Tim Dennis

Bee Faced Christmas Workshop
'VMM stitching, Sticks'
An Approach to Structuring for the VDM Specification Language

Stephen Bear
REQUIREMENTS

- Allow a large specification to be split into person sized units, in a natural way.

- Allow an individual module, or an incomplete group of modules to be checked.

- Limit interference between separate units.

- Provide a detailed semantic definition
CONRAINTS

• It must be possible to compare the semantics of a specification given in the core language with the semantics of a specification given using the structuring constructs.
  - so we use the same basic mathematical machinery.

• The definition of the structuring constructs must preserve the semantics of the core language.
  - so we do not extend the core language.
A CHOICE

- Use simple unsophisticated mathematics
OVERVIEW OF MODULES

- As in STC VDM, the basic specification unit is a MODULE.

- A module encapsulates a collection of related types, values, functions and operations.

- Operations within a module may interact by updating values of a shared state.

- A module is similar to an algebraic ADT. It defines a state type which may be used in other modules.

- A module is an object.
MODULE SYNTAX

Module :: intf : Interface
   body : \{Definitions\}

Module
   -- description of constructs provided by, or used by, the module

Definitions
   -- collection of definitions written in the core language, but using constructs introduced by the interface

end
OVERVIEW OF IMPORT-EXPORT CONSTRUCTS

- A module may EXPORT constructs.
- Exported constructs may be imported and used by another module.
- A group of modules may import constructs from each other.
- A construct which is not exported is "hidden". A hidden construct may not be referred to by any other module.
INTERFACE - EXPORT CLAUSE

Interface :: id : Id
   exp : ModSig

ModSig :: types : Name \( ^m \) [TypeDef]
   values : Name \( ^m \) [Type]
   fns : Name \( ^m \) [FnType]
   opns : Name \( ^m \) [OpSig]

Names of constructs with optional syntactic description
FULL CONCRETE SYNTAX FOR EXPORT

Module INTEGER_STACK

exports

types INTEGER_STACK

operations

POP () \rightarrow INTEGER using INTEGER_STACK

PUSH (INTEGER) using INTEGER_STACK

definitions

end
CONCRETE SYNTAX WITH IMPLICIT STATE TYPE

Module INTEGER_STACK

  exports
  operations  POP () -> INTEGER
              PUSH (INTEGER)

definitions

end

If an operation exported by module M does not specify
the state type explicitly, then the state type is M and
the type is implicitly imported.
LIGHTWEIGHT SYNTAX

Module INTEGER_STACK

exports

operations POP, PUSH

definitions

-----

POP () \rightarrow INTEGER
PUSH (INTEGER)

-----

end

Signatures provided by the definitions need not be repeated in the export clause.
INTERFACE - IMPORT CLAUSE

Interface :: id : Id

\[ \ldots \]

imp : Id \rightarrow Mod Sig

exp : ModSig

- Id of module providing imported constructs
- Syntactic description of constructs
IMPLICIT STATE TYPE

Module SYMBOL_TABLE

    imports from INTEGER_STACK

    operations

        POP () \rightarrow INTEGER

        PUSH (INTEGER)

end

If an operation imported from a module M does not specify the state type explicitly, then the name of the state type is also M and the type is implicitly imported.
LIGHTWEIGHT SYNTAX

Module INTEGER_STACK

Exports

Operations POP () → INTEGER
PUSH (INTEGER)

end

Module SYMBOL_TABLE

imports from INTEGER_STACK

Operations POP, PUSH

end

If a document contains a module which exports a construct and another which imports it, then the signatures need to be repeated.
NAMES

Name :: prefix : seq1 of Id
    local : Id

module INTEGER_STACK
    exports
        operations POP, PUSH

end

Names reflect module structure directly – the prefix indicates where the construct is defined.

The full name of POP is INTEGER_STACK.POP
NAMES OF IMPORTED CONSTRUCTS

Module INTEGER_STACK
    exports
    operations POP, PUSH
    ___
    end

Module SYMBOL_TABLE
    imports from INTEGER_STACK
    operations POP, PUSH
    ___
    end

- The full name of a construct is not changed if it is imported.
- In both modules the full names of POP and PUSH are
  INTEGER_STACK.POP
  INTEGER_STACK.PUSH
SEMANTICS OF THE CORE LANGUAGE

- The semantics of the core language is given in terms of models.
- A model is a mapping which gives a denotation to a named construct.

```
INTEGER_STACK -->
POP -->
- - -
PUSH -->
```

MODELS OF A SPECIFICATION

A model may - or may not - satisfy a specification

definitions

type RED = - - -
type BLUE = - - -

RED ->
- - -

BLUE ->
- - -
SEMANTICS OF A SPECIFICATION

• The models which satisfy a specification are picked out by a relation

$$\text{is-a-model-of} \subseteq \text{MODELS} \times \text{Definitions}$$

• The semantics of a specification is defined to be the set of all models which satisfy the specification.

$$[[\text{spec}]] \triangleq \{ M \in \text{MODELS} \mid M \text{ is-a-model-of spec} \}$$

