangleq$ *Expression*(i_1, ..., i_n)
  init mk-*SystemName*(i_1:*Type*, ..., i_n:*Type*) $\triangleq$ *Expression*(i_1, ..., i_n)

end

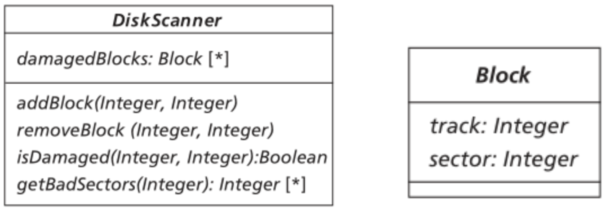
functions
  specification of functions .....

operations
  specification of operations .....

22

Vienna Development Method (VDM)

- Example: Consider a piece of software designed to keep track of damaged blocks on the surface of a disk. A disk is divided into a number of tracks and each track into a number of sectors. A block is identified, therefore, by giving both a track and sector number.


DiskScanner
damagedBlocks: Block [*]
addBlock(Integer, Integer)
removeBlock (Integer, Integer)
isDamaged(Integer, Integer):Boolean
getBadSectors(Integer): Integer [*]
Block
track: Integer
sector: Integer

UML specification of the DiskScanner class and Block type

23

Vienna Development Method (VDM)

- Specification in VDM

types
Block : : *track*: $\mathbb{N}$
sector: $\mathbb{N}$

state *DiskScanner* **of**
damagedBlocks: *Block*-**set**
init mk-*DiskScanner* (*dB*) $\underline{\Delta}$ $dB = \{ \}$
end

24

operations
addBlock (*trackIn*: $\mathbb{N}$, *sectorIn*: $\mathbb{N}$)
ext wr *damagedBlocks*: *Block*-**set**
pre mk-*Block* (*trackIn*, *sectorIn*) $\notin$ *damagedBlocks*
post $damagedBlocks = \overleftarrow{damagedBlocks} \cup \{\textbf{mk-}Block\ (trackIn, sectorIn)\}$

removeBlock (*trackIn*: $\mathbb{N}$, *sectorIn*: $\mathbb{N}$)
ext wr *damagedBlocks*: *Block*-**set**
pre mk-*Block* (*trackIn*, *sectorIn*) $\in$ *damagedBlocks*
post $damagedBlocks = \overleftarrow{damagedBlocks} \setminus \{\textbf{mk-}Block\ (trackIn, sectorIn)\}$

isDamaged (*trackIn*: $\mathbb{N}$, *sectorIn*: $\mathbb{N}$) *query*: $\mathbb{B}$
ext rd *damagedBlocks*: *Block*-**set**
pre TRUE
post *query* $\Leftrightarrow$ **mk**-*Block* (*trackIn*, *sectorIn*) $\in$ *damagedBlocks*

getBadSectors (*trackIn*: $\mathbb{N}$) *list*: $\mathbb{N}$-**set**
ext rd *damagedBlocks*: *Block*-**set**
pre TRUE
post $list = \{b.sector \mid b \in damagedBlocks \bullet b.track = trackIn\}$[1]

[1] {*expression* (*x*) | *binding* (*x*) • *test*(*x*)}

25

Industrial Applications of VDM

- VDM is a widely used formal method and has been used in industrial strength projects as well as by the academic community.
- VDM++ (the object-oriented version of VDM) has been applied to the formal specification of a real-time information system for tracking and tracing rail traffic (the CombiCom project provided a real-time information system for the Rotterdam, Cologne, Verona rail freight corridor).
- VDM++ was employed for the formal specification of a new generation of IC chip developed by FeliCa Networks in Japan.
- There is tool support available, for example, the IFAD VDM-SL toolbox and the open-source Overture IDE tool.

26