Probabilistic invariants for probabilistic machines

ZB2003, Turku, June 2003

Revision: 1.0, May 27, 2003

Thai Son Hoang, Zhendong Jin,
Ken Robinson, Annabelle McIver and Carroll Morgan

School of Computer Science and Engineering

The University of New South Wales, Sydney, Australia

1. Introduction

The work described in this paper is concerned with using probabilistic Generalised Substitution Language (pGSL). In order to achieve that goal we need to develop the concept of a probabilistic machine. That involves the extension of Abstract Machine Notation (AMN) to probabilistic AMN (pAMN) to express the notion of

- probabilistic choice substitution;
- probabilistic invariant, here called an expectation

Additionally, we are adapting the B-Toolkit to assist with the development of probabilistic B (pB) machines. This involves:

- new syntax;
- proof obligation generation for new constructs;
- reasoning over reals as well as Boolean.

2. pGSL

The following table shows the pGSL constructs, emphasizing those that are currently implemented in the modified B-Toolkit.

Substitution	Meaning

2. pGSL

The following table shows the pGSL constructs, emphasizing those that are currently implemented in the modified B-Toolkit.

Substitution	**Meaning**
$[x := E]exp$	The expectation obtained after replacing all free occurrences of x in *exp* by E, renaming bound variables in *exp* if necessary to avoid capture of free variables in E.

2. pGSL

The following table shows the pGSL constructs, emphasizing those that are currently implemented in the modified B-Toolkit.

Substitution	**Meaning**
$[x := E]$*exp*	The expectation obtained after replacing all free occurrences of x in *exp* by E, renaming bound variables in *exp* if necessary to avoid capture of free variables in E.
[skip]*exp*	*exp*

2. pGSL

The following table shows the pGSL constructs, emphasizing those that are currently implemented in the modified B-Toolkit.

Substitution	**Meaning**
$[x := E]$*exp*	The expectation obtained after replacing all free occurrences of x in *exp* by E, renaming bound variables in *exp* if necessary to avoid capture of free variables in E.
[skip]*exp*	*exp*
$[prog_1 \; {}_p\!\oplus \; prog_2]$*exp*	$p \times [prog_1]exp \; + \; (1-p) \times [prog_2]exp$

2. pGSL

The following table shows the pGSL constructs, emphasizing those that are currently implemented in the modified B-Toolkit.

Substitution	**Meaning**
$[x := E]exp$	The expectation obtained after replacing all free occurrences of x in exp by E, renaming bound variables in exp if necessary to avoid capture of free variables in E.
$[\mathsf{skip}]exp$	exp
$[prog_1\ {}_p\!\oplus\ prog_2]exp$	$p \times [prog_1]exp\ +\ (1-p) \times [prog_2]exp$
$[prog_1\ \parallel\ prog_2]exp$	$[prog_1]exp$ min $[prog_2]exp$
$[@y \cdot pred \Longrightarrow prog]exp$	min $(y) \cdot (pred \mid [prog]exp)$, where y does not occur free in exp.
$[pre \mid prog]exp$	$\langle pre \rangle \times [prog]exp$, where $0 \times \infty := 0$.
$[pre \rightarrow prog]exp$	$1/\langle pre \rangle \times [prog]exp$, where $\infty \times 0 := \infty$.
$prog_1 \sqsubseteq prog_2$	$[prog_1]exp \Rrightarrow [prog_2]exp$ for all exp

2.1. How pGSL extends GSL

In pGSL, predicates (functions from state to Boolean) are widened to functions from state to real number. We are replacing certainty by probabilty, which are called expectations.

- For consistency with Boolean logic, $\mathit{false} \mapsto 0$, $\mathit{true} \mapsto 1$.
- Notationally, we have kept the predicate as much as possible.

2.2. pGSL idioms

In standard the B Method (B) we are used to the idea of invariance:

$$pred \quad \equiv \quad [prog]pred. \qquad (1)$$

2.2. pGSL idioms

In standard the B Method (B) we are used to the idea of invariance:

$$pred \quad \equiv \quad [prog]pred. \tag{1}$$

In pB we will be using the corresponding idiom:

$$exp \quad \equiv \quad [prog]exp\ , \tag{2}$$

which expresses the relation between the expectation before and after the program *prog*.

