Lecture 9

The B Method

Refining B Specifications - Refinement of Nondeterminism,
Proof Obligations for Refinement

Lecture Outline

- References
- Nondeterminism in specifications
- Solving nondeterminism
- Relocating nondeterminism
- Proof obligations for refinement

References

- [1] Abrial, J.-R., *The B Book - Assigning Programs to Meanings*, Cambridge University Press, 1996. (chapter 11)
- [2] Schneider, S., *The B-Method - An Introduction*, Palgrave Macmillan, Cornerstones of Computing series, 2001. (chapters 9,13, 14)
- [3] Clearsy System Engineering, *AtelierB home page* `http://www.atelierb.eu/en/`
- [4] Clearsy System Engineering, *B Method home page* `http://www.methode-b.com/en/`

Nondeterminism in specifications

- Enable underspecification, providing implementation flexibility and deferring some decisions to the appropriate moment
- Nondeterministic substitutions
  - ANY clause (unbounded choice) - allows an arbitrary value to be chosen and executes a statement based on that value
    - Construct: ANY x WHERE Q THEN S END
    - Semantics:
      $[\ \texttt{ANY}\ x\ \texttt{WHERE}\ Q\ \texttt{THEN}\ S\ \texttt{END}\ ]\ P \Leftrightarrow \forall x \cdot (Q \Rightarrow [S]P)$
    - Example:

ANY *x* **WHERE** $x \in luckydip$
THEN $prize := x \parallel luckydip := luckydip - \{x\}$
END

  - Nondeterministic assignment - particular shape of ANY, assigns an arbitrary value from a set to a variable
    - Construct: $x :\in S$
    - Semantics: $x :\in S \Leftrightarrow \texttt{ANY}\ e\ \texttt{WHERE}\ e \in S\ \texttt{THEN}\ x := e\ \texttt{END}$

Nondeterminism in specifications (cont.)

- Nondeterministic substitutions
  - `CHOICE` clause (bounded choice) - arbitrary selection and execution of one of the possible branches
    - Construct: `CHOICE` S `OR` T `OR` ... `OR` U `END`
    - Semantics:
      $[$ `CHOICE` S `OR` T `OR` ... `OR` U `END` $]\ P \Leftrightarrow$
      $[S]P \wedge [T]P \wedge \ldots \wedge [U]P$
    - Example:

CHOICE *result* := *pass* || *licences* := *licences* ∪ {*examinee*}
OR *result* := *fail*
END

Nondeterminism in specifications (cont.)

- Nondeterministic substitutions
  - SELECT clause - allows a choice of statements, in which each statement has a guard that dictates when it is enabled
    - Construct and example:

SELECT Q_1 **THEN** T_1
WHEN Q_2 **THEN** T_2
WHEN ...
WHEN Q_n **THEN** T_n
ELSE V
END

SELECT $x > 1$ **THEN** $x := x - 1$
WHEN $x < 4$ **THEN** $x := x + 1$
WHEN $y > 1$ **THEN** $y := y - 1$
WHEN $y < 4$ **THEN** $y := y + 1$
END

    - Semantics:

$$\left[\begin{array}{l}\textbf{SELECT } Q_1 \textbf{ THEN } T_1\\ \textbf{WHEN } Q_2 \textbf{ THEN } T_2\\ \quad\vdots\\ \textbf{WHEN } Q_n \textbf{ THEN } T_n\\ \textbf{END}\end{array}\right] P = \begin{array}{l} Q_1 \Rightarrow [T_1]P\\ \wedge\ Q_2 \Rightarrow [T_2]P\\ \vdots\\ \wedge\ Q_n \Rightarrow [T_n]P\end{array}$$

Solving nondeterminism

- Within an abstract machine description, nondeterminism can appear in the initialisation and/or operations
- A nondeterministic statement has a number of possible executions
- A means of solving nondeterminism is providing a way of choosing between those possible executions
  - E.g.: a nondeterministic assignment such as $e :\in S$ can be refined by giving a strategy for chosing e, such as taking the minimum from S
- The nondeterminism solving strategy works on the variables of the refinement machine
  - The state of the refinement machine is linked to that of its corresponding abstract machine by means of a linking invariant placed inside the refinement
  - If the refinement and abstract machine have variables in common, then there is an implicit linking invariant stating that their values should be equal

Solving nondeterminism (cont.)

