Tim Denvir

BSS FACS CHRISTMAS WORKSHOP

'VDM Structuring, Slides'.

An Approach to Structuring for the VDM Specification Language

Stephen Bear

HPLABS - INFORMATION SYSTEMS CENTRE
SOFTWARE ENGINEERING DEPARTMENT

HEWLETT PACKARD

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

HPLABS – INFORMATION SYSTEMS CENTRE
SOFTWARE ENGINEERING DEPARTMENT

HEWLETT PACKARD

CONSTRAINTS

- 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 refered 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 () o-> INTEGER using INTEGER_STACK
            PUSH (INTEGER) using INTEGER_STACK
    definitions
        -----
    end
```

CONCRETE SYNTAX WITH IMPLICIT STATE TYPE

```
Module INTEGER_STACK
    exports
        operations    POP () o-> 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 () o-> INTEGER
        PUSH (INTEGER)
    -----
end
```

Signatures provided by the definitions need not be repeated in the export clause.

SB39

INTERFACE – IMPORT CLAUSE

```
Interface  ::     id : Id

           . . .

                      m
           imp   : Id -> Mod Sig

           exp   : ModSig
```

- Id of module providing imported constructs
- Syntactic description of constructs

HPLABS – INFORMATION SYSTEMS CENTRE
SOFTWARE ENGINEERING DEPARTMENT

HEWLETT PACKARD

IMPLICIT STATE TYPE

```
Module SYMBOL_TABLE

    imports from INTEGER_STACK

        operations    POP () o=> INTEGER
                      PUSH (INTEGER)

