



Proof rules for probabilistic loops

Carroll Morgan*

15 August 1996

Abstract

Probabilistic predicate transformers provide a semantics for imperative programs containing both demonic and probabilistic nondeterminism. Like the (standard) predicate transformers popularised by Dijkstra, they model programs as functions from final results to the initial conditions sufficient to achieve them.

This paper presents practical proof rules, using the probabilistic transformers, for reasoning about iterations when probability is present. They are thoroughly illustrated by example: probabilistic binary chop, faulty factorial, the *martingale* gambling strategy and Herman's *probabilistic self-stabilisation*.

Just as for traditional programs, weakest-precondition based proof rules for program derivation are an important step on the way to designing more general refinement techniques, or even a refinement calculus, for imperative probabilistic programming.

1 Introduction

The standard predicate transformers described by Dijkstra [3] provide a model in which a program is a function: it takes a set of desired final states to the set of all initial states from which the program's execution is guaranteed to produce one of those final states. Regarding sets of states as predicates over the state space, programs are thus predicate transformers.

A conspicuous feature of Dijkstra's presentation was the appearance of demonic nondeterminism, a form of choice in programs over which the users have no control and about which nothing can be predicted. It arises naturally in the predicate-transformer approach, and as a result benefits from a particularly simple treatment there.

In the work of Kozen [12] demonic nondeterminism is replaced by probabilistic nondeterminism. Probabilistic nondeterminism is not controllable by the user (either), but it is to some extent predictable: in repeated runs of a program

$$coin := heads \quad \frac{1}{2} \oplus \quad coin := tails ,$$

one would have the same expectations about the final value of the program variable *coin* as one would have about the repeated flipping of a real coin.

We have extended the above [18], presenting a system in which demonic and probabilistic nondeterminism are treated together in a simple way: as well as building on the original work of

*Morgan is a member of the Probabilistic Systems Group within the Programming Research Group at Oxford University; the other members are Annabelle McIver, Jeff Sanders and Karen Seidel. Our work is supported by the EPSRC.

Dijkstra and of Kozen, we took advantage of later work by Claire Jones and Gordon Plotkin [10] and a ‘relational’ probabilistic model proposed by JiFeng He [7] (who used ‘convex closure’ [19] to generalise an earlier imperative model due to Kozen [11]).

One of the principal results of our earlier work [18] is the exact determination of the ‘healthiness conditions’ that apply to probabilistic predicate transformers; they generalise the conditions given by Dijkstra for standard predicate transformers.

Our overall aim is to broaden the scope of refinement methods to include more aspects of ‘real’ system design, in this case that the ultimate components from which a system is built are never entirely reliable. When their unreliability can be quantified, probabilistic program derivation, or refinement, can be used to match low-level unreliability of components to high-level ‘tolerable’ unreliability in a specification.

The contribution of this paper specifically is to use the probabilistic healthiness conditions to propose and justify methods for the treatment of probabilistic loops; in that way we move the theory [18] towards everyday practice. The main theorems concern probabilistic invariants and variants, and generalise the corresponding standard theorems; our probabilistic healthiness conditions are crucial to their proofs, and to the separate treatment of partial and total correctness.

Informally, the use of invariants is just as in standard programs, based on the work of Hoare and Floyd [9, 4]: the invariant is established initially; it is maintained; and on termination additionally the negation of the repetition condition holds. Here however we use probabilistic invariants, as anticipated by Kozen, by Sharir, Pnueli and Hart [24], and finally by Jones [12, 10]; we have generalised their work by treating nondeterminism as well.

The probabilistic variant rule (and the related ‘0-1 Law’) was earlier proposed by Hart, Sharir and Pnueli [6] and shown to be sound and finitarily complete: a variant function must be bounded above and below, and have a nonzero probability of decrease. Our contribution here is to express that rule at the level of probabilistic predicate transformers, reproducing the proofs of soundness and finitary completeness in that context. We achieve a slight generalisation in that catastrophic failure (divergence, or **abort**) is included naturally as a possible behaviour of programs in our model.

Sections 3–4 give the main theorems for the use of invariants and the way in which they are combined with information about loop termination; they are illustrated by the examples of Sec. 5, chosen to reveal the various combinations of probabilistic and standard variants and invariants. Sections 6–8 treat termination on its own. Section 9 provides a final example, a recent ‘showcase’ for probabilistic formalisms in which certain termination is the principal feature.

2 Probabilistic predicate transformers

Standard predicates are sets of states, and can thus be regarded as characteristic functions from the state space to $\{0, 1\}$. In practice — that is, for reasoning about specific programs — they are written as Boolean-valued expressions (formulae) over program variables.

Probabilistic predicates are functions from the state space to the entire closed interval $[0, 1]$.¹ In practice they are written as real-valued expressions over the program variables.

¹Elsewhere [18] we take a more general but equivalent view, that they are functions into the non-negative reals $[0, \infty)$.

