Probabilistic invariants for probabilistic machines

Thai Son Hoang[1], Zhendong Jin[1], Ken Robinson[1], Annabelle McIver[2], and Carroll Morgan[1]

[1] School of Computer Science & Engineering, University of New South Wales, NSW 2052 Australia;
{htson, zjin, kenr, carrollm}@cse.unsw.edu.au
[2] Department of Computing, Macquarie University, NSW 2109 Australia;
anabel@ics.mq.edu.au

Abstract. Abrial's *Generalised Substitution Language* (*GSL*) [4] can be modified to operate on arithmetic expressions, rather than Boolean predicates, which allows it to be applied to probabilistic programs [13]. We add a new operator $_p\oplus$ to *GSL*, for probabilistic choice, and we get the *probabilistic Generalised Substitution Language* (*pGSL*): a smooth extension of *GSL* that includes random algorithms within its scope.
In this paper we begin to examine the effect of *pGSL* on *B*'s larger-scale structures: its *machines*. In particular, we suggest a notion of *probabilistic* machine invariant. We show how these invariants interact with *pGSL*, at a fine-grained level; and at the other extreme we investigate how they affect our general understanding "in the large" of probabilistic machines and their behaviour.
Overall, we aim to initiate the development of *probabilistic B* (*pB*), complete with a suitable *probabilistic AMN* (*pAMN*). We discuss the practical extension of the *B-Toolkit* [5] to support *pB*, and we give examples to show how *pAMN* can be used to express and reason about probabilistic properties of a system.
Keywords: Probability, program correctness, generalised substitutions, weakest preconditions, *the B Method* (*B*), probabilistic algorithms.

1 Introduction

Abrial's *Generalised Substitution Language* (*GSL*) [4] is a weakest-precondition based method of describing computations and their meaning; it is complemented by the structures of *Abstract Machines*, together with which it provides a framework for the development of mathematically verified systems.

GSL can be extended to the *probabilistic Generalised Substitution Language* (*pGSL*), in which the standard Boolean values—representing certainty—are replaced by real values—representing probabilities. In principle, the *standard* machines of *the B Method* (*B*) can be extended to *probabilistic B* (*pB*) machines, which would allow us to implement random algorithms, or to model faulty (unreliable) operations. For practical use, we need to extend the standard toolkit to be able to generate proof obligations for the probabilistic constructs, and to enable proofs to be conducted in the standard set of Booleans extended by the set of reals.

This paper is concerned with the development and tool support of probabilistic machines based on *pGSL*. There are many foundational issues on probabilistic computational models that are not the subject of this paper and are not addressed here; more complete references may be found elsewhere [13]. The theory on which this paper is based [9] requires that those real values—which we call "expectations"—are non-negative and bounded. To avoid clutter in the exposition, however, our examples below do not necessarily adhere to those constraints.

The contribution of this paper is to extend the concept of *invariant* to probabilistic machines, based on the theory of *pGSL*: we define probabilistic invariants; we set out the proof obligations for maintaining such invariants by extending the current rules in the B; we give informal interpretations of the meaning of those invariants in practice; we develop a machine construct and give examples of how to use it; we highlight possible pitfalls; and we suggest approaches to correct them.

2 An Introduction to Probability and *pGSL*

2.1 Elementary Probability Theory

We briefly review and define some elementary concepts in probability theory [6]: the principal concepts we need are distribution and expectation.

Experiment: Any process of observation or measurement.
Outcomes: The results obtained from an experiment.
Sample space: The set of all possible outcomes of an experiment.
Event: A subset of the sample space.
Probability distribution (discrete): A normalised function from the sample space to $[0, 1]$ giving the probability of each outcome.
Random variable: Any function from the sample space into the reals.
Characteristic function: The *characteristic function* of an event is a random variable that takes value 1 for outcomes in the event, and 0 otherwise. Given an event *pre* (written as a predicate) the expression $\langle pre \rangle$ is the characteristic function of that event.
Expected value (discrete): If f is a bounded random variable and μ is a discrete distribution, both over sample space S, then the expected value of f over μ is defined:

$$\sum_{s \in S} f(s) * \mu(s) \ .$$

This simplified presentation is sufficient for our purpose here.

As a consequence of the above definitions, it can be shown that the expected value of a characteristic function over a distribution is equal to the probability assigned to its underlying set by the distribution.

The probabilistic generalised substitution language *pGSL* acts over "expectations" rather than predicates. Expectations are bounded, non-negative, real-valued functions of the state space; with the exception that when dealing with miracles they can take a formal value ∞.

