Electronic Notes in Theoretical Computer Science 137 (2005) 183–204 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Tank Monitoring: A pAMN Case Study

# Steve Schneider[1](#_bookmark3)

*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](mailto:S.Schneider@surrey.ac.uk)

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2005.04.031

# Introduction

The B-Method [[1](#_bookmark17)] provides a framework for the development of provably cor- rect 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](#_bookmark18)], called *probabilistic B*. This approach builds on previous work which introduces probabilistic choice into program statements, and ex- tends the notion of weakest precondition semantics to deal with *expectations* [[5](#_bookmark21)]. 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, prob- abilistic B introduces expectations on aspects of the state, in addition to the existing parts of a B machine. Thus the relationship between the expected val- ues 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.

# Introducing Probability

* 1. *Probabilistic GSL*

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

*prog*1 *p* ⊕ *prog*2

execute *prog*2 with probability 1 − *p*. See [[6](#_bookmark22),[3](#_bookmark19),[4](#_bookmark20)] for a full introduction to An execution of this choice will execute *prog*1 with probability *p*, and will 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](#_bookmark22)] is given in Figure [1](#_bookmark1).

In this paper we will use a derived operator (also given in [[1](#_bookmark17)]) for assigning to a variable some element from a set *S* chosen nondeterministically. We define

*x* :∈ *S* =^ @*y.*(*y* ∈ *S* =⇒ *x* := *y* )

Thus

[*x* :∈ *S* ]*exp* = (min *x* | *x* ∈ *S.exp*)

We will also use a derived operator (also given in [[3](#_bookmark19)]) for expressing a

minimum probability on a choice. We define

*prog*1 ≥*p* ⊕ *prog*2 =^ @*q.*(*p* ≤ *q* ≤ 1) =⇒ *prog*1 *q* ⊕ *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* ≥0.9 ⊕ *incorrect*. This would be refined by a component that behaves correctly at least 95% of the time, for

example.

* 1. *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](#_bookmark19)]. 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* | *prog* ]*exp* ⟨*pre*⟩× [*prog* ]*exp*, where 0 ×∞ =b 0

*prog*1 []*prog*2 [*prog*1 ]*exp* min[*prog*2 ]*exp*

[*pre* =⇒ *prog* ]*exp* 1/⟨*pre*⟩× [*prog* ]*exp*, where ∞× 0 =b ∞

[*skip*]*exp exp*

[*prog*1 *p* ⊕ *prog*2 ]*exp p* × [*prog*1 ]*exp* + (1 − *p*) × [*prog*2 ]*exp*

[@*y* .*pred* =⇒ *prog* ]*exp* (min *y* | *pred* .[*prog* ]*exp*)

*prog*1 ± *prog*2 [*prog*1 ]*exp* $ [*prog*2 ]*exp* for all *exp*.

* *exp* is an expectation
* *pre* is a predicate (not an expectation)
* ⟨*pre*⟩ denotes predicate *pre* converted to an expectation, here restricted to the unit interval:

⟨*false*⟩ is 0 and ⟨*true*⟩ is 1.

* × 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 $ *exp*2 means that *exp*1 is everywhere no more than *exp*2 .

Fig. 1. *pGSL*—the probabilistic Generalised Substitution Language [[6](#_bookmark22)]

following laws from that Appendix will be used in this paper: Law 13:

(*prog*1 ≥*p* ⊕ *prog*2); *prog*3 = (*prog*1; *prog*3) ≥*p* ⊕ (*prog*2; *prog*3) Law 24:

(*prog*1 ≥*pq* ⊕ *prog*2)= *prog*1 ≥*p* ⊕ (*prog*1 ≥*q* ⊕ *prog*2)

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

*prog*2 ± *prog*1 ⇒ *prog*1 ≥*p* ⊕ *prog*2 = *prog*1 *p* ⊕ *prog*2

*flow*

*volume*

* 1. *Probabilistic B*

Fig. 2. The tank system

There are two aspects to the introduction of probabilistic behaviour into a B machine as proposed in [[2](#_bookmark18)]. The first is to allow operations to be constructed using probabilistic GSL, so probabilistic choices can be made within opera- tions. The second is to introduce an expectation clause into a B machine in order to express requirements on various expectations on the state. An expec- tation 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* $ *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* $ [*Init*]*V*

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

*V* $ [*Op*]*V*

In this paper we will use expectations of the form *V* . This is an abbre- viation for 0 $ *V* . Observe that this still gives rise to a non-trivial proof obligation *P* 1, that *V* is non-negative on initialisation.

