Tank monitoring: a pAMN case study

Steve Schneider, Thai Son Hoang,
Ken Robinson, and Helen Treharne

Technical Report

CSD-TR-03-17

March 4, 2004

Royal Holloway
University of London

Department of Computer Science
Egham, Surrey TW20 0EX, England

Contents

1 Introduction

The B-Method [Abr96] 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 [HJR+03], 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*. 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 [Mor98, MM04, MMH03] 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 [Mor98] is given in Figure 1.

In this paper we will use a derived operator (also given in [MM04]) for expressing a minimum probability on a choice. We define

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

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 $correct \ {}_{\geqslant .9}\!\oplus\ incorrect$. This would be refined by a component that behaves correctly at least 95% of the time, for example.

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 $0 \times \infty \widehat{=} 0$
$prog_1 [] prog_2$	$[prog_1]exp \min [prog_2]exp$
$[pre \Longrightarrow prog]exp$	$1/\langle pre \rangle \times [prog]exp$, where $\infty \times 0 \widehat{=} \infty$
$[skip]exp$	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.

Figure 1 *pGSL*—the probabilistic Generalised Substitution Language [Mor98]

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 [MM04]. The following laws from that Appendix will be used in this paper:

Law 13:

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

Law 24:

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

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

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

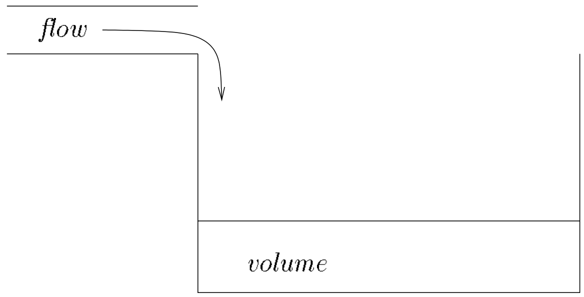

Figure 2 The tank system

2.3 Probabilistic B

There are two aspects to the introduction of probabilistic behaviour into a B machine as proposed in [HJR+03]. 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. This 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*.

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

```
MACHINE         Tank
CONSTANTS       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
```

Figure 3 The AMN description of the tank system

```
MACHINE        VolumeTracker1
INCLUDES       Tank
VARIABLES      rvolume
INVARIANT      rvolume : REAL
               & rvolume \times  (minflow/maxflow) <= volume
               & volume <= rvolume \times  (maxflow / minflow)
EXPECTATION    E1: rvolume - A\times volume,
               E2: B\times  volume - rvolume
INITIALISATION rvolume := 0
OPERATIONS
  poll = T: tock
       || S1: (rvolume := rvolume+flow
            .99 (+)
           S2: rvolume :: rvolume+[minflow,maxflow] )
END
```

Figure 4 The *VolumeTracker*1 machine

4 A monitoring system

4.1 The first simple system

4.1.1 Specification

We want 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 either perfectly accurate (at least 99% of the time), or else completely arbitrary over the range of possible readings [*minflow*, *maxflow*].

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 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.

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 $E1$:

$$\begin{array}{rcl}
([poll]E1) - E1 & = & ([T \parallel (S1 \;{}_{.99}\!\oplus\; S2)]E1) - E1 \\
& = & ([(T \parallel S1) \;{}_{.99}\!\oplus\; (T \parallel S2))]E1) - E1 \\
& = & (.99 \times [T \parallel S1]E1 + .01 \times [T \parallel S2]E1) - E1 \\
(*) & = & (.99 \times (rvolume + flow - A(volume + flow)) \\
& & \quad +.01 \times (rvolume + minflow - A(volume + flow))) \\
& & \qquad -(rvolume - Avolume) \\
& = & .99 \times (flow - A \times flow) + .01(minflow - A \times flow) \\
& = & (.99 - A) \times flow + .01 \times minflow
\end{array}$$