$[x := E\,]\mathit{exp}$	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.
$[y, x := F, E\,]\mathit{exp}$	The expectation obtained after replacing all free occurrences of $y$and x in exp by F and E respectivley, renaming bound variables in exp if necessary to avoid capture of free variables in F and E.
$[\mathit{pre} \mid \mathit{prog}\,]\mathit{exp}$	$\langle \mathit{pre}\rangle \times [\mathit{prog}\,]\mathit{exp}$, where $0 \times \infty \mathrel{\widehat{=}} 0$.
$[\mathit{prog}_1 \;[\!]\; \mathit{prog}_2]\mathit{exp}$	$[\mathit{prog}_1]\mathit{exp}\;\;\mathsf{min}\;\;[\mathit{prog}_2]\mathit{exp}$
$[\mathit{pre} \Longrightarrow \mathit{prog}\,]\mathit{exp}$	$1/\,\langle \mathit{pre}\rangle \times [\mathit{prog}\,]\mathit{exp}$, where $\infty \times 0 \mathrel{\widehat{=}} \infty$.
$[\mathrm{SKIP}]\mathit{exp}$	exp
$[\mathit{prog}_1\;{}_p\!\oplus\;\mathit{prog}_2]\mathit{exp}$	$p \times [\mathit{prog}_1]\mathit{exp} \;+\; (1{-}p) \times [\mathit{prog}_2]\mathit{exp}$
$[@y \cdot \mathit{pred} \Longrightarrow \mathit{prog}\,]\mathit{exp}$	$(\mathsf{min}\;y \mid \mathit{pred} \cdot [\mathit{prog}\,]\mathit{exp})$, where y does not occur free in exp.
$\mathit{prog}_1 \sqsubseteq \mathit{prog}_2$	$[\mathit{prog}_1]\mathit{exp} \Rrightarrow [\mathit{prog}_2]\mathit{exp}$ for all exp

- exp is an expectation (possibly but not necessarily $\langle \mathit{pred}\rangle$ for some predicate pred);
- pre is a predicate (not an expectation);
- $\langle \mathit{pre}\rangle$ denotes the predicate pre converted to an expectation, here restricted to the unit interval: $\langle \mathit{false}\rangle$ is 0 and $\langle \mathit{true}\rangle$ is 1.
- $\times$ is multiplication;
- $\mathit{prog}, \mathit{prog}_1, \mathit{prog}_2$ are probabilistic generalised substitutions;
- p is an expression over the program variables (possibly but not necessarily a constant), taking a value in $[0, 1]$; and
- x is a variable.
- y is a variable, or a vector of variables.
- E is an expression.
- F is an expression, or a vector of expressions

We give the definitions including infeasible or "miraculous" commands [12, Sec. 1.7], but omit them in the main text for brevity. Also, with these definitions, monotonicity is maintained; instead of conjuntivity, we have a more general property *sublinearity* [13]. We do not use sublinearity here.

Fig. 1. *pGSL*—the probabilistic Generalised Substitution Language [13]

2.2 Brief Introduction to *pGSL*

pGSL is a logic for reasoning about programs operating over a computational model in which initial states are taken to final distributions over states (or, in the case of demonic programs, to *sets* of final distributions). The details of that model can be found elsewhere [10].

For the details of *pGSL* itself we first refer the reader to our companion paper [11, Sec. 2.1 and Sec. 2.2]. Fig. 1 gives a summary of substitutions in *pGSL*.

2.3 Probabilistic *pGSL* Extends Standard *GSL*

Recall that in (standard) *GSL* we typically deal with conclusions of the form $\mathit{pre} \Rightarrow [\mathit{prog}]\mathit{post}$, which means "the final state is guaranteed to satisfy *post* if the initial state satisfied *pre*". In *pGSL*, instead we have conclusions of the form $\mathit{preE} \Rrightarrow [\mathit{prog}]\mathit{postE}$, which means "the expected value of *postE* in the final state is at least the expected value of *preE* in the initial state". In fact, the *pGSL* interpretation generalises the standard interpretation, as we now show.

Suppose our pre- and post-expectations are "standard", that is they are of the form $\langle \mathit{pre}\rangle$ and $\langle \mathit{post}\rangle$. In that case, our second interpretation becomes "the expected value of $\langle \mathit{post}\rangle$ in the final state is at least the expected value of $\langle \mathit{pre}\rangle$ in the initial state". But we know from elementary probability theory (Sec. 2.1) that the expected value of a characteristic function of a predicate (for us, a predicate is the same as a subset of state space—that is, it is an event) is just the probability that the predicate holds: so we have really said "the probability that *post* holds in the final state is at least the probability that *pre* held in the initial state".

For standard programs, predicates either hold (probability 1) or they do not (probability 0). And for x, y in $\{0, 1\}$ to say $x \leq y$ is only to say "y is 1 if x was 1". Thus, for those specifically, we have said "the probability that *post* holds in the final state is 1 if the probability that *pre* held in the initial state was 1"—and this is just the usual interpretation in standard *GSL*.

2.4 Some *pGSL* Idioms

In this paper, we will be dealing with conclusions of the form

$$\mathit{exp} \quad \equiv \quad [\mathit{prog}]\mathit{exp} \tag{1}$$

for some expectation *exp* and probabilistic substitution *prog*. By analogy with the standard case,

$$\mathit{pred} \quad \equiv \quad [\mathit{prog}]\mathit{pred},$$

we call *exp* an *invariant* of *prog*.