# 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](#_bookmark2).

The tank can be modelled using the machine given in Figure [3](#_bookmark5) [2](#_bookmark4) . This

2 An explanation of the ascii form of pGSL used in Figure [3](#_bookmark5) 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 speci- fications 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*].

# A monitoring system

* 1. *The ﬁrst simple system*
     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](#_bookmark6). (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](#_bookmark24)

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 specifica- tion 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 ex- pected 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* × *volume*

E2: *B* × *volume* − *rvolume*

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

that

*rvolume/B* ≤ *volume* ≤ *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 *E* 1 and *E* 2 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* × (*minflow/maxflow* ) ≤ *volume* ≤ *rvolume* × (*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*.

* + 1. *Deriving A and B*

For *VolumeTracker* 1 to meet its proof obligations, we require that the expec- tations 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

require that for any *flow* , *volume*, and *rvolume*, we must have that ([*poll* ]*E* 1)− achieve this. We require that *E* 1 $ [*poll* ]*E* 1 and *E* 2 $ [*poll* ]*E* 2. Thus we *E* 1 ≥ 0 and ([*poll* ]*E* 2) − *E* 2 ≥ 0.

([*poll* ]*E* 1) − *E* 1= ([*T* || (*V* 1*a* 0.99 ⊕ *V* 1*b*)]*E* 1) − *E* 1 We calculate the requirement on *A* from the requirement on *E* 1:

= ([(*T* || *V* 1*a*) 0.99 ⊕ (*T* || *V* 1*b*))]*E* 1) − *E* 1

= (0*.*99 × [*T* || *V* 1*a*]*E* 1+ 0*.*01 × [*T* || *V* 1*b*]*E* 1) − *E* 1

(∗)= (0*.*99 × (*rvolume* + *flow* − *A*(*volume* + *flow* ))

+0*.*01 × (*rvolume* + *minflow* − *A*(*volume* + *flow* )))

−(*rvolume* − *A.volume*)

= 0*.*99 × (*flow* − *A* × *flow* )+ 0*.*01(*minflow* − *A* × *flow* )

= (0*.*99 − *A*) × *flow* + 0*.*01 × *minflow*

Since this must be non-negative everywhere (i.e. for all possible values of

flow), we obtain that

*A* ≤ 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* ≤ 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:

([*poll* ]*E* 2) − *E* 2= ([*T* || (*V* 1*a* 0.99 ⊕ *V* 1*b*)]*E* 2) − *E* 2

= ([(*T* || *V* 1*a*) 0.99 ⊕ (*T* || *V* 1*b*))]*E* 2) − *E* 2

= (0*.*99 × [*T* || *V* 1*a*]*E* 2+ 0*.*01 × [*T* || *V* 1*b*]*E* 2) − *E* 2

(∗∗)= (0*.*99 × (*B* (*volume* + *flow* ) − (*rvolume* + *flow* ))

+0*.*01 × (*B* (*volume* + *flow* ) − (*rvolume* + *maxflow* )))

−(*B.volume* − *rvolume*)

= 0*.*99 × (*B.flow* − *flow* )+ 0*.*01(*B.flow* − *maxflow* )

= *B* × *flow* − 0*.*99 × *flow* − 0*.*01 × *maxflow*

We require that this is non-negative for any value of *flow* . Thus *B* ≥ 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* ≥ 0*.*99 + 0*.*01(*maxflow/minflow* )

Observe lines (\*) and (\*\*) concerning the evaluation of [*T* || *V* 1*b*] with respect to an expectation. Since *V* 1*b* is nondeterministic in the assignment to

*rvolume*, the minimum expectation over all possible assignments to *rvolume* must be taken. In *E* 1, *rvolume* is positive, so the smallest possible value of *rvolume* is used in the calculation of the pre-expectation of *E* 1. In *E* 2 *rvolume* is negative so the largest possible value of *rvolume* is used in the calculation of the pre-expectation of *E* 2. This means that however the nondeterminism is later resolved, the expectation will be at least the value calculated. Expec- tations should always be non-decreasing, so demonic nondeterminism always considers the worst case with respect to increases.