- E.g.: machine used to allocate phone numbers
  - Stores a set `allocated` of already allocated numbers
  - Allows to `choose` a new number (if not already allocated), `query` whether a number if allocated ot not and nondeterministically `allocate` a free number

MACHINE *Allocate*
VARIABLES *allocated*
INVARIANT $allocated \subseteq \mathbb{N}_1$
INITIALISATION $allocated := \emptyset$
OPERATIONS
  choose$(nn) \mathrel{\widehat{=}}$
  PRE $nn \in \mathbb{N}_1 \wedge nn \notin allocated$
  THEN $allocated := allocated \cup \{nn\}$
  END;

  $aa \leftarrow$ **query**$(nn) \mathrel{\widehat{=}}$
  PRE $nn \in \mathbb{N}_1$
  THEN IF $nn \in allocated$
    THEN $aa :=$ **TRUE**
    ELSE $aa :=$ **FALSE**
    END
  END;

  $nn \leftarrow$ **allocate** $\mathrel{\widehat{=}}$
  ANY *no*
  WHERE $no \in \mathbb{N}_1$ - *allocated*
  THEN $allocated := allocated \cup \{no\} \parallel nn := no$
  END

END

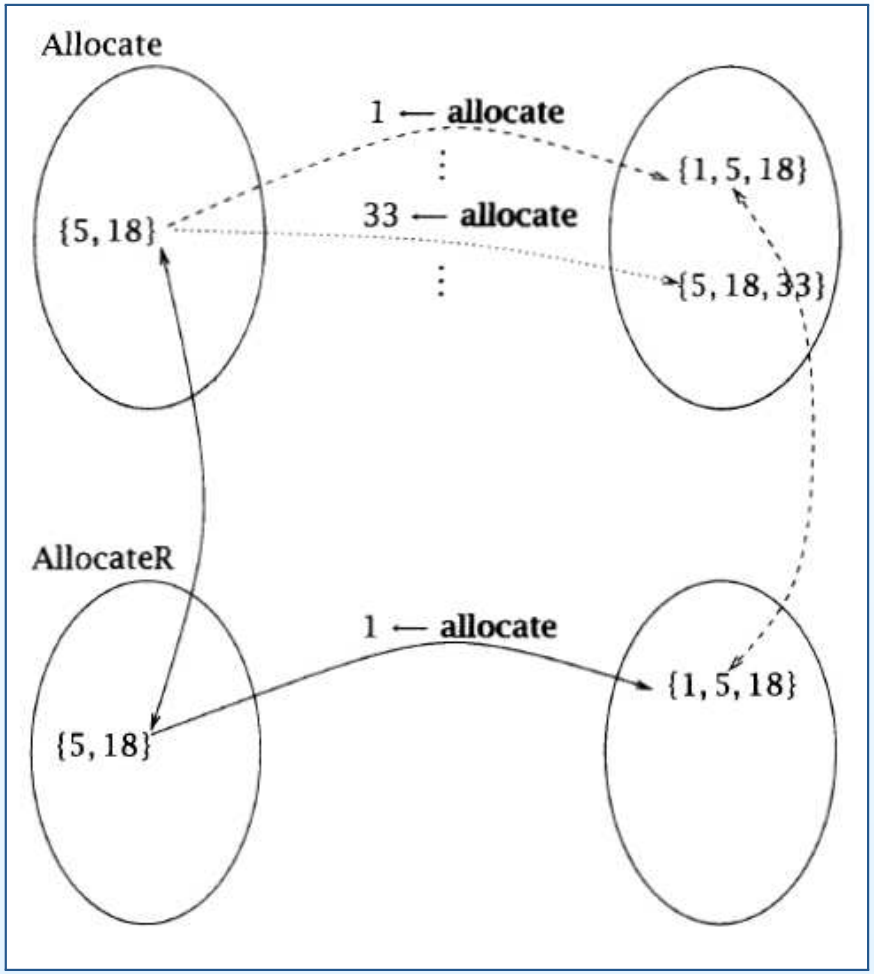
Solving nondeterminism (cont.)