3. An example

We will take the well-known "library" example, and use that as a basis for developing a probabilistic version. Our aim is: first, to show how probabilistic invariants capture some probabilistic properties and; second, to highlight some of the unexpected and subtle issues that can arise.

3.1. Standard library

```
MACHINE StandardLibrary ( totalBooks )
VARIABLES
    booksInLibrary , loansStarted , loansEnded
INVARIANT
    booksInLibrary ∈ ℕ ∧ loansStarted ∈ ℕ ∧ loansEnded ∈ ℕ ∧
    loansEnded ≤ loansStarted ∧
    booksInLibrary + loansStarted − loansEnded = totalBooks
INITIALISATION
    booksInLibrary , loansStarted , loansEnded := totalBooks , 0 , 0
```

```
OPERATIONS

    StartLoan $\widehat{=}$
      PRE  booksInLibrary $> 0$  THEN
         booksInLibrary := booksInLibrary $- 1$  $\|$
         loansStarted := loansStarted $+ 1$
      END ;

    EndLoan $\widehat{=}$
      PRE  loansEnded $<$ loansStarted  THEN
         booksInLibrary := booksInLibrary $+ 1$  $\|$
         loansEnded := loansEnded $+ 1$
      END

END
```

3.2. Probabilistic library

If we wanted to model a library in which books may become "lost", we might add a Lose operation of the form

$$Lose \quad \widehat{=} \quad booksInLibrary := booksInLibrary - 1 \ , \qquad (3)$$

and to arrange that every so often Lose is invoked, with some probability. The problem with this is that we have no way in B (or in pB for that matter) of modelling a probabilistically invoked operation; we have no control over the invocation of operations, other than through preconditions.

An alternative, in pB, is to model operations with probabilistic effects and we take that approach in the ProbabilisticLibrary machine.

```
MACHINE  ProbabilisticLibrary ( totalBooks )

SEES  Real_TYPE

CONSTANTS  pp

PROPERTIES  pp ∈ REAL ∧ pp ≤ real ( 1 ) ∧ real ( 0 ) ≤ pp

VARIABLES

    booksInLibrary , loansStarted , loansEnded , booksLost
```

INVARIANT

$\text{booksInLibrary} \in \mathbb{N} \land \text{loansStarted} \in \mathbb{N} \land$
$\text{loansEnded} \in \mathbb{N} \land \text{booksLost} \in \mathbb{N} \land$
$\text{loansEnded} \leq \text{loansStarted} \land$
$\text{booksInLibrary} + \text{booksLost} + \text{loansStarted} - \text{loansEnded} =$
$\quad \text{totalBooks}$

EXPECTATIONS

$\text{real} (0) \Rrightarrow \text{pp} \times \text{real} (\text{loansEnded}) - \text{real} (\text{booksLost})$

INITIALISATION

$\text{booksInLibrary} , \text{loansStarted} := \text{totalBooks} , 0 \;\; \|$
$\text{loansEnded} , \text{booksLost} := 0 , 0$

```
OPERATIONS

    StartLoan ≙
      PRE  booksInLibrary > 0  THEN
         booksInLibrary := booksInLibrary − 1  ‖
         loansStarted := loansStarted + 1
      END ;

    EndLoan ≙
      PRE  loansEnded < loansStarted  THEN
         PCHOICE  pp  OF
            booksLost := booksLost + 1
         OR
            booksInLibrary := booksInLibrary + 1
         END  ‖
         loansEnded := loansEnded + 1
      END
END
```

4. The expectations clause

Each predicate in the EXPECTATIONS clause defines a real-value function from the state and the lower bound of that function. Each has the form:

$$e \Rrightarrow V \ , \tag{4}$$

where

- V is an expression over program variables,
- e is the lower bound that must be established by the initialisation.

4. The expectations clause

Each predicate in the EXPECTATIONS clause defines a real-value function from the state and the lower bound of that function. Each has the form:

$$e \Rrightarrow V \ , \tag{4}$$

where

- V is an expression over program variables,
- e is the lower bound that must be established by the initialisation.

If a standard invariant, I, was written as an expectation, we would write:

$$\text{true} \Rightarrow I \ , \tag{5}$$

but that is simply I, so nothing would appear to be achieved. We will see that there is significant difference for the probabilistic invariant.

4. The expectations clause

Each predicate in the EXPECTATIONS clause defines a real-value function from the state and the lower bound of that function. Each has the form:

$$e \Rrightarrow V \;, \tag{4}$$

where

- V is an expression over program variables,
- e is the lower bound that must be established by the initialisation.