The manipulation of the predicate transformers in the two systems — standard and probabilistic — is very similar. For example, in both cases assignment is syntactic substitution, sequential composition (of programs) is functional composition (of the predicate transformers) and recursion is given by least fixed points. For a full presentation we refer the reader to our other publications [23, 18].²

Because symbols may be confused in the probabilistic case, however, we adopt the following notational conventions to separate them as much as possible.

Notation 2.1 Standard predicates are Boolean expressions over the state variables, and are written in the normal way. (We use \Leftrightarrow for bi-implication.)

Probabilistic predicates are real-valued expressions between 0 and 1 inclusive. The brackets $[\cdot]$ convert a standard predicate to a probabilistic predicate so that $[true]$ is (the constant expression) 1 and $[false]$ is 0. The overbar operator denotes subtraction from 1, so that $[\overline{P}]$ is the same as $[\neg P]$ for standard predicate P .

Minimum and maximum are written \sqcap, \sqcup respectively, with \sqcap binding more tightly.

The relations ‘everywhere no more than’, ‘everywhere equal’ and ‘everywhere no less than’ between probabilistic predicates are written \Rightarrow, \equiv and \Leftarrow respectively. \square

With the above conventions we have for example that

$$1/2 \quad \Rightarrow \quad [x \leq 0]/2 + [x \geq 0]/2$$

because for all values of the state-variable x the right-hand side is at least $1/2$. However the stronger claim

$$1/2 \quad \equiv \quad [x \leq 0]/2 + [x \geq 0]/2$$

is false, since when x is 0 the left-hand side is $1/2$ but the right-hand side is 1.

The basic properties of predicate transformers needed for our presentation are collected in App. B, and are referred to here as ‘facts’. Those concerning wp are consequences of the healthiness laws [18]. We make essential use also of weakest *liberal* probabilistic preconditions in some of our proofs; the facts concerning them are proved elsewhere [21]. Since wlp does not appear in the statements of the principal theorems, however, the wlp -theory is not needed for use of our results.

Notation 2.2 We write $f.x$ for the function f applied to the argument x (rather than $f(x)$). The application operator $.$ is left associative. \square

Notation 2.3 We write $:=$ for ‘is defined to be’. \square

3 Partial loop correctness

The *weakest liberal precondition* of a program describes its *partial* correctness, identifying those initial states from which the program either establishes a given postcondition or fails to terminate [3]. The more conventional *weakest* (not liberal) *precondition* requires termination as well, and thus describes *total* correctness. We write $wlp.prog.Q$ and $wp.prog.Q$ for the weakest liberal and weakest preconditions respectively of program $prog$ and postcondition Q .

²Section 4 of Morgan et al. [18] summarises the probabilistic wp -rules (Fig. 2 there) and treats some elementary examples. Section 7 (Fig. 4) collects the healthiness conditions.

The *wlp* semantics differs from the *wp* in these two respects.³

1. The nowhere-terminating program is defined $wlp.\mathbf{abort}.Q := 1$ for all postconditions Q . (Compare $wp.\mathbf{abort}.Q := 0$.)
2. The weakest liberal precondition semantics of a recursive program is given by a *greatest* (rather than least) fixed point.

When considering loops, a special case of recursion, the *wlp* semantics is therefore as given in Def. 3.3 following.

Notation 3.1 The program *loop* is defined

$$loop \quad := \quad \mathbf{do} \ G \rightarrow \mathit{body} \ \mathbf{od} \ ,$$

for standard predicate G (the loop guard) and program *body* (the loop body).

In calculations we always treat G as a probabilistic predicate, even though it takes only standard values, thus avoiding clutter by omitting the brackets $[G]$. □

Notation 3.2 We use η to indicate greatest fixed point. □

Definition 3.3 *Weakest liberal precondition for loop.* For any postcondition Q we define

$$wlp.loop.Q \quad := \quad (\eta P \cdot G \sqcap wlp.body.P \sqcup \overline{G} \sqcap Q) \ .$$

□

From Def. 3.3 we derive immediately the usual rule for partial correctness of loops, based on the preservation of an invariant.

Lemma 3.4 Let predicate I be a *wlp-invariant* of *loop*, thus satisfying

$$G \sqcap I \quad \Rightarrow \quad wlp.body.I \ . \tag{1}$$

Then in fact

$$I \quad \Rightarrow \quad wlp.loop.(\overline{G} \sqcap I) \ . \tag{2}$$

Proof: We substitute I for P in the right-hand side of Def. 3.3, setting $Q := \overline{G} \sqcap I$, and find

$$\begin{aligned} & G \sqcap wlp.body.I \sqcup \overline{G} \sqcap (\overline{G} \sqcap I) \\ \Leftarrow & G \sqcap (G \sqcap I) \sqcup \overline{G} \sqcap (\overline{G} \sqcap I) && \text{assumption} \\ \equiv & I \ , && G \text{ standard} \end{aligned}$$

obtaining the result immediately from the elementary property of greatest fixed points that $x \leq f.x$ implies $x \leq \eta.f$. □

³For uniformity we use probabilistic predicates throughout, even for standard programs, writing $wp.(x := y).[x = y] \equiv 1$ for example rather than the conventional $wp.(x := y).(x = y) \equiv true$.

