The Development of a Toolkit to Support the Probabilistic B Method

Thai Son Hoang

7th June 2002
Outline
Outline

• A brief introduction to B and GSL.
Outline

• A brief introduction to **B** and **GSL**.
• Introduction to **pGSL**.
Outline

- A brief introduction to \( \mathbb{B} \) and \( \text{GSL} \).
- Introduction to \( \text{pGSL} \).
- New \( \text{pB} \) construct.
Outline

• A brief introduction to B and GSL.
• Introduction to pGSL.
• New pB construct.
• Example of probabilistic Number.
Outline

• A brief introduction to B and GSL.
• Introduction to pGSL.
• New pB construct.
• Example of probabilistic Number.
• Conversion of B-Toolkit.
Outline

- A brief introduction to \texttt{B} and \texttt{GSL}.
- Introduction to \texttt{pGSL}.
- New \texttt{pB} construct.
- Example of probabilistic Number.
- Conversion of \texttt{B-Toolkit}.
- Conclusion.
Introduction to B and GSL
Introduction to B and GSL

- **B-Method** is a systematic development of large software systems from reusable fragments.
Introduction to B and GSL

- **B-Method** is a systematic development of large software systems from reusable fragments.

- **B-Toolkit** is built to illustrate all the aspects of B-Method.
Introduction to B and GSL (Cont.)

- **B-Method** based on **Generalized Substitution Language (GSL)** by Abrial.
Introduction to B and GSL (Cont.)

- **B-Method** based on **G**eneralized **S**ubstitution **L**anguage (**GSL**) by Abrial.

  - \([x := E]R \equiv \text{The predicate obtained after replacing all free occurrence of } x \text{ in } R \text{ by } E.\]
  - \([P \mid G]R \equiv P \& [G]R\)
  - \([P \implies G] \equiv P \Rightarrow [G]R\)
  - \([\text{skip}]R \equiv R\)
  - \(G \parallel H \equiv \text{apply the substitutions } G \text{ and } H \text{ concurrently.}\)
  - \([G; H]R \equiv [G]([H]R)\)
  - \([G,H]R \equiv [G]R \& [H]R\)
Introduction to B and GSL (Cont.)

- The developer write programs (machines) using Abstract Machine Notation (AMN) into the B-Toolkit.
Introduction to B and GSL (Cont.)

- The developer write programs (machines) using Abstract Machine Notation (AMN) into the B-Toolkit.

Simple machine written in AMN:

VARIABLES xx, yy
Introduction to B and GSL (Cont.)

- The developer writes programs (machines) using Abstract Machine Notation (AMN) into the B-Toolkit.

Simple machine written in AMN:

```
VARIABLES xx, yy
INARIANT xx : NAT & yy : NAT
```
Introduction to B and GSL (Cont.)

- The developer write programs (machines) using Abstract Machine Notation (AMN) into the B-Toolkit.

Simple machine written in AMN:

VARIABLES xx, yy
INVARIANT xx : NAT & yy : NAT
INITIALISATION xx, yy := 0, 0
Introduction to B and GSL (Cont.)

- The developer write programs (machines) using Abstract Machine Notation (AMN) into the B-Toolkit.

Simple machine written in AMN:

```
VARIABLES xx, yy
INVARIANT xx : NAT & yy : NAT
INITIALISATION xx, yy := 0, 0
OPERATIONS
```
Introduction to B and GSL (Cont.)

- The developer write programs (machines) using Abstract Machine Notation (AMN) into the B-Toolkit.

Simple machine written in AMN:

```
VARIABLES xx, yy
INVARIANT xx : NAT & yy : NAT
INITIALISATION xx, yy := 0, 0
OPERATIONS
    Increase =
    BEGIN
        xx := xx + 1 ||
        yy := yy + 1
    test
    END
END
```
Introduction to pGSL
Introduction to pGSL

- pGSL is the extension of GSL.
Introduction to pGSL

- **pGSL** is the extension of **GSL**.
- The differences between **GSL** and **pGSL**: is that predicates (functions from state to Boolean) have been widened to functions from state to number.
Introduction to pGSL

- **pGSL** is the extension of **GSL**.
- The differences between **GSL** and **pGSL**: is that predicates (functions from state to Boolean) have been widened to functions from state to number.
- For consistency with Boolean logic, $\text{false} \mapsto 0$, $\text{true} \mapsto 1$. In other words, it acts over 'expectations' rather than predicates.
Introduction to pGSL

- **pGSL** is the extension of **GSL**.
- The differences between **GSL** and **pGSL**: is that predicates (functions from state to Boolean) have been widened to functions from state to number.
- For consistency with Boolean logic, false $\mapsto 0$, true $\mapsto 1$. In other words, it acts over 'expectations' rather than predicates.
- Notationally, we have kept the predicate syntax as much as possible.
Introduction to pGSL

- **pGSL** is the extension of **GSL**.
- The differences between **GSL** and **pGSL**: is that predicates (functions from state to Boolean) have been widened to functions from state to number.
- For consistency with Boolean logic, false $\mapsto 0$, true $\mapsto 1$. In other words, it acts over 'expectations' rather than predicates.
- Notationally, we have kept the predicate syntax as much as possible.
- Example of an expression in **pGSL**:
  
  $$(yy + 1 \in \mathbb{N} \land \text{expectation}((yy + 1) - 2 \times xx)) \times \text{frac}(1, 2)$$
New pB construct
New pB construct