For example, we toss a fair coin and want to estimate the number of heads that turn up. Let *nn* be the number of times that we toss the coin and *cc* be the number of times that head turns up. Considering the expectation $\mathit{nn} - 2 \times \mathit{cc}$, we calculate

$$\left[\begin{array}{l} cc := cc + 1 \;{}_{\frac{1}{2}}\oplus\; cc := cc \;||\\ nn := nn + 1 \end{array}\right] (nn - 2 \times cc)$$

$\equiv$ parallel substitution[3]

$$\left[\begin{array}{ll} & (cc := cc + 1 \;||\; nn := nn + 1) \\ {}_{\frac{1}{2}}\oplus & (cc := cc \quad\;\; ||\; nn := nn + 1) \end{array}\right] (nn - 2 \times cc)$$

$\equiv$ ${}_{\frac{1}{2}}\oplus$ and $:=$

$$\begin{array}{l} \quad (1/2)((nn + 1) - 2 \times (cc + 1)) \\ + \; (1/2)((nn + 1) - 2 \times cc) \end{array}$$

$\equiv$ arithmetic

$$nn - 2 \times cc$$

According to our interpretation (Sec. 2.3) of *pGSL*, this calculation shows that the expected value of $nn - 2 \times cc$ is never decreased by this operation. If we initialise nn and cc both to 0 then that expectation is initially 0—then we have shown that the expected value of $nn - 2 \times cc$ is never negative. In other words, the expected value of cc is never greater than $nn/2$.

Such interpretations will be crucial to our understanding of "probabilistic invariant".

3 Using *pGSL*: Probabilistic Machines

In earlier sections we introduced *pGSL* and explained its use of expectations to interpret individual probabilistic substitutions. Here we focus on our main topic: the meaning of these expectations when used as "invariants" for a *pB* machine.

We begin with a standard specification—the well-known "library" example—and use that as a basis against which we can contrast a probabilistic version. Our aim is, first, to show how probabilistic invariants capture its probabilistic properties and, second, to highlight some of the unexpected and subtle issues that can arise.

3.1 A Simple Library in *B*

Consider the specification of a simple Library in Fig. 2. The state of the machine contains three variables, namely *booksInLibrary*, *loansStarted* and *loansEnded* representing: the number of books in the library; the number of book loans initiated by the library; and the number of book loans completed by the library, respectively. To keep the example simple, we ignore other functions of the library.

[3] Here we apply the simple rule to integrate parallel substitution ($||$) with probabilistic choice substitution (${}_{pp}\oplus$):
Provided S, T, U are all standard, we have:

$$(S \;{}_{p}\oplus\; T) \;||\; U = (S \;||\; U) \;{}_{p}\oplus\; (T \;||\; U) \;.$$

```
MACHINE StandardLibrary ( totalBooks )
VARIABLES
  booksInLibrary , loansStarted , loansEnded
INVARIANT
  $booksInLibrary \in \mathbb{N} \wedge loansStarted \in \mathbb{N} \wedge loansEnded \in \mathbb{N} \wedge$
  $loansEnded \leq loansStarted \wedge$
  $booksInLibrary + loansStarted - loansEnded = totalBooks$
INITIALISATION
  $booksInLibrary := totalBooks \;\|\; loansStarted := 0 \;\|\; loansEnded := 0$

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
```

Fig. 2. Standard specification of a Library

Initially, *booksInLibrary* has value *totalBooks* (a parameter of the machine). Both *loansStarted* and *loansEnded* are assigned 0 initially.

We have two operations that can modify the state of the machine, StartLoan, for starting a loan of a book, and EndLoan, for ending the loan of a book. The StartLoan operation has a precondition that there are books available for loan; it decrements the books held and increments the book loans. The EndLoan operation is complementary in the obvious way.

The (standard) invariant of this machine is

$$booksInLibrary + (loansStarted - loansEnded) \quad = \quad totalBooks \ , \qquad (2)$$

in which the term $loansStarted - loansEnded$ is an abstraction of the number of books that are in the on-loan database of the library—that is, books that are recorded as on loan.

3.2 Adding Probabilistic Properties

In the Boolean world of standard B, the operations and invariants express certainty: books are either in the library or they are on loan; they can't be anywhere

```
MACHINE  $ProbabilisticLibrary\ (\ totalBooks\ )$
SEES  $Real\_TYPE$
CONSTANTS  $pp$
PROPERTIES  $pp \in REAL \land pp \leq real\ (\ 1\ ) \land real\ (\ 0\ ) \leq pp$
VARIABLES
   $booksInLibrary\ ,\ loansStarted\ ,\ loansEnded\ ,\ booksLost$
INVARIANT
   $booksInLibrary \in \mathbb{N} \land loansStarted \in \mathbb{N} \land loansEnded \in \mathbb{N} \land booksLost \in \mathbb{N} \land$
   $loansEnded \leq loansStarted \land$
   $booksInLibrary + booksLost + loansStarted - loansEnded = totalBooks$
EXPECTATIONS
   $real\ (\ 0\ ) \Rrightarrow pp \times real\ (\ loansEnded\ ) - real\ (\ booksLost\ )$
INITIALISATION
   $booksInLibrary\ ,\ loansStarted\ ,\ loansEnded\ ,\ booksLost := totalBooks\ ,\ 0\ ,\ 0\ ,\ 0$

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

    EndLoan $\widehat{=}$
      PRE  $loansEnded < loansStarted$  THEN
         PCHOICE  $pp$  OF
            $booksLost := booksLost + 1$
         OR
            $booksInLibrary := booksInLibrary + 1$
         END  $\|$
         $loansEnded := loansEnded + 1$
      END
END
```

Fig. 3. Simple probabilistic Library

else. In a real library, books are occasionally lost. In this section, we discuss how that can be modelled in pB.

One approach might be to add a *Lose* operation of the form

$$Lose \quad \widehat{=} \quad booksInLibrary := booksInLibrary - 1\ , \tag{3}$$

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 can however model operations with probabilistic *effects* in pB, and so we take that approach.