It is worth noting that the assumption (1) of Lem. 3.4 is weaker than the one used in the standard rule for partial correctness of loops, where one conventionally finds wp instead:

$$G \sqcap I \quad \Rightarrow \quad wp.body.I . \quad (3)$$

The difference is real, even in the standard case, but only if we are genuinely interested in partial correctness. With Lem. 3.4 we can show for example

$$wlp.(do\ x \neq 0 \rightarrow \mathbf{abort\ od}).[x = 0] \quad \equiv \quad 1 , \quad (4)$$

choosing $I := 1$ to do so: if the loop terminates then it establishes $x = 0$.

The reason that (3) is used in the standard case is that it suffices for total correctness of the loop, and avoids introducing the extra concept of wlp : if indeed $I \Rightarrow wp.loop.1$ then we must have $G \sqcap I \Rightarrow wp.body.1$ in any case, making (1) and (3) equivalent.

For probabilistic programs the above analysis does not apply,⁴ and as shown below use of the stronger (3) is required for soundness in general (Ex. 4.7).

4 Total loop correctness

In the standard case Fact B.1 is used to combine partial loop correctness with a termination argument, to give total loop correctness. Here we rely on its probabilistic analogue Fact B.2.

Notation 4.1 The binary operator $\&$ is defined

$$Q_0 \& Q_1 \quad := \quad (Q_0 + Q_1 - 1) \sqcup 0 .$$

□

It is easily checked that $Q_0 \& Q_1 \equiv Q_0 \sqcap Q_1$ when either Q_0 or Q_1 is standard, and thus that Fact B.1 results when Fact B.2 is specialised to $Q_0, Q_1 := Q, 1$ for standard Q . We have further that $\&$ is commutative and associative with identity 1.

With Fact B.2 and Lem. 3.4 we have immediately a rule for total correctness of probabilistic loops.

Notation 4.2 The termination condition of *loop* is defined

$$T \quad := \quad wp.loop.1 .$$

□

⁴The reasoning fails at the point of concluding $G \sqcap I \Rightarrow wp.body.I$ from

$$G \sqcap I \Rightarrow wp.body.1 \quad \text{and} \quad G \sqcap I \Rightarrow wlp.body.I .$$

Applying Fact B.2 to those two inequalities gives only

$$wp.body.I \equiv wp.body.(1 \& I) \Leftarrow wlp.body.I \& wp.body.1 \Leftarrow (G \sqcap I) \& (G \sqcap I) \equiv G \sqcap (I \& I) ,$$

strictly weaker than the above when I is probabilistic since in that case $I \& I \neq I$. Note for example that

$$\begin{aligned} 1/2 &\equiv wlp.(x := 0 \text{ }_{1/2} \oplus \mathbf{abort}).[x = 1] \\ \text{and } 1/2 &\equiv wp.(x := 0 \text{ }_{1/2} \oplus \mathbf{abort}).1 \end{aligned}$$

do not imply $1/2 \equiv wp.(x := 0 \text{ }_{1/2} \oplus \mathbf{abort}).[x = 1]$ — which indeed is not true.

Lemma 4.3 Let invariant I satisfy $G \sqcap I \Rightarrow wp.body.I$. Then

$$I \& T \quad \Rightarrow \quad wp.loop.(\overline{G} \sqcap I) .$$

Proof:

$$\begin{aligned} & wp.loop.(\overline{G} \sqcap I) \\ \equiv & wp.loop.((\overline{G} \sqcap I) \& 1) \\ \Leftarrow & wlp.loop.(\overline{G} \sqcap I) \& wp.loop.1 && \text{Fact B.2} \\ \Leftarrow & I \& T . && \text{assumption, Lem. 3.4} \end{aligned}$$

□

Lemma 4.3 suffices for many situations, in particular those in which either I or T is standard since in that case $I \& T \equiv I \sqcap T$. When both I and T are probabilistic, however, the precondition of Lem. 4.3 can be too low (pessimistic, though still correct). But as the following⁵ shows, we cannot just replace $\&$ by \sqcap on the left-hand side.

Example 4.4 Take invariant $I := [n = 0]/2 + [n = 1]$ in the program *loop*, defined

```
do n = 0 → n := -1  $\frac{1}{2}$  ⊕ n := +1
| n > 0 → skip
od .
```

Then we have

$$\begin{aligned} T & \equiv [n < 0] + [n = 0]/2 \\ I \sqcap T & \equiv [n = 0]/2 \\ I \sqcap \overline{G} & \equiv 0 , \end{aligned}$$

but $I \sqcap G \equiv [n = 0]/2 \not\Rightarrow wp.loop.0 \equiv wp.loop.(\overline{G} \sqcap I)$. □

Thus we improve Lem. 4.3 in a different way, below, where for simplicity we assume that *body* is deterministic.⁶ The strategy is to develop a larger invariant I' than the I we are given, so that when we eventually form the precondition $I' \& T$ we recover the original I .