    -----

end
```

If an operation imported from a module M does not specify the state tpe 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 () o=> 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.

SB42

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

SB43

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

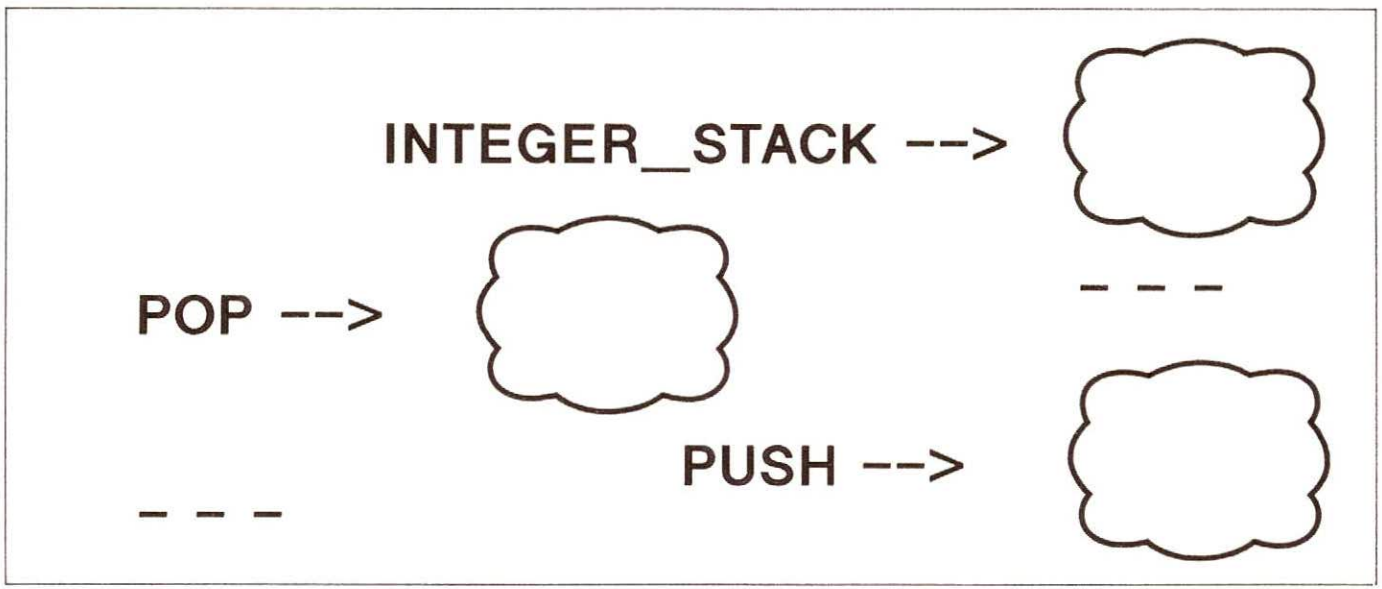
  INTEGER_STACK.POP

  INTEGER_STACK.PUSH

SB44

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.



SB45

MODELS OF A SPECIFICATION

