Nondeterminism in B specifications

- Nondeterminism occurs when there are several choices between different courses of action, and we have no control about which action will be chosen
- Nondeterminism allows the specifier to describe the acceptable system behaviour by including a number of possible solutions (behaviours)
- Nondeterminism provides flexibility for the implementor because it allows to postpone some decisions until the later stages of program development

B statement language (so far)

Statement	Informal meaning
v := e	assignment
v,w := e1,e2	multiple assignment
skip	no operation
PRE P THEN S END	if P holds, behave like S
IF P THEN S1 ELSE S2 END	if P, execute S1, otherwise S2
CASE E OF EITHER e1 THEN T1 OR e2 ... END	case analysis
S1 || S2	parallel execution of S1 and S2

The ANY statement

- Syntax:
  ANY x WHERE Q THEN T END
- a new local variable x is introduced; the scope of x is T
- x is initialised according to the predicate Q; if Q allows for more than one initial value for x, the specific value (satisfying Q) for x is chosen arbitrarily
- T is executed using x. Thus, different values for x result in different behaviours for T

Weakest precondition for the ANY statement

- Weakest precondition:
  [ANY x WHERE Q THEN T END] $P =$
  $\forall x \bullet \; (Q \Rightarrow [T]\, P)$
- ANY statement is guaranteed to establish the postcondition P if execution of T establishes P no matter what value, satisfying Q, is chosen for x

The LET statement

- Syntax:

  LET x BE $x = E$ IN S END

- a new local variable x is introduced and initialised with E

- the special case of ANY statement:

  LET x BE $x = E$ IN S END =

  ANY x WHERE $x = E$ THEN S END

- Weakest precondition:

  [LET x BE $x = E$ IN S END] P =

  $\forall x \bullet \ (x = E \ \Rightarrow \ [T]\, P)$

Nondeterministic assignment

- Syntax:

  $x \ :\in \ S$ (x :: S in ASCII notation)

- arbitrary value from a set S is assigned to a variable x

- the special case of ANY statement:

  $x \ :\in \ S$ =

  ANY e WHERE $e \in S$ THEN $x := e$ END

- Weakest precondition:

  $[x \ :\in \ S] \ P \quad = \quad \forall z \bullet \ (z \in S \ \Rightarrow \ P[z/x])$

The CHOICE statement

- Syntax:

  CHOICE $S1$ OR $S2$ OR ... OR Sn END

- one statement from $S1, ..., Sn$ is chosen in a completely arbitrary way and then executed

- Weakest precondition:

  [CHOICE $S1$ OR $S2$ OR ... OR Sn END] $P =$

  $[S1]\, P \wedge ... \wedge [Sn]\, P$

- All branches of CHOICE statement should establish the desired postcondition P

The SELECT statement

- Syntax:

  SELECT Q_1 THEN T_1

  WHEN Q_2 THEN T_2 ...

  WHEN Q_n THEN T_n

  [ELSE V] END

- if exactly one guard Q_i is true then the corresponding branch T_i is executed

- if more than one guard Q_i is true then any of the corresponding branches T_i can be executed

The SELECT statement (cont.)

- if none of the guards Q_i is true then the ELSE branch is executed; in case, when the ELSE branch is missing, SELECT statement gets into "waiting mode"
- Weakest precondition:
  [SELECT Q_1 THEN T_1 WHEN ...
  WHEN Qn THEN Tn END] $P =$
  $(Q_1 \Rightarrow [T_1]P) \wedge (Q_2 \Rightarrow [T_2]P) \wedge ... \wedge$
  $(Q_n \Rightarrow [T_n]P)$

Event-based system

- An event-based system is a system which reacts to the events triggered by the environment
- All machine operations of an event-based system B specification are of the form:
  SELECT Q THEN S END
  where a predicate Q describes an event, and
  a statement S – the system reaction
- If several operations are "enabled", one of them is chosen arbitrarily; if an operation is "disabled", it gets into "waiting mode"

```
MACHINE Jukebox
SETS TRACK
VARIABLES credit, playset
INVARIANT
  credit ∈ NAT ∧ playset ⊆ TRACK
INITIALISATION
  credit, playset := 0, {}

OPERATIONS
  pay(cc) =
    PRE cc ∈ NAT1
    THEN credit := credit + cc
    END;

  select(tt) =
    PRE credit > 0 ∧ tt ∈ TRACK
    THEN
      CHOICE
        credit := credit - 1 ||
        playset := playset ∪ {tt}
      OR
        ...
```



```
        ...
        playset := playset ∪ {tt}
      END
    END;

  tt ← play =
    PRE playset ≠ {}
    THEN
      ANY tr
      WHERE tr ∈ playset
      THEN
        tt := tr ||
        playset := playset − {tr}
      END
    END

END
```

```
MACHINE Task_Manager
SETS TASK, STATE
VARIABLES
   arrived_tasks, state,
   chosen_flag, chosen_task
INVARIANT
   arrived_tasks ∈ TASK → BOOL ∧
   state ∈ STATE ∧
   chosen_flag ∈ BOOL ∧
   chosen_task ∈ TASK ∧
   (chosen_flag = TRUE ⇒
    arrived_tasks(chosen_task) = TRUE)
INITIALISATION
   arrived_tasks := TASKS × {FALSE} ||
   state :∈ STATE ||
   chosen_flag := FALSE ||
   chosen_task :∈ TASK

OPERATIONS
   ...
```



```
...
task_arrival =
  SELECT
    FALSE ∈ ran(arrived_tasks)
  THEN
    ANY tt
    WHERE
      tt ∈ TASK ∧
      arrived_tasks(tt) = FALSE
    THEN
      arrived_tasks(tt) := TRUE
    END
  END;

task_selection =
  SELECT
    TRUE ∈ ran(arrived_tasks) ∧
    chosen_flag = FALSE
  THEN
    ANY tt
    WHERE
    ...
```

```
      ...
        tt ∈ TASK ∧
        arrived_tasks(tt) = TRUE
      THEN
        chosen_task := tt ||
        chosen_flag := TRUE
      END
    END;

  task_execution =
    SELECT
      chosen_flag = TRUE
    THEN
      state :∈ STATE ||
      chosen_flag := FALSE ||
      arrived_tasks(chosen_task) := FALSE
    END
END
```