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

$$A \leqslant .99 + .01(minflow/maxflow)$$

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:

$$\begin{array}{rcl}
([poll]E2) - E2 & = & ([T \mid\mid (S1 \;{}_{.99}\!\oplus\; S2)]E2) - E2 \\
 & = & ([(T \mid\mid S1) \;{}_{.99}\!\oplus\; (T \mid\mid S2))]E2) - E2 \\
 & = & (.99 \times [T \mid\mid S1]E2 + .01 \times [T \mid\mid S2]E2) - E2 \\
(**) & = & (.99 \times (B(volume + flow) - (rvolume + flow)) \\
 & & \quad +.01 \times (B(volume + flow) - (rvolume + maxflow))) \\
 & & \qquad -(B.volume - rvolume) \\
 & = & .99 \times (B.flow - flow) + .01(B.flow - maxflow) \\
 & = & B \times flow - 0.99 \times flow - 0.01 \times maxflow
\end{array}$$

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 $[T \mid\mid S2]$ with respect to an expectation. Since $S2$ 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 $B \geqslant 1.03$. Thus we know that

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

This implies for example that

$$0.97 \times rvolume \leqslant volume \leqslant 1.03 \times rvolume$$

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

However, if we have a requirement 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.

```
MACHINE Sensorb1
SEES Tank
OPERATIONS
sb, stb <--  pollb1 =
Sb1:            sb := flow || stb := ok
                    >=0.9 (+)
Sb2:            sb :: [minflow,maxflow] || stb := broken
        END

END
```

Figure 5 A *Sensor* machine

```
IMPLEMENTATION   VolumeTracker1I
REFINES           VolumeTracker1
IMPORTS   Tank, Sensora1, Sensorb1, Context
VARIABLES   rvolume
INVARIANT   rvolume : REAL
INITIALISATION   rvolume := 0
OPERATIONS
poll = VAR v1, v2, st1, st2, rflow
       IN
Pa:      v1,st1 <-- polla1;
Pb:      v2,st2 <-- pollb1;
F:       rflow <-- flow(v1,st1,v2,st2);
R:       rvolume := rvolume + rflow;
T:       tock
       END
END
```

Figure 6 The implementation *VolumeTracker1I*

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, *Sensora1* and *Sensorb1* to give readings to 99% accuracy. *Sensorb1* 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.

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 manipulating the probabilistic choices using the laws of [MM04] given in Section 2.2.

The *poll* operation in *VolumeTracker1I* is of the form $Pa;\ Pb;\ F;\ R;\ T$, where $v1, v2, st1, st2, rflow$ are all local variables. We show that this operation is equiva-

```
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
```

Figure 7 The AMN description of flow calculation

lent to *poll* given in the specification machine *VolumeTracker*1, as follows:

$$
\begin{array}{ll}
\multicolumn{2}{l}{Pa;\ Pb;\ F;\ R;\ T} \\
= & \{\text{expanding } Pa \text{ and } Pb\} \\
 & (Sa1 \ {}_{\geqslant 0.9}\!\oplus\ Sa2); \\
 & (Sb1 \ {}_{\geqslant 0.9}\!\oplus\ Sb2);\ F;\ R;\ T \\
= & \{\text{Law } 13\} \\
 & Sa1;\ (Sb1 \ {}_{\geqslant 0.9}\!\oplus\ Sb2);\ F;\ R;\ T \\
 & {}_{\geqslant 0.9}\!\oplus \\
 & Sa2;\ (Sb1 \ {}_{\geqslant 0.9}\!\oplus\ Sb2);\ F;\ R;\ T \\
= & \{\text{Law } 13\} \\
 & (Sa1;\ Sb1;\ F;\ R;\ T \ {}_{\geqslant 0.9}\!\oplus\ Sa1;\ Sb2;\ F;\ R;\ T) \\
 & {}_{\geqslant 0.9}\!\oplus \\
 & (Sa2;\ Sb1;\ F;\ R;\ T \ {}_{\geqslant 0.9}\!\oplus\ Sa2;\ Sb2;\ F;\ R;\ T) \\
= & \{\text{standard program algebra in each branch; removal of local variables}\} \\
 & (S1 \parallel T \ {}_{\geqslant 0.9}\!\oplus\ S1 \parallel T) \ {}_{\geqslant 0.9}\!\oplus\ (S1 \parallel T \ {}_{\geqslant 0.9}\!\oplus\ S2 \parallel T) \\
= & \{\text{idempotence of } {}_{\geqslant p}\!\oplus\ \} \\
 & S1 \parallel T \ {}_{\geqslant 0.9}\!\oplus\ (S1 \parallel T \ {}_{\geqslant 0.9}\!\oplus\ S2 \parallel T) \\
= & \{\text{Law } 24\} \\
 & (S1 \parallel T \ {}_{\geqslant 0.99}\!\oplus\ S2 \parallel T) \\
= & \{\text{Law A, since } S2 \sqsubseteq S1\} \\
 & (S1 \parallel T \ {}_{0.99}\!\oplus\ S2 \parallel T)
\end{array}
$$

