The original publication is available at http://dx.doi.org/10.1007/11415787_21
Appeared in Proceedings of the ZB2005 conference © Springer-Verlag

Development via Refinement in Probabilistic B — Foundation and Case Study

Thai Son Hoang[1,2], Zhendong Jin[1], Ken Robinson[1,2],
Annabelle McIver[3], and Carroll Morgan[1]

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

Abstract. In earlier work, we introduced probability to the *B-Method* (B) by providing a probabilistic choice substitution and by extending B's semantics to incorporate its meaning [8]. This, a first step, allowed probabilistic programs to be written and reasoned about within B.
This paper extends the previous work into refinement within B. To allow probabilistic *specification* and *development* within B, we must add a probabilistic *specification* substitution; and we must determine the rules and techniques for its rigorous refinement into probabilistic code.
Implementation in B frequently contains loops. We generalise the standard proof obligation rules for loops giving a set of rules for reasoning about the correctness of probabilistic loops. We present a small case-study that uses those rules, the randomised Min-Cut algorithm.

Keywords: Probability, program correctness, generalised substitutions, weakest preconditions, B, randomised algorithms, refinement.

1 Introduction

Our overall aim is to extend the *B-Method* (B) to incorporate probability, with the aim of allowing its rigorous development techniques to apply to random algorithms, probabilistic distributed systems (via for example *Event-B*) and safety-critical applications (using fully quantitative judgments of the "cost" of program outcomes).

We have made a number of extensions already at what would be called a "low-level". For example, we have extended B to allow the deduction of probability-one conclusions about programs containing probability: this is called qB [11], and would apply to the final stages of an algorithm like the IEEE 1394 (FireWire) protocol [2, 6] where a potential livelock is resolved with probability one.

A second extension is the incorporation of full probabilistic reasoning into B (that is, not just probability-one) via the introduction of a probabilistic-choice substitution, with its associated semantics: this is called *pB*. Unlike *qB*, whose logic remains Boolean, the *pB* logic is based on real numbers, necessary to be able to make judgments about probability.

The probabilistic-choice substitution is "code" in the sense that it can be (almost) directly translated into a programming language, and would typically be found in the last stages of a development. In that sense it can be considered "low-level".

Our aim in this paper is to begin to address the "higher-level" concerns of probabilistic development. Traditionally this involves some form of "specification", incorporating nondeterminism (interpreted as implementation freedom), together with an appropriate notion of "refinement" that leads from such specifications to the code that implements them.

We propose a probabilistic *specification* substitution (similar to the one proposed for *Z* [21]), and we recall the definition of probabilistic refinement [16]. We prove the "fundamental theorem" for the new construct, in the *B* context, by analogy with the fundamental theorem for the traditional specification statement [12] that shows the new statement's semantics properly interacts with refinement.

To illustrate and explore the extension, we extend the rules for partial and total correctness of standard loops to probabilistic loops, in the style of the probabilistic wp-logic [16]. Furthermore, the proof obligations for probabilistic loops can be separated so that we can prove standard (predicate) properties and probabilistic (quantitative) properties separately. The latter is especially important for a practical method, such as *B*, where we need to preserve as much as possible the facility and efficiency of dealing with a system's non-probabilistic components, limiting the new (and more complex) probabilistic reasoning to where it's required.

The new techniques mentioned are illustrated by a case study of a randomised algorithm. In the example, we go from specifying the algorithm, implementing it using a loop and reasoning about the correctness of the algorithm.

The paper is structured as follows: in Sec. 2 we briefly recall the details of the *probabilistic Generalised Substitution Language* (*pGSL*), and illustrate the expectation logic by using simple examples; in Sec. 3 we first review the traditional specification substitution for standard systems and then introduce probabilistic specification substitution to describe probabilistic systems; in Sec. 4 we appeal to the expectation semantics from *pGSL* and obtain the probabilistic fundamental theorem which is a generalised version of the corresponding standard theorem.

In Sec. 5 we discuss proof obligations for probabilistic loops, which is the generalisation of the variant and invariant technique for standard loops.

In Sec. 6 we first apply the fundamental theorem to the well-known example of "Min-Cut" algorithm, we also set out the proof obligations for maintaining the refinement; finally, we summarise the development, draw our conclusions and outline possible future work.

2 The probabilistic-choice substitution

The probabilistic-choice substitution has been introduced into *probabilistic B* (*pB*) already [8]; as we noted in the introduction, it can be considered as a "low-level" extension.

The numeric logic *pGSL* necessary to accommodate the extension uses real- rather than Boolean-valued expressions for its "predicates", which we call "expectations": the numbers represent "expected values" rather than the normal predicates that definitely do, or do not hold. In other words, we replace certainty by probability.

We can give only a very brief description of *pGSL* in the space available here; the reader is referred to our earlier work for a full introduction [11].

The probabilistic-choice substitution is the only extension to standard *Generalised Substitution Language* (*GSL*). It has the form

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

which means that with probability p, the substitution $prog_1$ is executed, and with probability $1 - p$, the substitution $prog_2$ is chosen.

Implication-like relations between expectations are

$$\begin{array}{lcl} exp_1 \Rrightarrow exp_2 & \widehat{=} & exp_1 \text{ is everywhere no more than } exp_2 \\ exp_1 \equiv exp_2 & \widehat{=} & exp_1 \text{ is everywhere equal to } exp_2 \\ exp_1 \Lleftarrow exp_2 & \widehat{=} & exp_1 \text{ is everywhere no less than } exp_2 . \end{array}$$

The refinement relationship in *pB* is defined accordingly:

$$prog_1 \sqsubseteq prog_2 \quad \text{if and only if} \quad [prog_1]exp \Rrightarrow [prog_2]exp \quad \text{for all } exp$$

The semantics of the substitutions in *pGSL* are given in Fig. 1.