* + 1. *Example*

As an illustration, we shall consider some concrete numbers: if *minflow* = 100 and *maxflow* = 400, then we obtain *A* ≤ 0*.*9925 and *B* ≥ 10*.*03. Thus we know that

(100*/*103) × *rvolume* ≤ *volume* ≤ *rvolume* × (400*/*397) This implies for example that

0*.*97 × *rvolume* ≤ *volume* ≤ 10*.*03 × *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

description cannot ensure that 0*.*99 × *rvolume* ≤ *volume*. This is because However, if our requirement is for 99% accuracy, this will not be met. The 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.

* + 1. *Implementation*

Our first implementation of *VolumeTracker* 1 will make use of two sensors, which provide readings for the flow, and also give diagnostic information stat- ing 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, *Sensor* 1*a* and *Sensor* 1*b* to give readings to 99% accuracy. *Sensor* 1*b* is given in Figure [5](#_bookmark8), and *Sensora*1 is entirely similar.

We propose an implementation *VolumeTracker* 1*I* of *VolumeTracker* 1 which uses two sensors in order to obtain a more reliable reading of the flow. This is given in Figure [6](#_bookmark9), and makes use of the Context machine of Figure [7](#_bookmark10).

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 re- finement 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 *VolumeTracker* 1*I*

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](#_bookmark19)] given in Section [2.2](#_bookmark0). The *poll* operation in *VolumeTracker* 1*I* of Figure [6](#_bookmark9) is of the particular form *P* 1*a*; *P* 1*b*; *F* ; *R*; *T* , where the variables *v* 1*, v* 2*, st*1*, st*2*, rflow* are all local.

We show that this operation is equivalent to *poll* given in the specification machine *VolumeTracker* 1, as follows:

*P* 1*a*; *P* 1*b*; *F* ; *R*; *T*

= {expanding *P* 1*a* and *P* 1*b*}

(*S* 1*al* ≥0.9 ⊕ *S* 1*ar* );

(*S* 1*bl* ≥0.9 ⊕ *S* 1*br* ); *F* ; *R*; *T*

= {Law 13}

*S* 1*al* ; (*S* 1*bl* ≥0.9 ⊕ *S* 1*br* ); *F* ; *R*; *T*

≥0.9 ⊕

*S* 1*ar* ; (*S* 1*bl* ≥0.9⊕ *S* 1*br* ); *F* ; *R*; *T*

= {Law 13}

(*S* 1*al* ; *S* 1*bl* ; *F* ; *R*; *T* ≥0.9 ⊕ *S* 1*al* ; *S* 1*br* ; *F* ; *R*; *T* )

≥0.9 ⊕

(*S* 1*ar* ; *S* 1*bl* ; *F* ; *R*; *T* ≥0.9 ⊕ *S* 1*ar* ; *S* 1*br* ; *F* ; *R*; *T* )

= {standard program algebra in each branch; removal of local variables} (*V* 1*a * *T* ≥0.9 ⊕ *V* 1*a * *T* ) ≥0.9 ⊕ (*V* 1*a * *T* ≥0.9⊕ *V* 1*b * *T* )

= {idempotence of ≥*p*⊕ on left-hand argument}

*V* 1*a * *T* ≥0.9 ⊕ (*V* 1*a * *T* ≥0.9 ⊕ *V* 1*b * *T* )

= {Law 24}

(*V* 1*a * *T* ≥0.99 ⊕ *V* 1*b * *T* )

= {Law A, since *V* 1*b* ± *V* 1*a*}

(*V* 1*a * *T* 0.99 ⊕ *V* 1*b * *T* )

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.

* + 1. *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 de- crease 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 implementa- tion.
      * The accuracy of the approximation *rvolume* to the tank value *volume* de- pends 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 implemen- tations 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.

* 1. *Introducing error margins*
     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](#_bookmark11).