Thus we arrive at the operation *poll* given in the specification machine *VolumeTracker*1. This demonstrates that *VolumeTrackerI*1 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.

```
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\times volume,
                E2: B\times volume - rvolume
INITIALISATION rvolume := 0
OPERATIONS
  poll = T: tock
       || S1: (rvolume := rvolume+flow+[lowerror,higherror]
             .99 (+)
            S2: rvolume :: rvolume+[minflow+lowerror,maxflow+higherror] )
END
```

Figure 8 The AMN description of the second monitoring system

- ◇ 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 probabilistic 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

We now allow for a margin of error in the addition of *flow* to the current reading of volume *rvolume*. 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, and is given in Appendix A. Now two sources of nondeterminism must be taken into account: the reading of the sensors in $S1$ (which can be most pessimistic with regard to $E1$ when *flow* is low) and the arbitrary reading in $S2$ (which can be most pessimistic for $E1$ when *flow* is high). This combination of considerations results in A taking the minimum of the following two values (recall *lowerror* is negative):

$1 + (\mathit{lowerror}/\mathit{minflow})$

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 the record 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 $S2$ (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 (as far as ensuring that $E2$ does not decrease is concerned) that can be obtained with a flow of *maxflow*.

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 the sensor errors within the description of the sensors, resulting in a new version of sensor description. For example, in $\mathit{Sensorb}2$ we will take the error range to be $[le2, he2]$. The resulting sensor is given in Figure 9.

The implementation $\mathit{VolumeTracker}2I$ will be the same as $\mathit{VolumeTracker}1I$, (though now importing $\mathit{Sensora}2$ and $\mathit{Sensorb}2$ instead of the original sensors). However, observe that two sensors working correctly might not agree on their readings. In this case the context machine specifies the average of the two readings to be taken.

The machine *VolumeTrackerI* provides an implementation of *poll*, provided $[le1, he1] \subseteq [\mathit{lowerror}, \mathit{higherror}]$ and $[le2, he2] \subseteq [\mathit{lowerror}, \mathit{higherror}]$: in other words, that the error ranges for each sensor are within those given in the specification. The proof of this is given in Appendix A.

4.2.3 Summary

This second example illustrates several points:

- We can specify error ranges for readings of *flow*.

```
MACHINE Sensorb2
SEES Tank
CONSTANTS       leb2, heb2
PROPERTIES      leb2 : REAL & leb2 <= 0
                & heb2 : REAL & reb2 >= 0
OPERATIONS
s2, st <--  pollb2 =
          S2a: s2 := flow+[le2,he2] || st := ok
                 >=0.9 (+)
          S2b: s2 :: [minflow+le2,maxflow+he2] || st := broken
        END

END
```

Figure 9 The machine *Sensorb*2

- 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 error ranges 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 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.

To be tolerant to one faulty reading, we need three sensors: *Sensora*3, *Sensorb*3, and *Sensorc*3. 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.

4.3.2 Specification

In fact here *VolumeTracker3I* is a refinement of the specification *VolumeTracker*3 given in Figure 12, provided all of the sensor errors are within the error given in *VolumeTracker*3, e.g. $[le3, he3] \subseteq [lowerror, higherror]$.