The loss of a book will be modelled by altering the EndLoan operation so that, with some probability pp, the user fails to return a book to the library; in that case the effect of EndLoan is to consider the book lost.

With the introduction of probabilistic-choice substitution (in Sec. 2), we can specify this behaviour within the EndLoan operation. The *PCHOICE* construct is the *probabilistic AMN* (*pAMN*) counterpart of ${}_p\oplus$. In this operation, the chance of a book being lost is pp—where *booksInLibrary* fails to increase; the other $1-pp$ of the time, *booksInLibrary* increases as normal. The variable *booksLost* is introduced to record the number of books lost and is initialised to 0. In the case of losing a book, *booksLost* will increase accordingly. We replace the standard substitution $booksInLibrary := booksInLibrary + 1$ with

```
PCHOICE  pp  OF
    booksLost := booksLost + 1
OR
    booksInLibrary := booksInLibrary + 1
END
```

With the introduction of the variable *booksLost*, we must of course adjust the standard invariant, to include the new variable:

$$\begin{array}{ll} & (booksInLibrary + booksLost) \\ + & (loansStarted - loansEnded) \\ = & totalBooks\ . \end{array} \tag{4}$$

The first term on the left-hand side is the number of books not in the on-loan database; the second term is the number of books that are in the on-loan database. This specification is simply modelling the effect of loss, without attempting to identify where it occurs. In practice, loss could be the consequence of a faulty (unreliable) loan or return operation. At some point, "loss" needs to be recognised and that is modelled by the probabilistic $booksLost{:=}booksLost{+}1$.

3.3 The *EXPECTATIONS* Clause

In Fig. 1 we introduced a new *EXPECTATIONS* clause into *pAMN* for declaring the probabilistic invariant. It gives an expression V over the program variables, denoting the random-variable invariant, and an initial expression e which is evaluated over the program variables when the machine is initialised. We write it $e \Rrightarrow V$. Its interpretation is that the expected value of V, at any point, is always at least the value of e initially. The value of e can be dependent on the context of the machine (machine's parameters, constants, etc.), but often e will just be a constant.

3.4 What Do Probabilistic Invariants Guarantee?

We answer that by analogy with standard invariants, which we review first.

Suppose a machine has initialisation $\textsc{Init}$ and two operations $\textsc{OpX}$ and $\textsc{OpY}$. If we satisfy the standard proof obligations with respect to some invariant I, viz.

$$\begin{aligned} \mathsf{true} &\Rightarrow [\textsc{Init}]I \\ I &\Rightarrow [\textsc{OpX}]I \\ I &\Rightarrow [\textsc{OpY}]I, \end{aligned} \tag{5}$$

then we are assured that

$$\mathsf{true} \Rightarrow [\textsc{Init};\ \textsc{Op?};\ \textsc{Op?};\ \ldots;\ \textsc{Op?}]I \tag{6}$$

holds for any (finite) sequence of operations $\textsc{Op?}$ each chosen from $\{\textsc{OpX}, \textsc{OpY}\}$. It doesn't matter when the choice between "do $\textsc{OpX}$ now", "do $\textsc{OpY}$ now" and "stop now" is made—that is whether the sequence of operations is chosen in advance or whether it is evolved "on-the-fly" on the basis of the machine state and/or outputs produced so far by the operations already executed.

The fact that (5) assures (6) is in fact the *soundness* of the (standard) invariant technique.

For soundness of the probabilistic invariant technique clearly there must be a similar situation—that is a probabilistic version of (5) and (6)—with the first implying the second: it is that if

$$\begin{aligned} E &\Rrightarrow [\textsc{Init}]I \\ I &\Rrightarrow [\textsc{OpX}]I \\ I &\Rrightarrow [\textsc{OpY}]I, \end{aligned} \tag{7}$$

then we are assured that

$$E \Rrightarrow [\textsc{Init};\ \textsc{Op?};\ \textsc{Op?};\ \ldots;\ \textsc{Op?}]I \tag{8}$$

for any finite sequence $\textsc{Op?};\ \textsc{Op?};\ ...;\ \textsc{Op?}$ of operations, no matter when or how chosen. (Recall that E is some "initial" expression, possibly depending on parameters to the machine.)

It might be surprising that in the probabilistic case the "when/how" makes a crucial difference; we show by example that it does.[4] Consider the *Counter* machine shown in Fig. 4

This machine *fails* to satisfy our probabilistic proof obligations (7) even though

$$0 \Rrightarrow [\textsc{Init};\ \textsc{Op?};\ \textsc{Op?};\ ...;\ \textsc{Op?}](\mathit{count}) \tag{9}$$

[4] It can be shown that it makes no difference in the standard case whether the operations are chosen beforehand or on-the-fly—the proof obligations are the same.

```
MACHINE Counter
SEES
    Int_TYPE , Real_TYPE
VARIABLES
    count
INVARIANT
    count ∈ INT
INITIALISATION
    count := 0

OPERATIONS
    cc ⟵ OpX ≙
        PCHOICE frac ( 1 , 2 ) OF
            count := count + 1 ||
            cc := count
        OR
            count := count − 1 ||
            cc := count
        END ;

    OpY ≙ count := 0

END
```

Fig. 4. The Counter Machine

is trivially true, for any finite sequence OP?; OP?; ...; OP? of operations *chosen in advance.*