If a standard invariant, I, was written as an expectation, we would write:

$$\text{true} \Rightarrow I \;, \tag{5}$$

but that is simply I, so nothing would appear to be achieved. We will see that there is significant difference for the probabilistic invariant.

Importantly, although $e \Rrightarrow V$ is invariant, it is **not** used as a standard predicate invariant.

5. What do expectations guarantee?

We wish to interpret the conditions on initialisation and operations in the context of expections: true $\Rightarrow I$ for standard programs; and $e \Rrightarrow V$ for probabilistic programs.

5. What do expectations guarantee?

We wish to interpret the conditions on initialisation and operations in the context of expections: true $\Rightarrow I$ for standard programs; and $e \Rrightarrow V$ for probabilistic programs.

Standard program:

$$\begin{aligned} \text{true} &\Rightarrow [\text{Init}]I \\ I &\Rightarrow [\text{OpX}]I \\ I &\Rightarrow [\text{OpY}]I, \end{aligned} \tag{6}$$

then we are assured that

$$\text{true} \Rightarrow [\text{Init; Op?; Op?; } \ldots \text{; Op?}]I \tag{7}$$

5. What do expectations guarantee?

We wish to interpret the conditions on initialisation and operations in the context of expections: true $\Rightarrow I$ for standard programs; and $e \Rrightarrow V$ for probabilistic programs.

Standard program:

$$\begin{aligned} \text{true} &\Rightarrow [\text{Init}]I \\ I &\Rightarrow [\text{OpX}]I \\ I &\Rightarrow [\text{OpY}]I, \end{aligned} \tag{6}$$

then we are assured that

$$\text{true} \Rightarrow [\text{Init; Op?; Op?; \ldots; Op?}]I \tag{7}$$

Probabilistic program:

$$\begin{aligned} e &\Rrightarrow [\text{Init}]V \\ V &\Rrightarrow [\text{OpX}]V \\ V &\Rrightarrow [\text{OpY}]V, \end{aligned} \tag{8}$$

then we are assured that

$$e \Rrightarrow [\text{Init; Op?; Op?; \ldots; Op?}]V \tag{9}$$

6. Proof obligations for probabilistic operations

Standard program:

N1: The initialisation needs to establish the invariant given the context of the machine (information about sets and constants)

$$[Init]I \ .$$

N2: The operations need to maintain the invariant

$$I \Rightarrow [Op]I \ .$$

6. Proof obligations for probabilistic operations

Standard program:

N1: The initialisation needs to establish the invariant given the context of the machine (information about sets and constants)

$$[Init]I \ .$$

N2: The operations need to maintain the invariant

$$I \Rightarrow [Op]I \ .$$

Probabilistic program:

P1: The initialisation needs to establish the lower bound of the probabilistic invariant, given the context of the machine (information about sets and constants)

$$e \Rrightarrow [Init]V \ .$$

P2: The operations do not decrease the expected value of the probabilistic invariant, i.e. the expected value of the invariant after the operation is at least the expected value before the operation

$$V \Rrightarrow [Op]V \ .$$

7. What the invariant means

The order of execution of operations of a machine is inherently demonically nondeterministic: we do not, and cannot specify the order in which operations are executed. So, machines involve nondeterminism, even when operations do not contain nondeterminism.

For standard machines, the trace of the values of the expectations of the standard invariant is true, true, . . . , true, and is not remarkable.

For probabilistic machines, the trace of the values of the expectations of the probabilistic invariant is $e_0, e_1, \ldots, e_n$, where $e_0 \Rrightarrow e_1 \Rrightarrow \ldots \Rrightarrow e_n$. That is,

> the trace of expectations must form a monotonically increasing chain, no matter how the nondeterminism is resolved.

Note: the trace above is a sequence of theoretical values, not measurements.

For those interested in an experimental view here is another story.

Over a large number of tests of the machine, carried out by an adversary, who can choose to resolve demonic choice within operations any way they wish, and who can choose to invoke operations in any order, we will observe that the average value of V is at least the stated value.

$*$

8. Standard and probabilistic invariant: the difference

There are some consequences of our use of expectations that are surprising if the difference between Boolean and probabilistic invariants is not fully appreciated.