For *VolumeTracker*3, carrying out the standard calculations on preservation of $E1$, we find that the best (highest) value we can obtain for A, which enables the

```
MACHINE Sensorb3
SEES Tank
CONSTANTS       le3, he3
PROPERTIES      le3 : REAL & le3 <= 0
                & he3 : REAL & re3 >= 0
OPERATIONS
sb <--  pollb3 =
            sb := flow+[le3,he3]
                    >=0.9 (+)
            sb :: [minflow+le3,maxflow+he3]
        END

END
```

Figure 10 A sensor without diagnostics

```
IMPLEMENTATION   VolumeTrackerI3
REFINES           VolumeTracker3
IMPORTS   Tank, Sensora3, Sensorb3, Sensorc3
VARIABLES   rvolume
INVARIANT   rvolume : REAL
INITIALISATION   rvolume := 0
OPERATIONS
poll = VAR v1, v2, v3
       IN
         v1 <-- polla3;
         v2 <-- pollb3;
         v3 <-- pollc3;
         rflow := median(v1,v2,v3);
R:       rvolume := rvolume + rflow;
         tock
       END
END
```

Figure 11 The implementation *VolumeTrackerI3*

expectation $E1$ to be preserved, is the minimum of

$1 + (lowerror/minflow)$

and

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

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

$1 + (higherror/maxflow)$

and

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

The second of these will always be the maximum, since $maxflow \geqslant minflow$. The situation is similar to the previous example considered in Section 4.2.2, but with

```
MACHINE        VolumeTracker3
INCLUDES       Tank
PROPERTIES     lowerror : REAL & lowerror <= 0
               & higherror : REAL & higherror >= 0
VARIABLES      rvolume
INVARIANT      rvolume : REAL
EXPECTATION    E1: rvolume - A\times volume,
               E2: B\times volume - rvolume
INITIALISATION rvolume := 0
OPERATIONS
  poll = T: tock
        || S1: (rvolume := rvolume+flow+[lowerror,higherror]
             .972 (+)
           S2: rvolume :: rvolume+[minflow+lowerror,maxflow+higherror] )
END
```

Figure 12 The third monitoring system specification

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 $maxflow = 400$, $minflow = 100$, $higherror = 1$, $lowerror = -1$.

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

- If $flow = minflow$, then *volume* is incremented by *minflow*, and the least that *rvolume* can be incremented by is $minflow + lowerror$. Thus in this case we obtain a possible value of $A = 0.99$.
- If $flow = maxflow$, then *volume* is increased by *maxflow*, and the least that *rvolume* can be incremented by is $minflow + lowerror$ if at least two sensors go wrong (which can happen with probability 0.028), otherwise $maxflow + lowerror$. 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 $E2 = volume - B \times rvolume$ must not decrease. Here we are concerned with the proportion by which *volume* can exceed *rvolume*, and the worst case always occurs when $flow = minflow$. In this case, the reading might at worst be $maxflow + higherror$ (with probability 0.028) and $minflow + higherror$ otherwise. This yields a value for B of at least 1.085 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 no more than 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 *volume* and *rvolume* to be established.

5 Separating system updates and tank updates

It could be useful to separate the model of the tank from the model of the system, and not refer to tank updates in the *poll* operation at all. Consequently, we could keep this abstract tank update operation throughout the refinement, and then throw it away once we have all the implementation. This is a small change from conventional B where we would expect to use all the operations of a machine's implementation, but it is appropriate in modelling embedded systems. [DT97]. Here, we want to keep only the operations which model the actual software functionality, and throw away the model of the environment once it is no longer required. Normally the environment model is only needed at the abstract level in order to specify a safety property between the approximated value *rvolume* of the volume in the tank and the real value *volume*, as we have seen in the expectations of the *VolumeTracker* machines previously.

To achieve this separation, consider two operations, one called *realPoll* which advances flow and volume, and *approxPoll*, which advances *rvolume*. The latter is the one we will want to implement.

Now it is possible (indeed inevitable) that there will be some states of the system where *volume* and *rvolume* do not match, because *realPoll* and *approxPoll* are out of step. Furthermore, any expectation of the form $rvolume - A \times volume$ must decrease on *realPoll*, since that increases *volume* but does not change *rvolume*. Similarly, an expectation of the form $B \times volume - rvolume$ must decrease on *approxPoll*, since that increases *rvolume* while leaving *volume* unchanged. Thus we require a way of dealing with the separation of *realPoll* from *approxPoll*.

There are in fact a variety of approaches we could take to dealing with this in the specification. In this section we will explore the introduction of auxiliary variables *rr* and *aa* to track the number of times the *realPoll* and *approxPoll* operations have respectively been called, and we will include this information in the expectations.

