Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 137 (2005) 183–204
www.elsevier.com/locate/entcs

Tank Monitoring: A pAMN Case Study

Steve Schneider[1]

Department of Computing, University of Surrey, UK

Thai Son Hoang

School of Computer Science and Engineering
University of New South Wales, Australia

Ken Robinson

School of Computer Science and Engineering
University of New South Wales, Australia

Helen Treharne

Department of Computing, University of Surrey, UK

Abstract

The introduction of probabilistic behaviour into the B-Method is a recent development. In addition to allowing probabilistic behaviour to be modelled, the relationship between expected values of the machine state can be expressed and verified. This paper explores the application of probabilistic B to a simple case study: tracking the volume of liquid held in a tank by measuring the flow of liquid into it. The flow can change as time progresses, and sensors are used to measure the flow with some degree of accuracy and reliability, modelled as non-deterministic and probabilistic behaviour respectively. At the specification level, the analysis is concerned with the EXPECTATION clause in the probabilistic B machine and its consistency with machine operations. At the refinement level, refinement and equivalence laws on probabilistic GSL are used to establish that a particular design of sensors delivers the required level of reliability.

Keywords: Probabilistic B, refinement, formal methods, probabilistic predicate transformers.

[1] Email: S.Schneider@surrey.ac.uk

1571-0661
doi:10.1016/j.entcs.2005.04.031

1 Introduction

The B-Method [1] provides a framework for the development of provably correct systems, based on the weakest precondition semantics of the *Generalised Substitution Language* (GSL), and structured around the concept of *Abstract Machines*.

The introduction of probabilistic behaviour into the B-Method has recently been proposed [2], called *probabilistic B*. This approach builds on previous work which introduces probabilistic choice into program statements, and extends the notion of weakest precondition semantics to deal with *expectations* [5]. An expectation can be considered as the expected value of a formula or expression. Thus programs can be viewed as *expectation transformers* rather than *predicate transformers*, and their semantics gives the expectation of an expression after the program has been executed in terms of expectations prior to execution.

In addition to allowing such probabilistic behaviour into programs, probabilistic B introduces expectations on aspects of the state, in addition to the existing parts of a B machine. Thus the relationship between the expected values of several components of the machine state can be expressed and formally verified.

This paper explores the application of probabilistic B to a simple case study: tracking the volume of liquid held in a tank by measuring the liquid flow into it. The flow can change as time progresses. Sensors with a given reliability are used to measure the flow and provide information to the system, so there is a small probability that the sensors will fail, giving an incorrect reading. The behaviour of the sensors is described using probabilistic B. We include the tank explicitly in our model so that we can describe the relationship between the actual volume of liquid it contains and our system's measurement for it. As well as probabilistic behaviour, our system exhibits nondeterministic behaviour in the reading that a failed sensor will give, and (after the first scenario we consider) in the reading that a correctly working sensor will give: any value from a particular range. Thus the case study also explores the interaction between probabilistic and nondeterministic behaviour.

The case study is concerned with two stages of the development process: specification, and refinement. At the specification level we are concerned with obtaining bounds on the accuracy of the system's value for the volume of liquid in the tank, given a particular level of reliability for the combination of sensors providing the readings. This analysis will be concerned with the EXPECTATION clause in the probabilistic B machine. At the refinement level, we are concerned with establishing that a particular combination of sensors does indeed deliver the required level of reliability. This analysis will make use of refinement and equivalence laws on probabilistic GSL.

2 Introducing Probability

2.1 Probabilistic GSL

pGSL is an extension of GSL to include a probabilistic choice statement:

$$prog_1 \;{}_p\!\oplus\; prog_2$$

An execution of this choice will execute $prog_1$ with probability p, and will execute $prog_2$ with probability $1 - p$. See [6,3,4] for a full introduction to pGSL.

To give a semantics to pGSL programs, we make use of expectations: bounded non-negative real-valued functions of the state space. These are generally expressed as formulas over the state variables. The weakest pre-expectation semantics for a program *prog* maps an expectation *exp* to another expectation $[prog]exp$, analogous to weakest precondition semantics. It gives the expected value for *exp* after *prog* in terms of expectations on the state before. The language and its semantics from [6] is given in Figure 1.

In this paper we will use a derived operator (also given in [1]) for assigning to a variable some element from a set S chosen nondeterministically. We define

$$x :\in S \mathrel{\widehat{=}} @y.(y \in S \Longrightarrow x := y)$$

Thus

$$[x :\in S]exp = (\min x \mid x \in S . exp)$$

We will also use a derived operator (also given in [3]) for expressing a minimum probability on a choice. We define

$$prog_1 \;{}_{\geqslant p}\!\oplus\; prog_2 \mathrel{\widehat{=}} @q.(p \leqslant q \leqslant 1) \Longrightarrow prog_1 \;{}_q\!\oplus\; prog_2$$

This program chooses $prog_1$ with a probability of at least p.

The operator is useful for describing systems with a minimum required reliability. If a component is required to behave correctly at least 90% of the time, then this may be described as $correct \;{}_{\geqslant 0.9}\!\oplus\; incorrect$. This would be refined by a component that behaves correctly at least 95% of the time, for example.

2.2 Some pGSL laws

The semantics supports a collection of algebraic laws concerning the various operators. An extended collection of laws is given in Appendix A.3 of [3]. The

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 ∞.