The calculation of appropriate *A* and *B* follows the same pattern as shown previously in Section [4.1.2](#_bookmark7). Now two sources of nondeterminism must be taken into account: the reading of the sensors in *V* 2*a* (which can be most pessimistic with regard to *E* 1 when *flow* is low) and the arbitrary reading in *V* 2*b* (which can be most pessimistic for *E* 1 when *flow* is high). This combination of considerations (recall *lowerror* is negative, so *A* ≤ 1) means that *A* is bounded above by both of the following values:

1+ (*lowerror/minflow* )

and

0*.*99 + (*lowerror/maxflow* )+ 0*.*01(*minflow/maxflow* )

For example, if *minflow* = 100, *maxflow* = 400, and *lowerror* = −10, then the first value is lower, and we obtain *A* = 0*.*9. On the other hand, if *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 *minflow* + *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 *V* 2*b* (1% of the time) dominates, and the worst case occurs when the flow is *maxflow* and *rvolume* is only incremented by *lowerror* + *minflow* .

Similar considerations for the expectation *E* 2 yield that the value obtained for *B* is the maximum of the following two values, the first for the case where *flow* = *maxflow* and the second when *flow* = *minflow* .

1+ (*higherror/maxflow* ) and

0*.*99 + (*higherror/minflow* )+ 0*.*01(*maxflow/minflow* )

In this case, the second value will always be higher, and hence will give the ap- propriate value for *B* , since *maxflow/minflow* ≥ 1, and *higherror/minflow* ≥ *higherror/maxflow* . This informs us that the worst case always occurs with a flow of *minflow* , and an incorrect reading of *maxflow* + *higherror* . This is worse than the worst outcome that can be obtained with a flow of *maxflow* , as far as ensuring that *E* 2 does not decrease is concerned.

* + 1. *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 *Sensor* 2*b* we will take the error range to be [*le*2*b, he*2*b*]. The resulting sensor is given in Figure [9](#_bookmark13).

The implementation *VolumeTracker* 2*I* will be the same as *VolumeTracker* 1*I* , except that it now importing *Sensor* 2*a* (with error range [*le*2*a, he*2*a*]) and *Sensor* 2*b*, instead of the original sensors. It is given for reference in Figure [B.1](#_bookmark25) of Appendix [B](#_bookmark26).

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 *Sensor* 2*b*

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.

vided that the following hold: that [*le*2*a, he*2*a*] ⊆ [*lowerror, higherror* ] and The machine *VolumeTracker* 2*I* provides an implementation of *poll* , pro- [*le*2*b, he*2*b*] ⊆ [*lowerror, higherror* ]. In other words, that the error ranges for

each sensor are within those given in *VolumeTracker* 2 for the overall combi-

nation. The proof of this is given in Appendix [B](#_bookmark26).

* + 1. *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.
  1. *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.

* + 1. *Implementation: sensor*

A sensor without diagnostic information about its status is given in Figure [10](#_bookmark14). It provides only a flow reading, without any information about its state.

To be tolerant to one faulty reading, we need three sensors: *Sensor* 3*a*, *Sensor* 3*b*, and *Sensor* 3*c*. 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](#_bookmark15). We still assume a 90% reliability on the reading of any individual sensor.

* + 1. *Speciﬁcation*

In fact here *VolumeTracker* 3*I* is a refinement of *VolumeTracker* 3 given in Figure [12](#_bookmark16), provided all of the sensor errors are within the error given in

*VolumeTracker* 3, e.g. [*le*3*, he*3] ⊆ [*lowerror, higherror* ].

For *VolumeTracker* 3, carrying out the standard calculations on preserva-

tion of *E* 1, we find that the best (highest) value we can obtain for *A*, which enables the expectation *E* 1 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

END

v1 <-- poll3a; v2 <-- poll3b; v3 <-- poll3c;

rflow := median(v1,v2,v3); rvolume := rvolume + rflow; tock

## END

Fig. 11. The implementation *VolumeTrackerI* 3

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(*maxflow/minflow* )+ (*higherror/minflow* )

The second of these will always be the maximum, since *maxflow* ≥ *minflow* . The situation is similar to the previous example considered in Section [4.2.2](#_bookmark12), 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* .