The machine fails to satisfy (7) because *count* $\Rrightarrow$ [OPY](*count*) cannot be proved. And the reason that it *must* fail is that (9) is not true—for this machine—if the operations can be chosen on-the-fly: consider for example the program fragment

$$\begin{aligned} Prog \mathrel{\widehat{=}}\ & \textsc{Init}; \\ & c \longleftarrow \textsc{OpX}; \\ & \textbf{IF } c = 1 \textbf{ THEN } \textsc{OpY} \textbf{ ELSE } \textsc{OpX} \textbf{ END}. \end{aligned} \tag{10}$$

The IF-statement represents a choice, on-the-fly, of whether to execute OPX or OPY as the second operation; and it is readily verified that the expected value of count after (10) is -0.5, which fails the instantiation (9) of the general (8) for this machine. That is, we do *not* have $0 \Rrightarrow [Prog](count)$.

Thus the answer to the title of this section is that

probabilistic invariants guarantee (8) provided (7) holds.

The strong constraint "no matter how the operations are chosen" in (4) is absolutely necessary: the (usual) situation is that our machine must behave correctly no matter what environment makes use of it. A system containing code like (10) is a perfectly reasonable use of MACHINE Counter—and any such system, if it depended on the expected value of count being non-negative afterwards, would fail.

3.5 A Probabilistic Invariant for the Library

In this section, we try to find the probabilistic invariant for a probabilistic library (Fig. 3) by "informal" reasoning.

With the introduction of probabilistic choice substitution in the new EndLoan operation, we want an estimated upper bound for the number of books lost. Informally, we believe that $pp * loansEnded$ is the expected value of the number of books actually lost. That informal reasoning leads to:

the expected value of $pp{*}loansEnded{-}booksLost$ is at least 0.

Thus we define $V \mathrel{\widehat{=}} pp * loansEnded - booksLost$ to be the expected-value invariant of the probabilistic library machine. The initial value for V is 0 (established by the initialisation).

When we claim that V is an expected-value invariant for this machine, we mean that, if we check the value of V many times during the running of operations of the Library, then the average of our observation of V will be at least 0. As explained in Sec. 3.4 that is the intended meaning of our probabilistic invariant. From that we can conclude for our probabilistic library machine, the expected number of books lost (value of *booksLost*) is bounded above by $pp{*}loansEnded$.

3.6 Proof Obligations

Recall that the proof obligations for a non-probabilistic machine are:

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

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

For probabilistic machines, the same ideas will be applied, except that the invariant now may take *real* values instead of Boolean. In order to prove that the *real* invariant is bounded below, we have to prove the following:

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

$$e \Rrightarrow [\mathit{Init}]\, V \ .$$

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

$$V \Rrightarrow [\mathit{Op}]\, V \ .$$

We have to prove the above for each *real-valued* invariant. The standard (Boolean) invariants can be treated the same as before (with probabilistic choice substitution being treated as demonic). Consequently, proof obligations for the *probabilistic* (expectation) and Boolean invariants may be generated, and proved, separately.

3.7 Proving the Obligations

Here we only discuss the proof of maintenance of the probabilistic invariant:

$$V \quad \widehat{=} \quad pp * \mathit{loansEnded} - \mathit{booksLost} \ . \tag{11}$$

In the example in Fig. 3, consider the proof obligation for the initialisation (*P1*).[5] We have to prove that

$$0 \Rrightarrow [\mathit{Initialisation}]\, V \ .$$

Consider the right-hand side of the inequality:

$$\begin{array}{ll} & [\mathit{Initialisation}]\, V \\ \\ \equiv & \left[\begin{array}{ll} \mathit{booksInLibrary}, \mathit{loansStarted}, & \\ \mathit{loansEnded}, \mathit{booksLost} & := \mathit{totalBooks}, 0, 0, 0 \end{array}\right] V \\ \\ \equiv & \left[\begin{array}{ll} \mathit{booksInLibrary}, \mathit{loansStarted}, & \\ \mathit{loansEnded}, \mathit{booksLost} & := \mathit{totalBooks}, 0, 0, 0 \end{array}\right] \left(\begin{array}{c} pp * \mathit{loansEnded} \\ -\mathit{booksLost} \end{array}\right) \\ \\ \equiv & pp * 0 - 0 \qquad \text{substitution} \\ \equiv & 0 \qquad \text{arithmetic} \end{array}$$

So we have shown that the initialisation establishes the initial lower bound for the probabilistic invariant.

For operation StartLoan, since the operation both increases *loansStarted* and decreases *booksInLibrary* deterministically, and since the expected-value invariant does not contain *loansStarted* and *booksInLibrary*, we can easily prove that the operation maintains the invariant.

