Software Engineering
Lecture 04: The B Specification Method

Peter Thiemann

University of Freiburg, Germany

SS 2013
The B specification method

- **B-Method**: formal approach to specification and development of software systems
- Developed by Jean-Raymond Abrial, late 1980es
- Supports all phases of software development
- Emphasis on simplicity
- Amenable to formal verification
- Tool support: Atelier-B, B-Toolkit
- Industrial use
- **Syntax** [http://www.stups.uni-duesseldorf.de/ProB/index.php5/Summary_of_B_Syntax](http://www.stups.uni-duesseldorf.de/ProB/index.php5/Summary_of_B_Syntax)
Central concept: Abstract Machine

Example: The Ticket Dispenser
MACHINE Ticket
VARIABLES serve, next
INVARIANT serve : NAT & next : NAT & serve <= next
INITIALISATION serve, next := 0, 0
OPERATIONS
  ss <-- serve_next =
    PRE serve < next
    THEN ss, serve := serve + 1, serve + 1
    END ;
  tt <-- take_ticket =
    PRE true
    THEN tt, next := next, next + 1
    END
END
MACHINE, VARIABLES, INVARIANT

MACHINE name

- uniquely names a machine in a project

VARIABLES name, ...

- components of local machine state space
- all distinct names

INVARIANT formula

Conjunction of

- type of each variable, e.g., serve : NAT
- relations between variables, e.g., serve <= next
OPERATIONS

List of operation definitions

\[\text{output, ...} \leftarrow \text{name (input, ...)} = \]
\[
\begin{align*}
\text{PRE} & \quad \text{formula} \\
\text{THEN} & \quad \text{statement} \\
\text{END} &
\end{align*}
\]

- Name of operation
- Names of input and output parameters
- PRE precondition
  - Must be true to invoke
  - May be dropped if true
- THEN body: specification of output, effect on state space
  - Must specify each output variable
  - May update the machine state
Abstract Machines

Statement / Assignment

Simple Assignment

\[ \text{name} := \text{expression} \]

Multiple Assignment

\[ \text{name, ...} := \text{expression, ...} \]

- all distinct names on left hand side
- simultaneous assignment — evaluate all right hand sides, then assign to left hand sides all at once
INITIALISATION

INITIALISATION statement

- possible initial states
- all variables of the machine state must be assigned
Sets and Logic
Sets

- B builds on *typed* set theory
- Standard mathematical notation for operations is ok, but we use the syntax of the tools
- Predefined sets:
  - `BOOL = { TRUE, FALSE }`
  - `INT, NAT, NAT1` machine integers and natural numbers (without 0)
  - `STRING` with elements of the form "string content’’
- Types of variables can be defined by predicates
  - `v:S` the value of `v` is an element of set `S`
  - `v<:S` the value of `v` is a subset of set `S`
Set Formation

SETS declaration; ...

► another MACHINE clause
► declaration can be
  ► set-name: set with unspecified elements
  ► set-name = \{ element-name, ...\}: set with named elements
► example

SETS COLOR = \{red, green, blue\}; KEY; PERSON
Set Expressions

Excerpt

If $S$ and $T$ are sets, then so are . . .

\[
\begin{align*}
\{\}, \{E\}, \{E, \ldots\} & \quad \text{empty set, singleton set, set enumeration} \\
\{x \mid P\} & \quad \text{comprehension set} \\
S \cup T, S \cap T, S - T & \quad \text{set union, set intersection, set difference} \\
S \times T & \quad \text{Cartesian product} \\
\text{POW}(S), \text{POW1}(S) & \quad \text{power set, set of non-empty subsets}
\end{align*}
\]

Properties of sets

\[
\begin{align*}
E : S, E / : S & \quad \text{element of, not element of} \\
S \subseteq T, S / \subseteq T & \quad \text{subset of, not subset of} \\
S \subset T, S / \subset T & \quad \text{strict subset of, not strict subset of} \\
\text{card}(S) & \quad \text{cardinality}
\end{align*}
\]
Typed set expressions

\[1 :: \mathbb{N} \quad \text{NAT} :: \mathcal{P}(\mathbb{N})\]

\[\text{SETS } M = \{x_1, \ldots, x_n\}\]

\[x_i :: M \quad M :: \mathcal{P}(M)\]

\[\{\} :: \mathcal{P}(A)\]

\[E_1 :: A \quad \{E_1, \ldots\} :: \mathcal{P}(A)\]