3 The probabilistic-specification substitution

The probabilistic-specification substitution, and its properties, are the "high-level" subjects of this paper. Because our concern here is with larger-scale structures, we must turn to a specification construct, since that is the starting point for the refinement steps that are characteristic of *B* developments, whether standard or probabilistic. We begin by reviewing the "standard" specification that *B* already contains, where by *standard* we mean "without probability".

3.1 Standard specification substitution

In this section, we briefly review the interaction of specifications and so-called "specification substitutions"[4] for standard systems.

[4] For those familiar with the refinement calculus, these will correspond to pre-postcondition specifications [4], specification statements [13], and prescription [19]; we are going to treat the *B* version of those.

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

- *exp* is an expectation (possibly but not necessarily $\langle \mathit{pred}\rangle$ for some predicate *pred*);
- *pre* is a predicate (not an expectation);
- $\langle \mathit{pre}\rangle = 1$ when *pre* holds, $\langle \mathit{pre}\rangle = 0$ when *pre* does not hold;
- $*$ is multiplication;
- $\mathit{prog}, \mathit{prog}_1, \mathit{prog}_2$ are probabilistic generalised substitutions;
- p is an expression over the program variables (possibly but not necessarily a constant), taking a value in $[0,1]$; and
- v is a variable (or a vector of variables).

pGSL [15] acts over "expectations" rather than predicates: *expectations* take values in $[0,1] \cup \{\infty\}$.
We give the definitions including infeasible or "miraculous" commands [13, Sec. 1.7], but omit them in the main text.

Fig. 1. *pGSL* semantics

In the specification stage of a development, it is traditional to use pre- and post-conditions to describe the desired behavior of the system to be built. In general, there are many forms of this; one version is "Specification statements" [13]:

$$v : [P, Q]$$

where v is the *frame*, a sub-vector of the program variables whose values may change. P and Q are predicates describing the initial state and the final state, respectively.

In B [1], we find the same idea though with a different syntax. In this paper we will use the syntax

$$v : \{P, Q\} \ , \quad ^{5} \tag{1}$$

with the meaning that the substitution will establish Q under the precondition P, and change only the variables in v. In this form, we will always assume that P and Q are predicates over x and over x_0, v. The variables v are those that can be possibly changed by the substitution. The variables x_0 are distinct from x and represent their original values.

3.2 Probabilistic specification substitution

We now show how the ideas of Sec. 3.1 can be generalised to the probabilistic context, that is, we will propose a probabilistic generalisation of (1) which will play the same role in probabilistic specification and refinement as the original (1) does in the standard case.

In the expectation logic of Sec. 2, we write

$$A \quad \Rrightarrow \quad [S]B \ , \tag{2}$$

to mean that execution of S must establish that the expected value of B over final state distributions is bounded below by A's value in the initial state. By analogy with the connection between Dijkstra-style specification and the specification statement, we propose a probabilistic specification substitution written as in the standard case, that is

$$v : \{A, B\} \ , \tag{3}$$

except that A now is an expectation defined over the program variables, B is an expectation that may additionally refer to x_0 and v as before are variables that are allowed to change.

For example, if we want to specify a coin that with probability at least one-half comes up heads, then in the style of (2) we would write

$$\frac{1}{2} \quad \Rrightarrow \quad [\mathit{Flip}]\, \langle c = H \rangle \ ,$$

where c (for "coin") is the state variable with possible values $\{H, T\}$. In the style of (3), we would instead specify the substitution *Flip* as the substitution

$$c : \left\{ \frac{1}{2}, \langle c = H \rangle \right\} \ , \tag{4}$$

[5] Actually, in B it is written as

$$P \ \mid \ v \in Q$$

.

for the following reason: it achieves $c = H$ (post-expectation $\langle c = H \rangle$) with probability at least $\frac{1}{2}$ (pre-expectation). Thus the *probabilistic specification substitution* generalises the traditional specification substitution into the probabilistic program domain.

We now give the semantic definition for (3) so that we can explain why the specifications like (4) have the meaning we claim for them.

Definition 1. *The semantics of the specification substitution $v : \{A, B\}$, with respect to arbitrary post-expectation C (containing no x_0), is given by*

$$[v : \{A, B\}]C \quad \widehat{=} \quad A \quad * \quad [x_0 := x]\,(\sqcap x \cdot C \;\div\; B^w) \ , \tag{5}$$

where x is the vector of all variables appearing in A, B or C; w is the vector of unchanging variables, in x but not in v; and B^w is $B * \langle w = w_0 \rangle$. The symbols $*, \div$ denote multiplication, division respectively of real numbers.

In general, $(\sqcap x \cdot D)$ means the greatest lower bound of the expression (expectation) D over the possible values of x. We use explicit brackets to indicate the scope of the minimum, so in the definition, $(\sqcap x \cdot C \;\div\; B^w)$ means the minimum of $C \;\div\; B^w$ over all x.

We give the intuitive justification for Def. 1 as follows: it says that the specification takes an initial state to any one of a number of final state distributions, all of which satisfy the requirement that the expectation of B over that final distribution is bounded below by A evaluated on the initial state. Given that, the definition calculates the expected value of C (instead of B), using algebraic properties of these substitutions.

Taking the example of (4), we can calculate the probability that the outcome is heads. From Def. 1 it is given as

$$\begin{array}{lll}
 & [c : \{\frac{1}{2}, \langle c = H \rangle\}]\,\langle c = H \rangle & \\
\equiv & \frac{1}{2} \quad * \quad [c_0 := c]\,(\sqcap c \cdot \langle c = H \rangle \;\div\; \langle c = H \rangle) & \text{Def. 1} \\
\equiv & \frac{1}{2} \quad * \quad [c_0 := c]\ 1 & \text{arithmetic}\ ^{6} \\
\equiv & \frac{1}{2}\ . & \text{arithmetic and simple substitution}
\end{array}$$

So indeed the probability that (4) establishes $c = H$ is at least $\frac{1}{2}$.

