B structuring mechanisms

- Large specifications must be structured in order to control inevitable complexity
- B-Method provides structuring mechanisms which enable machines to be expressed as combinations of simpler machines
- Structuring mechanisms allow distinct parts be described and understood separately; Also, internal consistency conditions can be verified independently

B structuring mechanisms (cont.)

- Machine state can be separated into different machines which will be responsible for the operations on that part of state
- B-Method allows us also to describe relationships between different components (machines)
- The mechanisms that B provides to compose specifications are the **INCLUDES, EXTENDS, USES,** and **SEES** access mechanisms

Inclusion

- MACHINE M2
  INCLUDES M1
- M1 is considered to be part of the description of M2, and its state is part of M2 state
- Sets, constants, and variables of M1 are visible in M2 (read access)
- Invariant of M1 is implicitly included in M2 invariant
- M1 variables can be updated only via M1 operations; So M1 is responsible for preserving its own invariant

Inclusion (cont.)

- If M1 is a parameterised machine, then its parameters should be instantiated in INCLUDES clause
- M2 initialisation first initialises all its included machines, then executes its own initialisation
- M2 has complete control over M1 because M1 cannot be included in any other machine
- M1 should be defined completely independently of M2; No references to M2 sets, constants, variables, and operations are allowed

Promotion

- Operations of M1 are available for M2, but NOT for M2 environment, i.e. they are NOT part of M2 interface
- the PROMOTES clause lifts an operation from an included machine to have the status of an operation of the including machine
- If all operations of M1 are promoted, then M2 is really extension of M1; Then we can write EXTENDS instead of INCLUDES

Included operations

- The bodies of M2 operations can contain calls to any operations of included machines
- The syntax of operation calls is

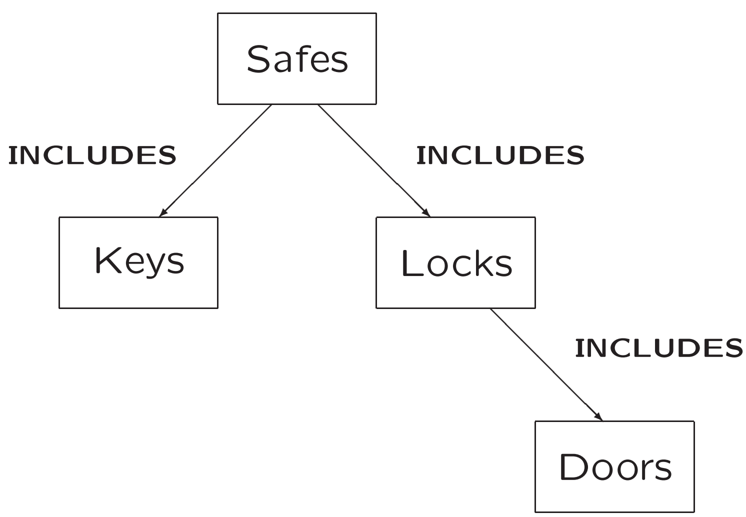
$$\mathtt{x}_1, \mathtt{x}_2, \ldots \leftarrow \mathtt{op}(\mathtt{e1}, \mathtt{e2}, \ldots)$$

- $\mathtt{e1}, \mathtt{e2}, \ldots$ are concrete value expressions, and $\mathtt{x}_1, \mathtt{x}_2, \ldots$ are distinct variables standing for actual result parameters

Multiple inclusion

- A machine can include a number of other machines, and those machines can themselves include machines
- Inclusion is transitive, i.e. sets, constants, and variables of included machines are visible independently how deeply included a machine is
- However, access to operations is not transitive
- A machine can call several operations of included machines in one step. However, those operations should be from different machines

Example



```
MACHINE Doors
SETS
  DOOR;
  POSITION = {open,closed}
VARIABLES position
INVARIANT
  position ∈ DOOR → POSITION
INITIALISATION
  position := DOOR × {closed}
OPERATIONS
  opening(dd) =
    PRE dd ∈ DOOR
    THEN position(dd) := open
    END;
  closedoor(dd) =
    PRE dd ∈ DOOR
    THEN position(dd) := closed
    END
END
```



```
MACHINE Locks
INCLUDES Doors
PROMOTES closedoor
SETS
  STATUS = {locked, unlocked}
VARIABLES status
INVARIANT
  status ∈ DOOR → STATUS ∧
  ∼position[{open}] ⊆ ∼status[{unlocked}]
INITIALISATION
  status := DOOR × {locked}
OPERATIONS
  opendoor(dd) =
    PRE
      dd ∈ DOOR ∧
      status(dd)=unlocked
    THEN
      opening(dd)
    END;
    ...
```

```
  ...
  unlockdoor(dd) =
    PRE
      dd ∈ DOOR
    THEN
      status(dd) := unlocked
    END;

  lockdoor(dd) =
    PRE
      dd ∈ DOOR  ∧
      position(dd)=closed
    THEN
      status(dd) := locked
    END

END
```



```
MACHINE Keys
SETS KEY

VARIABLES keys
INVARIANT
  keys ⊆ KEY
INITIALISATION
  keys := {}

OPERATIONS
  insertkey(kk) =
    PRE kk ∈ KEY
    THEN keys := keys ∪ {kk}
    END;

  removekey(kk) =
    PRE kk ∈ KEY
    THEN keys := keys − {kk}
    END

END
```

```
MACHINE Safes
INCLUDES Locks, Keys
PROMOTES
   opendoor, closedoor, lockdoor
CONSTANTS unlocks
PROPERTIES
  unlocks ∈ KEY ⤖ DOOR
INVARIANT
   ~status [{unlocked}] ⊆ unlocks [keys]
OPERATIONS
   insert(kk,dd) =
     PRE
        kk ∈ KEY ∧ dd ∈ DOOR ∧
        unlocks(kk) = dd
     THEN
        insertkey(kk)
     END;
     ...
```



```
  extract(kk,dd) =
    PRE
      kk ∈ KEY ∧ dd ∈ DOOR ∧
      unlocks(kk) = dd ∧
      status(dd) = locked
    THEN  removekey(kk)
    END;
  unlock(dd) =
    PRE
      dd ∈ DOOR ∧
      ~unlocks(dd) ∈ keys
    THEN  unlockdoor(dd)
    END;
  quicklock(dd) =
    PRE
      dd ∈ DOOR ∧
      position(dd) = closed
    THEN  lockdoor(dd) ||
          removekey(~unlocks(dd))
    END
END
```