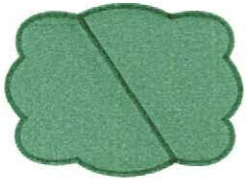
A model may – or may not – satisfy a specification

definitions
type RED = – – –
type BLUE = – – –

RED–>  – – –

BLUE –> 
– – –

RED –>  – – –

BLUE –> 
– – –

SB46

SEMANTICS OF A SPECIFICATION

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

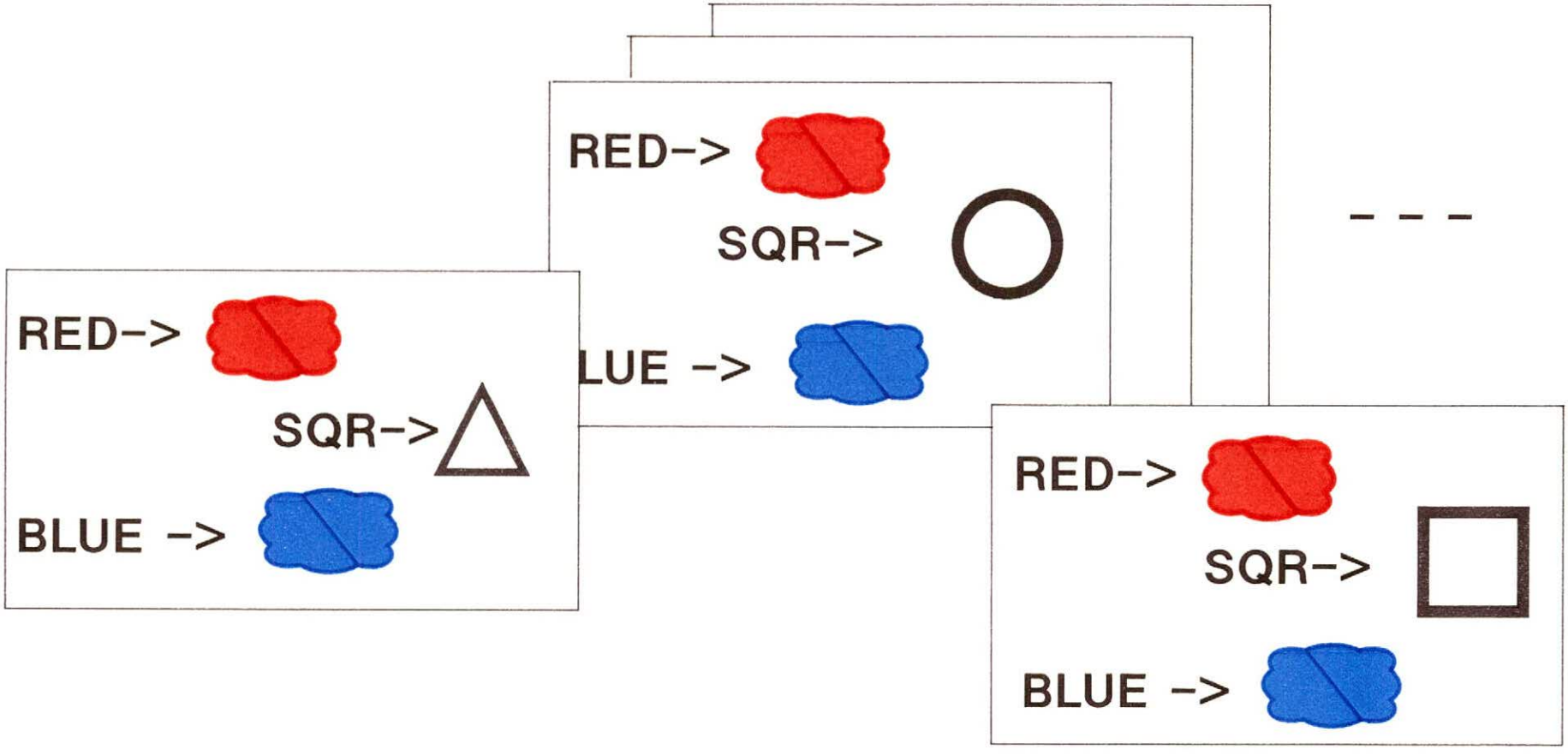
$$\text{is-a-model-of} \subseteq \text{MODELS} \times \text{Definitions}$$

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

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