\[P \Rightarrow x :: A \quad \{x | P\} :: \mathcal{P}(A)\]

\[S :: \mathcal{P}(A) \quad T :: \mathcal{P}(A)\]

\[S \cup T :: \mathcal{P}(A) \quad S \cap T :: \mathcal{P}(A) \quad S \setminus T :: \mathcal{P}(A)\]

\[S :: \mathcal{P}(A) \quad T :: \mathcal{P}(B)\]

\[S \ast T :: \mathcal{P}(A \times B)\]

\[S :: \mathcal{P}(A) \quad \text{POW}(S) :: \mathcal{P}(\mathcal{P}(A))\]

\[E :: A \quad S :: \mathcal{P}(A)\]

\[E : S :: \text{PROP}\]

\[S :: \mathcal{P}(A) \quad T :: \mathcal{P}(A)\]

\[S <: T :: \text{PROP}\]

\[\text{card}(S) :: \mathbb{N}\]
First-Order Predicate Logic

- Atoms are expressions of type $\text{PROP}$
- Standard connectives
  - $P \& Q$ : conjunction
  - $P \lor Q$ : disjunction
  - $P \Rightarrow Q$ : implication
  - $P \Leftrightarrow Q$ : equivalence
  - $\neg P$ : negation
  - $(x). (P \Rightarrow Q)$ : universal quantification
  - $(x). (P \& Q)$ : existential quantification

- In quantification, predicate $P$ must fix the type of $x$

- Example
  - $(m). (m: \text{NAT} \Rightarrow \#(n). (n: \text{NAT} \& m < n))$
Weakest Preconditions
State Space

- State space of a B machine = type of its variables restricted by invariant $I$
- Specification of operation = relation on state space
- Questions
  1. Is an operation executable?
  2. Does an operation preserve the invariant?
State Space

- State space of a B machine = type of its variables restricted by invariant $I$
- Specification of operation = relation on state space
- Questions
  1. Is an operation executable?
  2. Does an operation preserve the invariant?
- Formalized for operation `PRE P THEN S END`
  1. Executable: $I \land P$
  2. Preservation: if executable, does $I$ hold after $S$?
State Space

- State space of a B machine = type of its variables restricted by invariant $I$
- Specification of operation = relation on state space
- Questions
  1. Is an operation executable?
  2. Does an operation preserve the invariant?
- Formalized for operation PRE $P$ THEN $S$ END
  1. Executable: $I$ & $P$
  2. Preservation: if executable, does $I$ hold after $S$?
- Tool: Weakest Precondition (WP) $[S] Q$ (a predicate)
  - If $[S] Q$ holds before executing $S$, then $Q$ holds afterwards
  - For all $R$ that hold before $S$ and guarantee that $Q$ holds afterwards, $R \Rightarrow [S] Q$
State Space

- State space of a B machine = type of its variables restricted by invariant $I$
- Specification of operation = relation on state space
- Questions
  1. Is an operation executable?
  2. Does an operation preserve the invariant?
- Formalized for operation \( \text{PRE } P \text{ THEN } S \text{ END} \)
  1. Executable: $I \land P$
  2. Preservation: if executable, does $I$ hold after $S$?
- Tool: \textit{Weakest Preccondition} (WP) $[S]Q$ (a predicate)
  - If $[S]Q$ holds before executing $S$, then $Q$ holds afterwards
  - For all $R$ that hold before $S$ and guarantee that $Q$ holds afterwards, $R \Rightarrow [S]Q$
- WP can be calculated for each statement of the AMN
Example

VARIABLES x, y
INVARIANT x:{0,1,2} & y:{0,1,2}
OPERATIONS
  f =
    y := max { 0, y - x }
END

Weakest precondition

[ y := max { 0, y - x } ] (y > 0)
<=>
  (y = 1) & (x = 0)
  or (y = 2) & (x = 0)
  or (y = 2) & (x = 1)
Weakest Preconditions

Calculation of the Weakest Precondition

WP for Assignment

\[ [x := E]P = P[E/x] \]

Example

\[ [y := \max \{0, y - x\}] \quad (y > 0) \]
\(\equiv\)
\(\max \{0, y - x\} > 0\)
\(\equiv\)
\(y - x > 0\)
\(\equiv\)
\((y = 1) \& (x = 0)\)
\(\text{or } (y = 2) \& (x = 0)\)
\(\text{or } (y = 2) \& (x = 1)\)
Calculation of the Weakest Precondition

WP for skip

\[[\text{skip}]P = P\]