- **pGSL**: Probabilistic choice substitution $S_{p \oplus T}$. 
New pB construct

- **pGSL**: Probabilistic choice substitution $S_{p \oplus T}$.
- **pAMN**:

```
PCHOICE p OF
  S
OR
  T
END
```
Example of probabilistic Number
Example of probabilistic Number

A machine that has two counting devices on it. The machine has one operation namely Increase. When invoking this operation
Example of probabilistic Number

A machine that has two counting devices on it. The machine has one operation namely Increase. When invoking this operation

- the first device increases its value probabilistically. Half of the time, it increases the value by 1. The other half of the time, it keeps the value the same.
Example of probabilistic Number

A machine that has two counting devices on it. The machine has one operation namely Increase. When invoking this operation

- the first device increases its value probabilistically. Half of the time, it increases the value by 1. The other half of the time, it keeps the value the same.
- the second device increases its value deterministically by 1.
Example of probabilistic Number

A machine that has two counting devices on it. The machine has one operation namely Increase. When invoking this operation

- the first device increases its value probabilistically. Half of the time, it increases the value by 1. The other half of the time, it keeps the value the same.

- the second device increases its value deterministically by 1.

And we expect that the value on the second device is always twice the value on the first device.
Example of probabilistic Number (Cont.)

Using \texttt{pAMN}, below is the \texttt{Increase} operation:

\begin{verbatim}
Increase =
BEGIN
  PCHOICE 1/2 OF
  xx := xx + 1
  OR
  skip
  END ||
  yy := yy + 1
END
\end{verbatim}
Specification of probabilistic Number is shown below (in pAMN notation).

```plaintext
MACHINE pNumber
SEES Real_TYPE, Bool_TYPE
VARIABLES xx, yy
INVARIANT
   xx : NAT & yy : NAT &
   expectation(yy - 2 * xx)
INITIALISATION
   xx, yy := 0, 0
```
OPERATIONS

Increase =

BEGIN

PCHOICE 1/2 OF

xx := xx + 1

OR

skip

END

∥

yy := yy + 1

END

END
Proof obligations generator
Proof obligations generator

The rules are:

- The initialisation needs to establish the invariant on the assumption of the context of the machine.
Proof obligations generator

The rules are:

• The initialisation needs to establish the invariant on the assumption of the context of the machine.

• The operations need to maintain the invariant.
Proof obligation for initialisation
Proof obligation for initialisation

Proving by using pB’s rules:

\[ xx, yy := 0, 0 \] xx ∈ \mathbb{N} ∧ yy ∈ \mathbb{N} ∧ \text{expectation}(yy − 2 × xx) \\
≡ 0 ∈ \mathbb{N} ∧ 0 ∈ \mathbb{N} ∧ \text{expectation}(0) \]
Proof obligation for initialisation

Proving by using pB’s rules:

\[
[xx, yy := 0, 0] xx \in \mathbb{N} \land yy \in \mathbb{N} \land \text{expectation}(yy - 2 \times xx)
\]
\[
\equiv 0 \in \mathbb{N} \land 0 \in \mathbb{N} \land \text{expectation}(0)
\]

We need to have precondition in the initialisation.

**PRE** \text{expectation}(0)

**THEN**

\[
xx, yy := 0, 0
\]

**END**
Proof obligation for Increase operation
Proof obligation for Increase operation

- Rule for probabilistic choice substitution.

\[ [S_p \oplus T]R \equiv p \times [S]R + (1 - p) \times [T]R \]
Proof obligation for Increase operation

- Rule for probabilistic choice substitution.
  
  \[ [S_{p \oplus T}]R \equiv p \times [S]R + (1 - p) \times [T]R \]

- Arithmetic with Real number.
Modifying the B-Toolkit
Modifying the B-Toolkit

- Internal structure of the B-Toolkit
Modifying the B-Toolkit

- Internal structure of the B-Toolkit
  1. Motif interface.
Modifying the B-Toolkit

- Internal structure of the B-Toolkit
  1. Motif interface.
  2. Theories driven processes.
Modifying the B-Toolkit

• Internal structure of the B-Toolkit
  1. Motif interface.
  2. Theories driven processes.

• What to do
  — Analyzer.
Modifying the B-Toolkit

- Internal structure of the **B-Toolkit**
  1. Motif interface.
  2. Theories driven processes.

- What to do
  - Analyzer.
  - PO generator.
Modifying the B-Toolkit

- Internal structure of the **B-Toolkit**
  1. Motif interface.
  2. Theories driven processes.

- What to do
  - Analyzer.
  - PO generator.
  - Prover.
Case studies
Case studies

- Random algorithms.
Case studies

- Random algorithms.
- Uncertainties in Networking
Conclusion
Conclusion

• The introduction of **pGSL** helps the programmers handle probabilistic properties of software formally.
Conclusion

• The introduction of pGSL helps the programmers handle probabilistic properties of software formally.

• The new Toolkit will assist in developing and maintaining software with probabilistic properties.
Conclusion

- The introduction of **pGSL** helps the programmers handle probabilistic properties of software formally.

- The new **Toolkit** will assist in developing and maintaining software with probabilistic properties.

- Further more, in the future, the **Toolkit** can be upgraded to support other properties of software development.