- Defining this relation is a non-trivial task

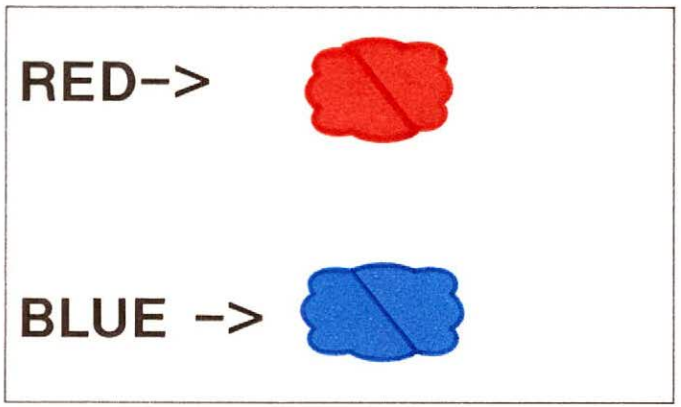
UNDEFINED CONSTRUCTS ARE UNDETERMINED



SB48

MODELS MAY CONTAIN JUNK

- ME [[spec]] ∧ n ∉ dom (m) ==> m u [n->v] ∈ [[spec]]

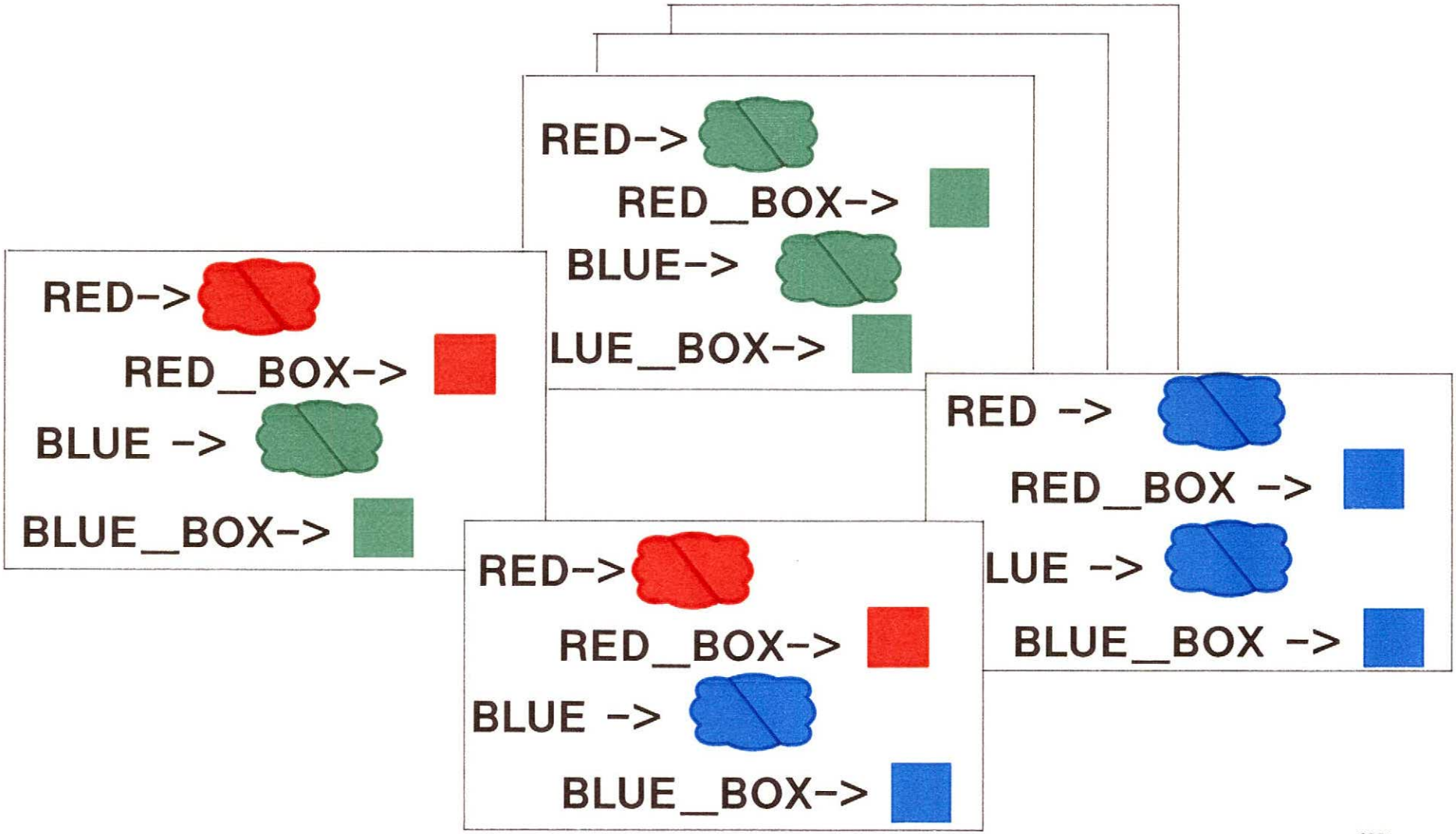


- For any two specifications S, T, if [[S]] is non-empty it contains models which provide denotations for constructs defined or used by T.

SB49

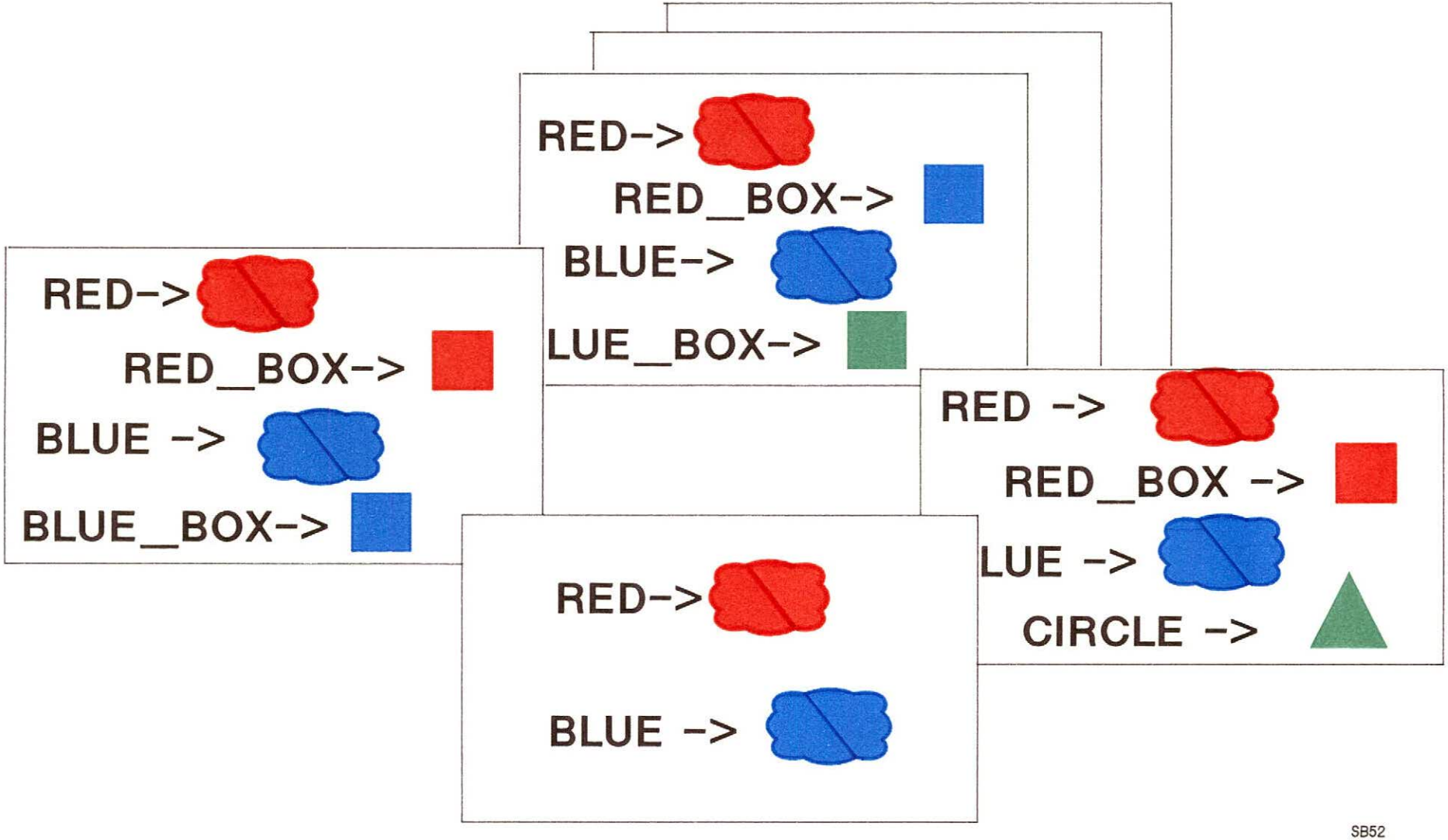
```
module COLOURS
    exports
        types RED, BLUE
definitions
- - -
end