If we calculate the probability that the outcome of the same program is tails, however, we have

$$\begin{array}{lll}
 & [c : \{\frac{1}{2}, \langle c = H \rangle\}]\,\langle c = T \rangle & \\
\equiv & \frac{1}{2} \quad * \quad [c_0 := c]\,(\sqcap c \cdot \langle c = T \rangle \;\div\; \langle c = H \rangle) & \text{Def. 1} \\
\equiv & \frac{1}{2} \quad * \quad [c_0 := c]\ 0 & \text{minimum } 0 \div 1 \text{ occurs at } C = H \\
\equiv & 0\ . & \text{arithmetic and simple substitution}
\end{array}$$

The conclusion is that (4) does not give any guarantee at all that the outcome is tails. We address this point later, in Sec. 6.5.

[6] We assume that $x \div 0$ is ∞ for any x so that the $\sqcap$ ignores it.

4 The fundamental theorems for specifications

In this section we justify the semantics given in Def. 1 by looking at a fundamental theorem that such semantics should satisfy. There is a standard fundamental theorem already; we propose a corresponding probabilistic fundamental theorem.

4.1 The standard fundamental theorem

This theorem comes from the refinement calculus [13, 12]; here we explain it in terms of B-style notation.

Theorem 1. *Let $v : \{P, Q\}$ be defined as in (1) and T be any program written in* GSL *with state variables x, then $v : \{P, Q\} \sqsubseteq T$ if and only if*

$$P \quad \Rrightarrow \quad [x_0 := x][T]Q^w \ ,$$

where Q^w is $Q \wedge w = w_0$. Similar theorems and their proofs can be found in [17, 3]. The theorem states that if the before state satisfies P then the substitution T will guarantee to establish Q in the after state, and change only variables in v. And therefore T satisfies the specification $v : \{P, Q\}$.

4.2 The probabilistic fundamental theorem

Now we return to the issue of the probabilistic fundamental theorem. It is Theorem 2 as follows:

Theorem 2. *Let $v : \{A, B\}$ be defined as in Def. 1 and T be any* pGSL *substitution and be free from variables x_0. Assume B satisfies the assumption: $\forall\, x_0 \cdot (\exists\, v \cdot (B \neq 0))$. Then*

$$v : \{A, B\} \ \sqsubseteq \ T \quad \textit{iff} \quad A \ \Rrightarrow \ [x_0 := x][T]B^w \ .$$

Proof. We now prove the theorem in each direction separately using Lemma 1 and Lemma 2 below.

Lemma 1. *Let $v : \{A, B\}$ and T be the same as in Theorem 2. If $v : \{A, B\} \sqsubseteq T$ then we have*

$$A \quad \Rrightarrow \quad [x_0 := x][T]B^w \ ,$$

where as usual x is "all variables", i.e. those occurring in A, B or T.

Proof. We begin the proof from the right-hand side. The first few lines for the proof is for the fact that the post-expectation in Def. 1 does not contain x_0. Also notice that the substitutions are right-associative:

$$\begin{array}{cll} & [x_0 := x][T]B^w & \\ \equiv & [x_0 := x]([x' := x_0][T][x_0 := x']B^w) & \begin{array}{r} x' \text{ are fresh variables} \\ T \text{ contains no } x_0 \end{array} \end{array}$$

$$
\begin{array}{lll}
\equiv & [x' := x][T]([x_0 := x']B^w) & \text{sequential substitution} \\
 & & \text{no } x_0 \text{ in } [T][x_0 := x']B^w \\
\Lleftarrow & [x' := x]([v : \{A, B\}][x_0 := x']B^w) & \text{monotonicity and assumption} \\
\equiv & [x' := x]\,(A \;*\; [x_0 := x]\,(\sqcap x \cdot [x_0 := x']B^w \;\div\; B^w)) & \text{from Def. 1} \\
\equiv & & \text{simple substitution } [x' := x] \\
 & A \;*\; [x' := x][x_0 := x]\,(\sqcap x \cdot [x_0 := x']B^w \;\div\; B^w) & \\
\equiv & A \;*\; [x_0 := x][x' := x_0]\,(\sqcap x \cdot [x_0 := x']B^w \;\div\; B^w) & [x_0 := x] \text{ is free of } x' \\
\equiv & A \;*\; [x_0 := x]\,(\sqcap x \cdot [x' := x_0][x_0 := x']B^w \;\div\; B^w) & \text{properties of } \sqcap \\
\equiv & A \;*\; [x_0 := x]\,(\sqcap x \cdot B^w \;\div\; B^w) & \text{sequential substitution} \\
 & & \text{no } x' \text{ in } B \\
\equiv & A\ , & \text{non-zero assumption on } B\text{, arithmetic}
\end{array}
$$

which completes the proof.

Lemma 2. *Let $v : \{A, B\}$ and T be the same as in Theorem 2. If*

$$A \quad \Rrightarrow \quad [x_0 := x][T]B^w \tag{6}$$

then we have

$$v : \{A, B\} \sqsubseteq T\ .$$

Proof. We begin by calculating the application of substitution $v : \{A, B\}$ to any expectation C which is free from x_0:

$$
\begin{array}{lll}
 & [v : \{A, B\}]C & \\
\equiv & A \;*\; [x_0 := x]\,(\sqcap x \cdot C \;\div\; B^w) & \text{Def. 1} \\
\Rrightarrow & [x_0 := x][T]B^w \;*\; [x_0 := x]\,(\sqcap x \cdot C \;\div\; B^w) & \text{Assumption (6)} \\
\equiv & [x_0 := x]\,([T]B^w \;*\; (\sqcap x \cdot C \;\div\; B^w)) & \text{simple substitution } [x_0 := x] \\
\equiv & [x_0 := x][T]\,((\sqcap x \cdot C \;\div\; B^w) * B^w) & T \text{ free from } x_0 \text{ and scaling } [T]\text{; see below} \\
\Rrightarrow & [x_0 := x][T]((C \;\div\; B^w) \;*\; B^w) & \text{monotonicity} \\
 & & (\sqcap x \cdot C \;\div\; B^w) \Rrightarrow C \;\div\; B^w \text{ as } C \text{ free from } x_0 \\
\equiv & [x_0 := x][T]C & \text{non-zero assumption on } B \\
\equiv & [T]C\ . & \text{both } T \text{ and } C \text{ free from } v_0
\end{array}
$$

Since C was arbitrary, we have that $v : \{A, B\} \sqsubseteq T$, which completes the proof.

