

Outline

Contents

1	Introduction	1
2	MACHINE	2
3	VARIABLES	2
4	INVARIANT	2
5	INITIALISATION	3
6	OPERATIONS	4
7	Example	5
8	Summary	6

1 Introduction

Philosophy

- Build *functional specifications*
- User observed behaviour is the focus
- Divide and Conquer: one abstract machine per system component
- Assemblies of abstract machines are themselves abstract machines
- Emphasize the **what** and not the **how** - the black box approach
- Introduce the Abstract Machine Notation (AMN)

Abstract Machine Notation

- Specify abstract machines through a formal notation (AMN)
- Five main parts to the specification:
 - MACHINE** machine name
 - VARIABLES** machine variables or state
 - INVARIANT** constraints on the machine variables
 - INITIALISATION** starting state
 - OPERATIONS** define how machine changes state

2 MACHINE

<+-->

Declaring an Abstract Machine

- An abstract machine has a name
- All machines in a development must have different names
- Declare the machine name in the MACHINE heading:

```
MACHINE CoffeeClub0
```

3 VARIABLES

Declaring the Abstract Machine State

- Variables in the abstract machine must be *declared*
- The *values* of these variables have *type*
- This type is defined by a *set* of values, from which may be drawn the value of the variable.
- This is a set in *the mathematical sense*
- *Hence the need for a thorough understanding of Set Theory*

An Aside on Sets

- A set in B Tools is a collection of set elements, where every element is consistent in some way with its fellow elements
- Any finite set can be considered to be a subset of some “grand master set”, which is the *set type*
- *VOLVOS*, *HOLDENS* and *FORDS* can be considered to be subsets of the (set) type *CARS* (or even *VEHICLES*)
- Elements of sets may themselves be sets, but they must be sets of the same thing (type)
- *functions* (ordered pairs of sets) are the primary source of data modelling in B

4 INVARIANT

<+-->

Typing the Variables

- Unlike conventional programming languages, B types its variables in a separate clause
- Reason is that types are a *constraint* on the variable's behaviour
- Variable typing is called a *trivial constraint*

Publication	Programming
INVARIANT	INVARIANT
$a \in \mathbb{N} \wedge$	$a : \text{NAT} \ \&$
$b \in \text{CARS}$	$b : \text{CARS}$

- Note the *conjunction* (anding, \wedge) of the constraints

Constraining the Variables

- We often want to restrict the values of variables in some way
- Use a *non-trivial constraint*
- For example, suppose we want to declare a variable that represents a Holden car. It is a variable of type *CARS*, but can only have values from some subset of car:

INVARIANT
 $holden \in \text{CARS} \wedge$
 $holden \in \text{HOLDENS}$

- Constraining variables is one way of applying *divide and conquer*

5 INITIALISATION

Initialising the Variables

- The abstract machine must start in a known state
- Define this state by giving initialisations for all variables
- Initial values *must* satisfy the invariant!
- Example:

INITIALISATION
 $holden := barina$

- (We assume that $barina \in \text{HOLDENS}$ is true!)

6 OPERATIONS

Changing the State of the Machine

- Operations are the *only* way in which the state of the abstract machine may be changed
- Whenever variables are changed, the new values must still satisfy the invariant
- The changing of all variables is *atomic*
- Example:

OPERATIONS

```
upgrade ( new )  $\hat{=}$   
PRE  
    new  $\in$  HOLDENS  
THEN  
    holden := new  
END
```

Operation Definitions

- Example:

OPERATIONS

```
output  $\leftarrow$  name ( input )  $\hat{=}$   
PRE  
    preconditions  
THEN  
    variable updates (including output)  
END
```

- Name
- Input Parameters
- Output Parameters
- Preconditions
- Variable Updates or state change definition (atomic)
- Output variable definition

7 Example

A Simple Abstract Machine

As a simple first machine we will model a simple coffee club, in which the members add money to a "piggy bank" and occasionally take money out of the bank to buy provisions for the club.

In our model we will use a variable `piggybank` whose value is a natural number, representing the contents in cents.

Also, we will use operations

FeedBank(amount) add amount to the piggy bank;

RobBank(amount) take amount out of the piggy bank;

money ← CashLeft an enquiry operation that returns, in money, the total of the contents of the piggy bank.

First Abstract Machine

```
MACHINE           CoffeeClub0
VARIABLES         piggybank
INVARIANT         piggybank : NAT
INITIALISATION   piggybank := 0
```

First Abstract Machine

```
OPERATIONS
FeedBank(amount) =
  PRE amount : NAT
  THEN piggybank := piggybank + amount
  END;
RobBank(amount) =
  PRE amount : NAT
  THEN piggybank := piggybank - amount
  END;
money <-- CashLeft =
  BEGIN money := piggybank END
  END
```

First Abstract Machine: Problem!

- There is a problem with `CoffeeClub0`
- Can you spot it?
- One way to check is to *animate* the specification

```
INVARIANT         piggybank : NAT
...
RobBank(amount) =
```

```
PRE amount : NAT
THEN piggybank := piggybank - amount
END;
```

8 Summary

Summary

- Describe behaviour as that of an abstract machine
- Declaration of abstract machine components
- Simple example of an abstract machine