Refinement

- Refinement is the process of moving from abstract specifications to less abstract specifications while preserving the behaviour of the former
- Refinement is based on data or operation transformation which allows the behaviour of the abstract system to be simulated by the more refined (concrete, detailed) system
- Two kinds of refinement:
  - data refinement
  - refinement of nondeterminism

Refinement (cont.)

- Intermediate specifications are presented in REFINEMENT components
- A REFINES clause identifies the component that this refinement is refining
- The abstract component M and the concrete component N are two alternative descriptions of the same system, with the same external interface, but with different internal implementations

Refinement (cont.)

- The states of the abstract machine M and the concrete refinement N must be composed of distinct sets of variables
- These states are linked by a refinement relation which expresses the correspondence between the abstract and concrete states
- This correspondence must be
  - established as a result of corresponding initialisations
  - preserved by each of the operations

Sequential composition

- In addition to the AMN statements introduced so far, in refinements we are allowed to use sequential composition of several statements
- The syntax: $S; T$
- The standard meaning – the statement T is executed after the statement S (if the latter terminates)
- The weakest precondition rule:
  $[S; T]\, P \quad = \quad [S]\, ([T]\, P)$

Data refinement

- In data refinement we represent abstract data by data that is more concrete:
  - sets by sequences
  - sequences by arrays
  - relations by two arrays
  - ...

Refinement of nondeterminism

- In refinement of nondeterminism we resolve nondeterminism presented in a specification by describing how the choice is to be made.
  Examples:
  - $e :\in S \quad \sqsubseteq$ (is refined by) $\quad e := min(S)$
  - CHOICE $S1$ OR S2 END $\quad \sqsubseteq \quad$ S1
  - PRE P THEN S END $\quad \sqsubseteq$
    IF P THEN S ELSE skip END
  - ANY zz WHERE P THEN S END $\quad \sqsubseteq$
    ANY zz WHERE $P \wedge Q$ THEN S END

```
MACHINE Team
SETS ANSWER = {in, out}
VARIABLES team
INVARIANT team ⊆ 1..22 ∧ card(team) = 11
INITIALISATION  team := 1..11

OPERATIONS
  substitute(pp, rr) =
    PRE
      pp ∈ team ∧ rr ∈ 1..22 ∧ rr ∉ team
    THEN
      team := (team ∪ {rr}) − {pp}
    END;

  aa ← query(pp) =
    PRE pp ∈ 1..22
    THEN
      IF  pp ∈ team  THEN  aa := in
      ELSE  aa := out  END
    END;
END
```



```
REFINEMENT TeamR
REFINES Team
VARIABLES teamr
INVARIANT
  teamr ∈ 1..11 >+>> 1..22 ∧
  ran(teamr) = team
INITIALISATION  teamr := id(1..11)

OPERATIONS
  substitute(pp, rr) =
    BEGIN
      teamr (∼teamr(pp)) := rr
    END;

  aa ← query =
    IF  pp ∈ ran(teamr)
    THEN  aa := in
    ELSE  aa := out
    END
END
```

```
REFINEMENT TeamR
REFINES Team
VARIABLES teama
INVARIANT
  teama ∈ 1..22 → ANSWER ∧
  team = ∼teama [{in}]
INITIALISATION
  teama := (1..11)×{in} ∪ (12..22)×{out}
OPERATIONS
  substitute(pp, rr) =
    BEGIN
      teama(pp) := out;
      teama(rr) := in
    END;
  aa ← query =
    BEGIN
      aa := teama(pp)
    END
END
```



```
REFINEMENT JukeboxR
REFINES Jukebox
CONSTANTS freefreq
PROPERTIES freefreq ∈ NAT1
VARIABLES creditr, playlist, free
INVARIANT
  creditr ∈ NAT ∧ creditr = credit ∧
  playlist ∈ iseq(TRACK) ∧
  ran(playlist) = playset ∧
  free ∈ 0..freefreq
INITIALISATION
  creditr, playlist := 0, {}
OPERATIONS
  pay(cc) =
    BEGIN
      creditr := creditr + cc
    END;
  ...
```

```
  select(tt) =
    BEGIN
      IF tt ∉ ran(playlist)
      THEN  playlist := playlist ← tt
      END;
      IF free = freefreq
      THEN
        free := 0
      ELSE
        free := free + 1;
        creditr := creditr − 1
      END
    END;
  tt ← play =
    BEGIN
      tt := first(playlist);
      playlist := tail(playlist)
    END
END
```