5.1 A first attempt

The description of the tank model is given in Figure 13. It incorporates a new variable *rr* to track the number of times *realPoll* has been called. Observe that *flow* can change on each occurrence of this operation.

The new monitoring system is given in Figure 14. This includes the model of the tank, and introduces its own counter *aa* for tracking the number of calls to *approxPoll*.

The expectations $E1$ and $E2$ that would be appropriate to include will need to take into account the difference between *aa* and *rr*. The general form of such expectations will be as follows:

$E1$ $rvolume - A \times volume - (aa - rr) \times A'$

$E2$ $B \times volume - rvolume - (rr - aa) \times B'$

These expectations must be non-decreasing on every operation. Thus they must both be preserved by both *approxPoll* and *realPoll*. In the case of $E1$, *approxPoll* will increase *rvolume* and *aa*, so the increase in *aa* can be used to offset the increase in *rvolume*, which in this operation is not matched by a corresponding increase in *volume*. Similarly, *realPoll* will increase *volume* and *rr*. Thus the decrease in $(aa - rr)$ will be used to offset the decrease in $rvolume - A \times volume$, so that the

```
MACHINE Tank
CONSTANTS  minflow, maxflow
PROPERTIES minflow : REAL & maxflow : REAL
            & minflow > 0 & maxflow >= minflow

VARIABLES flow, volume, rr
INVARIANT flow : REAL & volume : REAL & rr : NAT
INITIALISATION volume := 0 || flow :: [minflow, maxflow] || rr := 0

OPERATIONS realPoll =
           BEGIN
                 flow :: [minflow, maxflow] ||
                 volume := volume + flow ||
                 rr := rr + 1
           END
END
```

Figure 13 The new model of the tank, tracking the number of updates

overall expectation does not decrease. The appropriate values for A and A' can be calculated by using the inequalities $[approxPoll]E1 \Rrightarrow E1$ and $[realPoll]E1] \Rrightarrow E1$.

A similar form of reasoning applies to $E2$, and we obtain the following instantiations:

E1 $rvolume - (minflow/maxflow)volume - (aa - rr)minflow$

E2 $(maxflow/minflow)volume - rvolume - (rr - aa)maxflow$

These expectations do not provide very tight bounds. The difficulty that this calculation has highlighted is that *approxPoll* and *realPoll* will in general be updating *rvolume* and *volume* with different values of *flow*. In the most extreme case, *realPoll* could perform a number of updates with $flow = maxflow$, and then *approxPoll* could perform a number of updates with $flow = minflow$. In general, if the machines become more out of step (which is certainly allowed within the specification), then *volume* and *rvolume* might be incremented with different values of *flow*, and so could become quite different. Since *flow* is updated nondeterministically on occurrences of *realPoll*, we must consider the worst case possibility, and this is so bad that it completely overshadows any probabilistic behaviour that we might hope to describe in the expectation.

5.1.1 Summary

In this example we have seen how the updates to the monitoring system and to the tank can be separated. This separation introduces the possibility that the real value *volume* and the system value *rvolume* can diverge quite considerably, for two reasons: firstly, *realPoll* and *approxPoll* might not occur together in general, so one might occur much more than the other; and secondly, *realPoll* and *approxPoll* in general will read different values of *flow*, and so the updates they effect can be different, even if they are reasonably closely in step.

The expectations must be non-decreasing for both operations however the nondeterminism is resolved. The separation of *realPoll* and *approxPoll* means that the relationship between the expected values of *volume* and *rvolume* is weakened.

Note that the implementation of the *approxPoll* operation in terms of sensors will be the same as it was previously (except that *tock* will not now be included). A

```
MACHINE VolumeTracker4
INCLUDES Tank
PROMOTES  realPoll
VARIABLES rvolume, aa
INVARIANT rvolume : REAL & aa : NAT
EXPECTATION E1, E2
INITIALISATION rvolume := 0 || aa := 0

OPERATIONS

approxPoll = P1: BEGIN
                   S1 : (rvolume := rvolume + flow
                                0.99 (+)
                   S2 : rvolume :: rvolume + [minflow, maxflow])
                              ||
                          aa := aa + 1
                 END
END
```

Figure 14 A tank monitoring system separating system from tank updates

reading of the flow by means of two or three sensors as in the Section 4 will provide a suitable implementation of the operation exactly as it did before.