The skip statement has no effect on the state.
Calculation of the Weakest Precondition

WP for conditional

- Syntax: IF $E$ THEN $S$ ELSE $T$ END for statements $S$ and $T$
- Weakest precondition

$$\bigl[\text{IF } E \text{ THEN } S \text{ ELSE } T \text{ END}\bigr]P = (E \& [S]P) \text{ or } (\neg E \& [T]P)$$

Example

$$\bigl[\text{IF } x<5 \text{ THEN } x := x+4 \text{ ELSE } x := x-3 \text{ END}\bigr] (x < 7)$$

$$\iff$$

$$\begin{align*}
(x < 5) & \& [x := x+4] (x < 7) \\
\text{or} & \not (x < 5) & \& [x := x-3] (x < 7) \\
\iff
(x < 5) & \& (x+4 < 7) \\
\text{or} & (x \geq 5) & \& (x-3 < 7) \\
\iff
(x < 3) \\
\text{or} & (x \geq 5) & \& (x < 10)
\end{align*}$$
Machine Consistency
IN Variant and INITIALISATION

Objectives

1. The state space must not be empty
2. Initialization must be successful

INVARIANT \( I \)

State space is non-empty if \( #(v). (I) \)

INITIALISATION \( T \)

Success if \( [T]I \)
INVESTMENT and INITIALISATION

Objectives

1. The state space must not be empty
2. Initialization must be successful

INVESTMENT $I$

State space is non-empty if $\#(v). (I)$

INITIALISATION $T$

Success if $[T] I$

Example Ticket Dispenser

1. For $\text{serve} = 0$ and $\text{next} = 0$, $\text{serve} \leq \text{next}$ holds
2. $[\text{serve}, \text{next} := 0, 0] I = 0 : \text{NAT} \land 0 : \text{NAT} \land 0 \leq 0$
Proof Obligation for Operations

Consider

- **INVARIANT** \( I \)
- operation **PRE** \( P \) **THEN** \( S \) **END**

Consistent if

\[ I \land P \Rightarrow [S]I \]
Machine Consistency

Proof Obligation for Operations

Consider

- IN Variant $I$
- operation PRE $P$ THEN $S$ END

Consistent if

$I \land P \Rightarrow [S]I$

Example Ticket Dispenser serve\_next

\[(serve:\text{NAT} \land next:\text{NAT} \land serve \leq next) \land (serve < next) \Rightarrow [serve := serve + 1] (serve:\text{NAT} \land next:\text{NAT} \land serve \leq next) \]

\[\Leftrightarrow\]

\[(serve:\text{NAT} \land next:\text{NAT} \land serve < next) \Rightarrow (serve:\text{NAT} \land next:\text{NAT} \land serve + 1 \leq next)\]
Relations
Printer Permissions

MACHINE Access
SETS USER; PRINTER; OPTION; PERMISSION = { ok, noaccess } 
CONSTANTS options 
PROPERTIES 
  options : PRINTER <-> OPTION & 
  dom( options ) = PRINTER & ran( options ) = OPTION 
VARIABLES access 
INVARIANT access : USER <-> PRINTER 
INITIALISATION access := {} 
OPERATIONS 
  add (uu, pp) = 
    PRE uu:USER & pp:PRINTER 
    THEN access := access \/ { uu |-> pp } 
    END ; 
  ...
New Machine Clauses

CONSTANTS \textit{name}, \ldots

- \textit{name} is a fixed, but unknown value
- Type determined by PROPERTIES

PROPERTIES \textit{formula}

- Describes conditions that must holds on SETS and CONSTANTS
- \textit{Must} specify the types of the constants
- \textit{Must not} refer to VARIABLES

About clauses

- Clauses must appear in same order as in example!
- No forward references allowed
Relational Operations

- Binary relation between $S$ and $T$
  \[ S <-> T = \text{POW}(S \times T) \]

- Elements of a relation $R : S <-> T$ are pairs, written as $uu \mid-> pp$, where $uu : S$ & $pp : T$

- Predefined symbols for domain and range of a relation
  \[
  \text{dom}(R) = \{ s \mid s : S \& \#(t).(t : T \& s \mid-> t : R) \} \\
  \text{ran}(R) = \{ t \mid t : S \& \#(s).(s : S \& s \mid-> t : R) \}
  \]
Relational Operations

- Binary relation between $S$ and $T$
  
  \[ S \leftrightarrow T = \text{POW}(S \times T) \]