$[x := E]exp$	$exp[E/x]$
$[x, y := E, F]exp$	$exp[E, F/x, y]$
$[pre \mid prog]exp$	$\langle pre \rangle \times [prog]exp$, where $0 \times \infty \mathrel{\widehat{=}} 0$
$prog_1 [] prog_2$	$[prog_1]exp \min [prog_2]exp$
$[pre \Longrightarrow prog]exp$	$1/\langle pre \rangle \times [prog]exp$, where $\infty \times 0 \mathrel{\widehat{=}} \infty$
$[skip]exp$	exp
$[prog_1 \ {}_p\oplus \ prog_2]exp$	$p \times [prog_1]exp + (1-p) \times [prog_2]exp$
$[@y.pred \Longrightarrow prog]exp$	$(\min y \mid pred.[prog]exp)$
$prog_1 \sqsubseteq prog_2$	$[prog_1]exp \Rrightarrow [prog_2]exp$ for all exp.

- exp is an expectation
- pre is a predicate (not an expectation)
- $\langle pre \rangle$ denotes predicate pre converted to an expectation, here restricted to the unit interval: $\langle false \rangle$ is 0 and $\langle true \rangle$ is 1.
- $\times$ is multiplication.
- $prog$, $prog_1$, $prog_2$ are probabilistic generalised substitutions.
- p is an expression over the program variables (possibly but not necessarily constant), taking a value in $[0, 1]$.
- 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.
- $exp_1 \Rrightarrow exp_2$ means that exp_1 is everywhere no more than exp_2.

Fig. 1. *pGSL*—the probabilistic Generalised Substitution Language [6]

following laws from that Appendix will be used in this paper:

Law 13:

$$(prog_1 \ {}_{\geqslant p}\oplus \ prog_2);\ prog_3 = (prog_1;\ prog_3) \ {}_{\geqslant p}\oplus \ (prog_2;\ prog_3)$$

Law 24:

$$(prog_1 \ {}_{\geqslant pq}\oplus \ prog_2) = prog_1 \ {}_{\geqslant p}\oplus \ (prog_1 \ {}_{\geqslant q}\oplus \ prog_2)$$

We also make use of the following law, which we will call Law A:

$$prog_2 \sqsubseteq prog_1 \Rightarrow prog_1 \ {}_{\geqslant p}\oplus \ prog_2 = prog_1 \ {}_p\oplus \ prog_2$$

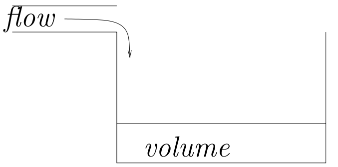

Fig. 2. The tank system

2.3 Probabilistic B

There are two aspects to the introduction of probabilistic behaviour into a B machine as proposed in [2]. The first is to allow operations to be constructed using probabilistic GSL, so probabilistic choices can be made within operations. The second is to introduce an EXPECTATION clause into a B machine in order to express requirements on various expectations on the state. An EXPECTATION clause will in general contain a collection of expectation expressions. This clause plays a role for expectations analogous to the INVARIANT clause on predicates on the state. The associated proof obligations are that every operation, from any legitimate state (i.e. any state that meets the invariant), must not decrease any of the expectations.

Each expectation is of the form $e \Rrightarrow V$, meaning that the expected value of V is always at least the value of e initially. The new proof obligations associated with each such expectation are the following:

P1 Initialisation must establish the lower bound of the invariant:

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

P2 Each operation must not decrease the expected value of V:

$$V \Rrightarrow [Op]\, V$$

In this paper we will use expectations of the form V. This is an abbreviation for $0 \Rrightarrow V$. Observe that this still gives rise to a non-trivial proof obligation $P1$, that V is non-negative on initialisation.

3 The Tank

The system we aim to model is a tank being filled with a liquid. The liquid flows into the tank through a pipe. We wish to track the volume of liquid in the tank. This is illustrated in Figure 2.

The tank can be modelled using the machine given in Figure 3 [2]. This

[2] An explanation of the ascii form of pGSL used in Figure 3 and elsewhere in this paper is

```
MACHINE          Tank
CONSTANTS        minflow, maxflow
PROPERTIES       minflow : REAL & maxflow : REAL
                   & minflow > 0
                   & maxflow >= minflow
VARIABLES        flow, volume
INVARIANT        flow : REAL & volume : REAL
INITIALISATION   volume := 0 || flow :: [minflow,maxflow]
OPERATIONS
  tock = flow :: [minflow,maxflow] || volume := volume + flow
END
```

Fig. 3. The AMN description of the tank system

describes a model of the real tank, and will therefore be included in the specifications we will give, so that we can relate the state of the monitoring system to the real state of the tank.

Here we assume that in one time unit (as represented by *tock*), the volume of liquid increases by the value of *flow*. The value of *flow* can itself be any value between *minflow* and *maxflow*, and can change on every time step.

An interval of real numbers between l and h is denoted $[l, h]$. The interval $[x + l, x + h]$ is abbreviated $x + [l, h]$.

4 A monitoring system

4.1 The first simple system

4.1.1 Specification

We wish to produce a software system that tracks the volume of liquid in the tank to some level of accuracy. The system we require can be specified using the probabilistic B machine *VolumeTracker*1 of Figure 4. (The expectation makes use of values of A and B that will be given later.) For this first example, we take a simple approach where a single *poll* operation updates both the tank and the monitoring system state at the same time. Later in the paper we will consider the separation of system updates from tank updates.