We have to do similar reasoning with operation EndLoan, i.e. to prove that $V \Rrightarrow [\mathit{EndLoan}]\, V$ (proof obligation *P2*). We calculate

[5] All calculations use real numbers, but we will omit any type casting.

$$[\mathit{EndLoan}]\,V$$

$$\equiv \quad \left[\begin{array}{l} \begin{pmatrix} (\mathit{booksLost} := \mathit{booksLost} + 1 \\ {}_{pp}\oplus \\ \mathit{booksInLibrary} := \mathit{booksInLibrary} + 1) \end{pmatrix} \\ || \\ \mathit{loansEnded} := \mathit{loansEnded} + 1 \end{array}\right] V$$

$\equiv$ parallel substitution with ${}_{pp}\oplus$

$$\left[\begin{array}{l} \begin{pmatrix} (\mathit{booksLost} := \mathit{booksLost} + 1 \\ || \\ \mathit{loansEnded} := \mathit{loansEnded} + 1) \end{pmatrix} \\ {}_{pp}\oplus \\ \begin{pmatrix} (\mathit{booksInLibrary} := \mathit{booksInLibrary} + 1 \\ || \\ \mathit{loansEnded} := \mathit{loansEnded} + 1) \end{pmatrix} \end{array}\right] \begin{pmatrix} pp * \mathit{loansEnded} \\ -\mathit{booksLost} \end{pmatrix}$$

$\equiv$ ${}_{pp}\oplus$

$$\begin{array}{rl} pp * & \left[\begin{array}{l} \mathit{booksLost} := \mathit{booksLost} + 1 \\ || \\ \mathit{loansEnded} := \mathit{loansEnded} + 1 \end{array}\right] \begin{pmatrix} pp * \mathit{loansEnded} \\ -\mathit{booksLost} \end{pmatrix} \\ +\,(1 - pp) * & \left[\begin{array}{l} \mathit{booksInLibrary} := \mathit{booksInLibrary} + 1 \\ || \\ \mathit{loansEnded} := \mathit{loansEnded} + 1 \end{array}\right] \begin{pmatrix} pp * \mathit{loansEnded} \\ -\mathit{booksLost} \end{pmatrix} \end{array}$$

$\equiv$ parallel substitution and $:=$

$$\begin{array}{l} \quad pp * (pp * (\mathit{loansEnded} + 1) - (\mathit{booksLost} + 1)) \\ +\,(1 - pp) * (pp * (\mathit{loansEnded} + 1) - \mathit{booksLost}) \end{array}$$

$\equiv \quad pp * \mathit{loansEnded} - \mathit{booksLost}$ arithmetic

$\equiv \quad V$

So we have shown that $V \Rrightarrow [\mathit{EndLoan}]\,V$. (In fact, the expectation is unchanged since there is no demonic nondeterminism).

In this example, we have specified a library system that includes the chance of books being lost. From the probabilistic invariant, we can estimate the cost of maintaining the library (the number of books lost). Furthermore, we have discussed how we can reason about the specification and how to write it in *pB*.

3.8 What the Invariant Means

With the two calculations of the previous section we have established the mathematical validity of the invariant V for the machine of Fig. 3, in the sense that the proof obligations are satisfied. How do we interpret that validity?

Recall Sec. 3.4: it means that over a large number of tests of the machine, carried out by an adversary, who can choose to resolve demonic choice within

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

Fig. 5. StockTake operation

operations any way he wishes (although there is none in our example), 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.

In the machine of Fig. 3, we conclude therefore that the expected value of $pp * loansEnded - booksLost$ is at least 0; no matter what the adversary does. We wrote the invariant that way so that we could give an expected upper bound for *booksLost*—it is $pp * loansEnded$.

In general, we might wish to establish several such average-case inequalities. For each one we would formulate a suitable probabilistic invariant and lower bound; and each would generate its own proof obligations (*P1*), (*P2*) as above.

4 Pitfalls: Mixing Demonic and Probabilistic Choice

The validity of a probabilistic invariant assures us of a lower bound for its average value over many sequences of machine-operation invocations. In this section we show by example just how strong a requirement that is, given an adversarial tester who has complete freedom in choosing which operations to invoke. We show how the mathematical constraint of having to prove the invariant's validity guides us in the design of machines that are well-behaved even against such adversaries.

4.1 StockTake Breaks the Probabilistic Invariant

Imagine that every year, the library needs to do a stocktake: update the number of book lost, and reset the information about the status of the library. The library wants to estimate the cost for doing such operations annually. Assuming that the cost for replacing a book is a constant, *cost*, the operation StockTake is defined in Fig. 5.

The StockTake operation is very similar to the initialisation, but with an extra output to represent the cost for replacing the books lost. One can easily prove that the operation maintains the standard invariant. The surprise comes when trying to prove the obligation for maintaining the probabilistic invariant by this operation. We have to prove that $V \Rrightarrow [\mathit{StockTake}]\, V$. Consider the right-hand side of that inequality (considering the effect of variables *loansEnded* and *booksLost* only):

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

So to show the invariant does not decrease we must prove that

$$\mathit{pp} * \mathit{loansEnded} - \mathit{booksLost} \quad \Rrightarrow \quad 0\ , \tag{12}$$

which we cannot prove in this context. The question here is what did we do wrong in the above operation.

4.2 Surprising Interaction of Demonic and Probabilistic Choice

To understand the failure to maintain the probabilistic invariant we will discuss a number of aspects.

Initialisation is not forever It is first worth reviewing why we might have expected the StockTake operation to be satisfactory. We might have observed that StockTake is very similar to the machine initialisation. In standard B it is obvious that we can repeat the initialisation whenever we wish, and the standard invariant will be maintained. However, maintaining the probabilistic invariant means not decreasing the expected value. If the standard B invariant is viewed in this light, it is a Boolean expression that is expected to evaluate to true (1) after the initialisation. This represents a monotonic increase over its value before initialisation, which was either false (0) or true (1). If the initialisation is re-run as an operation it starts from an expected value of true (1) and so guarantees not to decrease the expected value.