- Refinement of machine `Allocate`
  - Stores its own copy of `allocated` (that should have the same values as the homonymous variable of the abstract machine)
  - `Choose` and `query` are not modified (their preconditions are not repeated, but inherited from the abstract machine)
  - `Allocate` is refined by giving a strategy to choose a new number (the minimum of those non-allocated)

```
REFINEMENT AllocateR
REFINES Allocate
VARIABLES allocated
INITIALISATION allocated := ∅
OPERATIONS
  choose(nn) ≙
  BEGIN
    allocated := allocated ∪ { nn }
  END;

  aa ← query (nn) ≙
  BEGIN
    IF nn ∈ allocated
    THEN aa := TRUE
    ELSE aa := FALSE
    END
  END;

  nn ← allocate ≙
  BEGIN
    nn := min( ℕ1 - allocated);
    allocated := allocated ∪ {nn}
  END

END
```

Solving nondeterminism (cont.)

- (Informally) a refinement is cosidered correct if its initialisation and operations perform only steps that match some steps allowed by the abstract machine
  - Whenever the refinement and abstract machine are in linked states, then any output of the refinement must be allowed by the abstract machine and any state reached by the refinement must match some state that the abstract machine can reach



Relocating nondeterminism

- Refinement is a relationship between machines as a whole, including both states and operations
- Sometimes, resolving the nondeterministic behavior of an operation involves relocating nondeterminism to another part of a machine
  - This relocation may happen as a result of data refinement
  - This is ok, as long as the behavior of the refinement machine is consistent with the abstract behavior
- Therefore, refinement machines may preserve some nondeterminism, resulting in a number of possible executions, if each such execution matches an execution of the refined abstract machine

Relocating nondeterminism (cont.)

- E.g.: machine used to allocate books to be read
  - Books are dispensed from BOOK, provided as set of the machine
  - Read books are stored in the `read` set, initialy empty
  - A single operation is provided, which nondeterministically selects the next book to be read, returns it and adds it to the set of already read books

```
MACHINE Books
SETS BOOK
VARIABLES read
INVARIANT read ⊆ BOOK
INITIALISATION read := ∅
OPERATIONS
   bb ← newbook ≙
   PRE read ≠ BOOK
   THEN
      ANY bk
      WHERE bk ∈ BOOK - read
      THEN read := read ∪ {bk} || bb := bk
      END
   END

END
```

Relocating nondeterminism (cont.)

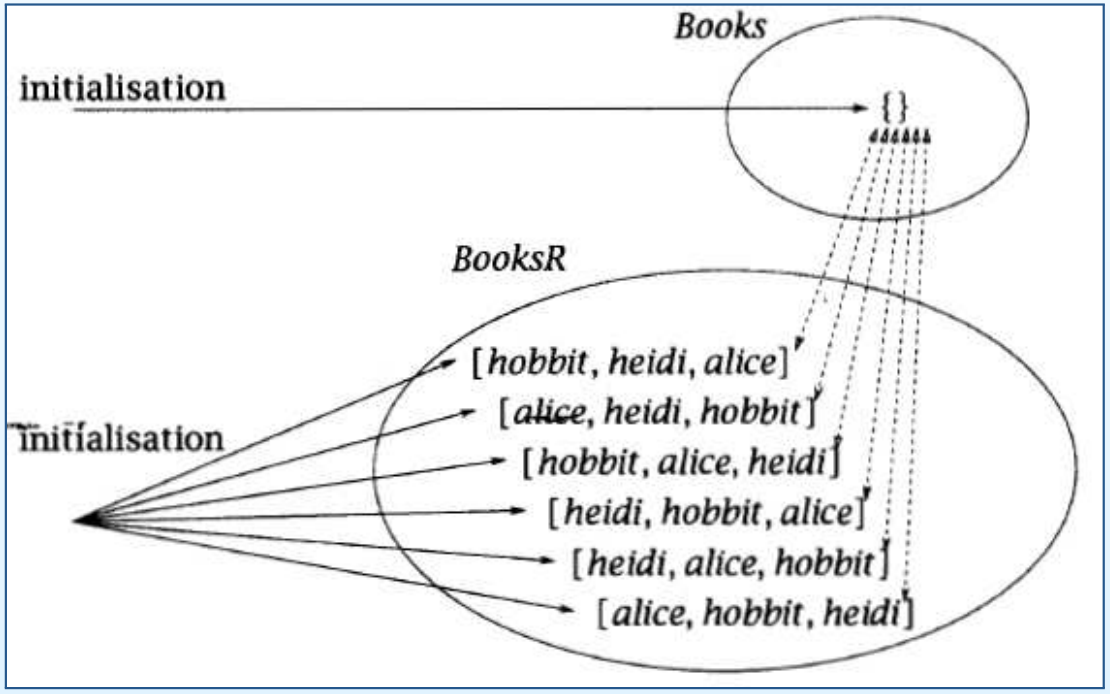
- A possible refinement of Books provides a selection order
  - The sequence variable scheme contains the list of unread books (books should not be repeated)
  - scheme is initialized to some arbitrary permutation of the set BOOKS
  - Nondeterminism is relocated from operation to initialisation due to the data refinement strategy