Our first specification, *VolumeTracker*1, requires that a state update is perfectly accurate at least 99% of the time. Otherwise (i.e. up to 1% of the time) it can be completely arbitrary over the range of possible readings $[minflow, maxflow]$.

given in Appendix A

```
MACHINE         VolumeTracker1
INCLUDES        Tank
VARIABLES       rvolume
INVARIANT       rvolume : REAL
                & rvolume * (minflow/maxflow) <= volume
                & volume <= rvolume *  (maxflow / minflow)
EXPECTATION     E1: rvolume - A * volume,
                E2: B * volume - rvolume
INITIALISATION rvolume := 0
OPERATIONS
  poll = T: tock
         || V1a: (rvolume := rvolume+flow
                      0.99 (+)
             V1b: rvolume :: rvolume+[minflow,maxflow] )
END
```

Fig. 4. The *VolumeTracker*1 machine

The system maintains a single state variable *rvolume*, which contains the value the system has for the volume of liquid in the tank. Thus our specification will be concerned with the relationship between *rvolume* and the actual volume *volume*.

It is natural to have two expectations to provide a range on what the expected value for *volume* can be, given a particular value for the expected value of *rvolume*. Because *rvolume* and *volume* are increased on each step with some value from a fixed range of possible values, we consider expectations as linear combinations of *rvolume* and *volume*. Thus they would be of the form:

E1: $rvolume - A \times volume$

E2: $B \times volume - rvolume$

These must both be non-negative, so we can deduce for the expected values that

$$rvolume/B \leqslant volume \leqslant rvolume/A$$

Thus given an expected value for *rvolume* we have a range for the expected value of *volume*. The required degree of accuracy as given by A and B will naturally emerge as part of the specification.

Since both $E1$ and $E2$ must be greater than 0, and non-decreasing on every occurrence of *poll*, we obtain some constraints on the possibilities for A and B.

Observe that any absolute restrictions on the relationship between *volume* and *rvolume* will appear in the invariant. In particular, the lower and upper bounds on *volume* for any given value of *rvolume* are given by the following inequalities:

$$rvolume \times (minflow/maxflow) \leqslant volume \leqslant rvolume \times (maxflow/minflow)$$

This will always be true, so it is included in the invariant. However, it does not provide a very tight relationship between *volume* and *rvolume*.

4.1.2 Deriving A and B

For *VolumeTracker*1 to meet its proof obligations, we require that the expectations will never decrease on any call of the operation *poll*, from any state.

We can carry out some calculations to derive conditions for A and B to achieve this. We require that $E1 \Rrightarrow [poll]E1$ and $E2 \Rrightarrow [poll]E2$. Thus we require that for any *flow*, *volume*, and *rvolume*, we must have that $([poll]E1) - E1 \geqslant 0$ and $([poll]E2) - E2 \geqslant 0$.

We calculate the requirement on A from the requirement on $E1$:

$$\begin{aligned}
([poll]E1) - E1 &= ([T \mid\mid (V1a \;{}_{0.99}\!\oplus\; V1b)]E1) - E1 \\
&= ([(T \mid\mid V1a) \;{}_{0.99}\!\oplus\; (T \mid\mid V1b))]E1) - E1 \\
&= (0.99 \times [T \mid\mid V1a]E1 + 0.01 \times [T \mid\mid V1b]E1) - E1 \\
(*) &= (0.99 \times (rvolume + flow - A(volume + flow)) \\
&\qquad\quad +0.01 \times (rvolume + minflow - A(volume + flow))) \\
&\qquad\qquad -(rvolume - A.volume) \\
&= 0.99 \times (flow - A \times flow) + 0.01(minflow - A \times flow) \\
&= (0.99 - A) \times flow + 0.01 \times minflow
\end{aligned}$$

Since this must be non-negative everywhere (i.e. for all possible values of flow), we obtain that

$$A \leqslant 0.99 + 0.01(minflow/flow)$$

for any value of *flow*. The bound takes its minimal value when *flow* is *maxflow*, so we obtain that

$$A \leqslant 0.99 + 0.01(minflow/maxflow)$$

Thus the closer to 1 the ratio between *minflow* and *maxflow*, the closer A can be to 1 and the more accurate the upper bound on the expected value for *volume* for any given expectation on *rvolume*. However, note that A can always be at least 0.99.

For B we perform the following calculation:

$$\begin{aligned}([poll]E2) - E2 &= ([T \mid\mid (V1a \ {}_{0.99}\oplus \ V1b)]E2) - E2 \\ &= ([(T \mid\mid V1a) \ {}_{0.99}\oplus \ (T \mid\mid V1b))]E2) - E2 \\ &= (0.99 \times [T \mid\mid V1a]E2 + 0.01 \times [T \mid\mid V1b]E2) - E2 \\ (**) &= (0.99 \times (B(volume + flow) - (rvolume + flow)) \\ &\qquad +0.01 \times (B(volume + flow) - (rvolume + maxflow))) \\ &\qquad\quad -(B.volume - rvolume) \\ &= 0.99 \times (B.flow - flow) + 0.01(B.flow - maxflow) \\ &= B \times flow - 0.99 \times flow - 0.01 \times maxflow\end{aligned}$$

We require that this is non-negative for any value of $flow$. Thus $B \geqslant 0.99 + 0.01(maxflow/flow)$ for any value of $flow$. The largest value for the expression (i.e. the largest lower bound for B) is given when $flow = minflow$, and we obtain

$$B \geqslant 0.99 + 0.01(maxflow/minflow)$$

Observe lines (*) and (**) concerning the evaluation of $[T \mid\mid V1b]$ with respect to an expectation. Since $V1b$ is nondeterministic in the assignment to $rvolume$, the minimum expectation over all possible assignments to $rvolume$ must be taken. In $E1$, $rvolume$ is positive, so the smallest possible value of $rvolume$ is used in the calculation of the pre-expectation of $E1$. In $E2$ $rvolume$ is negative so the largest possible value of $rvolume$ is used in the calculation of the pre-expectation of $E2$. This means that however the nondeterminism is later resolved, the expectation will be at least the value calculated. Expectations should always be non-decreasing, so demonic nondeterminism always considers the worst case with respect to increases.