400, *minflow* = 100, *higherror* = 1, *lowerror* = −1. For example, consider the situation where we have the following: *maxflow* =

Since the expectation *E* 1 = *rvolume* − *A* × *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 *E* 2 = *volume* − *B* × *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

10*.*085 if the expectation of *E* 2 is not to decrease. This is a margin of error of 8*.*5%.

* + 1. *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 read- ing 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% reliabil-

ity 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.

# 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 consid- ered 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 specifica- tion, 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 ex- plored some of the interesting kinds of behaviour that can arise in such sys- tems, 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.

1. S. Schneider. *The B-Method: an Introduction*. Palgrave, 2001.

# 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](#_bookmark17),[7](#_bookmark23)].

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 *VolumeTracker* 2*I*

*x* := *E* x:=E

*x* :∈ *S* x :: S

*x , y* := *E, F* x,y := E,F

*pre* | *prog* pre | prog

*prog*1[]*prog*2 prog1 [] prog2

*pre* =⇒ *prog* pre ==> prog

*skip* skip

*prog*1 *p* ⊕ *prog*2 prog1 p (+) prog2 *prog*1 ≥*p* ⊕ *prog*2 prog1 >=p (+) prog2 @*y.pred* =⇒ *prog* @ y . pred ==> prog

1. Verifying the implementation of *poll* in *VolumeTracker* 2*I*

The *poll* operation in *VolumeTracker* 2*I* is of the form *P* 2*a*; *P* 2*b*; *F* ; *R*; *T* , where *v* 1*, v* 2*, st*1*, st*2*, rflow* are all local variables. We show that this operation is equivalent to *poll* given in the specification machine *VolumeTracker* 2, as follows:

*P* 2*a*; *P* 2*b*; *F* ; *R*; *T*

= {expanding *P* 2*a* and *P* 2*b*}

(*S* 2*al* ≥0.9 ⊕ *S* 2*ar* ); (*S* 2*bl* ≥0.9⊕ *S* 2*br* ); *F* ; *R*; *T*

= {Law 13, twice }

(*S* 2*al* ; *S* 2*bl* ; *F* ; *R*; *T* ≥0.9 ⊕ *S* 2*al* ; *S* 2*br* ; *F* ; *R*; *T* )

≥0.9 ⊕

(*S* 2*ar* ; *S* 2*bl* ; *F* ; *R*; *T* ≥0.9 ⊕ *S* 2*ar* ; *S* 2*br* ; *F* ; *R*; *T* )

= {standard program algebra in each branch; removal of local variables} (*rvolume* :: *rvolume* + *flow* + [(*le*2*a* + *le*2*b*)*/*2*,* (*he*2*a* + *he*2*b*)*/*2]; *T*

≥0.9⊕ *rvolume* :: *rvolume* + *flow* + [*le*2*a, he*2*a*]; *T* )

≥0.9 ⊕

(*rvolume* :: *rvolume* + *flow* + [*le*2*b, he*2*b*]; *T*

≥0.9⊕ *rvolume* :: *rvolume* + *flow* + [*minflow , maxflow* ]; *T* )

@ {expanding the ranges of the nondeterministic choices, provided *lowerror* ≤ *le*2*a*, *lowerror* ≤ *le*2*b*,

*he*2*a* ≤ *higherror* , *he*2*b* ≤ *higherror* }

(*rvolume* :: *rvolume* + *flow* + [*lowerror , higherror* ]; *T*

≥0.9⊕ *rvolume* :: *rvolume* + *flow* + [*lowerror , higherror* ]; *T* )

≥0.9 ⊕

(*rvolume* :: *rvolume* + *flow* + [*lowerror , higherror* ]; *T*

≥0.9 ⊕ *rvolume* :: *rvolume* + *flow*

+[*minflow* + *lowerror, maxflow* + *higherror* ]; *T* )

= {Laws 13 and 24}

*V* 2*a*; *T* ≥0.99 ⊕ *V* 2*b*; *T*

= {Laws 13 and A, since *V* 2*b* ± *V* 2*a*; *T* independent of *V* 2*a* and *V* 2*b* }

(*V* 2*a * *T* 0.99 ⊕ *V* 2*b * *T* )

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.