• Defining this relation is a non-trivial task
UNDEFINED CONSTRUCTS ARE UNDETERMINED
MODELS MAY CONTAIN JUNK

\[
\text{ME } [\text{spec}] \land n \notin \text{dom } (m) \implies m \cup [n \rightarrow v] \in [\text{spec}]
\]

- For any two specifications S, T, if [S] is non-empty it contains models which provide denotations for constructs defined or used by T.
module COLOURS
    exports
        types RED, BLUE
    definitions
        -
end

module COLOURED_BOXES
    imports from COLOURS
        types RED, BLUE
    definitions
        RED_BOX = box of RED
        BLUE_BOX = box of BLUE
end
MODELS OF THE COMPLETE DOCUMENT

\[
\text{[[Document]]} \bigtriangleup \text{[[COLOURS]]} \land \text{[[COLOURED_BOXES]]}
\]
OVERVIEW OF PARAMETERISATION

• A module may be parameterised by formal parameters - types, values, functions or operations.

• Within the parameterised module, the formal parameters may be used like any other construct.

• A parameterised module may be INSTANTIATED within another module. Formal parameters are replaced by actual parameters.

• Within the instantiating module, the newly instantiated constructs may be used like any other construct.
INTERFACE – PARAMETER CLAUSE

Interface::
  id : Id
  par : ModSig
  imp : Id $\rightarrow$ ModSig
  exp : ModSig

Syntactic description of formal parameters

- types, values, functions or operations.
PARAMETERISED MODULE

Module SORT

parameters
types ITEM
functions ARE_ORDERED (ITEM,ITEM) -->> Boolean
exports
functions DO_SORT (seq of ITEM) -->> seq of ITEM

end

Within a parameterised module, formal parameters may be used like any other constructs.
INSTANTIATION

Interface::
  id : Id
  part : ModSig
  imp : Id \rightarrow ModSig
  inst : Id \rightarrow Instance
  exp : ModSig

Instance::
  mod : Id
  view : Id \rightarrow Name
  sig : ModSig

An instance of a parameterised module may be created — instantiated — by another module.
INSTANTIATION OF SORT

Module SORT
parameters
types ITEM
functions ARE_ORDERED (ITEM, ITEM) --> Boolean
exports
functions DO_SORT (seq of ITEM) --> seq of ITEM
end

Module MAILING_LIST
instantiates
INTEGER_SORT as new SORT
(ITEM --> INTEGER, ARE_ORDERED --> GE)
Functions DO_SORT (seq of INTEGER) --> seq of INTEGER
end
NAMES OF INSTANTIATED CONSTRUCTS

• Suppose that a parameterised module defines a type T. Then the full name is
  P.T

• If a module M creates an instance I of P
  module M
  instantiates
    I as new P (- - - )
    type T

The full name of the instantiated construct is
  M.I.T
Module MAILING_LIST

  instantiates
    INTEGER_SORT as new SORT
    (ITEM --> INTEGER, ...)

  Functions DO_SORT (seq of INTEGER) --> seq of INTEGER

ADDRESS_SORT as new SORT

  (ITEM --> ADDRESS, ...)

  Functions DO_SORT (seq of ADDRESS) -->
    seq of ADDRESS

end

- MAILING_LIST.INTEGER_SORT.DO_SORT
- MAILING_LIST.ADDRESS_SORT.DO_SORT
Module COLOURED_BOX

parameters
  type COLOUR

exports
  type SHAPE

definitions
  SHAPE = Box of COLOUR

end

Module BOXES

instantiater
  RED_BOX is new COLOURED_BOX (COLOUR --> RED)

  type SHAPE

end
MODELS OF COLOURED BOX

COLOUR --> COLOUR
SHAPE --> SHAPE

COLOUR --> COLOUR
SHAPE --> SHAPE

COLOUR --> COLOUR
SHAPE --> SHAPE
MODELS OF BOXES

RED BOX.SHAPE

RED

RED_BOX.SHAPE

RED

RED BOX.SHAPE

RED

RED BOX.SHAPE

RED

RED BOX SHAPE

RED

RED BOX.SHAPE

RED

HPLABS - INFORMATION SYSTEMS CENTRE
SOFTWARE ENGINEERING DEPARTMENT

HEWLETT PACKARD
MODELS OF AN INSTANTIATION

- The models of the parameterised module and the instantiating module cannot be compared directly.

- Models of the instantiation.

\[
\begin{array}{ccc}
\text{Name} & R & \text{Name} \\
\text{rng } R \triangleleft M_p & \text{VAL} & \text{rng } R \triangleleft M_i \\
\end{array}
\]

\[M_i \text{ is a model of the instantiation iff there is a model } M_p \text{ of the parameterised module such that the diagram commutes.}\]
MODELS OF THE INSTANTIATION

RED

RED_SOX.SHAPE

RED

RED_SOX.SHAPE

HPLABS - INFORMATION SYSTEMS CENTRE
SOFTWARE ENGINEERING DEPARTMENT

HEWLETT PACKARD
MODELS OF THE DOCUMENT

[[document]] = [[BOXES]] \cap [[instantiate (COLOURED_BOX)]]
SUMMARY

- A specification may be defined as a collection of (parameterized) modules.

- Modules define types – decomposition by types is an established approach.

- The semantics of structuring is defined in terms of the core language.

- The semantics of the core language has not been changed.