4.1.3 Example

As an illustration, we shall consider some concrete numbers: if $minflow = 100$ and $maxflow = 400$, then we obtain $A \leqslant 0.9925$ and $B \geqslant 10.03$. Thus we know that

$$(100/103) \times rvolume \leqslant volume \leqslant rvolume \times (400/397)$$

This implies for example that

$$0.97 \times rvolume \leqslant volume \leqslant 10.03 \times rvolume$$

so if we have a requirement for 97% accuracy, this will be met.

```
MACHINE Sensor1b
SEES Tank
OPERATIONS
sf, st <--  poll1b =
S1bl:           sf := flow || st := ok
                   >=0.9 (+)
S1br:           sf :: [minflow,maxflow] || st := broken

END
```

Fig. 5. A *Sensor* machine

However, if our requirement is for 99% accuracy, this will not be met. The description cannot ensure that $0.99 \times rvolume \leqslant volume$. This is because an incorrect reading, that could occur with probability 0.01, could be wrong by a factor of 4, leading to a large increase of *rvolume* over the real value of *volume*. The level of accuracy is concerned not only with the probability of correct readings, but also with the amount by which a flawed reading could be out.

To ensure 99% accuracy we would either have to reduce the ratio between *minflow* and *maxflow* (so bad readings cannot be so wildly out), or decrease the probability of a bad reading. Observe that these alterations are concerned only with the specification machine. This machine gives the probability of an accurate reading that is required for ensuring the expectations.

4.1.4 Implementation

Our first implementation of $VolumeTracker1$ will make use of two sensors, which provide readings for the flow, and also give diagnostic information stating whether they are broken or not. We will firstly consider sensors which can fail on any particular reading independently of any other reading. We will consider sensors which have a reliability of at least 90%. We will need to make use of two of these, $Sensor1a$ and $Sensor1b$ to give readings to 99% accuracy. $Sensor1b$ is given in Figure 5, and $Sensora1$ is entirely similar.

We propose an implementation $VolumeTracker1I$ of $VolumeTracker1$ which uses two sensors in order to obtain a more reliable reading of the flow. This is given in Figure 6, and makes use of the Context machine of Figure 7.

Observe that the implementation contains its own variable *rvolume*. To avoid complicating this example with imported state, we relax the normal restriction that implementation machines cannot have their own state.

We need to prove that the *poll* operation in the implementation is a refinement of the *poll* operation in the specification. This can be done by ma-

```
IMPLEMENTATION   VolumeTracker1I
REFINES           VolumeTracker1
IMPORTS   Tank, Sensor1a, Sensor1b, Context
VARIABLES   rvolume
INVARIANT   rvolume : REAL
INITIALISATION   rvolume := 0
OPERATIONS
poll = VAR v1, v2, st1, st2, rflow
       IN
P1a:     v1,st1 <-- poll1a;
P1b:     v2,st2 <-- poll1b;
F:       rflow <-- flow(v1,st1,v2,st2);
R:       rvolume := rvolume + rflow;
T:       tock
       END
END
```

Fig. 6. The implementation *VolumeTracker1I*

```
MACHINE      Context
OPERATIONS
  ff <-- flow(v1,st1,v2,st2) =
    PRE   v1 : REAL & v2 : REAL
          & st1 : STATUS & st2 : STATUS
    THEN
F:      IF st1 = broken & st2 = broken THEN ff :: [minflow,maxflow]
        ELSIF st1 = broken & st2 = ok THEN ff := v2
        ELSIF st1 = ok & st2 = broken THEN ff := v1
        ELSIF st1 = ok & st2 = ok THEN ff := (v1+v2)/2
    END
END
```

Fig. 7. The AMN description of flow calculation

nipulating the probabilistic choices using the laws of [3] given in Section 2.2.

The *poll* operation in *VolumeTracker1I* of Figure 6 is of the particular form $P1a;\ P1b;\ F;\ R;\ T$, where the variables $v1, v2, st1, st2, rflow$ are all local. We show that this operation is equivalent to *poll* given in the specification machine *VolumeTracker1*, as follows:

$$
\begin{aligned}
& P1a;\ P1b;\ F;\ R;\ T \\
& = \{\text{expanding } P1a \text{ and } P1b\} \\
& \quad (S1al \ {}_{\geqslant 0.9}{\oplus}\ \ S1ar);
\end{aligned}
$$

