Semantics of machine operations

- To be able to check formally that operations work as they supposed to, we have to assign precise mathematical meaning (semantics) to them
- Operations in general are concerned with changing the local state and setting output variables
- A specification of an operation basically describes the relationship between initial (before) and final (after) states

Machine states

- Machine states include the combinations of all possible values of (input, output, local) machine variables
- Machine states can be modelled as values of cartesian product on machine variable types, for example, $NAT \times NAT$ for two variables of natural numbers
- An operation is then state transformation described using Abstract Machine Notation

Different kinds of state transformation

- Functional (deterministic) execution —
  one-to-one relationship between initial and final states
- Non-deterministic execution — one-to-many relationship
- Aborting execution — one-to-zero relationship
- Infeasible execution — execution is in "waiting mode" or "hybernation"

Weakest preconditions

- We are often interested only in certain "expected" or "acceptable" final states
- A predicate P describing a set of "acceptable" final states is called a *postcondition*
- $[S]P$ denotes all initial states from which execution of S is guaranteed to achieve P. $[S]P$ – *weakest precondition*
- The B Method provides rules for calculating weakest preconditions for different statements

Weakest precondition rules for some B Statements

[x := e] P	$\equiv$	P[e/x]
[x,y := e1,e2] P	$\equiv$	P[e1,e2/x,y]
[skip] P	$\equiv$	P
[PRE E THEN S END] P	$\equiv$	E $\wedge$ [S]P
[IF E THEN S1 ELSE S2 END] P	$\equiv$	(E $\Rightarrow$ [S1]P) $\wedge$ (¬E $\Rightarrow$ [S2]P)
[CASE E OF EITHER e1 THEN S1 OR e2 THEN S2 OR ... OR en THEN Sn ELSE T END] P	$\equiv$	(E=e1 $\Rightarrow$ [S1]P) $\wedge$ (E=e2 $\Rightarrow$ [S2]P) $\wedge$ ... (E=en $\Rightarrow$ [Sn]P) $\wedge$ (E $\neq$ e1 $\wedge$ ...E $\neq$ en $\Rightarrow$ [T]P)

Need for consistent specifications

- Software development using B is based on mathematical (logical) proof
- In logics,

  false $\Rightarrow$ any_statement

  i.e. anything can be proved from false assumptions
- Analogously, an inconsistent (contradictory) specification can be implemented by any program
- To prevent this, B Method forces us to check consistency of an initial specification

Machine consistency conditions

- To check machine consistency, B generates four sets of proof obligations:
  - Constraint POs – to check that sets and constants given as machine parameters exist
  - Context POs – to check that local sets and constants satisfying given properties exist
  - Initialisation POs – to prove that initialisation assignment establishes a state satisfying the machine invariant
  - Operation POs (for each operation) – to prove that the operations maintain (preserve) the invariant.

Inconsistency of operations

Operations can be inconsistent because

- Operation precondition is too weak
- Operation body is not correct
- Invariant is too strong
- Invariant is too weak
- Invariant is simply wrong

Full machine consistency

Consistency conditions for a machine

```
MACHINE M(p)
CONSTRAINTS C
SETS St
CONSTANTS k
PROPERTIES B
VARIABLES v
INVARIANT I
INITIALISATION T
OPERATIONS
  y ← op(x) =
    PRE P
    THEN S
    END;
  ...
END
```


Full machine consistency (cont.)

- Consistency of constraints C:
  $$\exists p \bullet C$$
- Consistency of properties B:
  $$C \Rightarrow \exists St, k \bullet B$$
- Consistency of invariant I:
  $$C \wedge B \Rightarrow \exists v \bullet I$$
- Consistency of initialisation T:
  $$C \wedge B \Rightarrow [T]\, I$$
- Consistency of operations:
  $$C \wedge B \wedge P \wedge I \Rightarrow [S]\, I$$