- Elements of a relation $R : S \leftrightarrow T$ are pairs, written as $uu \rightarrow pp$, where $uu : S$ & $pp : T$

- Predefined symbols for domain and range of a relation
  
  \[
  \text{dom} (R) = \{ s \mid s : S \& \#(t). (t : T \& s \rightarrow t : R) \} \\
  \text{ran} (R) = \{ t \mid t : S \& \#(s). (s : S \& s \rightarrow t : R) \}
  \]

- Example:
  
  $\text{PRINTER} = \{ \text{PL, PLDUPLEX, PLCOLOR} \}$
  
  $\text{options} = \{ \text{PL} \rightarrow \text{ok, PLCOLOR} \rightarrow \text{noaccess} \}$
  
  $\text{dom (options)} = \{ \text{PL, PLCOLOR} \}$
  
  $\text{ran (options)} = \{ \text{ok, noaccess} \}$
MACHINE Access ...

OPERATIONS ...

  ban (uu) =
    PRE uu:USER
    THEN access := { uu } |<<| access
    END ;

  nn <-- printnumquery (pp) =
    PRE pp:PRINTER
    THEN nn := card (access |> { pp })
    END ;
Relational Operations II

Domain and range restriction

Let $R:S \leftrightarrow T$

**Domain restriction: Remove elements from $\text{dom} \ (R)$**

- Keep domain elements in $U$:
  $$U <\mid R = \{ s \mid \to t \mid (s \mid \to t):R \ & s:U \}$$

- Drop domain elements in $U$ (anti-restriction, subtraction):
  $$U \ll\mid R = \{ s \mid \to t \mid (s \mid \to t):R \ & s:/\!\!\!/:U \}$$

**Range restriction: Remove elements from $\text{ran} \ (R)$**

- Keep range elements in $U$:
  $$R \mid U = \{ s \mid \to t \mid (s \mid \to t):R \ & t:U \}$$

- Drop range elements in $U$:
  $$R \mid \mid U = \{ s \mid \to t \mid (s \mid \to t):R \ & t:/\!\!\!/:U \}$$
Relational Operations III
Further Relational Operations

id(S)                identity relation
R−                   inverse relation
R[U]                 relational image
(R1;R2)              relational composition
R1<+R2               relational overriding
Relations

Relational Operations III
Further Relational Operations

id(S)     identity relation
R⁻        inverse relation
R[∪]      relational image
(R₁;R₂)   relational composition
R₁<+R₂    relational overriding

Overriding . . .

- R₁<+R₂ means R₂ overrides R₁
- Union of R₁ and R₂, but in the intersection of $\text{dom}(R₁)$ and $\text{dom}(R₂)$, the elements of R₂ take precedence
- $R₁<+R₂ = (\text{dom}(R₂) <<| R₁) \setminus R₂$
Functions
Functions

- In B a function is an unambiguous relation, i.e., a set of pairs

- Shorthand notation to indicate properties of functions
  - $S\rightarrow T$ partial function
  - $S\rightarrow\rightarrow T$ partial surjection
  - $S\rightarrow\rightarrow\rightarrow T$ partial injection
  - $S\rightarrow\rightarrow\rightarrow\rightarrow T$ partial bijection
  - $S\rightarrow T$ total function
  - $S\rightarrow\rightarrow T$ total surjection
  - $S\rightarrow\rightarrow\rightarrow T$ total injection
  - $S\rightarrow\rightarrow\rightarrow\rightarrow T$ total bijection

- Using functions
  - $f \ (E)$ function application
  - $\%x \ (P \mid E)$ lambda abstraction, $P$ gives type of $x$
Example: Reading Books / Declarations

MACHINE Reading
SETS READER; BOOK; COPY; RESPONSE = { yes, no }
CONSTANTS copyof
PROPERTIES copyof : COPY --> BOOK
VARIABLES hasread, reading
INVARIANT
  hasread : READER <-> BOOK &
  reading : READER --> COPY &
  (reading ; copyof) \ hasread = {} 
INITIALISATION
  hasread := {} || reading = {}
Example: Reading Books / Operations (Excerpt)

OPERATIONS (excerpt)

\[
\text{start} (rr, cc) = \\
\text{PRE} \\
rr:\text{READER} & cc:\text{COPY} & \text{copyof} (cc)/:\text{hasread}(rr) & \\
rr/:\text{dom} (\text{reading}) & cc/:\text{ran} (\text{reading}) \\
\text{THEN} \\
\text{reading} := \text{reading} \setminus \{ rr \rightarrow cc \} \\
\text{END} \\
\]