$$
\begin{aligned}
& (S1bl \;{}_{\geqslant 0.9}\!\oplus\; S1br);\; F;\; R;\; T \\
= & \{\text{Law 13}\} \\
& S1al;\; (S1bl \;{}_{\geqslant 0.9}\!\oplus\; S1br);\; F;\; R;\; T \\
& {}_{\geqslant 0.9}\!\oplus \\
& S1ar;\; (S1bl \;{}_{\geqslant 0.9}\!\oplus\; S1br);\; F;\; R;\; T \\
= & \{\text{Law 13}\} \\
& (S1al;\; S1bl;\; F;\; R;\; T \;{}_{\geqslant 0.9}\!\oplus\; S1al;\; S1br;\; F;\; R;\; T) \\
& {}_{\geqslant 0.9}\!\oplus \\
& (S1ar;\; S1bl;\; F;\; R;\; T \;{}_{\geqslant 0.9}\!\oplus\; S1ar;\; S1br;\; F;\; R;\; T) \\
= & \{\text{standard program algebra in each branch; removal of local variables}\} \\
& (V1a \parallel T \;{}_{\geqslant 0.9}\!\oplus\; V1a \parallel T) \;{}_{\geqslant 0.9}\!\oplus\; (V1a \parallel T \;{}_{\geqslant 0.9}\!\oplus\; V1b \parallel T) \\
= & \{\text{idempotence of } {}_{\geqslant p}\!\oplus \text{ on left-hand argument}\} \\
& V1a \parallel T \;{}_{\geqslant 0.9}\!\oplus\; (V1a \parallel T \;{}_{\geqslant 0.9}\!\oplus\; V1b \parallel T) \\
= & \{\text{Law 24}\} \\
& (V1a \parallel T \;{}_{\geqslant 0.99}\!\oplus\; V1b \parallel T) \\
= & \{\text{Law A, since } V1b \sqsubseteq V1a\} \\
& (V1a \parallel T \;{}_{0.99}\!\oplus\; V1b \parallel T)
\end{aligned}
$$

Thus we arrive at the operation *poll* given in the machine *VolumeTracker*1. This demonstrates that *VolumeTracker*1*I* indeed provides an implementation of *VolumeTracker*1.

4.1.5 Summary

This first example has illustrated several points:

- The expected value of the machine expectation expression should be non-decreasing on every occurrence of the operation.
- However, the actual value of the machine expectation expression can decrease on some operation calls (provided its expected value does not).
- Expectations can be used to express a relationship between the expected values of state variables, in our case providing a range for the expected value of *volume* in terms of the expected value of *rvolume*. This is checked as part of machine consistency, and is independent of any particular implementation.
- The accuracy of the approximation *rvolume* to the tank value *volume* depends not only on the probability of an incorrect reading, but also on the ratio between *minflow* and *maxflow*, since this affects the maximum possible error in *rvolume*.
- Probabilistic operations can be implemented using combinations of proba-

```
MACHINE         VolumeTracker2
INCLUDES        Tank
CONSTANTS       lowerror, higherror
PROPERTIES      lowerror : REAL & lowerror <= 0
                & higherror : REAL & higherror >= 0
VARIABLES       rvolume
INVARIANT       rvolume : REAL
EXPECTATION     E1: rvolume - A * volume,
                E2: B * volume - rvolume
INITIALISATION rvolume := 0
OPERATIONS
 poll =
  T: tock
  || V2a: rvolume :: rvolume+flow+[lowerror,higherror]
              0.99 (+)
     V2b: rvolume :: rvolume+[minflow+lowerror,maxflow+higherror]
END
```

Fig. 8. The AMN description of the second monitoring system

bilistic components (sensors) in the way we would expect. Such implementations need only be checked for refinement against the machine descriptions of the operations. The machine consistency checks ensure that the machine operations provide the overall requirements on the expectations.

4.2 Introducing error margins

4.2.1 Specification

In the previous example, correct readings of *flow* were exactly accurate. We now allow for a margin of error in readings of *flow*. Specifically, the error can be any value in the range $[lowerror, higherror]$. Typically the possibility of no error at all should be within the range, so *lowerror* will be negative and *higherror* will be positive. The revised machine is given in Figure 8.

The calculation of appropriate A and B follows the same pattern as shown previously in Section 4.1.2. Now two sources of nondeterminism must be taken into account: the reading of the sensors in $V2a$ (which can be most pessimistic with regard to $E1$ when *flow* is low) and the arbitrary reading in $V2b$ (which can be most pessimistic for $E1$ when *flow* is high). This combination of considerations (recall *lowerror* is negative, so $A \leqslant 1$) means that A is bounded above by both of the following values:

$1 + (lowerror/minflow)$

and

$$0.99 + (\mathit{lowerror}/\mathit{maxflow}) + 0.01(\mathit{minflow}/\mathit{maxflow})$$

For example, if $\mathit{minflow} = 100$, $\mathit{maxflow} = 400$, and $\mathit{lowerror} = -10$, then the first value is lower, and we obtain $A = 0.9$. On the other hand, if $\mathit{lowerror} = -0.1$, then the second value is lower and we obtain $A = 0.9915$. In the first case the possible error in any reading of the flow is 10% of *minflow*, so the worst case occurs when the flow is *minflow* and $\mathit{minflow} + \mathit{lowerror}$ is added to *rvolume*: the resulting *rvolume* could be 10% out. On the other hand, in the second case the error in the flow can be at most 0.1%, so the error that can be introduced by $V2b$ (1% of the time) dominates, and the worst case occurs when the flow is *maxflow* and *rvolume* is only incremented by $\mathit{lowerror} + \mathit{minflow}$.

Similar considerations for the expectation $E2$ yield that the value obtained for B is the maximum of the following two values, the first for the case where $\mathit{flow} = \mathit{maxflow}$ and the second when $\mathit{flow} = \mathit{minflow}$.

$$1 + (\mathit{higherror}/\mathit{maxflow})$$

and

$$0.99 + (\mathit{higherror}/\mathit{minflow}) + 0.01(\mathit{maxflow}/\mathit{minflow})$$

In this case, the second value will always be higher, and hence will give the appropriate value for B, since $\mathit{maxflow}/\mathit{minflow} \geqslant 1$, and $\mathit{higherror}/\mathit{minflow} \geqslant \mathit{higherror}/\mathit{maxflow}$. This informs us that the worst case always occurs with a flow of *minflow*, and an incorrect reading of $\mathit{maxflow} + \mathit{higherror}$. This is worse than the worst outcome that can be obtained with a flow of *maxflow*, as far as ensuring that $E2$ does not decrease is concerned.

