

The B-Method

Wai Keong Ip

Dept. of Information Management
National Taiwan University

December 14, 2011

Main References

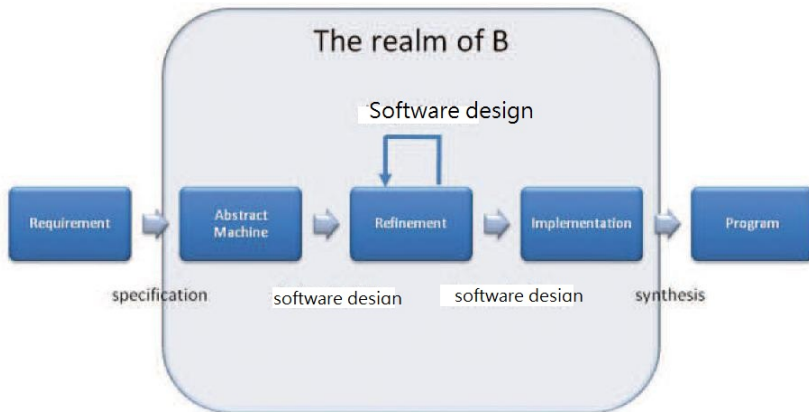
- ① Chi-Shiang Liu, [The B-Method.pdf](#)
- ② J.-R. Abrial, [The B-Book](#)
Cambridge University Press, 1996
- ③ J.B. Wordsworth, [Software Engineering with B](#)
Addison-Wesley, 1996

- 🌐 Introduction
- 🌐 Specification
- 🌐 Refinement
- 🌐 Implementation

What is the B

- B is a method for specifying, designing, and coding software systems
- B uses a simple “pseudo” programming language to model requirements, to specify interfaces, and to provide implementations and intermediate designs
- Compared to Z, B is slightly more low-level and more focused on refinement to code rather than just formal specification
- The language is known as **AMN** (Abstract Machine Notation)
- Event-B has been developed based on B, it is a simpler notation, which is easier to learn and use

- In B method, we use abstract machine represents the software model.
- We care about the essential properties of the internal data, not the implementation details.
- Tells what it does, rather than how it does.
- Abstract Machine Notation allows specifications to be statically type checked, dynamically validated, and mathematically verified by proof to ensure the correctness of the design process.



- First-order logic
- Set theory
- Generalized substitutions

Generalized Substitutions

- The mean to describe state changes
- A generalized substitution acts as a predicate transformer.

The substitution $[V := V + 1]$ substitutes all occurrences of V with the expression $V+1$, e.g.

$$[V := V + 1]V > 0 \iff V + 1 > 0$$

Generalized Substitutions (cont'd)

- The simple substitution $[V := E]$ has the usual meaning
- Generalized substitutions are expressed by means of simple substitutions and logic operations
- The application of a generalized substitution S to a predicate P yields a new predicate denoted $[S]P$
- If predicate P characterizes a set of states, substitution S represents a **state transformation**, then $[S]P$ is the predicate characterizing the states such that, when S is applied, the resulting states are in P
- $[S]P$ can be read as “ S establishes P ”

Generalized Substitutions Forms

simple substitution

$[x := E]R \Leftrightarrow$ replacing all free occurrences of x in R by E .

multiple substitution

$[x_1, \dots, x_n := E_1, \dots, E_n]R \Leftrightarrow$ simultaneously replacing all free occurrences of x_1, \dots, x_n in R by E_1, \dots, E_n respectively.

no operation

$[skip]R \Leftrightarrow R$

substitution with precondition

$[P|S]R \Leftrightarrow P \wedge [S]R$

when the pre-condition P does not hold, the substitution $P|S$ is not guaranteed to achieve anything which means $P|S$ is a **non-terminating substitution**.

Generalized Substitutions Forms (cont'd)

⊕ guarded substitution

$$[P \implies S]R \Leftrightarrow P \Rightarrow [S]R$$

We say P guards S .

It means under the assumption of a predicate P , S can establish R . However, when P does not hold, the substitution $P \implies S$ is able to establish everything.

⊕ bounded choice

$$[S1 \square S2]R \Leftrightarrow [S1]R \wedge [S2]R.$$

" \square " is bounded non-determinism which means abstract machine can do $S1$ or $S2$ and both can establish postcondition

⊕ unbounded choice

$$[@z \cdot S]R \Leftrightarrow \forall z \cdot [S]R, \text{ where } z \text{ is not free in } R.$$

$@z \cdot S$, we say **any** z S . That means S depend on z , z can be any value.