5.2 Restricting flow changes

The first approach to separating *realPoll* from *approxPoll* yielded a very weak relationship between the system and the real values for the volume of the liquid in the tank. However, in practice we might have certain assumptions about the way these operations might be called. For example, we might expect *approxPoll* to be called at roughly the same rate as *realPoll*, and to read the same values for *flow*. Thus we would not expect *flow* to change between *realPoll* and the next *approxPoll*. We can incorporate this into the model by introducing an explicit operation *setFlow* which is the only operation that changes the flow; and we can include an assumption (by means of a precondition) that this only occurs when *realPoll* and *approxPoll* are in step. This builds our environmental assumptions, that *realPoll* and *approxPoll* will effectively be in step, into the model. In fact we allow still allow *rvolume* and *volume* to become out of step, but in a controlled way, as we will see below. The updated version of the tank is given in Figure 15.

One way to incorporate this is to call *setFlow* from within the monitoring system, under a precondition so that the flow can be changed only when *rvolume* and *volume* are in step. Of course this precondition involves both the software and the tank system, and incorporates a modelling assumption.

If the flow can be changed while the system is out of step, then a much weaker expectation would result, in the extreme case corresponding to the expectations derived in Section 5.1 — a rather weaker relationship between *rvolume* and *volume*. Since we are concerned to exclude that, we will include the precondition in *setNewFlow*. The resulting machine is given in Figure 16. The expectations in *VolumeTracker4a* must be non-decreasing under all three operations. We obtain the following expectations:

$$\begin{aligned} \text{E1} \quad & (rvolume - (0.99 + 0.01(minflow/maxflow)) \times volume) \\ & \quad +(rr - aa) \times ((0.99 + 0.01(minflow/maxflow))flow) \end{aligned}$$

```
MACHINE Tank
SEES Bool_TYPE
CONSTANTS  minflow, maxflow
PROPERTIES minflow : REAL & maxflow : REAL
            & minflow > 0 & maxflow >= minflow

VARIABLES flow, volume, rr
INVARIANT flow : REAL & volume : REAL & rr : NAT
INITIALISATION volume := 0 || flow :: [minflow, maxflow] || rr : NAT

OPERATIONS realPoll =
              BEGIN
                 volume := volume + flow ||
                 rr := rr + 1
              END;

           setFlow(ff) =
              PRE ff : minflow..maxflow
              THEN flow := ff
              END
END
```

Figure 15 A tank monitoring system separating system from tank updates

$$\text{E2}\quad (0.99 + 0.01(\mathit{maxflow}/\mathit{minflow}) \times \mathit{volume} - \mathit{rvolume}) \\ +(aa - rr) \times ((0.99 + 0.01(\mathit{maxflow}/\mathit{minflow}))\mathit{flow})$$

Observe that *setNewFlow* does not change the expectations $E1$ and $E2$, because its precondition states that $rr = aa$, so the part of the expectation dependent on *flow* evaluates to 0. Observe also that the extent by which *rvolume* and *volume* are out of step is accounted for by a multiple of *flow*, which has been constant since the last time *volume* and *rvolume* were properly aligned.

5.2.1 Summary

The first approach to separating *realPoll* from *approxPoll* yielded a very weak relationship between the system and the real values for the volume of the liquid in the tank, because assumptions about the way the operations would be executed were not built into the model. In this section we built in the assumption that *realPoll* and *approxPoll* were dealing with the same values for *flow* by controlling more carefully when *flow* can be changed. We introduced a new operation *setFlow* to do this, but only when *volume* and *rvolume* were in step. We then required that the other operations could not alter *flow*. The expected operation of the system, whereby *realPoll* and *approxPoll* will essentially occur together, is incorporated within this model. But unexpected operation, in which *realPoll* and *approxPoll* occurring together will read completely different values of *flow*, is not permitted within this mode of the system.

The result is a much tighter relationship on the expected values of *volume* and *rvolume*.