For the deferred judgment, we using the scaling property of substitutions which states that multiplication by a non-negative constant distributes through substitutions [16].

5 Refining probabilistic specifications to loops

We now turn to our second major topic, the development of loops in *pB*. In following section we will show how loops and specification substitutions fit together.

We will first recall the proof obligations for standard loops, then apply the theorems stated in Sec. 4 in order to set out the generalised proof obligations for probabilistic loops.

5.1 Proof obligations for standard loops

For a standard loop, such as

$$\text{loop} \quad \widehat{=} \quad \textsc{while}\ G\ \textsc{do}\ S\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}\ ,$$

we recall the proof obligations for its correctness in the context of an initialisation which it occurs in a fragment: *init*; *loop*. Then we have that

$$P \Rightarrow [\mathit{init}; \mathit{loop}]\, Q$$

holds if the well-known variant-and-invariant rules are satisfied [7, 1].

S1: The invariant must hold before the while-test is made for the first time, which is formulated as: $P \Rightarrow [\mathit{init}]I$.
S2: The invariant is maintained by the loop body: $G \wedge I \Rightarrow [S]I$.
S3: When the loop ends, i.e. the while-test is false and the invariant is still true, the loop establishes the postcondition: $\neg G \wedge I \Rightarrow Q$.
S4: The invariant guarantees that the variant denotes a natural number, which is formulate as: $I \Rightarrow V \in \mathbb{N}$.
S5: The loop body decreases the variant: for some fresh variable n we have: $G \wedge I \quad \Rightarrow \quad [n := V][S](V < n)$.

5.2 Proof obligations for probabilistic loops

In setting up the proof obligations for *probabilistic loops*, we try to mimic the obligations for standard loops. We need to calculate the pre-expectation of a probabilistic substitution with respect to a particular post-expectation, which usually is a product of an embedded predicate[7] and another (general) expectation. The embedded predicate captures the normal invariant and the other deals with the quantitative property of the loop. We need to be able to separate them, for which we use “probabilistic conjunction operator”.

Recall the *probabilistic conjunction operator “&”* defined over the expectation space [15]:

$$(E\ \&\ F).x \quad \widehat{=} \quad (E.x + F.x - 1) \quad \sqcup \quad 0\ , \quad ^{8}$$

for expectations E, F and all $x \in X$. It is easy to see that “ &” is monotonic with respect to $\Rrightarrow$, and $\langle P \rangle * E \equiv \langle P \rangle\ \&\ E$, for predicate P and general expectation E. Moreover, from an earlier work [15], we know that for any probabilistic substitution S has the following *sub-conjunctivity* property:

$$[S](E\ \&\ F) \quad \Lleftarrow \quad [S]E\ \&\ [S]F\ , \tag{7}$$

for all expectations E, F (The properties of & operator can be seen in [16]).

We begin with a lemma that will allow us to deal with the standard and probabilistic expectations separately.

[7] Recall that an embedded predicate $\langle P \rangle$ is 1 if P holds and 0 otherwise.
[8] The definition of $\sqcup$ is: $a \sqcup b \mathrel{\widehat{=}} a$ max b

Lemma 3. *Let S be a probabilistic substitution written in* pGSL*; let P, Q be predicates; and let A, B be expectations. If we have*

$$\langle P\rangle \quad \Rrightarrow \quad [S]\,\langle Q\rangle \ \ , \textit{ and} \tag{8}$$

$$\langle P\rangle * A \quad \Rrightarrow \quad [S]B \ , \textit{ then we have} \tag{9}$$

$$\langle P\rangle * A \quad \Rrightarrow \quad [S](\langle Q\rangle * B) \tag{10}$$

Proof. We begin with the left-hand side:

$$\begin{array}{lll}
 & \langle P\rangle * A & \\
\equiv & \langle P\rangle * \langle P\rangle * A & \text{arithmetic} \\
\equiv & \langle P\rangle \ \& \ (\langle P\rangle * A) & \langle P\rangle \text{ is standard} \\
\Rrightarrow & [S]\,\langle Q\rangle \ \& \ [S]B & \text{(8), (9) and monotonicity of `` \&''} \\
\Rrightarrow & [S](\langle Q\rangle \ \& \ B) & \text{sub-conjunctivity (7)} \\
\equiv & [S](\langle Q\rangle * B) \ , & \langle Q\rangle \text{ is standard}
\end{array}$$

which completes the proof.

We now use probabilistic conjunction to explain the generalisation of Sec. 5.1 to probabilistic loops. In fact, we will just study one kind of probabilistic loops, whose partial correctness is probabilistic while its total correctness is absolutely trivial. Such loops can be written in *pGSL* as follows:

$$\text{loop} \quad \widehat{=} \quad \text{WHILE } G \text{ DO } S \text{ INVARIANT } I \text{ EXPECTATION } E \text{ VARIANT } V \text{ END} \ .$$

Assuming as before that the loop follows an initialisation, we will state and justify the proof obligations for its correctness with respect to the probabilistic implication

$$\langle P\rangle * A \quad \Rrightarrow \quad [\mathit{init};\mathit{loop}](\langle Q\rangle * B) \ ,$$

where A, B are expectations and P, Q are predicates [14, 16].