Multiple Substitution

$$[V, W := E, F]P \iff [tmp := F][V := E][W := tmp]P$$

where tmp is a fresh variable

$$[V, W := W, V]V > W$$

$$\iff [tmp := V][V := W][W := tmp]V > W$$

$$\iff [tmp := V][V := W]V > tmp$$

$$\iff [tmp := V]W > tmp$$

$$\iff W > V$$

- 🌐 Introduction
- 🌐 Specification
- 🌐 Refinement
- 🌐 Implementation

Formal Specification

- Specification(functional) describes an abstract program
- If the initial state satisfies the precondition, then it changes only variables and terminates in a final state satisfying the postcondition
- For **validation** of the user requirements
“Is the specified system what the customer wants?”
- To support **verification** of the code
“Is the code a correct implementation of the specification?”

Abstract Machines in B

- 🌐 The basic module for specification in B is the **abstract machine**
- 🌐 The abstract machine is a module that consists of
 - ☀️ a **static** part defining the state
 - 👤 **variables** (the local state of the abstract machine)
 - 👤 **invariant** (the static laws of the system, properties and requirements)
 - ☀️ a **dynamic** part modifying this state
 - 👤 **operations** (modify the state according to the invariant, model services)
- 🌐 Valid states need to be explicitly specified with an **invariant** predicate

Basic Structure of the Abstract Machine

MACHINE

Name(Parameters)

VARIABLES

list of variables

INVARIANT

invariant predicate

specify the types of variables

INITIALISATION

initialization substitution

OPERATIONS

outputs \leftarrow *name(inputs)* = *substitution*

END

- **Parameter**: user decide the value in the set
- **Set internal**: implementer decide the value in the set
- a name in abstract machine notation must be at least two characters, and the first must be alphabetic

More Clause

CONSTRAINT

restrict predicate

restrict predicate for parameter

SEE

list of machine names

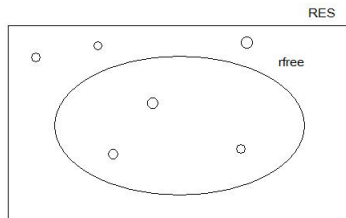
introduces the names of machines for supplying information in the machine definition

ASSERTIONS

predicate

additional information help us understand the mathematical properties of machine

Abstract Machine Example



MACHINE

$RMan (RES)$

VARIABLES

$rfree$

INVARIANT

$rfree \subseteq RES$

INITIALISATION

$rfree := \phi$

Abstract Machine Example (cont'd)

OPERATIONS

alloc (*rr*) =

PRE $rr \in rfree$

THEN

$rfree := rfree - \{rr\}$

END;

free (*rr*) =

PRE

$rr \in RES \wedge rr \notin rfree$

THEN

$rfree := rfree \cup \{rr\}$

END;

END

Verification of the Specification

- ➊ The abstract machine shall initiate in a valid state:
The initialization shall establish the invariant
- ➋ The operations of the abstract machine shall not take it into an invalid state, assuming that their pre-conditions are respected:
The operations shall preserve the invariant
- ➌ Need to generate and discharge proof obligations

The initialization predicate shall establish the invariant:

$$[G]Inv$$

where G is the initialization substitution, and Inv is the invariant predicate
 In the case of the *Resource manager* machine:

$$\begin{aligned}
 [rfree := \phi] \ rfree &\subseteq RES \\
 \iff (\phi &\subseteq RES) \\
 \iff \top
 \end{aligned}$$

Operations shall preserve the invariant, assuming their precondition is satisfied:

$$Inv \wedge P \Rightarrow [S]Inv$$

where

- 🌐 Inv is the invariant predicate
- 🌐 P is the pre-condition of the operation
- 🌐 S is the substitution of the operation

Example of the Specification's Operation

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

($Inv \wedge P$)

$\Rightarrow [S]Inv$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

```
alloc(rr) =  
  PRE rr ∈ rfree  
  THEN  
    rfree := rfree - {rr}  
  END
```

($r_{free} \subseteq RES \wedge P$)
 $\Rightarrow [S]Inv$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

$(r_{free} \subseteq RES \wedge rr \in r_{free})$

$\Rightarrow [S]Inv$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

$(r_{free} \subseteq RES \wedge rr \in r_{free})$

$\Rightarrow [r_{free} := r_{free} - \{rr\}]Inv$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