It is obvious that when we move to real-valued expectations, the obligation of maintaining the expected value is stronger, and some notions taken from the simpler Boolean context will fail.

The initialisation of a pB machine establishes the probabilistic invariant on the assumption of a lower bound of the expectation; for the probabilistic library machine this was 0. Since a sequence of operations monotonically increases the probabilistic invariant it is presumptuous to expect that the initialisation, if run again at an arbitrary time would maintain the invariant. Thus, in general, there is no guarantee that an operation that duplicates the initialisation will maintain the probabilistic invariant.

The effect of demonic nondeterminism It is worth reminding ourselves that the choice of operations for a machine is demonically nondeterministic. As a consequence the machines must be designed to ensure that undesirable operation sequences do not lead to the violation of critical properties of machine behaviour. It is precisely for this reason that we use invariance in both non-probabilistic and probabilistic machines to establish that such critical properties are maintained regardless of the choice of operation sequence.

The probabilistic library machine is intended to achieve an upper bound of $pp * loansEnded$ for *booksLost*. Before the addition of the StockTake operation this was being controlled by the probabilistic choice in the EndLoan operation. StockTake now provides an opportunity for demonic nondeterminism to subvert that expectation, according to the following scenario. Suppose 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".

Notice that in a probabilistic machine there can be times when loss rate will be higher than expected. The problem with the above scenario is that the operation is chosen demonically to run only at those times. Consider the testing of a system. We might suggest that a machine-tester, in selecting what operation to run next, should not be able to see the current state of the machine.[6] We say that the demonic choice (taken by the tester), of which operation to run next, is *omniscient* if he is allowed to see the machine's state, and *oblivious* if he is not. Omniscient testing is clearly more severe than oblivious; so our proof obligations, which are sufficient to guarantee correct behaviour under omniscient testing, are stronger than alternative obligations we might formulate to guarantee survival under oblivious testing. (In fact, operation StockTake would probably be admitted by proof obligations designed for oblivious testing.[7])

Finally, we note that there is no difference between omniscient and oblivious testing of standard machines: if a standard machine is guaranteed to survive oblivious testing, then it is also guaranteed to survive omniscient testing. Only for probabilistic[8] machines does the omniscient/oblivious distinction matter.

4.3 Capturing Long-Term Behaviour

In our failure to satisfy the proof obligation for StockTake, the mathematics is in fact suggesting what we should do. *Formally* we introduce a new variable—call

[6] That this is reasonable can be seen by testing a coin: it would be wrong to flip the coin until heads shows, and then say "look, it always gives heads".

[7] We do not pursue the mathematical formulation of oblivious testing here, as the notion of what "can be seen" and what cannot turns out to be surprisingly complex. But we recognise it as a fruitful line of further research.

[8] In fact, adding angelic choice also reveals the distinction: any two of probabilistic/angelic/demonic are sufficient.

it "*fix*" for now—with the sole purpose of being able to satisfy that obligation. Routine calculation shows that we must modify the machine as follows: *fix* is given the value 0 initially; the value of *fix* is unchanged in all other operations; but in StockTake, we use *fix* to maintain information that is critical to the expectation:

$$\mathit{fix} := \mathit{pp} * \mathit{loansEnded} - \mathit{booksLost} + \mathit{fix}\ . \tag{13}$$

The expectation invariant is modified as follows:

$$V' \quad \widehat{=} \quad \mathit{pp} * \mathit{loansEnded} - \mathit{booksLost} + \mathit{fix}\ . \tag{14}$$

The new expectation invariant has the lower bound 0 established by the initialisation. Since the operations StartLoan and EndLoan do not change the value of *fix*, they will maintain invariant V' (in those cases *fix* acts as a constant). For the StockTake operation, we can prove that it maintains the probabilistic invariant V' (considering the changes for *loansEnded*, *booksLost* and *fix* only):

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

So $V' \Rrightarrow [\mathit{StockTake}]\, V'$—that is, the StockTake operation does not decrease the expected value of V'.

But what is the interpretation of *fix*? It is in fact a "running" long-term surplus/deficit indicator of books lost compared with what we expected to lose; and our new invariant tells us that we expect that indicator to be zero.

More abstractly, we see that the invariant is forcing us not to "lose information" as the original version of StockTake did. The reason—we can now see—is that meaningful statements about long-term behaviour can only be made if there are variables which record it; and operations that somehow "erase" the long-term behaviour will fail proof obligations.

In this specific case, we can say that—without *fix*—a hostile library administrator could decide (demonic choice) to run StockTake only when the rate of stolen books was "running high", and so give a false picture in that "snapshot" of the long-term behaviour of the library. Including *fix* makes sure that the snapshot includes *all* the behaviour up to that point.

5 Modifying the *B-Toolkit*

The *B-Toolkit* is a configuration management tool that assists a developer to produce a logically consistent set of *B* machines. Some of the important services provided by the tool are: analysis including syntax and type checking;

proof obligation generation; proof assistance (both automatic and interactive); and machine markup. The replacement of *GSL* by *pGSL*—with the consequent replacement of *Abstract Machine Notation* (*AMN*) by *pAMN* obviously affects those processes. 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 $\mathit{frac}(m, n)$.