P1: The expectation E together with the invariant I must be bounded below by the pre-expectation A, with the precondition P, before the while-test is first made. This is precisely formulated as follows:

$$\langle P\rangle * A \quad \Rrightarrow \quad [\mathit{init}]\,(\langle I\rangle * E) \ .$$

According to Lemma 3, this can be achieved by the following two proof obligations:

P1a: The precondition P must guarantee that the invariant I is established before the while-test is made for the first time: $\langle P\rangle \Rrightarrow [\mathit{init}]\,\langle I\rangle$, or equivalent to

$$P \Rightarrow [\mathit{init}]\,I \ .$$

P1b: The expectation E must be bounded below by the pre-expectation A with the precondition P before the while-test is first made:

$$\langle P \rangle * A \Rrightarrow [init]E \ .$$

P2: The loop body cannot decrease the expected value of E with the invariant I and the guard G:

$$\langle G \wedge I \rangle * E \Rrightarrow [S](\langle I \rangle * E) \ .$$

According to Lemma 3, this is achieved by the following two proof obligations:

P2a: The invariant I must hold within the loop body with probabilistic choice substitution being treated as demonic — this is called *demonic retraction.* If $\llbracket S \rrbracket$ represents the demonic retraction of S [9], then this rule can be formulated by $\langle G \wedge I \rangle \Rrightarrow [S] \langle I \rangle$, or equivalent to

$$G \wedge I \Rightarrow \llbracket S \rrbracket I \ .$$

P2b: The expectation E must not decrease within the loop body , i.e. the operation within the loop body can not decrease the expectation E by the invariant I and the guard G:

$$\langle G \wedge I \rangle * E \Rrightarrow [S]\, E \ .$$

P3: When terminating, the loop establishes the post-expectation B with postcondition Q, i.e:

$$\langle \neg G \wedge I \rangle * E \Rrightarrow \langle Q \rangle * B \ .$$

According to Lemma 3 this can be achieved by the following:

P3a: When terminating, the loop establishes the post-condition Q, that is we have: $\langle \neg G \wedge I \rangle \Rrightarrow \langle Q \rangle$. We can rewrite this without embedding as:

$$\neg G \wedge I \Rightarrow Q \ .$$

P3b: When terminating, the loop establishes the post-expectation B:

$$\langle \neg G \wedge I \rangle * E \Rrightarrow B \ .$$

P4: The standard invariant guarantees that the variant denotes a natural number as is the case in standard rule:

$$I \Rightarrow V \in \mathbb{N} \ .$$

P5: The loop body decreases the variant as is the case in standard rule, but the probabilistic choice within the body is treated as demonic retraction:

$$G \wedge I \Rightarrow [n := V] \llbracket S \rrbracket (V < n) \ .$$

[9] This demonic retraction is defined in [11] as $\llbracket S \rrbracket I \equiv ([S] \langle I \rangle = 1)$. This is defined to take advantage of the fact that $[S] \langle I \rangle$ can only take values in $0, 1$, and can be easily calculated by replacing all probabilistic choice substitutions by non-deterministic ones.

6 Case study: randomised Min-Cut

In this section, we show how to use the theorems of Sec. 4.2 and Sec. 5.2 in practice to develop a probabilistic algorithm. We will be using the well known technique of "probabilistic amplification".

In particular we take the example of finding a Min-Cut of a graph, the smallest number of edges whose removal would disconnect the graph. The algorithm contains two parts: the first part is to find a Min-Cut probabilistically, but at low probability; and the second part is to use probabilistic amplification to improve the probable correctness of the algorithm.

We will first briefly describe the Min-Cut algorithm and the probabilistic amplification technique; then we discuss how to code the Min-Cut algorithm in B, and we look in particular at the proof obligations required.

6.1 Informal description of the Min-Cut algorithm: contraction

The Min-Cut algorithm operates on undirected and connected graphs. A *cut* is a set of edges such that if we remove just those edges, the graph will become disconnected.

Deterministic algorithms' complexities are often improved by randomisation, and Min-Cut is an example of that. The result for randomised algorithms is much better than for the deterministic one, especially for dense graphs [20].

The randomised algorithm consists of a number of "contraction" steps. In a *contraction*, two connected nodes are chosen randomly and merged together. The contracted graph then has one node less than the original one. It can be proved that the connectivity of the contracted graph is always no less than the original one and that any specific minimum cut in the original graph remains in the contraction with probability at least $\frac{N-2}{N}$ (where N is the number of nodes of the graph). This contraction is done repeatedly until there are only two nodes left. At that point the only cut left is the (multiple) edges connecting the last two nodes. This will therefore be the cut that is chosen.

The above contraction procedure does not guarantee to find the minimum cut for the original graph, but there is a non-zero lower bound of probability that it will. By multiplying the probabilities for the successive stages, we see that probability is at least

$$p(N) \quad = \quad \frac{N-2}{N} * \frac{N-3}{N-1} * .... * \frac{2}{4} * \frac{1}{3} \quad = \quad \frac{2}{N*(N-1)} \; ; \tag{11}$$

Further, independent repetitions of the process can reduce the probability that a witness (solution to the problem) is not found on any of the repetitions, using the *probabilistic amplification* technique we describe below.

Full details of this algorithm are given by Motwani and Raghavan [20].

6.2 Probabilistic amplification

Intuitively, because it is difficult to find solutions in a search space which contains a large number of witnesses, it often suffices to choose an element at random from the space. The randomly chosen element is likely to be a witness; further, independent repetitions of the process reduce the probability that a witness is not found on any of the repetitions. This improvement is known as *probabilistic amplification.*

As we saw at (11) above, the probability of finding the minimum cut in one test is quite small. For $N = 10$, it would be $\frac{2}{10*9}$, that is approximately 2%. In order to improve that, we use probabilistic amplification to find the minimum cut repeatedly. The probability that we find the right minimum cut is the probability that the minimum cut is found in any one of those tests, which for M tests is at least is $P(N, M) = 1 - (1 - p(N))^M$, where $p(N)$ is as above.