$(r_{free} \subseteq RES \wedge rr \in r_{free})$

$\Rightarrow [r_{free} := r_{free} - \{rr\}] r_{free} \subseteq RES$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

$(r_{free} \subseteq RES \wedge rr \in r_{free})$

$\Rightarrow [r_{free} := r_{free} - \{rr\}] r_{free} \subseteq RES$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

$(r_{free} \subseteq RES \wedge rr \in r_{free})$

$\Rightarrow r_{free} - \{rr\} \subseteq RES$

Example of the Specification's Operation (cont'd)

In the case of the *Resource manager* machine:

INVARIANT $r_{free} \subseteq RES$

$alloc(rr) =$

PRE $rr \in r_{free}$

THEN

$r_{free} := r_{free} - \{rr\}$

END

$(r_{free} \subseteq RES \wedge rr \in r_{free})$

$\Rightarrow r_{free} - \{rr\} \subseteq RES$

$\Leftrightarrow \top$

- 🌐 Introduction
- 🌐 Specification
- 🌐 Refinement
- 🌐 Implementation

Generalities on Refinement

Refinement

- ➊ Is a step on the path from **specification to implementation**
- ➋ Reflects a design decision
- ➌ Is better in the sense that it is more accurate, applies in more situations, or runs more efficiently
- ➍ Defines the concrete version of the specification variables in the initialization and operations
- ➎ A so-called **refinement relation** establishes the mapping between the values of concrete variables and abstract variables

the Different Kinds of Refinement

The different kinds of refinement correspond to the different kinds of software design:

- 🌐 **Data design**
introducing data structures that can be easily programmed
- 🌐 **Algorithm design** provides indications of how operations are to be computed by algorithms, and may also:
 - ☀ allow more input values
 - ☀ restrict or even remove non-determinism

Refinements in B

- ➊ A refinement is a module introduced by the keyword: **REFINEMENT**
- ➋ The refined machine (or refinement) is referenced explicitly with a refinement clause (keyword: **REFINES**)
- ➌ A refinement may refine an abstract machine or a previous refinement
- ➍ The **invariant** establishes the type of the variables of the refinement and the **retrieve relation**
- ➎ Turning an Abstract Machine into a more concrete one
 - ☀ preserving **signature**: name, parameters and results
 - ☀ a different state or a different specification of the operation

Example of an Abstract Machine

MACHINE

ExampleM

VARIABLES

y

INVARIANT

$y \in F(N_1)$

INITIALISATION

$y := \phi$

Example of an Abstract Machine (cont'd)

OPERATIONS

enter(n) =

PRE $n \in N_1$

THEN

$y := y \cup \{n\}$

END

$m \leftarrow$ *getmax* =

PRE

$y \neq \phi$

THEN

$m := \max(y)$

END;

END

Example of Refinement

REFINEMENT

ExampleR

REFINES

ExampleM

VARIABLES

z

INVARIANT

$z \in N \wedge z = \max(y \cup \{0\})$

INITIALISATION

$z := 0$

Example of Refinement (cont'd)

OPERATIONS

enter(*n*) =

PRE $n \in N_1$

THEN

$z := \max(\{z, n\})$

END

$m \leftarrow \text{getmax} =$

PRE

$z \neq 0$

THEN

$m := z$

END;

END

Verification of the Refinement

- The refinement has to be shown compliant with respect to the specification
 - the **initialization** of the refinement $INIT_R$ shall be compatible with the initialization of the specification $INIT_M$
 - the refined **operations** OP_R shall also be compatible with the specified operations OP_M
- **compatible** means that:
 - the concrete operations shall be possible whenever the corresponding specification is possible
 - the values established by the concrete initialization and operations shall be mapped, by the **refinement relation** INV_R , to a subset of those established in the specification

Relation between Specification and Refinement

$y := \emptyset$

$z := 0$

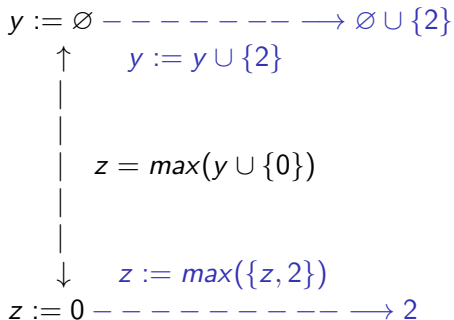
Relation between Specification and Refinement