4.2.2 Implementation: sensors

The error is likely to have been included in the specification because the sensors introduce some error. We can include these errors within the sensor descriptions, resulting in a new version of sensor description. For example, in $\mathit{Sensor}2b$ we will take the error range to be $[le2b, he2b]$. The resulting sensor is given in Figure 9.

The implementation $\mathit{VolumeTracker}2I$ will be the same as $\mathit{VolumeTracker}1I$, except that it now importing $\mathit{Sensor}2a$ (with error range $[le2a, he2a]$) and $\mathit{Sensor}2b$, instead of the original sensors. It is given for reference in Figure B.1 of Appendix B.

```
MACHINE Sensor2b
SEES Tank
CONSTANTS        le2b, he2b
PROPERTIES       le2b : REAL & le2b <= 0
                 & he2b : REAL & re2b >= 0
OPERATIONS
sf, st <--  poll2b =
S2bl:        sf :: flow+[le2b,he2b] || st := ok
                 >=0.9 (+)
S2br:        sf :: [minflow+le2b,maxflow+he2b] || st := broken

END
```

Fig. 9. The machine *Sensor2b*

Observe that in this scenario two sensors working correctly might not agree on their readings. In this case the context machine specifies that the average of the two readings should be taken.

The machine *VolumeTracker2I* provides an implementation of *poll*, provided that the following hold: that $[le2a, he2a] \subseteq [lowerror, higherror]$ and $[le2b, he2b] \subseteq [lowerror, higherror]$. In other words, that the error ranges for each sensor are within those given in *VolumeTracker2* for the overall combination. The proof of this is given in Appendix B.

4.2.3 Summary

This second example illustrates several points:

- We can specify error ranges for readings of *flow*.
- Such ranges have an impact on the expectations that will be non-decreasing on operations: the nondeterminism in the state updates means that the relationship between *rvolume* and *volume* will be weaker.
- The particular relationships that can be guaranteed between *volume* and *rvolume* depend on the error ranges of readings and also on the the ratio of *maxflow* to *minflow*. Each of these dominates in some cases.
- The flow readings can be implemented by sensors whose errors are within the specified range.

4.3 Removing sensor diagnostics

We now consider the situation where the sensors do not provide explicit status information. In this case the only way faulty readings can be identified is by

```
MACHINE Sensor3c
SEES Tank
CONSTANTS       le3c, he3c
PROPERTIES      le3c : REAL & le3c <= 0
                & he3c : REAL & re3c >= 0
OPERATIONS
sc <--  poll3c =
            sc :: flow+[le3c,he3c]
                    >=0.9 (+)
            sc :: [minflow+le3c,maxflow+he3c]

END
```

Fig. 10. A sensor without diagnostics

comparison with other readings.

In this example we will work from the sensors to the specification: we will derive the specification that the combination of sensors delivers.

4.3.1 Implementation: sensor

A sensor without diagnostic information about its status is given in Figure 10. It provides only a flow reading, without any information about its state.

To be tolerant to one faulty reading, we need three sensors: $Sensor3a$, $Sensor3b$, and $Sensor3c$. By taking the median value of the three readings we obtain an accurate reading, provided no more than one of them goes wrong. This suggests the implementation given in Figure 11. We still assume a 90% reliability on the reading of any individual sensor.

4.3.2 Specification

In fact here $VolumeTracker3I$ is a refinement of $VolumeTracker3$ given in Figure 12, provided all of the sensor errors are within the error given in $VolumeTracker3$, e.g. $[le3, he3] \subseteq [lowerror, higherror]$.

For $VolumeTracker3$, carrying out the standard calculations on preservation of $E1$, we find that the best (highest) value we can obtain for A, which enables the expectation $E1$ to be preserved, is the minimum of

$$1 + (lowerror/minflow)$$

and

$$0.972 + 0.028(minflow/maxflow) + lowerror/maxflow$$

```
IMPLEMENTATION   VolumeTrackerI3
REFINES           VolumeTracker3
IMPORTS   Tank, Sensora3, Sensor3b, Sensor3c
VARIABLES   rvolume
INVARIANT   rvolume : REAL
INITIALISATION   rvolume := 0
OPERATIONS
poll = VAR v1, v2, v3
       IN
         v1 <-- poll3a;
         v2 <-- poll3b;
         v3 <-- poll3c;
         rflow := median(v1,v2,v3);
         rvolume := rvolume + rflow;
         tock
       END
END
```

Fig. 11. The implementation *VolumeTrackerI3*

```
MACHINE         VolumeTracker3
INCLUDES        Tank
PROPERTIES      lowerror : REAL & lowerror <= 0
                & higherror : REAL & higherror >= 0
VARIABLES       rvolume
INVARIANT       rvolume : REAL
EXPECTATION     E1: rvolume - A * volume,
                E2: B * volume - rvolume
INITIALISATION rvolume := 0
OPERATIONS
 poll =
   tock
   || S3a: rvolume := rvolume+flow+[lowerror,higherror]
              0.972 (+)
      S3b:  rvolume :: rvolume+[minflow+lowerror,maxflow+higherror]
END
```

Fig. 12. The third monitoring system specification

Similarly, the best (lowest) value we can obtain for B is the maximum of

$1 + (higherror/maxflow)$

and

$0.972 + 0.028(\mathit{maxflow}/\mathit{minflow}) + (\mathit{higherror}/\mathit{minflow})$

The second of these will always be the maximum, since $\mathit{maxflow} \geqslant \mathit{minflow}$. The situation is similar to the previous example considered in Section 4.2.2, but with a probability of an incorrect reading now at 0.028 rather than 0.01. Thus the expectations on the relationship between *rvolume* and *volume* are correspondingly weaker, since more weighting is given to the ratio between *maxflow* and *minflow*.