First we show strict *wp*-invariance of T itself.

Lemma 4.5 For all *loop* we have $G \sqcap T \equiv G \sqcap wp.body.T$.

Proof: We calculate

$$\begin{aligned} & G \sqcap T \\ \equiv & G \sqcap wp.loop.1 \\ \equiv & G \sqcap (G \sqcap wp.body.(wp.loop.1) \sqcup \overline{G} \sqcap 1) && \text{wp-semantics of } \mathbf{do} \cdots \mathbf{od} \\ \equiv & G \sqcap wp.body.T . && G \text{ standard} \end{aligned}$$

⁵Annabelle McIver suggested this possibility, and found the counterexample.

⁶We recall [18] that *deterministic* means ‘contains no demonic nondeterminism unless aborting’, so that for example

```
x := 0  $\frac{1}{2}$  ⊕ abort
```

is deterministic (as is **abort** itself). Syntactically, deterministic means (roughly) ‘contains no explicit nondeterministic choice operation’; our excuse for calling (for example) a coin-flipping program deterministic is that the *distribution* of its results is predictable. (Also, it is maximal in the refinement ordering.)

□

We now prove our main theorem for total correctness of deterministic loops; note that we assume a *wp*-invariance property (stronger than the *wlp*-invariance assumption of Lem. 4.3).

Theorem 4.6 If I is a *wp*-invariant of *loop* with deterministic *body*, and $I \Rightarrow T$, then

$$I \Rightarrow wp.loop.(\overline{G} \sqcap I) .$$

Proof: We show first that *wp*-invariance of I implies *wlp*-invariance of

$$I' := I + 1 - T .$$

Note we rely on $I \Rightarrow T$ for well-definedness (that $I' \Rightarrow 1$). We reason

$$\begin{array}{ll}
& wlp.body.I' \\
\equiv & wlp.body.(I + (1 - T)) \qquad \text{definition } I' \\
\equiv & wp.body.I + wlp.body.1 - wp.body.T \qquad \text{Fact B.3 twice; } body \text{ deterministic} \\
\equiv & wp.body.I + 1 - wp.body.T \qquad \text{Fact B.4} \\
\Leftarrow & G \sqcap (wp.body.I + 1 - wp.body.T) \\
\equiv & G \sqcap wp.body.I + G - G \sqcap wp.body.T \qquad G \text{ standard} \\
\equiv & G \sqcap wp.body.I + G - G \sqcap T \qquad \text{Lem. 4.5} \\
\Leftarrow & G \sqcap (G \sqcap I) + G - G \sqcap T \qquad \text{assumed } wp\text{-invariance of } I \\
\equiv & G \sqcap (I + 1 - T) \qquad G \text{ standard} \\
\equiv & G \sqcap I' . \qquad \text{definition } I'
\end{array}$$

From Lem. 4.3 we then conclude immediately

$$I \equiv I' \& T \Rightarrow wp.loop.(\overline{G} \sqcap I') \equiv wp.loop.(\overline{G} \sqcap I) ,$$

since for the last step we have

$$\begin{array}{ll}
& \overline{G} \sqcap I' \\
\equiv & \overline{G} \sqcap (I + 1 - T) \\
\equiv & \overline{G} \sqcap I + \overline{G} - \overline{G} \sqcap T \qquad G \text{ standard} \\
\equiv & \overline{G} \sqcap I . \qquad \overline{G} \text{ implies immediate termination: thus } \overline{G} \equiv \overline{G} \sqcap T
\end{array}$$

□

Thm. 4.6 is extended to the nondeterministic case by Thm. A.3, and it is not hard to show that the latter in turn implies Lem. 4.3: thus they are of equal power.

The following example shows the *wp*- (rather than *wlp*-) invariance of the invariant I to be necessary for soundness of Thm. 4.6 in general. (Recall from Sec. 3 that it is not necessary in the standard case.)

Example 4.7 For this example let *loop* be

```

do b →
  b := false 1/2 ⊕ abort
od ,

```

for Boolean b , and note that we have for termination

$$T \equiv [-b] + [b]/2 .$$

Define $I := 1/2$, so that $I \Rightarrow T$ as required by Thm. 4.6, and reason

$$\begin{aligned}
& wlp.body.I \\
\equiv & wlp.(b := false_{1/2} \oplus \mathbf{abort}).(1/2) \\
\equiv & (1/2)(wlp.(b := false).(1/2)) + (1/2)(wlp.\mathbf{abort}.(1/2)) \\
\equiv & (1/2)(1/2) + (1/2)(1) \\
\equiv & 3/4 \\
\Leftarrow & [b] \sqcap 1/2 \\
\equiv & G \sqcap I
\end{aligned}$$