 $y := \emptyset$ 

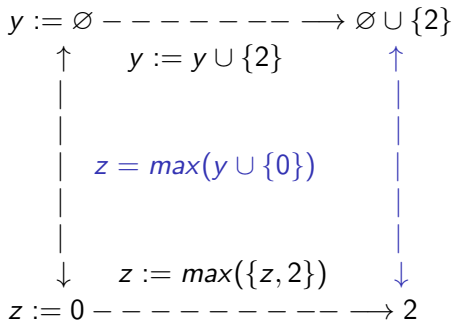
$$z = \max(y \cup \{0\})$$

 $z := 0$

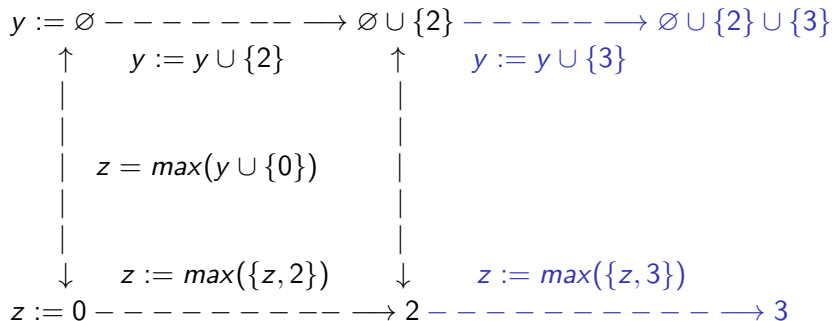
Relation between Specification and Refinement



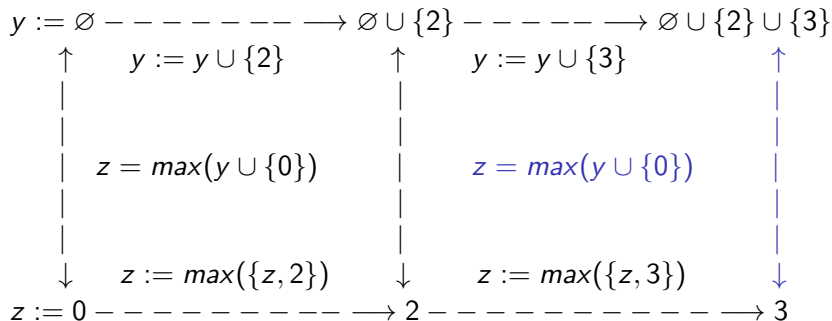
Relation between Specification and Refinement



Relation between Specification and Refinement



Relation between Specification and Refinement



Substitution Refinement Pattern Formula

- 🌐 S being a substitution, R a predicate:
 - ☀️ $[S]R$ means that all executions of S establish R
 - ☀️ $\neg[S]\neg R$ means that there **exists** an execution of S establishing R
- 🌐 Let INV_R be the refinement relation, S_M be a substitution on the abstract state, and S_R be a substitution on the concrete state, the formula:

$$[S_R]\neg[S_M]\neg INV_R$$

means that all executions of the concrete substitution S_R establish that there exists an execution of the abstract substitution S_M establishing INV_R

- The invariant of the refinement INV_R defines the refinement relation
- It is assumed that the specification machine is consistent:
the abstract invariant INV_M is established by all abstract initial states: $[INIT_M]INV_M$
- $[INIT_M]\neg INV_R$ characterizes the set of concrete states that are not related with an abstract initial state by INV_R
- The concrete initialization ($INIT_R$) shall guarantee that no concrete initial state satisfies $[INIT_M]\neg INV_R$, i.e. that the following formula is valid:

$$[INIT_R]\neg[INIT_M]\neg INV_R$$

Example of the Refinement's Initialization

$$[INIT_R] \neg [INIT_M] \neg INV_R$$

$$\iff [z := 0] \neg [y := \phi] \neg (z \in N \wedge z = \max(y \cup \{0\}))$$

$$\iff [z := 0] \neg \neg (z \in N \wedge z = \max(\phi \cup \{0\}))$$

$$\iff [z := 0] z \in N \wedge z = \max(\phi \cup \{0\})$$

$$\iff 0 \in N \wedge 0 = \max(\{0\}) \quad (\text{using } \max(\{n\}) = n)$$

$$\iff 0 \in N \wedge 0 = 0$$

$$\iff \top$$

Proof Obligations for the Refinement's Output-less Operations

- The verification of the correctness when refining an operation without output is:

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow PRE_R \wedge [OP_R] \neg [OP_M] \neg INV_R$$

- PRE_M and OP_M are respectively the precondition and the substitution of the specified operation
- PRE_R and OP_R are respectively the precondition and the substitution of the refined operation

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow PRE_R \wedge [OP_R] \neg [OP_M] \neg INV_R$$

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow PRE_R \wedge [OP_R] \neg [OP_M] \neg INV_R$$

$$\begin{aligned} \iff & y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ & n \in N_1 \wedge [z := \max(\{z, n\})] \neg [y := y \cup \{n\}] \\ & \neg(z \in N \wedge z = \max(y \cup \{0\})) \end{aligned}$$

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow PRE_R \wedge [OP_R] \neg [OP_M] \neg INV_R$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \neg [y := y \cup \{n\}] \\ \neg (z \in N \wedge z = \max(y \cup \{0\})) \end{aligned}$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \\ \neg \neg (z \in N \wedge z = \max(y \cup \{n\} \cup \{0\})) \end{aligned}$$

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow PRE_R \wedge [OP_R] \neg [OP_M] \neg INV_R$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \neg [y := y \cup \{n\}] \\ \neg (z \in N \wedge z = \max(y \cup \{0\})) \end{aligned}$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \\ \neg \neg (z \in N \wedge z = \max(y \cup \{n\} \cup \{0\})) \end{aligned}$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \\ (z \in N \wedge z = \max(y \cup \{n\} \cup \{0\})) \end{aligned}$$

Example of the Refinement's Output-less Operations (cont'd)

$$\begin{aligned} \iff & y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ & n \in N_1 \wedge [z := \max(\{z, n\})] \\ & (z \in N \wedge z = \max(y \cup \{n\} \cup \{0\})) \end{aligned}$$

Example of the Refinement's Output-less Operations (cont'd)

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \\ (z \in N \wedge z = \max(y \cup \{n\} \cup \{0\})) \end{aligned}$$

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge \max(\{z, n\}) \in N \wedge \\ \max(\{z, n\}) = \max(y \cup \{n\} \cup \{0\}) \end{aligned}$$

Example of the Refinement's Output-less Operations (cont'd)

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge [z := \max(\{z, n\})] \\ (z \in N \wedge z = \max(y \cup \{n\} \cup \{0\})) \end{aligned}$$

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge \max(\{z, n\}) \in N \wedge \\ \max(\{z, n\}) = \max(y \cup \{n\} \cup \{0\}) \end{aligned}$$

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ n \in N_1 \wedge \max(\{z, n\}) \in N \wedge \\ \max(\{z, n\}) = \max(y \cup \{n\} \cup \{0\}) \end{aligned}$$

Example of the Refinement's Output-less Operations (cont'd)

$$\Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \\ \max(\{z, n\}) = \max(y \cup \{n\} \cup \{0\})$$

$$\text{(using } \max(\max(S1 \cup S2) \cup S3) = \max(S1 \cup S2 \cup S3)\text{)}$$

Example of the Refinement's Output-less Operations (cont'd)

$$\iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge n \in N_1 \Rightarrow \max(\{z, n\}) = \max(y \cup \{n\} \cup \{0\})$$

$$\text{(using } \max(\max(S1 \cup S2) \cup S3) = \max(S1 \cup S2 \cup S3)\text{)}$$

$$\iff \top$$

Proof Obligations for the Refinement's Output Operations

- The verification of the correctness when refining an operation with output is:

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow PRE_R \wedge [[o := o']OP_R] \neg [OP_M] \neg (INV_R \wedge o = o')$$

- o is the identifier of the output and o' is a fresh identifier.

Example of the Refinement's Output Operations

$$\begin{aligned} INV_M \wedge INV_R \wedge PRE_M &\Rightarrow \\ PRE_R \wedge [[o := o']OP_R] \neg [OP_M] \neg (INV_R \wedge o = o') \end{aligned}$$

Example of the Refinement's Output Operations

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow \\ PRE_R \wedge [[o := o']OP_R] \neg [OP_M] \neg (INV_R \wedge o = o')$$

$$\Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [[m := m']m := z] \neg ([m := \max(y)]) \\ \neg (z \in N \wedge z = \max(y \cup \{0\}) \wedge m = m')$$

Example of the Refinement's Output Operations

$$INV_M \wedge INV_R \wedge PRE_M \Rightarrow \\ PRE_R \wedge [[o := o']OP_R] \neg [OP_M] \neg (INV_R \wedge o = o')$$

$$\Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [[m := m']m := z] \neg ([m := \max(y)]) \\ \neg (z \in N \wedge z = \max(y \cup \{0\}) \wedge m = m')$$

$$\Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [[m := m']m := z] \\ \neg \neg (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m')$$

Example of the Refinement's Output Operations (cont)

$$\begin{aligned} \iff & y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ & z \neq 0 \wedge [[m := m']m := z] \\ & (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m') \end{aligned}$$

Example of the Refinement's Output Operations (cont.)