; 

\[
\text{bb} \leftarrow \text{currentquery} (rr) = \\
\text{PRE} \\
rr:\text{READER} & rr:\text{dom} (\text{reading}) \\
\text{THEN} \\
\text{bb} := \text{copyof} (\text{reading} (rr)) \\
\text{END} \\
\]
Sequences and Arrays

Sequences

- A sequence is a *total* function from an initial segment of \( \text{NAT1} \) to another set
- \( \text{seq} \ (S) = (1..N \rightarrow S) \), where \( N : \text{NAT} \)
- Notation for manipulating sequences: formation, concatenation, first, last, etc

Arrays

- An array is a *partial* function from an initial segment of \( \text{NAT1} \) to another set
- \( (1..N \rightarrow S) \), where \( N : \text{NAT} \)
- Notation for updating arrays
  \[
  a \ (i) := E = a := a \ <+ \ \{ \ i \rightarrow E \} 
  \]
Nondeterminism
Nondeterminism in Specifications

- Up to now: high-level programming with sets
  - deterministic machines
  - abstraction from particular data structures
  - abstraction from realization of operations

- Further abstraction
  - specification may allow a range of acceptable behaviors
  - specification describes possible choices
  - subsequent refinement narrows down towards an implementation

- This section
  - AMN operations that exhibit nondeterminism
Example: Jukebox / Declarations

MACHINE Jukebox
SETS TRACK
CONSTANTS limit
PROPERTIES limit: NAT1
VARIABLES credit, playset
INVARIANT credit: NAT & credit <= limit & playset <=: TRACK
INITIALISATION credit, playset := 0, {}

OPERATIONS
   pay (cc) =
      PRE cc: NAT1
      THEN credit := min ( {credit + cc, limit}) END ;
Nondeterminism

Example: Jukebox / Operations (excerpt)

OPERATIONS

\[
\begin{align*}
    \text{tt } & \leftarrow \text{ play } = \\
    & \quad \text{PRE} \quad \text{playset} \neq \{\} \\
    & \quad \text{THEN} \quad \text{ANY} \quad \text{tr} \quad \text{WHERE} \quad \text{tr}:\text{playset} \\
    & \quad \quad \quad \text{THEN} \quad \text{tt} \,:= \, \text{tr} \, \| \, \text{playset} \,:= \, \text{playset} \,-\, \{\text{tr}\} \\
    & \quad \quad \quad \text{END} \\
    & \quad \quad \text{END} \\
    & ; \\
    \text{select} \, (\text{tt}) \, = \\
    & \quad \text{PRE} \quad \text{credit} > 0 \, \& \, \text{tt}:\text{TRACK} \\
    & \quad \text{THEN} \quad \text{playset} \,:= \, \text{playset} \, \setminus \, \{\text{tt}\} \\
    & \quad \quad \quad \| \quad \text{CHOICE} \quad \text{credit} \,:= \, \text{credit} \,-\, 1 \\
    & \quad \quad \quad \| \quad \text{OR} \quad \text{skip} \\
    & \quad \quad \text{END} \\
    & \quad \text{END}
\end{align*}
\]
Nondeterminism

ANY statement

\[
\text{ANY } x \text{ WHERE } Q \text{ THEN } S \text{ END}
\]

- \(x\) fresh variable, only visible in \(Q\) and \(S\)
- \(Q\) predicate; type of \(x\); other constraints
- \(S\) the body statement
- executes \(S\) with an arbitrary value for \(x\) fulfilling \(Q\)

Examples

1. \[
\text{ANY } n \text{ WHERE } n:\text{NAT1} \text{ THEN total := total} \ast n \text{ END}
\]
2. \[
\text{ANY } t \text{ WHERE } t:\text{NAT} \& t \leq \text{total} \& 2 \ast t \geq \text{total} \text{ THEN total := t END}
\]
Nondeterminism

ANY weakest precondition

\[ [\text{ANY } x \; \text{WHERE } Q \; \text{THEN } S \; \text{END}]P = !(x).(Q \Rightarrow [S]P) \]

Examples

1. \[ \text{[ANY } n \; \text{WHERE } n:\text{NAT1} \; \text{THEN } \text{total} := \text{total} \ast n \; \text{END}] \; (\text{total} > 1) \]
   \[ = !(n).(n:\text{NAT1} \Rightarrow [\text{total} := \text{total} \ast n] \; (\text{total} > 1)) \]
   \[ = !(n).(n:\text{NAT1} \Rightarrow (\text{total} \ast n > 1)) \]
   \[ = (\text{total} > 1) \]