REFINEMENT *BooksR*
REFINES *Books*
VARIABLES *scheme*
INVARIANT $scheme \in \mathbf{iseq}(BOOK) \wedge \mathbf{ran}(scheme) = BOOK - read$
INITIALISATION $scheme :\in \mathbf{perm}(BOOK)$
OPERATIONS
$bb \leftarrow$ **newbook** $\widehat{=}$
BEGIN
$bb := \mathbf{first}(scheme);$
$scheme := \mathbf{tail}(scheme)$
END

END

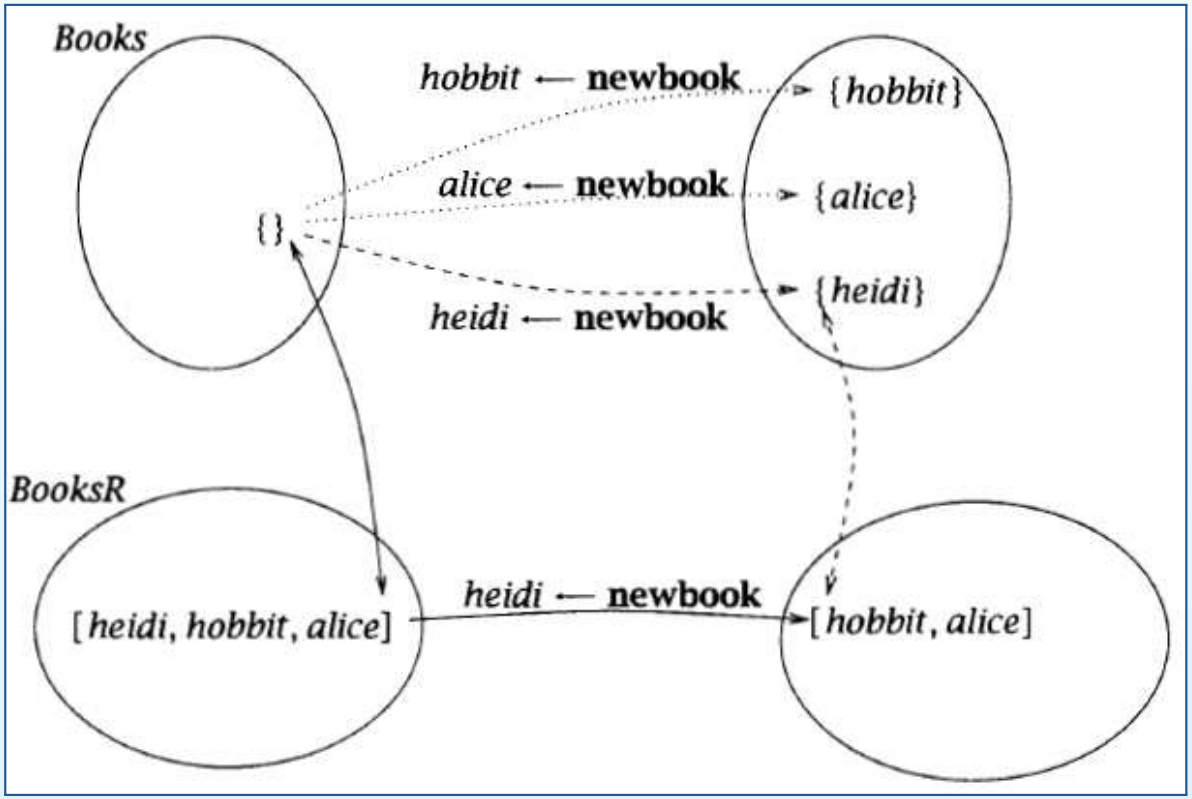
Relocating nondeterminism (cont.)

- The initialisation of the refinement is nondeterministic, while the one in the abstract machine was deterministic
  - It is ok, since each possible execution of the initialization from the refinement guarantees that `scheme` contains all books, thus leads to a state that is linked (by the linking invariant) to the abstract state ensured by the initialisation of the abstract machine (that no book has been read yet).



Relocating nondeterminism (cont.)