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [[m := m']m := z] \\ (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m') \end{aligned}$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [m' := z] \\ (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m') \end{aligned}$$

Example of the Refinement's Output Operations (cont)

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [[m := m']m := z] \\ (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m') \end{aligned}$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [m' := z] \\ (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m') \end{aligned}$$

$$\begin{aligned} \Leftrightarrow y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge [m' := z] \\ (z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = m') \end{aligned}$$

Example of the Refinement's Output Operations (cont)

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = z \end{aligned}$$

Example of the Refinement's Output Operations (cont)

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = z \end{aligned}$$

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge \max(y) = z \end{aligned}$$

Example of the Refinement's Output Operations (cont)

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge \max(y) = z \end{aligned}$$

$$\begin{aligned} \iff y \in F(N_1) \wedge z \in N \wedge z = \max(y \cup \{0\}) \wedge y \neq \phi \Rightarrow \\ z \neq 0 \wedge \max(y) = z \end{aligned}$$

$$\iff \top$$

- 🌐 Introduction
- 🌐 Specification
- 🌐 Refinement
- 🌐 Implementation

Generalities on Implementations in B

- Implementation is the input to programming language source code synthesis
- Implementation is a special case of refinement (same proof obligations)
- Implementation cannot be further refined
- Replace **REFINEMENT** to **IMPLEMENTATION**
- Additional constraints:
 - Implementation cannot contain non-deterministic constructs such as **ANY-WHERE-THEN-END**
 - Implementation do not contain variables, but use **IMPORTS** clause imports them from pre-existing B modules: library machines

Example of non-determinism

SELECT substitution

[*SELECT P THEN G WHEN Q THEN H END*]R

$\Leftrightarrow (P \Rightarrow [G]R) \wedge (Q \Rightarrow [H]R)$

Example of non-determinism

SELECT substitution

[*SELECT P THEN G WHEN Q THEN H END*]R

$\Leftrightarrow (P \Rightarrow [G]R) \wedge (Q \Rightarrow [H]R)$

SELECT

$x \geq 0$ THEN $x := x + 1$

WHEN

$x > 0$ THEN skip

WHEN

$x > 1$ THEN $x := 1$

Example of non-determinism

SELECT substitution

$[\text{SELECT } P \text{ THEN } G \text{ WHEN } Q \text{ THEN } H \text{ END }]R$

$\Leftrightarrow (P \Rightarrow [G]R) \wedge (Q \Rightarrow [H]R)$

SELECT

$x \geq 0 \text{ THEN } x := x + 1$

WHEN

$x > 0 \text{ THEN skip}$

WHEN

$x > 1 \text{ THEN } x := 1$

i.e. $x=1$ (non-determined)

Example of non-determinism

SELECT substitution

[*SELECT P THEN G WHEN Q THEN H END*] *R*

$\Leftrightarrow (P \Rightarrow [G]R) \wedge (Q \Rightarrow [H]R)$

SELECT

x ≥ 0 *THEN* *x* := *x* + 1

WHEN

x > 0 *THEN* *skip*

WHEN

x > 1 *THEN* *x* := 1

i.e. *x*=1 (non-determined)

x=2 (non-determined)

Example of non-determinism

SELECT substitution

[*SELECT P THEN G WHEN Q THEN H END*] *R*

$\Leftrightarrow (P \Rightarrow [G]R) \wedge (Q \Rightarrow [H]R)$

SELECT

$x \geq 0$ THEN $x := x + 1$

WHEN

$x > 0$ THEN skip

WHEN

$x > 1$ THEN $x := 1$

i.e. $x=1$ (non-determined)

$x=2$ (non-determined)

$x=0$ (determined)

Example of non-determinism

SELECT substitution

[SELECT *P* THEN *G* WHEN *Q* THEN *H* END]*R*

$\Leftrightarrow (P \Rightarrow [G]R) \wedge (Q \Rightarrow [H]R)$

SELECT

$x \geq 0$ THEN $x := x + 1$

WHEN

$x > 0$ THEN skip

WHEN

$x > 1$ THEN $x := 1$

i.e. $x=1$ (non-determined)