to show wlp -invariance of I , the other requirement of the theorem. But

$$\begin{aligned}
& wp.loop.(\overline{G} \sqcap I) \\
& wp.loop.([\neg b]/2) \\
\equiv & wp.(\mathbf{if } b \mathbf{ then } b := false_{1/2} \oplus \mathbf{abort} \mathbf{ fi}).([\neg b]/2) && \text{unfold loop} \\
\equiv & [b] \sqcap ((1/2)(1/2) + (1/2)(0)) \sqcup [\neg b] \sqcap [\neg b]/2 \\
\equiv & [b]/4 \sqcup [\neg b]/2 ,
\end{aligned}$$

showing the conclusion of Thm. 4.6 to be false in this case: the precondition $[b]/4 \sqcup [\neg b]/2$ is not at least I , since when b for example the former is $1/4$ and the latter $1/2$. \square

5 Three examples of total correctness

With Thm. 4.6 we are able to discover total correctness properties of loops, provided we are given their termination conditions. Rigorous termination arguments themselves are the subject of Sec. 7 below; here we treat termination informally.

The examples below illustrate the interplay of invariant and termination condition in the three possible probabilistic cases: one, the other, or both are probabilistic.

5.1 Uniform binary selection

In this example the termination condition is standard, indicating either certain termination (when 1) or failure to terminate (when 0). Given a positive integer N , an integer l is to be chosen uniformly so that $0 \leq l < N$; the method is by successive divisions of the choice interval into roughly equal halves.

Example 5.1 Let $prog$, $init$ and $loop$ be as in Fig. 1: given arbitrary integer C , we are interested in the probability that $l = C$ finally. We define

$$I := [l \leq C < h]/(h - l) ,$$

and with the following calculation show it to be invariant.⁷

In the calculation, we start with the overall postcondition and reason backwards towards the precondition, indicating between predicates when wp is applied to give the lower (the right-hand side of a reasoning step) from the upper (the left-hand side). We have

⁷We define $0/0 := 0$ for convenience, so that I is identically 0 when $l = h$. For any $x \neq 0$ we stipulate further that $x/0$ is some real number, but we do not care which. Note that the program itself does not divide by 0.

```

init  →      l, h := 0, N;
loop  →      do l + 1 ≠ h →
                p := (l + h) ÷ 2;
                l := p  $\oplus_{h-p}$   $\oplus_{p-l}$  h := p
                od

```

The program *prog* is the whole of the above.

We write $m \oplus_n$ as a convenient abbreviation for $m/(m+n) \oplus$.

Figure 1: Example 5.1, uniform binary selection.

$$\begin{aligned}
& [l \leq C < h]/(h-l) \\
\equiv & \quad ((h-p)/(h-l))[p \leq C < h]/(h-p) && \text{after applying } wp.(l := p \oplus_{h-p} \oplus_{p-l} h := p) \\
& + \quad ((p-l)/(h-l))[l \leq C < p]/(p-l) \\
\Leftarrow & \quad \text{in spite of 0-divisions: } lower \text{ is } 0 \text{ whenever } upper \text{ contains divisions by } 0 \\
& [l < p < h] \sqcap [l \leq C < h]/(h-l) \\
\equiv & \quad [l < (l+h) \div 2 < h] \sqcap [l \leq C < h]/(h-l) && \text{after applying } wp.(p := (l+h) \div 2) \\
\Leftarrow & \quad [l+1 \neq h] \sqcap [l \leq C < h]/(h-l), && \text{in spite of 0-divisions}
\end{aligned}$$

as required. Standard reasoning with variant $h-l$ shows that termination is certain because $N > 0$ initially; thus $T \equiv 1$, implying $I \Rightarrow T$ trivially, and we have immediately from Thm. 4.6 that

$$[l \leq C < h]/(h-l) \equiv I \Rightarrow wp.loop.(\overline{G} \sqcap I) \equiv wp.loop.[l = C],$$

and so finish with

$$[0 \leq C < N]/N \equiv wp.init.([l \leq C < h]/(h-l)) \Rightarrow wp.prog.[l = C].$$

We conclude overall that for any integer C the probability of *prog*'s setting l to C finally is at least $1/N$ provided $0 \leq C < N$, and that since there are exactly N such values for C we have achieved uniform selection from the given interval. The probability is (only) at least 0 otherwise — when C lies outside the interval we should assume that we have ‘no chance’ of establishing $l = C$ finally. \square

Note that the proof in Ex. 5.1 of invariance of I would succeed even if p were chosen nondeterministically between l and h rather than being assigned the specific value $(l+h) \div 2$. In that case we would appeal to the more general Thm. A.3 to reach the same conclusion.⁸

⁸In the notation of a refinement calculus [16] such a program *choose* could be written $p : [l+1 \neq h, l < p < h]$, with meaning given by

$$wp.choose.Q := [l < p < h] \sqcap (\sqcap p \mid l < p < h \cdot Q).$$

```

init  →      n, f := N, 1;
loop  →      do n ≠ 0 →
              f := f × n;
              n := n - 1  $_p \oplus$  n := n + 1
            od

```

The program *prog* is the whole of the above.