Consider the introduction into our probabilistic library of an operation to model stocktaking:

```
totalCost $\longleftarrow$ StockTake $\widehat{=}$
    BEGIN
        totalCost := cost $\times$ booksLost $\|$
        booksInLibrary := booksInLibrary + booksLost $\|$
        loansStarted := loansStarted $-$ loansEnded $\|$
        loansEnded := 0 $\|$
        booksLost := 0
    END
```

in which we simply “rerun” the initialisation, and output the cost of replacing the books lost.

It is clear the the standard invariant is maintained, but for the probabilistic invariant we require $V \Rrightarrow [\mathit{StockTake}]\,V$. Consider the right-hand side of that inequality (considering the effect of variables $\mathit{loansEnded}$ and $\mathit{booksLost}$ only):

$$\begin{array}{ll} & [\mathit{StockTake}]\,V \\ \equiv & [\mathit{loansEnded}, \mathit{booksLost} := 0, 0]\,V \\ \equiv & [\mathit{loansEnded}, \mathit{booksLost} := 0, 0] \\ & (pp * \mathit{loansEnded} - \mathit{booksLost}) \\ \equiv & 0\ . \end{array}$$

This requires us to prove

$$pp * \mathit{loansEnded} - \mathit{booksLost} \quad \Rrightarrow \quad 0\ , \tag{10}$$

which we cannot prove in this context.

What went wrong?

The problem is a consequence of us naively carrying forward from standard machines the idea that initialisation is always applicable. With standard invariants the lower bound is true, which is also the upper bound.

It is not normally the case with probabilistic invariants that the lower bound is the upper bound. If it were then there would be no difference between standard and probabilistic machines.

Consider the following scenario. A malevolent library administrator wishes to show that library loan system is “broken”: that the rate of book loss is higher than the advertised claim of *pp*. If the administrator adopts a policy of running StockTake whenever $booksLost$ is large relative to $pp * loansEnded$, then the library managers will indeed see that system is “broken”.

A “fix” of the StockTake operation is given in the paper.

9. Modifying the B-Toolkit

The changes required to adapt the B-Toolkit consisted of

Introduction of Real numbers: We use a read-only (seen) machine to introduce a REAL type. Currently this type is the set of non-negative rational numbers, with numbers being denoted by a constructor $frac(m, n)$.

Acceptance of pAMN: The parser had to be modified to accept the new pAMN constructs of: EXPECTATIONS clause, probabilistic choice construct (PCHOICE)

Analyser: The type and construct analysis had to be modified or extended. The analyser produces a canonic, (abstract) syntactic parse and separate canonic type information for each machine.

Proof obligation generator: The B-Toolkit needed to generate proof obligations for the new PCHOICE clause and for the probabilistic invariant. For the normal invariant, the PCHOICE substitution is treated as a non-deterministic CHOICE substitution; for the new expectation invariant, the proof obligations must follow the rules shown in section 6. Notice that while normal Boolean expressions could be converted to numeric expressions, we leave Boolean expressions unchanged. This has the effect of ensuring that the proof of all Boolean goals or sub-goals will proceed using the standard proof rules.

Provers: No change was required for the provers, but we needed to add new rules to support real number evaluations that arise as a consequence of expectations.

Mark-up: Small changes were required to mark-up the new EXPECTATIONS and PCHOICE constructions and the $\Rrightarrow$ expectation order.

The B-Toolkit is implemented on top of a theorem prover (the B-Tool prover), so every toolkit process is driven by a set of proof rules. A consequence of the separation of canonic (abstract) parse and type information by the analyser for each machine is that, after the analysis phase all other phases can be based purely on syntax. This considerably simplified the conversion of the B-Toolkit to handle numeric, rather than Boolean, logic, since proof obligations and proof rules are typeless. Some existing proof rules had to be modified and new rules added to support the the new syntax and proof theory of pAMN and pGSL. Currently, the probabilistic analysis (of expectations) of a machine is stored separately from the unaltered standard (non-probabilistic) analysis, but they could be merged.

10. Summary

We have extended standard AMN (to pAMN) to enable the concept of a probabilistic machine, which supports the following pB constructs:

1. probabilistic invariants or expectations;
2. probabilistic choice;

and we have modified the B-Toolkit to support:

1. parsing and analysis of the new constructs;
2. generation of proof obligations for the new constructs;
3. use of the rational number subset of reals.