$x=2$ (non-determined)

$x=0$ (determined)

- They do not need to be mutually exclusive, but they should cover every case.(this example doesn't cover every case)

- 🌐 The following substitutions provide support for programming in the B notation:
 - ☀ Sequencing: $S_1; S_2$
 - ☀ Choice: IF P THEN S_1 ELSE S_2 END
 - ☀ Loop: WHILE C DO S INVARIANT I VARIANT V END

- Syntax: $S_1; S_2$ denotes substitution S_1 , followed by substitution S_2
- Semantics:
$$[S_1; S_2]P \iff [S_1][S_2]P$$
- Sequencing is left associative:
$$S_1; S_2; S_3 = (S_1; S_2); S_3$$

📍 Syntax: **IF T THEN S_1 ELSE S_2**

📍 Semantics:

$$[\mathbf{IF\ T\ THEN\ } S_1 \ \mathbf{ELSE\ } S_2]P \iff (T \implies [S_1]P) \wedge (\neg T \implies [S_2]P)$$

Construction of Loops

- Programming languages usually have several loop constructs
- The B notation provides the *WHILE* construct:
 - the control of the iteration is a test condition
 - the body of the loop is executed while the test is true
 - there is usually some initialization before the iteration

```
WHILE T : formula DO B : substitution  
VARIANT V : expression INVARIANT I : formula  
END
```

- The loop **variant** (**rank function**) states the maximum number of times that the body will be executed
- The loop **invariant** is a formula that shall be valid each time the control condition is evaluated

Loop Example

$y := x; ctr := 0;$

WHILE $ctr < 5$ **DO**

$x := x + 1; ctr := ctr + 1$

VARIANT $6 - ctr$

INVARIANT $ctr \in 0..5 \wedge x = y + ctr$

END

Loop Verification

Loop verification:

$[INIT; \text{WHILE } T \text{ DO } B \text{ VARIANT } V \text{ INVARIANT } I \text{ END}]R$

Rules for the correctness of a loop:

- I-rule ($[INIT]I$)
- F-rule ($I \wedge \neg T \Rightarrow R$)
- T1-rule ($I \Rightarrow V \in N$)
- T2-rule ($I \wedge T \Rightarrow [V_i := V][B]V < V_i$, V_i is the initial variant)
- P-rule ($I \wedge T \Rightarrow [B]I$)

Verifying a Loop

```
[y := x; ctr := 0;  
WHILE ctr < 5 DO  
x := x + 1; ctr := ctr + 1  
VARIANT 6 - ctr  
INVARIANT ctr ∈ 0..5 ∧ x = y + ctr  
END]x = y + 5
```

Verifying the I-rule

$$[INIT]I \iff [x := y; ctr := 0]ctr \in 0..5 \wedge x = y + ctr$$

Verifying the I-rule

$$\begin{aligned} [INIT]I &\iff [x := y; ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \end{aligned}$$

Verifying the I-rule

$$\begin{aligned} [INIT]I &\iff [x := y; ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \end{aligned}$$

Verifying the I-rule

$$\begin{aligned} [INIT]I &\iff [x := y; ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y]0 \in 0..5 \wedge x = y + 0 \end{aligned}$$

Verifying the I-rule

$$\begin{aligned} [INIT]I &\iff [x := y; ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y]0 \in 0..5 \wedge x = y + 0 \\ &\iff 0 \in 0..5 \wedge y = y + 0 \end{aligned}$$

Verifying the I-rule

$$\begin{aligned} [INIT]I &\iff [x := y; ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y][ctr := 0]ctr \in 0..5 \wedge x = y + ctr \\ &\iff [x := y]0 \in 0..5 \wedge x = y + 0 \\ &\iff 0 \in 0..5 \wedge y = y + 0 \\ &\iff \top \end{aligned}$$

Verifying the F-rule

$$I \wedge \neg T \Rightarrow R \iff (\neg(ctr < 5) \wedge ctr \in 0..5 \wedge x = y + ctr) \\ \Rightarrow x = y + 5$$