Note that the implementation of the *approxPoll* operation in terms of sensors will again be the same as it was previously. All the changes we have made are at the level of the specification.

```
MACHINE VolumeTracker4a
INCLUDES Tank
PROMOTES  realPoll
VARIABLES rvolume, aa
INVARIANT rvolume : REAL & aa : NAT
EXPECTATION E1, E2
INITIALISATION rvolume := 0 || aa := 0

OPERATIONS

approxPoll = P1: BEGIN
                   S1 : (rvolume := rvolume + flow
                       p (+)
                   S2 : rvolume :: rvolume + [minflow, maxflow])
                          ||
                        aa := aa + 1
                 END;

       setNewFlow(ff) =
            PRE rr = aa & ff : minflow..maxflow
            THEN setFlow(ff)
            END
END
```

Figure 16 The revised tank monitoring system incorporating explicit flow changes

6 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 two progressions of scenarios. The first progression was given in Section 4. 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.

In the second set of scenarios, we separated the model of the tank from the description of the monitoring system. This approach is more common in the development of embedded systems [DT97], since the separation allows a cleaner development of the system. The fact that different operations were used to update the states of the tank and of the monitoring system had a significant impact on the relationship between the expectations of the real volume and the monitoring system's value for it. We found that the first approach gave too weak a relationship, essentially no stronger than that provided by the invariant (which is concerned only with all possible reachable states). The reason for this is that probabilistic B does not provide any control on the invocation of machine operations, or assumptions on the order and frequency of their occurrence, so it must allow for the machine to be placed in any environment. The fact that the flow could change on any update of the tank meant that the system readings and the real flow values could be wildly different for some sequences of operation calls.

In the second scenario, we introduced behaviour incorporating realistic assumptions: that the flow would not change while the system updates and tank updates were out of step. We considered this reasonable because in practice these updates would tend to be in step. This assumption meant that the system readings for flow corresponded to the real flow into the tank, and we regained a tighter relationship between the expected values of the measured volume and the real volume of liquid.

We have seen that the requirement that every operation should not decrease the machine's expectation introduces a consistency condition between the expectation and the probabilistic and nondeterministic behaviour in the machine operations. This need for consistency can be pushed in either direction: either starting with a required expectation and then deriving the reliability requirements and flow parameters necessary to achieve that; or starting with a given combination of sensors with some known reliability and obtaining the tightest possible bounds on the expectation.

7 Further work

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.

Incorporating some information about the interactions between different operations raises some interesting problems. The final scenario we considered is quite relaxed in that it allows the measured volume to become quite out of step with the real volume. There are other possibilities for modelling such a scenario. For example, it might be preferable to introduce a stronger model of control flow to ensure that real updates and system updates occur alternately. This might require the introduction of flags to track which operation should be performed next, and guards to block operations from executing out of turn.

As an alternative, it may be appropriate to introduce controllers separately for probabilistic B machines, and combine them in the style of CSP||B [TS00, TSB03]. Thus CSP processes will describe the permitted or expected sequences of operations, and could be used to drive the probabilistic B machine. This would allow some weaker requirements on expectations to be introduced in the context of such control loops: such expectations might need to be non-decreasing over the body of a control loop, rather than the stronger requirement that each operation individually should not decrease it. This is a topic for future research.

References

[Abr96] J-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[DT97] J. Draper and H. Treharne. The refinement of embedded software with the B-Method. In *Northern Formal Methods Workshop*. Springer, 1997.

[HJR+03] 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.

[MM04] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer, 2004.

[MMH03] 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.

[Mor98] C. Morgan. The generalised substitution language extended to probabilistic programs. In *B'98: the 2nd International B Conference*, number 1393 in LNCS. Springer, 1998.

[TS00] H. Treharne and S. Schneider. How to drive a B machine. In *ZB2000: International conference of Z and B Users*, number 1878 in LNCS. Springer, 2000.

[TSB03] H. Treharne, S. Schneider, and M. Bramble. Combining specification with composition. In *ZB2003: 3rd International Conference of Z and B users*, number 2651 in LNCS. Springer, 2003.

A Calculation of expectation coefficients in $VolumeTracker2$

to be completed

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

to be completed