The decrementing of n fails probabilistically, sometimes incrementing instead.

Figure 2: Example 5.2, faulty factorial.

5.2 Faulty factorial

In this example both the termination condition and the invariant are probabilistic. Given a natural number N , the program is to (attempt to) set f to $N!$ in spite of its containing a probabilistically faulty subtraction.

Example 5.2 The program is shown in Fig. 2, and is the conventional factorial algorithm except that the decrement of n sometimes increments instead.⁹

When p is 1, making the program standard (and decrementing of n certain), the invariant $N! = f \times n!$ suffices in the usual way to show that $wp.prog.[f = N!] \equiv 1$. In general, however, that postcondition is achieved only if the decrement alternative is chosen on each of the N executions of the loop body, thus with probability p^N . More rigorously we define invariant $I := p^n[N! = f \times n!]$, showing its preservation with the calculation

$$\begin{aligned}
& p^n[N! = f \times n!] \\
\equiv & \quad p(p^{n-1})[N! = f \times (n-1)!] \quad \text{after applying } wp.(n := n - 1 \oplus n := n + 1) \\
& + \quad \bar{p}(p^{n+1})[N! = f \times (n+1)!] \\
\Leftarrow & \quad p^n[N! = f \times (n-1)!] \quad \text{dropping the right additive term} \\
\equiv & \quad p^n[N! = f \times n \times (n-1)!] \quad \text{after applying } wp.(f := f \times n) \\
\Leftarrow & \quad [n \neq 0] \sqcap p^n[N! = f \times n],
\end{aligned}$$

as required.

The exact termination condition depends on p . Standard *random walk* results [5] show that *loop* terminates certainly when $p \geq 1/2$, but with probability only $(p/\bar{p})^n$ otherwise. In either case, however, the termination condition is at least p^n and so exceeds the invariant: thus Thm. 4.6 applies. We conclude

$$wp.prog.[f = N!] \Leftarrow wp.init.(p^n[N! = f \times n!]) \Leftarrow p^N[N! = 1 \times N!] \equiv p^N,$$

as suggested by our informal analysis earlier. □

⁹Perhaps it is struck by a cosmic ray.

```

init  →      c, b: = C, 1;
loop  →      do b ≠ 0 →
              if b ≤ c then
                c: = c - b;
                c, b: = c + 2b, 0 1/2 ⊕ b: = 2b
              fi
              od

```

The program *prog* is the whole of the above.

The gambler's capital c is initially C , and his intended bet b is initially 1.

On each iteration, if his intended bet does not exceed his capital, he is allowed to place it and has $1/2$ chance of winning.

If he wins, he receives twice his bet in return, and sets his intended bet to 0 to indicate he is finished; if he loses, he receives nothing and doubles his intended bet — hoping to win next time.

If he loses sufficiently often (in succession), his intended bet b will eventually be more than he can afford — his remaining capital c — and he will then be 'trapped' forever within the iteration.

Figure 3: Example 5.3, the martingale.

5.3 The martingale

Here the termination condition is probabilistic and the invariant is standard. The *martingale* is the gambling strategy of doubling one's bet after each loss of an even wager: since the wager is won eventually, with probability 1, an overall profit seems guaranteed.¹⁰

As is well known however, the flaw with the martingale is that the gambler runs the risk of using all his capital before the probabilistically certain win: his capital is finite, but the number of bets before the eventual win can be arbitrarily large.

Example 5.3 We model the martingale as in Fig. 3. If the gambler cannot place his bet, because his capital has become too small, he simply remains within the loop.

It is easy to show that $I := [c + b = C + 1]$ is an invariant of *loop*; and with some arithmetic it can be shown informally that — with the given initialisation — the chance of losing consistently until the capital is exhausted is $2/P$, where P is the smallest power of two exceeding $C+1$. Thus *wp.prog.1* is just $\overline{2/P}$.

There are two problems in applying Thm. 4.6 at this point, however. The first is that, although $\overline{2/P}$ is the termination condition of *prog* as a whole, we have not established the termination condition of *loop* itself; though we could calculate it, it would be a messy expression in terms of general initial values for b and c .

The second problem is that the invariant is *not* less than the termination condition: after the initialisation shown, for example, we have $I \equiv 1$ but $1 \not\equiv T$.

Both problems can be solved by using Lem. 4.3 in this case rather than Thm. 4.6; whatever T is in general, still we have

$$I \& T$$

¹⁰Karen Seidel suggested using the martingale for this example.

$$\begin{aligned}
&\equiv [c + b = C + 1] \& T \\
&\equiv 1 \& wp.prog.1 && \text{after applying } wp.(c, b := C, 1) \\
&\equiv \frac{1 \& (2/P)}{2/P} \\
&\equiv \overline{2/P},
\end{aligned}$$

showing that indeed

$$wp.prog.[c = C + 1] \Leftarrow \overline{2/P}.$$

With probability at least $\overline{2/P}$ the gambler eventually increases his capital by exactly 1. \square

6 The 0-1 law for termination

Beyond its use for specific programs, Thm. 4.6 has a general consequence that will be of importance to our later analysis of termination.¹¹ The *0-1 Law* of Hart et al. [6] reads informally as follows.