Verifying the F-rule

$$\begin{aligned} I \wedge \neg T \Rightarrow R &\iff (\neg(\mathit{ctr} < 5) \wedge \mathit{ctr} \in 0..5 \wedge x = y + \mathit{ctr}) \\ &\Rightarrow x = y + 5 \\ &\iff (\mathit{ctr} \geq 5 \wedge \mathit{ctr} \in 0..5 \wedge x = y + \mathit{ctr}) \\ &\Rightarrow x = y + 5 \end{aligned}$$

Verifying the F-rule

$$\begin{aligned} I \wedge \neg T \Rightarrow R &\iff (\neg(ctr < 5) \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \end{aligned}$$

Verifying the F-rule

$$\begin{aligned} I \wedge \neg T \Rightarrow R &\iff (\neg(ctr < 5) \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \end{aligned}$$

Verifying the F-rule

$$\begin{aligned} I \wedge \neg T \Rightarrow R &\iff (\neg(ctr < 5) \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \end{aligned}$$

Verifying the F-rule

$$\begin{aligned} I \wedge \neg T \Rightarrow R &\iff (\neg(ctr < 5) \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \\ &\iff (x = y + 5) \Rightarrow x = y + 5 \end{aligned}$$

Verifying the F-rule

$$\begin{aligned} I \wedge \neg T \Rightarrow R &\iff (\neg(ctr < 5) \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr \geq 5 \wedge ctr \in 0..5 \wedge x = y + ctr) \\ &\Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \\ &\iff (ctr = 5 \wedge x = y + ctr) \Rightarrow x = y + 5 \\ &\iff (x = y + 5) \Rightarrow x = y + 5 \\ &\iff \top \end{aligned}$$

Verifying the T1-rule

$$I \Rightarrow (V \in N) \iff ctr \in 0..5 \wedge x = y + ctr \Rightarrow 6 - ctr \in N$$

Verifying the T1-rule

$$\begin{aligned} I \Rightarrow (V \in N) &\iff ctr \in 0..5 \wedge x = y + ctr \Rightarrow 6 - ctr \in N \\ &\iff \top \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned} I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\ &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\ &\quad [V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned} I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\ &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\ &\quad [V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \\ &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\ &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned} I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\ \iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) &\Rightarrow \\ &[V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \\ \iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) &\Rightarrow \\ &[V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\ \iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) &\Rightarrow \\ &[V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned}
 I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1]6 - (ctr + 1) < V_i
 \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned}
 I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1]6 - (ctr + 1) < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr]6 - (ctr + 1) < V_i
 \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned}
 I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1]6 - (ctr + 1) < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr]6 - (ctr + 1) < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad 5 - ctr < 6 - ctr
 \end{aligned}$$

Verifying the T2-rule

$$\begin{aligned}
 I \wedge T &\Rightarrow [V_i := V][B]V < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1; ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1][ctr := ctr + 1]6 - ctr < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr][x := x + 1]6 - (ctr + 1) < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad [V_i := 6 - ctr]6 - (ctr + 1) < V_i \\
 &\iff (ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5) \Rightarrow \\
 &\quad 5 - ctr < 6 - ctr \\
 &\iff T
 \end{aligned}$$

Verifying the P-rule

$$I \wedge T \Rightarrow [B]I \iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\ [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr$$

Verifying the P-rule

$$\begin{aligned} I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\ &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\ &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\ &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \end{aligned}$$

Verifying the P-rule

$$\begin{aligned}
 I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr
 \end{aligned}$$

Verifying the P-rule

$$\begin{aligned}
 I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1]ctr + 1 \in 0..5 \wedge x = y + ctr + 1
 \end{aligned}$$

Verifying the P-rule

$$\begin{aligned}
 I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1]ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1
 \end{aligned}$$

Verifying the P-rule

$$\begin{aligned}
 I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1]ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1
 \end{aligned}$$

Verifying the P-rule

$$\begin{aligned}
 I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1]ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff x = y + ctr \Rightarrow x + 1 = y + ctr + 1
 \end{aligned}$$

Verifying the P-rule

$$\begin{aligned}
 I \wedge T \Rightarrow [B]I &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1; ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1][ctr := ctr + 1]ctr \in 0..5 \wedge x = y + ctr \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad [x := x + 1]ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff ctr \in 0..5 \wedge x = y + ctr \wedge ctr < 5 \Rightarrow \\
 &\quad ctr + 1 \in 0..5 \wedge x + 1 = y + ctr + 1 \\
 &\iff x = y + ctr \Rightarrow x + 1 = y + ctr + 1 \\
 &\iff T
 \end{aligned}$$

Thanks for your attention!