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. Every process after analysis will use the canonic information rather than using the raw *AMN*.

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 *pGSL* as stated in figure Fig. 1. 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.

It should be noted that the ProbabilisticLibrary machine has been: analysed; proof obligations have been generated; proof obligations have been discharged; and the machine marked up using the modified *B-Toolkit*. The marked-up text of machines appearing in this paper have been included directly from the *B-Toolkit*.

6 Further Work

The loss of conjunctivity—to be replaced by sublinearity—brings some complicated problem when it comes to refinement under *pGSL*. In general, refinement is a second-order property, but with the interpretation from Gries [7], we get an equivalent first-order refinement rule for the standard case of *B* (using *GSL*). Unfortunately, the introduction of probabilistic choice substitution does not preserve the necessary condition for the maintenance of the first-order equivalence. A current challenge is the development of a suitable strategy for refinement under *pGSL*. A possible solution might be the use of auxiliary variables and additional special rules for maintaining the meaning of refinement using first-order logic. It should be noted that first-order rules offer considerable advantages for the implementation of tools.

7 Conclusions

In this paper, we have presented a practical approach to extending the *B* to include probabilistic choice. We have extended *pB* machines to include a probabilistic choice construct and a probabilistic state invariant. New proof obligations for the establishment and maintenance of the probabilistic invariant have been described. A simple case study of a library, in which books may be lost, has been used to illustrate how we can use *pB* and how we can reason formally about expected outcomes of the system. We have shown that there are significant differences between standard *B* operations and *pB* operations.

The *B-Toolkit* has been modified to incorporate the new *pAMN* constructs and also to provide the generation and proof of proof obligations. In some cases, we have had to strike a balance between the theoretical and practical ideas to accommodate probability successfully into the *B-Toolkit*. Further investigation is required on refinement and machine composition using the *pB*.

Beyond a guarantee of "absolute" correctness, other major aspects of system/software design are essentially quantitative. Often the operating conditions provide an environment whose behaviour can only be estimated to within a "probabilistic margin of error". But even in these situations useful information about the operating "performance" of the implemented system can still be guaranteed. Other situations call for probability to be deliberately "programmed into the system" when standard methods fail to produce a guarantee of termination.

The main case study of this paper provides an example of the former situation, whilst there are many instances of the latter situation in distributed computing, with the FireWire protocol [14] and Rabin's distributed consensus [2] providing typical examples. In both situations the quantitative specification can be expressed as a numeric constraint and validated by the techniques set out in this paper. We have a complete development—through to implementation using probabilistic termination [11]—of Rabin's distributed consensus algorithm in *pB*.

Indeed many more performance-style specifications lend themselves to an approach based on invariants including "the expected time to achieve a stated goal" [8] or the "probability that a goal will be achieved within a specific time". Expected times to achieve "stability" for instance are of particular importance when systems use probability as an "in-built" facility.

Other tools such as the model checker PRISM [1] can also deal with these problems, however the model-checking approach contrasts fundamentally with the *B* in that model checkers are analysis- rather than design tools.

8 Acknowledgements

We wish to acknowledge the assistance of B-Core[5] for the modification of the *B-Toolkit*. We thank the anonymous reviewers for comments that we have used to improve the paper.

The authors at University of New South Wales gratefully acknowledge the support of the Australian Research Council under the large grant A00103115.

References

1. Probabilistic symbolic model checker. http://www.cs.bham.ac.uk/˜dxp/prism/publications.html.
2. Specification and development of probabilistic systems. http://web.comlab.ox.ac.uk/oucl/research/areas/probs/.
3. *Proceeding of the 3rd International Conference of B and Z Users*. Springer, 2003.
4. J-R. Abrial. *The B-Book*. Cambridge University Press, 1996.
5. B-Core(UK) Ltd. B Toolkit. http://www.b-core.com.
6. John E. Freund. *John E. Freund's Mathematical Statistics*. Prentice Hall International, Inc., 6 edition, 1999.
7. D. Gries and J. Prins. A new notion of encapsulation. In *Symposium on Language Issues in Programming Environments*. SIGPLAN, June 1985.
8. A. K. McIver. Quantitative program logic and counting rounds in probabilistic distributed algorithms. In *Proc. 5th Intl. Workshop ARTS '99*, volume 1601, 1999.
9. A. K. McIver and C. C. Morgan. Demonic, angelic and unbounded probabilistic choices in sequential programs. *Acta Informatica*, 37:329–354, 2001.
10. C. C. Morgan, A. K. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
11. A. K. McIver, C. C. Morgan, and Thai Son Hoang. Probabilistic termination in B. In *Proceeding of the 3rd International Conference of B and Z Users* [3].
12. C. C. Morgan. *Programming from Specifications*. Prentice-Hall, second edition, 1994. At `web.comlab.ox.ac.uk/oucl/publications/books/PfS`.
13. C. C. Morgan. The generalised substitution language extended to probabilistic programs. In *Proceedings B'98: the 2nd International B Conference*, volume 1393 of *LNCS*, Montpelier, April 1998. Also available at [2, B98].
14. Stoelinga and Vaandrager. Root contention in IEEE 1394. In *Proceedings of the 5th AMAST workshop on real time and probabilistic systems Bamberg, Germany, ARTS' 1999*, volume 1061 of *LNCS*.