For example, consider the situation where we have the following: $\mathit{maxflow} = 400$, $\mathit{minflow} = 100$, $\mathit{higherror} = 1$, $\mathit{lowerror} = -1$.

Since the expectation $E1 = \mathit{rvolume} - A \times \mathit{volume}$ must not decrease, whatever the value of *flow*, we have two extremes to consider:

- If $\mathit{flow} = \mathit{minflow}$, then *volume* is incremented by *minflow*, and the least that *rvolume* can be incremented by is $\mathit{minflow} + \mathit{lowerror}$. Thus in this case we obtain a possible value of $A = 0.99$.
- If $\mathit{flow} = \mathit{maxflow}$, then *volume* is increased by *maxflow*, and the least that *rvolume* can be incremented by is $\mathit{minflow} + \mathit{lowerror}$ if at least two sensors go wrong (which can happen with probability 0.028), otherwise $\mathit{maxflow} + \mathit{lowerror}$. Thus the most pessimistic expectation gives a possible value of $A = 0.9765$. Here the ratio between *maxflow* and *minflow* is more significant than the ratio between *minflow* and *lowerror* in contributing to the amount by which *rvolume* can be down, and we obtain a value of 0.9765 for A.

We also require that the expectation $E2 = \mathit{volume} - B \times \mathit{rvolume}$ must not decrease. Here we are concerned with the proportion by which *volume* can exceed *rvolume*, and the worst case always occurs when $\mathit{flow} = \mathit{minflow}$. In this case, the reading might at worst be $\mathit{maxflow} + \mathit{higherror}$ (with probability 0.028) and $\mathit{minflow} + \mathit{higherror}$ otherwise. This yields a value for B of at least 10.085 if the expectation of $E2$ is not to decrease. This is a margin of error of 8.5%.

4.3.3 Summary

This version of the tank monitoring system has considered a version of sensor which does not provide feedback on its status. Thus a sensor's incorrect reading can only be discovered by comparing it with other sensors. We considered an implementation which uses three sensors in such a way that if at most one has failed then an accurate reading is obtained. We found that if each sensor has at least 90% reliability, then the combination has at least 97.2% reliability in terms of providing an accurate reading. This allowed us to construct the specification that was guaranteed by the implementation. This in turn enables the relationship between the expected values of *volume* and *rvolume* to be established.

5 Discussion

The case study in this paper has shown how probabilistic B can be applied to specify and refine a system which naturally includes both probabilistic and nondeterministic behaviour, and has highlighted a number of issues that can arise in this process.

We considered a progression of scenarios. In the first scenario, we considered the simple case where sensor readings are either perfectly accurate, or completely arbitrary, with the sensors indicating whether they are working correctly or not. This enabled a value for the accuracy of the system's value *rvolume* to be given, given in terms of the range of possible flows. Essentially the accuracy is calculated by allowing for the worst case of nondeterminism, in accordance with the demonic approach to nondeterminism reflected in the semantics of the language. We obtained the expected result that the larger the ratio between the maximum and minimum flow, the less accurate the value we could expect.

In the second scenario, we allowed some error range on the values read even when the sensors were working correctly. This additional nondeterminism also entered into the calculation to determine the level of accuracy of *rvolume*, and again we saw that the wider the range of possibilities, for flow readings, and for the possible flows, the lower the level of accuracy for the system's record of the volume of liquid.

In the third scenario, the sensors no longer provided a direct indication of whether they were giving a correct reading or not, so it was necessary to use three sensors and compare readings to deduce which values are most likely correct. In this example we worked from the implementation to the specification, firstly obtaining the reliability provided by the combination of sensors, and then calculating the level of accuracy that the system could deliver.

All three of these scenarios were modelled using a machine which had only a single operation, which synchronised updates of the real tank and updates of the monitoring system.

Although the case study was of a simple system, this paper has only explored some of the interesting kinds of behaviour that can arise in such systems, and many other scenarios remain ready to be explored. For example, we might wish to model sensors that take some time to be repaired once they break. Such modelling would most likely require some auxiliary variable to track the time left until the sensor is working correctly again, and the best way of modelling such a system in probabilistic B is far from clear.

Acknowledgement

We are grateful to Neil Evans, Carroll Morgan, and Annabelle McIver for comments and discussions on this work.

This research was initiated during Ken Robinson's and Thai Son Hoang's visit to Royal Holloway, University of London, in July 2003, and thanks are due to EPSRC for providing funds under grant GR96859/01 to support this visit.

References

[1] J-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] T.S. Hoang, Z. Jin, K. Robinson, A. McIver, and C. Morgan. Probabilistic invariants for probabilistic machines. In *ZB2003: Third International Conference of B and Z Users*, number 2651 in LNCS. Springer, 2003.

[3] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2004.

[4] A. McIver, C. Morgan, and T.S. Hoang. Probabilistic termination in B. In *ZB2003: Third International Conference of B and Z Users*, number 2651 in LNCS. Springer, 2003.