module COLOURED_BOXES
imports from COLOURS
    types RED, BLUE
definitions
    RED_BOX = box of RED
    BLUE_BOX - box of BLUE

end
```

MODELS OF COLOURED_BOXES



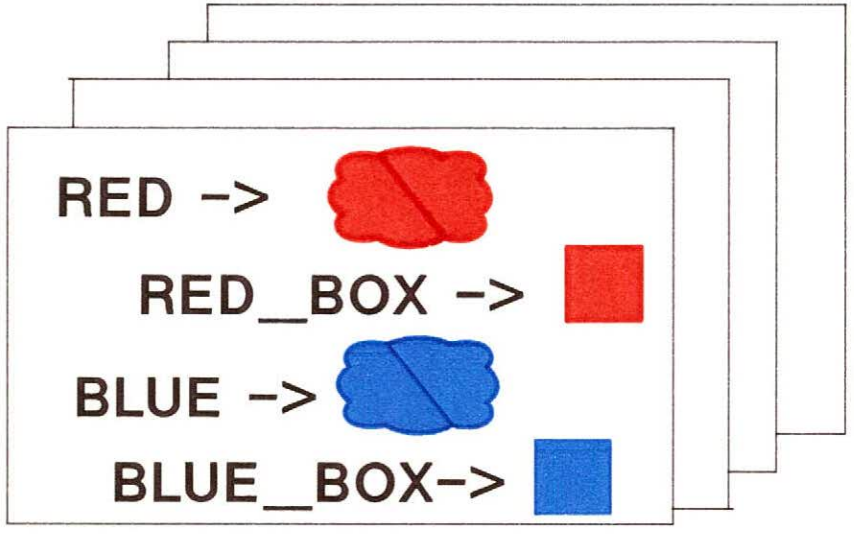
SB51

MODELS OF COLOURS



SB52

MODELS OF THE COMPLETE DOCUMENT



SB53

OVERVIEW OF PARAMETERISATION

- A module may be paramerised 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 $\overset{m}{\text{---}}$>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 $\xrightarrow{m}$ ModSig
inst ; Id $\xrightarrow{m}$ Instance
exp : ModSig

Instance:: mod : Id
view : Id $\xrightarrow{m}$ Name
sig : ModSig

An instance of a parameterised module may be created – instantiated – by another module

SB57

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
```

SB58

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

SB59

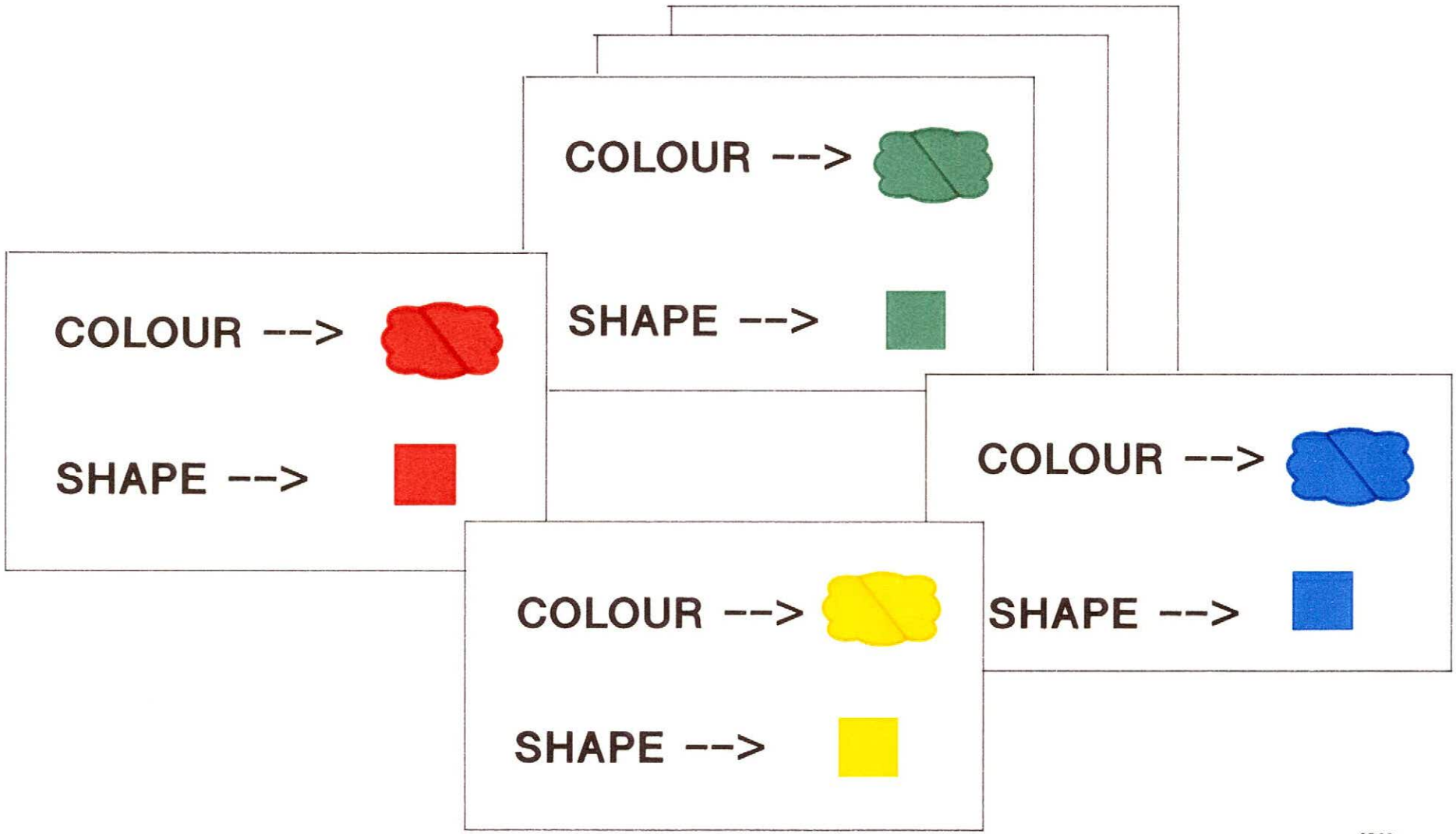
```
Module MAILING_LIST
    instantiates
        INTEGER_SORT as new SORT
        (ITEM --> INTEGER, ...)
    Functions DO_SORT (seq of INTEGER) --> seq of INTEGER
    ADDRESS_SORT as new SORT
        (ITEM --> ADDRESS, ...)
    Functions DO_SORT (seq of ADDRESSS) -->
                    seq of ADDRESS
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
```