For example, if we run the $N = 10$ case 120 times, the error probability would only be around 10%, that is, our probability of sucess is increased from 2% to $100 - 10 = 90\%$.

6.3 Formal development of contraction

In this section, we will see how the contraction steps are specified, and then implemented in *pGSL*; and we see the proof obligations for preserving the refinement relationship between the specification and the implementation.

Specification of contraction We look at the specification of the contraction, i.e. of the one test to find the minimum cut. Since the probability of the outcome for the test only depends on the number of nodes in the graph, we can take an abstract view for the specification. The machine (program) has one operation to model one test, with the input N being the number of nodes for the original graph. The output *ans* is *TRUE* when we have found the right minimum cut, and *FALSE* otherwise. In this specification, we want to state that for any input N, the probability that the output *ans* is *TRUE* on termination is at least $\frac{2}{N*(N-1)}$ (as at (11)). The specification is shown below:

$$\mathit{ans} \longleftarrow \textbf{contraction}(\ N\ ) \qquad \widehat{=} \qquad \mathit{ans} : \{\langle N \in \mathbb{N} \wedge 2 \leq N \rangle * p(N), \langle \mathit{ans} \rangle\}$$

An implementation of contraction A loop implementation of the contraction using *pGSL* is given in Fig. 2. In this implementation, we have a local variable n to keep the number of nodes in the current graph, and so we start with $n = N$ (original graph). At each stage, variable *ans* is *TRUE* just when the actual Min-Cut has not yet been destroyed by any merge so far. We keep merging while the number of nodes is greater than 2. The operation *merge(n, ans)* is specified in the machine merge as below.

$$\mathit{ans} \longleftarrow \textbf{merge}\ (\ n\ ,\ a\ ) \qquad \widehat{=} \qquad n \in \mathbb{N} \wedge a \in \mathit{BOOL} \mid \mathit{ans} := \mathit{FALSE} \ {}_{\leq \frac{2}{n}}\!\oplus\ a$$

The operation says that with probability at most $\frac{2}{n}$, the minimum cut will be destroyed by the contraction. Otherwise, if the minimum cut has not been destroyed, it will be kept.

```
$ans \longleftarrow$ contraction( $N$ ) $\widehat{=}$
VAR $n$ IN
    $n := N$; $ans := TRUE$;
    WHILE $2 < n$ DO
        $ans \longleftarrow merge(n, ans)$;        /* Select two nodes and merge */
        $n := n$ - 1
    INVARIANT    $n \in \mathbb{N} \;\wedge\; n \leq N \;\wedge\; 2 \leq n \;\wedge\; ans \in BOOL$
    EXPECTATION    $(2 \div (n * (n-1))) \;*\; \langle ans \rangle$
    END
END
```

Fig. 2. Implementation of contraction in *pGSL*

Proof obligations of contraction We now will apply the generalised proof obligations for the probabilistic loop to prove the correctness of the implementation to the specification of the contraction process.

To prove the refinement relationship between the specification and the implementation in Fig. 2 of the contraction process, Theorem 2 is applied to those programs. We thus have to prove that

$$\begin{array}{rl} & \langle N \in \mathbb{N} \wedge 2 \leq N \rangle * p(N) \\ \Rrightarrow & [ans_0 := ans][contraction]\,\langle ans \rangle \ , \end{array}$$

which can be simplified to

$$\langle N \in \mathbb{N} \wedge 2 \leq N \rangle * p(N) \quad \Rrightarrow \quad [contraction]\,\langle ans \rangle \ ,$$

where we have used the fact that there is no ans_0 on the right-hand side, so that the substitution $[ans_0 := ans]$ is redundant. Further more, we have used the fact that proving $\langle P \rangle * E \Rrightarrow F$ is equivalent to prove $E \Rrightarrow F$ under the assumption that P holds, so it is necessary to prove: $p(N) \Rrightarrow [contraction]\,\langle ans \rangle$, given that $N \in \mathbb{N} \wedge 2 \leq N$, which means that the implementation succeeds with probability (finding the right minimum cut) at least $p(N)$.

We first state all the components of the loop within our reasoning context. Referring to Sec. 5.2, we have the following information.

- The initialisation for the loop is: $init_1 \widehat{=} n := N; ans := TRUE$.
- The standard invariant is: $I_1 \widehat{=} n \in \mathbb{N} \wedge n \leq N \wedge 2 \leq n \wedge ans \in BOOL$.
- The guard for the loop is: $G_1 \widehat{=} 2 < n$.
- The body of the loop is: $S_1 \widehat{=} ans \longleftarrow merge(n, ans);\ n := n - 1$.
- The expectation (probabilistic invariant) is: $E_1 \widehat{=} \frac{2}{n*(n-1)} * \langle ans \rangle$.
- The precondition is: $P_1 \widehat{=} N \in \mathbb{N} \wedge 2 \leq N$.
- The pre-expectation is: $A_1 \widehat{=} p(N)$.
- The postcondition Q_1 is the constant predicate *true*.
- The post-expectation is $B_1 \widehat{=} \langle ans \rangle$.

$$
\begin{array}{l}
ans \longleftarrow \mathbf{minCut}(\ N,\ M\ )\ \widehat{=} \\
\quad ans : \{\langle N \in \mathbb{N} \wedge 2 \leq N \wedge M \in \mathbb{N}_1\rangle * P(N, M), \langle ans\rangle\}
\end{array}
$$

Fig. 3. Specification of probabilistic amplification

From this information there are 14 proof obligations for the implementation of the contraction, all of which have been proved using the *B-Toolkit* with some extra proof rules.