2. \[ \text{[ANY } t \; \text{WHERE } t:\text{NAT} \; \& \; t \leq \text{total} \; \& \; 2 \ast t \geq \text{total} \; \ldots] \; (\text{total} > 1) \]
   \[ = !(t).(t:\text{NAT} \; \& \; t \leq \text{total} \; \& \; 2 \ast t \geq \text{total} \Rightarrow [\text{total} := t](\text{total} > 1)) \]
   \[ = !(t).(t:\text{NAT} \; \& \; t \leq \text{total} \; \& \; 2 \ast t \geq \text{total} \Rightarrow (t > 1)) \]
   \[ = (\text{total} > 2) \]
Nondeterminism

CHOICE statement

CHOICE \( S_1 \) OR \( S_2 \) OR ... END

- choice between unrelated statements \( S_1, S_2, ... \)

Example

Outcome of a driving test

CHOICE result := pass || licences := licences \( \backslash / \) \{examinee\}
OR result := fail
END
Nondeterminism

**CHOICE weakest precondition**

\[
[\text{CHOICE } S \text{ OR } T \text{ END}] P = [S] P \& [T] P
\]

**Example**

Check that all licensed persons are old enough.

\[
\begin{cases}
  \text{CHOICE result := pass} \\
  \text{ licences := licences} \setminus \{\text{examinee}\} \\
  \text{ OR result := fail} \\
  \text{ END}
\end{cases}
\]

\[
\text{(licences<:ofAge)} = \begin{cases}
  \text{result := pass} \\
  \text{ licences := licences} \setminus \{\text{examinee}\} \\
  & \text{ (licences<:ofAge)} \\
  \text{ & [result := fail] (licences<:ofAge)}
\end{cases}
\]

\[
\begin{cases}
  \text{ licences := licences} \setminus \{\text{examinee}\} \\
  & \text{ (licences<:ofAge)} \\
  \text{ & (licences<:ofAge)}
\end{cases}
\]

\[
\text{ = (licences<:ofAge) & examinee:ofAge}
\]
Refinement
Refinement

- Refinement formalizes design decisions
- Transforms specification towards implementation
- In B, refinement comes with proof obligations that relate the participating machines

Data refinement

- Formalizes change of data representation
- Usually from abstract to concrete
- Example: set $\rightarrow$ list or array

Refinement of nondeterminism

- Formalizes selection of particular behavior from a nondeterministic specification
- Refined operations are “more deterministic”
Example: Jukebox / Declarations

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:=0 ; playlist:= [] ; free:=0
Example: Jukebox / Operations (excerpt)

select (tt) =

BEGIN
  IF tt/:ran (playlist) THEN playlist := playlist <- tt END ;
  IF free = freefreq
      THEN CHOICE free := 0 OR creditr := creditr-1 END
  ELSE free := free+1 ; creditr := creditr-1
  END
END

END ;

tt <-- play =

PRE playlist /= []
BEGIN tt := first (playlist) ;
    playlist := tail (playlist)
END
Proof Obligation for Refinement

- IN Variant of the Refinement specifies the linking invariant between state spaces of original and refinement.
- Let INVARIANT I in original and INVARIANT IR in refinement.
- For INITIALISATION T in original and INITIALISATION TR in the refinement, it must hold that
  \[ [TR] \land \neg [T] (\neg IR) \]
Proof Obligation for Refinement

- INVARIANT of the REFINEMENT specifies the linking invariant between state spaces of original and refinement.

- Let INVARIANT I in original and INVARIANT IR in refinement.

- For INITIALISATION T in original and INITIALISATION TR in the refinement, it must hold that

  \[ TR \Rightarrow (\text{not } T \Rightarrow \text{not } IR) \]

- For operation PRE P THEN S END in original and PRE PR THEN SR END in refinement, it must hold that

  \[ I \land IR \land P \Rightarrow [SR] \Rightarrow (\text{not } S \Rightarrow \text{not } IR) \]
Summary

- **B** — an industrial strength formal method that supports all phases of software development

- **Approach:**
  - start with high-level spec
  - apply refinement steps until level of implementation reached
  - (code generation tools exist)

- Each refinement step results in proof obligations that must be discharged

- Omitted from lecture
  - structuring: machine parameters, inclusion, extension, state and type export
  - implementation machines, loops, library machines
  - more notation...