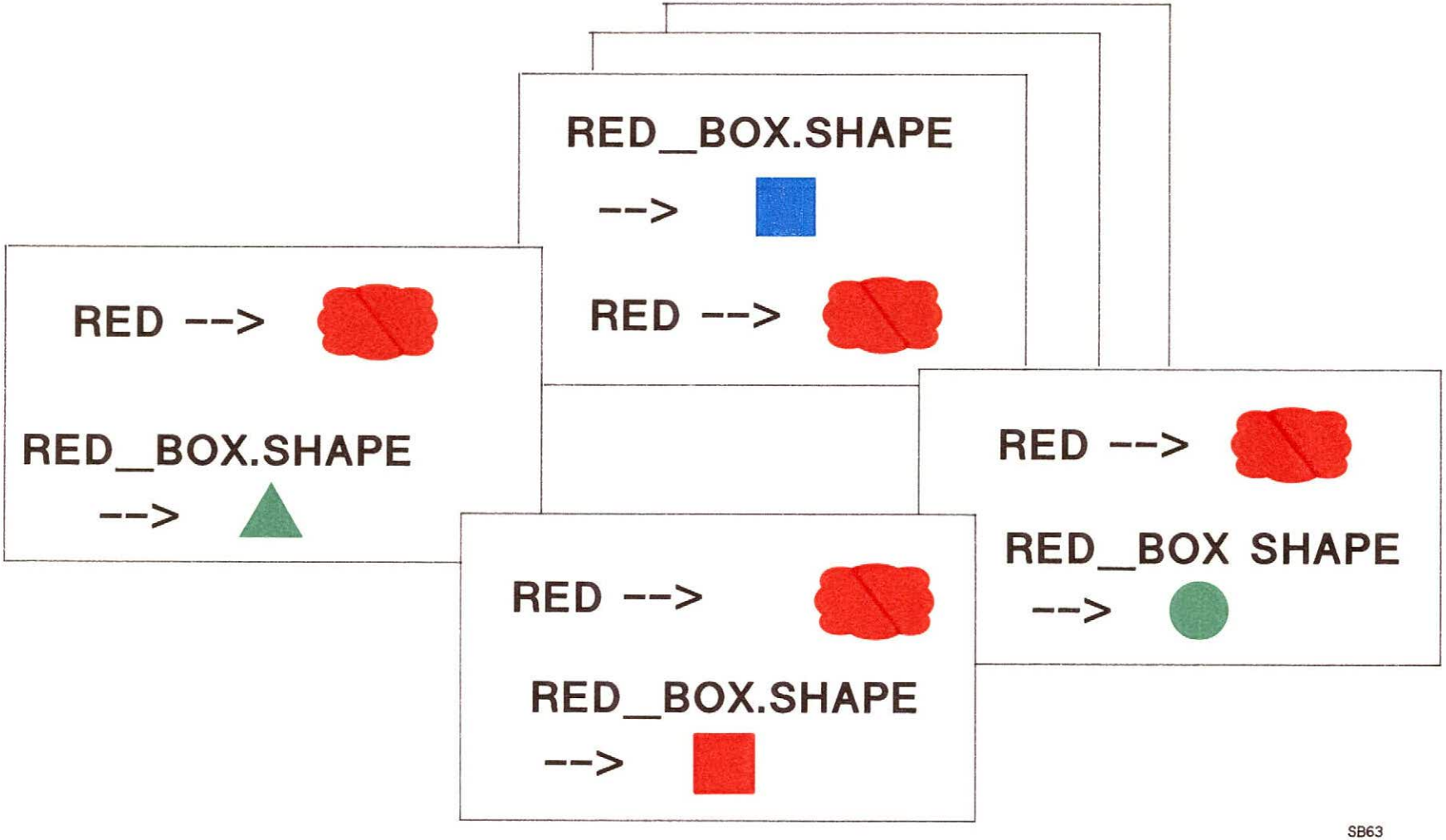
SB61

MODELS OF COLOURED_BOX



SB62

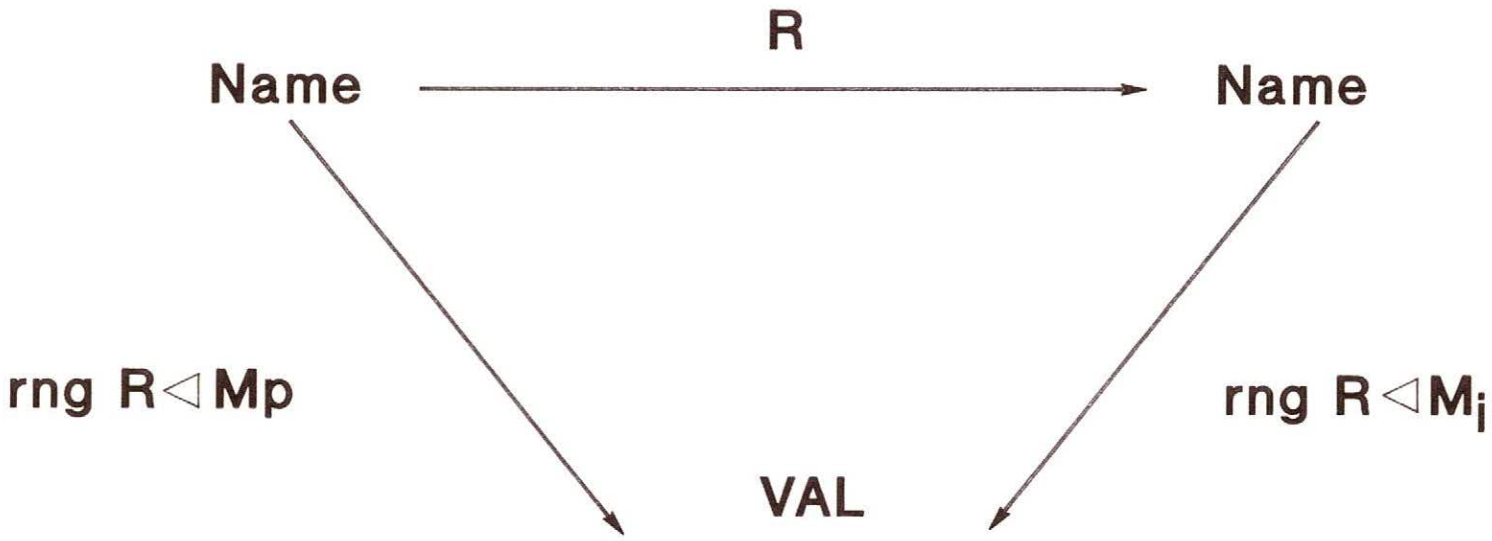
MODELS OF BOXES



SB63

MODELS OF AN INSTANTIATION

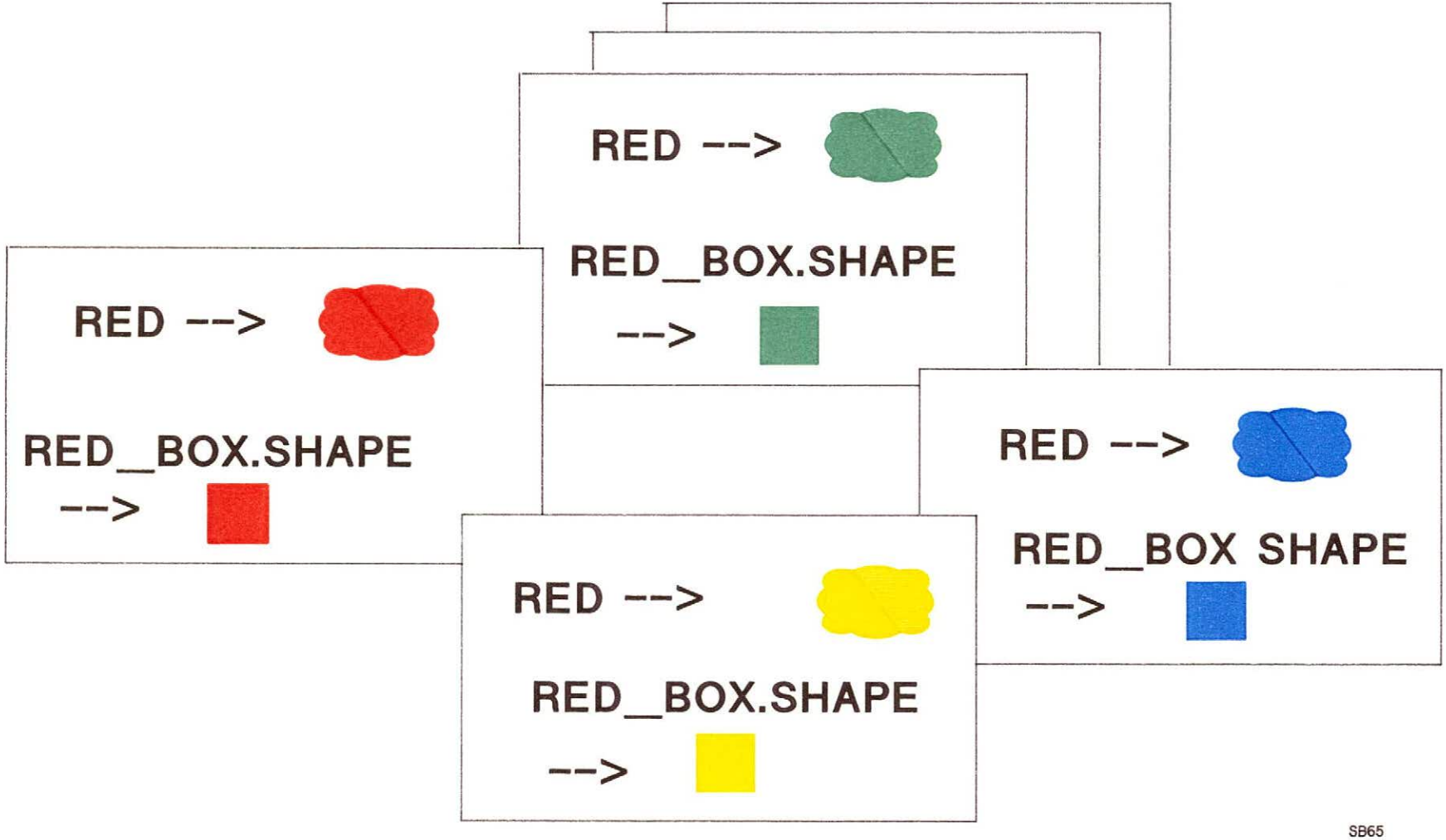
- The models of the parameterised module and the