Let process P be defined over a state space S , and suppose that from every state in some subset S' of S the probability of P 's eventual escape from S' is at least p , for some fixed $p > 0$.

Then P 's escape from S' is certain, occurring with probability 1.

More succinctly one could say that the infimum over S' of eventual escape probability is either 0 or 1: it cannot lie properly in between.

Note that we do not require that for every state in S' the probability of *immediate* escape is at least p — that is a much stronger condition, from which the certainty of eventual escape is obvious.

In our context we fix *loop* and choose an invariant I : the process is then the iteration of *body*, leading to eventual escape from the set of states $G \sqcap I$ — leading thus equivalently to eventual termination of the loop. The 0-1 Law in that form is easily proved from our Thm. 4.6.

Lemma 6.1 Let I be a wp -invariant of *loop* with termination condition T (as in Not. 4.2).¹² If for some fixed probability $p > 0$ we have $p(I) \Rightarrow T$, then in fact $I \Rightarrow T$.

Proof: With G 's being standard, wp -invariance of I and Fact B.6 we have

$$G \sqcap p(I) \equiv p(G \sqcap I) \Rightarrow p(wp.body.I) \equiv wp.body.(p(I)),$$

so that also $p(I)$ is a wp -invariant of *loop*. We then reason

$$\begin{aligned}
&p(I) \\
\Rightarrow &wp.loop.(\overline{G} \sqcap p(I)) && wp\text{-invariance of } p(I); p(I) \Rightarrow T, \text{ Thm. 4.6} \\
\equiv &wp.loop.(p(\overline{G} \sqcap I)) && G \text{ standard} \\
\equiv &p(wp.loop.(\overline{G} \sqcap I)) && \text{Fact B.6} \\
\Rightarrow &p(wp.loop.1) && \text{Fact B.7} \\
\equiv &p(T) && \text{definition } T
\end{aligned}$$

and, since $p \neq 0$, our result follows by dividing both sides by p . \square

Aside from its intrinsic interest, the importance of Lem. 6.1 is that it will give us a very general variant-based argument for establishing termination of probabilistic loops.

¹¹From this point we will refer to Theorems 4.6 and A.3 together as just Thm. 4.6. All results proved hold in the general nondeterministic case.

¹²This lemma holds even for probabilistic I — but in that case one cannot speak so readily of ‘states satisfying I ’ in our informal discussion.

7 Probabilistic variant arguments

Termination of standard loops is conventionally shown using ‘variants’ based on the state: they are integer-valued expressions over the state variables that are bounded below but still strictly decreased by each iteration of the loop. That method is complete (up to expressibility) since informally one can always define a variant

variant := ‘the largest number of iterations still possible from the current state’,

which satisfies the above conditions trivially if the loop indeed terminates.

For probabilistic programs however the standard variant method is not complete (though clearly it remains sound): for example the program

$$\begin{array}{l} \mathbf{do} \ (n \bmod N) \neq 0 \rightarrow \\ \quad n := n + 1 \quad \text{with prob } 1/2 \oplus \quad n := n - 1 \\ \mathbf{od} \end{array} \quad (5)$$

over natural number n is certain to terminate, yet from the fact that its body can both increment and decrement n it is clear there can be no strictly decreasing variant.

With the 0-1 Law of Lem. 6.1 we are able to justify the following variant-based rule for probabilistic termination, sufficient for many practical cases including (5). In Sec. 8 we show it complete over finite state spaces.

Lemma 7.1 Let V be an integer-valued expression in the program variables, defined at least over some subset I of the state space S . Suppose further that for iteration *loop*

1. there are fixed integer constants L (low) and H (high) such that

$$G \sqcap I \quad \Rightarrow \quad [L \leq V < H] ,$$

and

2. the subset I , as a (standard) predicate, is at least¹³ *wlp*-invariant for *loop* and
3. for some fixed probability $p > 0$ and for all integers N we have

$$p(G \sqcap I \sqcap [V = N]) \quad \Rightarrow \quad wp.body.[V < N] .$$

Then termination is certain from any state in which I holds: we have $I \Rightarrow T$, where T is the termination condition of *loop*.

Proof: We show first that Assumption 2 allows Assumption 3 to be strengthened as follows:

$$\begin{array}{ll} wp.body.(I \sqcap [V < N]) & \\ \equiv wp.body.(I \ \& \ [V < N]) & \text{standard predicates} \\ \Leftarrow wlp.body.I \ \& \ wp.body.[V < N] & \text{Fact B.2} \\ \Leftarrow (G \sqcap I) \ \& \ p(G \sqcap I \sqcap [V = N]) & \text{Assumptions 2,3} \\ \equiv p(G \sqcap I \sqcap [V = N]) . & G, I \text{ standard} \end{array}$$

¹³Being *wp*-invariant is a stronger requirement, therefore sufficient also.