[5] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.

[6] C. Morgan. The generalised substitution language extended to probabilistic programs. In *B'98: the 2nd International B Conference*, number 1393 in LNCS. Springer, 1998.

[7] S. Schneider. *The B-Method: an Introduction*. Palgrave, 2001.

A Machine Readable pGSL

This table gives the ascii form of statements in pGSL, used in the AMN descriptions presented in this paper. For a fuller account of machine-readable AMN, see [1,7].

```
IMPLEMENTATION   VolumeTracker2I
REFINES           VolumeTracker2
IMPORTS   Tank, Sensor2a, Sensor2b, Context
VARIABLES   rvolume
INVARIANT   rvolume : REAL
INITIALISATION   rvolume := 0
OPERATIONS
poll = VAR v1, v2, st1, st2, rflow
       IN
P2a:     v1,st1 <-- poll2a;
P2b:     v2,st2 <-- poll2b;
F:       rflow <-- flow(v1,st1,v2,st2);
R:       rvolume := rvolume + rflow;
T:       tock
       END
END
```

Fig. B.1. The implementation *VolumeTracker2I*

$x := E$	`x:=E`
$x :\in S$	`x :: S`
$x, y := E, F$	`x,y := E,F`
$pre \mid prog$	`pre | prog`
$prog_1 [] prog_2$	`prog1 [] prog2`
$pre \Longrightarrow prog$	`pre ==> prog`
$skip$	`skip`
$prog_1 \;{}_p\!\oplus\; prog_2$	`prog1 p (+) prog2`
$prog_1 \;{}_{\geqslant p}\!\oplus\; prog_2$	`prog1 >=p (+) prog2`
$@y.pred \Longrightarrow prog$	`@ y . pred ==> prog`

B Verifying the implementation of *poll* in *VolumeTracker2I*

The *poll* operation in *VolumeTracker2I* is of the form $P2a;\ P2b;\ F;\ R;\ T$, where $v1, v2, st1, st2, rflow$ are all local variables. We show that this operation is equivalent to *poll* given in the specification machine *VolumeTracker2*, as follows:

$$P2a;\ P2b;\ F;\ R;\ T$$

$$
\begin{aligned}
&= \{\text{expanding } P2a \text{ and } P2b\}\\
&\quad (S2al \;{}_{\geqslant 0.9}\!\oplus\; S2ar);\; (S2bl \;{}_{\geqslant 0.9}\!\oplus\; S2br);\; F;\; R;\; T\\
&= \{\text{Law 13, twice }\}\\
&\quad (S2al;\; S2bl;\; F;\; R;\; T \;{}_{\geqslant 0.9}\!\oplus\; S2al;\; S2br;\; F;\; R;\; T)\\
&\quad\; {}_{\geqslant 0.9}\!\oplus\\
&\quad (S2ar;\; S2bl;\; F;\; R;\; T \;{}_{\geqslant 0.9}\!\oplus\; S2ar;\; S2br;\; F;\; R;\; T)\\
&= \{\text{standard program algebra in each branch; removal of local variables}\}\\
&\quad (\mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [(\mathit{le}2a + \mathit{le}2b)/2, (\mathit{he}2a + \mathit{he}2b)/2];\; T\\
&\qquad {}_{\geqslant 0.9}\!\oplus\; \mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [\mathit{le}2a, \mathit{he}2a];\; T)\\
&\quad\; {}_{\geqslant 0.9}\!\oplus\\
&\quad (\mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [\mathit{le}2b, \mathit{he}2b];\; T\\
&\qquad {}_{\geqslant 0.9}\!\oplus\; \mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [\mathit{minflow}, \mathit{maxflow}];\; T)\\
&\Lleftarrow \{\text{expanding the ranges of the nondeterministic choices,}\\
&\qquad \text{provided } \mathit{lowerror} \leqslant \mathit{le}2a,\; \mathit{lowerror} \leqslant \mathit{le}2b,\\
&\qquad\qquad\qquad \mathit{he}2a \leqslant \mathit{higherror},\; \mathit{he}2b \leqslant \mathit{higherror}\}\\
&\quad (\mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [\mathit{lowerror}, \mathit{higherror}];\; T\\
&\qquad {}_{\geqslant 0.9}\!\oplus\; \mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [\mathit{lowerror}, \mathit{higherror}];\; T)\\
&\quad\; {}_{\geqslant 0.9}\!\oplus\\
&\qquad (\mathit{rvolume} :: \mathit{rvolume} + \mathit{flow} + [\mathit{lowerror}, \mathit{higherror}];\; T\\
&\quad\; {}_{\geqslant 0.9}\!\oplus\; \mathit{rvolume} :: \mathit{rvolume} + \mathit{flow}\\
&\qquad\qquad\qquad + [\mathit{minflow} + \mathit{lowerror}, \mathit{maxflow} + \mathit{higherror}];\; T)\\
&= \{\text{Laws 13 and 24}\}\\
&\quad V2a;\; T \;{}_{\geqslant 0.99}\!\oplus\; V2b;\; T\\
&= \{\text{Laws 13 and A, since } V2b \sqsubseteq V2a;\ T \text{ independent of } V2a \text{ and } V2b\ \}\\
&\quad (V2a \parallel T \;{}_{0.99}\!\oplus\; V2b \parallel T)
\end{aligned}
$$

Thus we arrive at the operation *poll* given in the machine *VolumeTracker*2. This demonstrates that *VolumeTracker*2*I* indeed provides an implementation of *VolumeTracker*2.