- The operation `newbook` is deterministic in the refinement
  - Nondeterminism has been relocated to the initialization
  - For each state of the refinement there is a single output and a single subsequent state reached after executing the refined operation



Relocating nondeterminism (cont.)

- Machine `BooksR` can be further refined
  - Nondeterminism is elliminated from the initialization, by providing the initial array of books as a constant
  - Data is further refined by means of a `counter` tracking the position of the last read book in the array

REFINEMENT *BooksRR*
REFINES *BooksR*
CONSTANTS *bookarr*
PROPERTIES $bookarr \in 1 \,..\, \mathbf{card}(BOOK) \rightarrowtail BOOK$
VARIABLES *counter*
INVARIANT $counter \in 0 \,..\, \mathbf{card}(BOOK) \wedge$
$((1 \,..\, counter) \lhd bookarr) \frown scheme = bookarr$
INITIALISATION $counter := 0$
OPERATIONS
$bb \leftarrow$ **newbook** $\widehat{=}$
BEGIN
$counter := counter + 1;$
$bb := bookarr(counter)$
END

END

Proof obligations for refinement

- Refinement pattern

MACHINE $M_1(X_1, x_1)$ **CONSTRAINTS** C_1 **SETS** S_1; $T_1 = \{a_1, b_1\}$ **(ABSTRACT_)CONSTANTS** c_1 **PROPERTIES** P_1 **(CONCRETE_)VARIABLES** v_1 **INVARIANT** I_1 **ASSERTIONS** J_1 **INITIALIZATION** U_1 **OPERATIONS** $u_1 \leftarrow O(w_1)\hat{=}$ **PRE** Q_1 **THEN** V_1 **END**; ... **END**	**REFINEMENT** $M_n(X_n, x_n)$ **REFINES** M_{n-1} **SETS** S_n; $T_n = \{a_n, b_n\}$ **(ABSTRACT_)CONSTANTS** c_n **PROPERTIES** P_n **INCLUDES** $M(N, n)$ **(CONCRETE_)VARIABLES** v_n **INVARIANT** I_n **ASSERTIONS** J_n **INITIALIZATION** U_n **OPERATIONS** $u_1 \leftarrow O(w_1)\hat{=}$ **PRE** Q_n **THEN** V_n **END**; ... **END**

Proof obligations for refinement (cont.)

- Appropriate parameter initialisation of the included machine

$$\overbrace{A_1 \wedge B_1 \wedge \ldots \wedge B_n \wedge C_1 \wedge P_1 \wedge \ldots \wedge P_n}^{Including} \Rightarrow \overbrace{[X, x := N, n](A \wedge C)}^{Included}$$

- Deducibility of assertions in refinement n

$$\overbrace{A_1 \wedge B_1 \wedge \ldots \wedge B_n \wedge C_1 \wedge P_1 \wedge \ldots \wedge P_n}^{Including} \wedge \overbrace{I_1 \wedge \ldots \wedge I_n \wedge J_1 \wedge \ldots \wedge J_{n-1}}^{Including} \wedge \overbrace{B \wedge P \wedge [X, x := N, n](I \wedge J)}^{Included} \Rightarrow J_n$$

- Initialisation in refinement n is consistent with initialisation in refinement $n-1$

$$\overbrace{A_1 \wedge B_1 \wedge \ldots \wedge B_n \wedge C_1 \wedge P_1 \wedge \ldots \wedge P_n}^{Including} \wedge \overbrace{B \wedge P}^{Included} \Rightarrow \overbrace{[[X, x := N, n]\, U; U_n]\neg[U_{n-1}]\neg I_n}^{Included}$$

Proof obligations for refinement (cont.)

- Implementations of operations in refinement n are consistent with their correspondents in refinement $n-1$

$$\overbrace{A_1 \wedge B_1 \wedge \ldots \wedge B_n \wedge C_1 \wedge P_1 \wedge \ldots \wedge P_n}^{Including} \wedge$$
$$\overbrace{I_1 \wedge \ldots \wedge I_n \wedge J_1 \wedge \ldots \wedge J_n \wedge Q_1 \wedge \ldots \wedge Q_{n-1}}^{Including} \wedge$$
$$\overbrace{B \wedge P \wedge [X, x := N, n](I \wedge J)}^{Included}$$
$$\Rightarrow Q_n \wedge [[u_1 := u_1']V_n]\neg[V_{n-1}]\neg(I_n \wedge u_1 = u_1')$$