Proving the obligations The proofs of the obligations can be found in [9].

6.4 Formal development of probabilistic amplification

In this section, we use probabilistic amplification in order to increase the probability of finding the minimum cut. We will again look at the specification and its implementation, and then look at the proof obligations for the refinement step. We will see that a slightly more specialised version of the probabilistic specification (and its fundamental theorem) is necessary for developments of this kind.

Specification of Min-Cut probabilistic amplification The machine has only one operation, namely minCut. This operation has two inputs: they are N for the number of nodes in the original graph and M for the number of times that we do the amplification, i.e. the number of times that the contraction process is used. The output *ans* of the operation abstractly models whether we find the right minimum cut or not after having done one amplification step. The specification states that the probability of finding the correct minimum cut should be at least $P(N, M) \;=\; 1 - (1 - p(N))^M$. The specification is shown in Fig. 3.

Implementation of Min-Cut probabilistic amplification The implementation of the probabilistic amplification is shown in Fig. 4. In the implementation, we have two auxiliary variables m and a, which represent the counter and the recent output from the contraction process, respectively. Initially, m is assigned M and *ans* is assigned *FALSE*, which means that we intend to repeat the test process M times and initially have not found the right minimum cut yet. In the body of the loop, the contraction process is taken and its result is returned in a; then *ans* is the disjunction of the new result a and the old *ans* (since if we find the correct (least) cut once, we can never lose it); and finally, the counter decreases accordingly.

```
$ans \longleftarrow$ minCut( $N$, $M$ ) $\widehat{=}$
VAR $m$, $a$ IN
    $m := M$; $ans := FALSE$;
    WHILE $m \neq 0$ DO
        $a \longleftarrow contraction(N)$;
        $ans := ans \vee a$;
        $m := m - 1$
    INVARIANT $m \in \mathbb{N} \wedge m \leq M \wedge ans \in BOOL$
    EXPECTATION $\langle ans \rangle + \langle m \neq 0 \rangle * \langle \neg ans \rangle * P(N, m)$
    END
END
```

Fig. 4. Probabilistic implementation of the specification in Fig. 3

Proof obligations of Min-Cut probabilistic amplification Again, we will apply the generalised proof obligations for the probabilistic loop to prove the correctness of the implementation to the specification of probabilistic amplification. To prove the refinement relationship, Theorem 2 is applied between the programs in Fig. 3 and Fig. 4; it states that we must show

$$\begin{array}{ll} & \langle N \in \mathbb{N} \ \wedge \ 2 \leq N \ \wedge \ M \in \mathbb{N}_1 \rangle * P(N, M) \\ \Rrightarrow & [ans_0 := ans][minCut]\, \langle ans \rangle \end{array} \qquad (12)$$

in order to establish the refinement. The implication (12) can be simplified to

$$\langle N \in \mathbb{N} \ \wedge \ 2 \leq N \ \wedge \ M \in \mathbb{N}_1 \rangle * P(N, M) \quad \Rrightarrow \quad [minCut]\, \langle ans \rangle \ ,$$

by noting that there is no ans_0 on the right-hand side. Also, we again separate the standard predicate and expectation, i.e. we will prove $P(N, M) \Rrightarrow [minCut]\, \langle ans \rangle$, under the assumption that $N \in \mathbb{N} \ \wedge \ 2 \leq N \ \wedge \ M \in \mathbb{N}_1$.

We first state all the components of the loop within our reasoning context. Referring to Sec. 5.2, we have the following information.

- The initialisation for the loop is: $init_2 \mathrel{\widehat{=}} m := M; ans := FALSE$.
- The standard invariant is: $I_2 \mathrel{\widehat{=}} m \in \mathbb{N} \ \wedge \ m \leq M \ \wedge \ ans \in BOOL$.
- The guard for the loop is: $G_2 \mathrel{\widehat{=}} m \neq 0$.
- The body of the loop is

$$S_2 \quad \widehat{=} \quad ans \longleftarrow contraction(N); \ ans := ans \vee a; m := m - 1 \ .$$

- The expectation (probabilistic invariant) is:

$$E_2 \mathrel{\widehat{=}} \langle ans \rangle + \langle m \neq 0 \rangle \ * \ \langle \neg ans \rangle * P(N, m) \ .$$

- The precondition is': $P_2 \mathrel{\widehat{=}} N \in \mathbb{N} \ \wedge \ 2 \leq N \ \wedge \ M \in \mathbb{N}_1$.
- The pre-expectation is $A_2 \mathrel{\widehat{=}} P(N, M)$.

- The postcondition Q_2 is the constant predicate *true*.
- The post-expectation is: $B_2 \mathrel{\widehat{=}} \langle ans \rangle$.

Here, we concentrate only on the proving of *P2b*; the other proofs can be seen elsewhere [9]. We find that the obligation *P2b* cannot be proved because of termination (details also can be seen in [9]). This is not surprising in retrospect, because for example a specification $v : \{p, \langle Q \rangle\}$ ensures termination in state satisfied Q with probability p only; with probability $1 - p$, abortion is possible. Here, we must ensure additionally that in the latter case, termination occurs although we do not care about the postcondition in that case. We address this briefly in the next section.

6.5 The "terminating" probabilistic specification substitution

In order to avoid the problem revealed in the last section, we would have to introduce the concept of "terminating probabilistic specification substitution" and a corresponding fundamental theorem for it as well.

To do that we would consider a special case of the probabilistic substitution, where the post-expectation B is standard, i.e is $\langle Q \rangle$ for some predicate Q, and the pre-expectation A is the probability — still a function of the state — that Q will be achieved. For consistency with probability elsewhere, we use lower-case p for pre-expectation.