instantiating module can not be compared direclty.
- Models of the instantiation.



M_i is a model of the instantiation iff there is a model Mp of the parameterised module such that the diagram commutes.

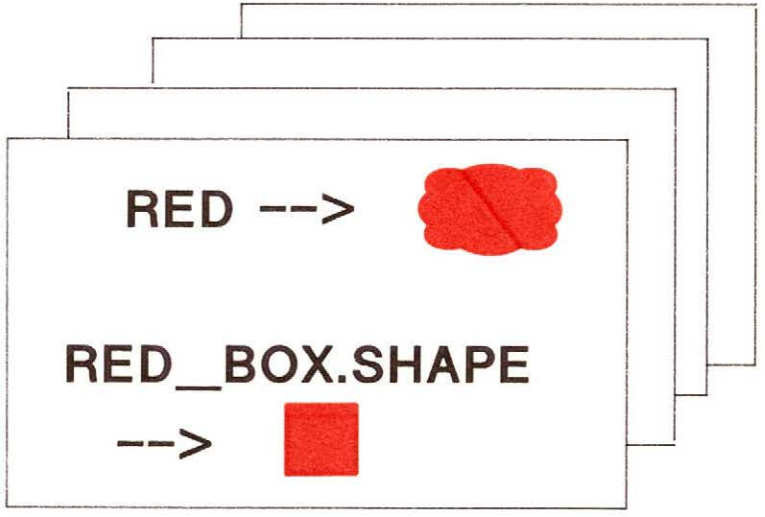
SB64

MODELS OF THE INSTANTIATION



SB65

MODELS OF THE DOCUMENT



$$[[\text{document}]] = [[\text{BOXES}]] \frown [[\text{instantiate (COLOURED\_BOX)}]]$$

HPLABS - INFORMATION SYSTEMS CENTRE
SOFTWARE ENGINEERING DEPARTMENT

HEWLETT PACKARD

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.