Definition 2. *Let p be a probabilistic expression over x and free from x_0; Q a predicate defined over x_0, v and satisfying $\forall\, x_0 \cdot (\exists\, v \cdot Q)$. The specification $v : \{\!\{p, \langle Q \rangle\}\!\}$ is defined by:*

$$v : \{\!\{p, \langle Q \rangle\}\!\} \quad \mathrel{\widehat{=}} \quad v : \{1, \langle Q \rangle\} \;\; {}_p\!\oplus \;\; x : \{1, 1\} \ . \tag{13}$$

And accordingly, we introduce the fundamental theorem for the above substitution as follows.

Theorem 3. *Let p be an expression over x and let Q be a predicate defined over x_0, v, and satisfying $\forall\, x_0 \cdot (\exists\, v \cdot Q)$; and T a program written in* pGSL*. For all such programs T, if $x : \{1, 1\} \sqsubseteq T$ and $v : \{p, \langle Q \rangle\} \sqsubseteq T$ then $v : \{\!\{p, \langle Q \rangle\}\!\} \sqsubseteq T$.*

With the new terminating version of the specification and fundamental theorem, we can reconstruct and prove all the proof obligations for the implementation of probabilistic amplification, which can be seen in [9].

We now can prove the correctness of the refinement for the contraction steps and the probabilistic amplification technique, both containing probabilistic specification substitution.

7 Conclusion, and challenges

We have taken a "second step" into the probabilistic-B world, by adding above our earlier work [8] the "superstructure" required for following the specify-refine-code path embodied in the *B-Method*. We have been successful in the sense that the new constructs are shown to be well defined, and interact properly with each other. In addition, the case-study is not completely trivial.[10]

The approach follows earlier work by Neil White, in his MSc thesis at Oxford [21], transporting the ideas from Z into B.

The B context however provides a number of new challenges, some of which we have addressed here. The issue of *separation* of standard reasoning from probabilistic reasoning is (or will be) of crucial importance if the probabilistic B is to handle developments of anything like the same size and scope as standard B. And the "terminating" specification substitution (mentioned here in the case study) will probably become the one used in practice.

8 Acknowledgments

We wish to acknowledge the assistance of B-Core [5] for the modification of the B-Toolkit.

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

References

1. Jean-Raymond Abrial. *The B-Book*. Cambridge University Press, 1996.
2. Jean-Raymond Abrial, Dominique Cansell, and Dominique Mery. A mechanically proved and incremental development of IEEE 1394 firewire tree identify protocol. *Formal Aspects of Computing*, 14(3):215–227, 2003.
3. Ralph-Johan Back. *On the correctness of refinement in program development*. PhD thesis, Department of Computer Science, University of Helsinki, 1978. Report A-1978-4.
4. Ralph-Johan Back and Joakim Von Wright. *Refinement Calculus, a Systematic Itroduction*. Springer-Verlag New York, Inc., 1998.
5. B Core(UK) Ltd. B Toolkit. `http://www.b-core.com`.
6. Colin J. Fidge and Carron Shankland. But what if I don't want to wait forever? *Formal Aspects of Computing*, 14(3):281–294, 2003.
7. David Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2:207–214, 1984.
8. Thai Son Hoang, Zhendong Jin, Ken Robinson, Annabelle McIver, and Carroll Morgan. Probabilistic Invariants for Probabilistic Machines. In D. Bert, J. P. Bowen, S. King, and M. Waldén, editors, *ZB2003: Formal Specification and Development in Z and B, Proceedings of the 3rd International Conference of B and Z Users*, volume 2651 of *LNCS*, Turku, Finland, June 2003. Springer-Verlag.
9. Thai Son Hoang, Zhendong Jin, Ken Robinson, Annabelle McIver, and Carroll Morgan. Proofs of the Min-Cut development. `http://www.cse.unsw.edu.au/~htson/b/minCutProofs.pdf`, April 2004.
10. INRIA. The Coq proof assistant. `http://coq.inria.fr/`.

[10] It was suggested to us by a case-study of the same algorithm done in the theorem-proving environment Coq [10] by Christine Paulin of the *LRI* in Paris.

11. Annabelle McIver, Carroll Morgan, and Thai Son Hoang. Probabilistic termination in B. In D. Bert, J. P. Bowen, S. King, and M. Waldén, editors, *ZB2003: Formal Specification and Development in Z and B, Proceedings of the 3rd International Conference of B and Z Users*, volume 2651 of *LNCS*, Turku, Finland, June 2003. Springer-Verlag.
12. Carroll Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3), July 1988. Reprinted in [18].
13. Carroll Morgan. *Programming from Specifications.* Prentice-Hall, second edition, 1994. At `web.comlab.ox.ac.uk/oucl/publications/books/PfS`.
14. Carroll Morgan. Proof rules for probabilistic loops. In He Jifeng, John Cooke, and Peter Wallis, editors, *Proceedings of the BCS-FACS 7th Refinement Workshop*, Workshops in Computing. Springer-Verlag, July 1996.
15. Carroll Morgan. The Generalised Substitution Language extended to probabilistic programs. In *Proceedings B'98: the 2nd International B Conference*, volume 1393 of *LNCS*, Montpelier, April 1998.
16. Carroll Morgan and Annabelle McIver. *Abstraction, Refinement and Proof for Probabilistic Systems.* Monographs in Computer Science. Springer-Verlag, 2004.
17. Carroll Morgan and Ken Robinson. *On the Refinement Calculus*, chapter Specification Statements and Refinements, pages 23–45. Springer-Verlag, 1992.
18. Carroll Morgan and Trevor Vickers, editors. *On the Refinment Calculus.* FACIT Series in Computer Science. Springer-Verlag, Berlin, 1994.
19. J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9(3):287–306, December 1987.
20. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
21. Neil White. Probabilistic Specification and Refinement. Master's thesis, Keble College, September 1996.