Thus we can add $(I \sqcap)$ to the right-hand side of Assumption 3.

Now we continue with induction to show that for all $n \geq 0$ we have

$$p^n(I \sqcap [V < L + n]) \quad \Rightarrow \quad T . \tag{6}$$

For the base case we reason from Assumption 1 that

$$p^0(I \sqcap [V < L]) \quad \Rightarrow \quad \overline{G} \quad \Rightarrow \quad T .$$

For the step case we reason

$$\begin{aligned} & p^{n+1}(I \sqcap [V < L + n + 1]) \\ \equiv & \quad G \sqcap p^{n+1}(I \sqcap [V < L + n + 1]) && G \text{ standard} \\ & \sqcup \overline{G} \sqcap p^{n+1}(I \sqcap [V < L + n + 1]) \\ \Rightarrow & \quad G \sqcap p^{n+1}(I \sqcap [V < L + n + 1]) \sqcup T && \overline{G} \Rightarrow T \\ \equiv & \quad p^{n+1}(G \sqcap I \sqcap [V < L + n]) && G \text{ standard} \\ & \sqcup p^{n+1}(G \sqcap I \sqcap [V = L + n]) \\ & \sqcup T \\ \Rightarrow & \quad p(T) \sqcup p^{n+1}(G \sqcap I \sqcap [V = L + n]) \sqcup T && \text{inductive hypothesis} \\ \Rightarrow & \quad p^n(wp.body.(I \sqcap [V < L + n])) \sqcup T && p(T) \Rightarrow T; \text{ Assumption 3 strengthened} \\ \equiv & \quad wp.body.(p^n(I \sqcap [V < L + n])) \sqcup T && \text{Fact B.6} \\ \Rightarrow & \quad wp.body.T \sqcup T && \text{inductive hypothesis} \\ \equiv & \quad T . && wp.body.T \Rightarrow T \end{aligned}$$

With (6) now established, from Assumption 1 we can conclude

$$\begin{aligned} & p^{H-L}(I) \\ \equiv & \quad p^{H-L}(G \sqcap I) \sqcup p^{H-L}(\overline{G} \sqcap I) && G \text{ standard} \\ \equiv & \quad p^{H-L}(G \sqcap I) \sqcup T && \overline{G} \Rightarrow T \\ \Rightarrow & \quad p^{H-L}(I \sqcap [V < H]) \sqcup T && \text{Assumption 1 (weakened)} \\ \equiv & \quad T \sqcup T && (6) \text{ above} \\ \equiv & \quad T . \end{aligned}$$

That, with Assumption 2 and $p^{H-L} \neq 0$, gives us $I \Rightarrow T$ directly from Lem. 6.1. \square

Informally, Lem. 7.1 shows termination given an integer-valued variant bounded *above and below* such that on each iteration a strict decrease is guaranteed *with at least fixed probability* $p > 0$. Note that the probabilistic variant *is allowed to increase* — but not above H . (We have emphasised the parts that differ from the standard variant rule.)

The termination of Program (5) now follows immediately from Lem. 7.1 with variant $n \bmod N$, taking $L, H := 0, N$.

In some circumstances it is convenient to use other forms of variant argument, variations on Lem. 8.1; one easily proved from it is the more conventional rule in which

the variant is bounded below (but not necessarily above), must decrease with fixed probability $p > 0$ and cannot increase.

That rule follows (informally) from Lem. 8.1 by noting that since the variant cannot increase its initial value determines the upper bound H required by the lemma, and it shows termination for example of the loop

```

do  $n > 0 \rightarrow$ 
   $n := n - 1$   $\frac{1}{2} \oplus$  skip
od ,

```

for which variant n suffices with $L := 0$.

8 Finitary completeness of variants

We now show that, if the state space is finite, the technique set out in Lem. 7.1 is complete for proving certain termination. We construct a variant explicitly: for any state it is the least n such that the iteration has nonzero probability of terminating in no more than n steps from that state. The following lemma establishes its existence and properties.

Lemma 8.1 Let the state space be S , and take arbitrary *loop*. Note that $[T] \equiv [wp.loop.1]$ is (the characteristic function of) that subset of S from which termination of *loop* is certain.

Then there is an integer function V of the state such that whenever $G \sqcap [T] \equiv 1$ (termination is certain but has not yet occurred) the probability of V 's strict decrease in the very next iteration is nonzero; more precisely, we construct V such that

$$G \sqcap [T] \sqcap [V = N] \quad \Rightarrow \quad [wp.body.[V < N]]$$

for all N .¹⁴

Proof: Define the \mathbb{N} -indexed probabilistic predicates

$$\begin{aligned} T_0 &:= \overline{G} \\ T_{n+1} &:= \overline{G} \sqcup wp.body.T_n , \end{aligned}$$

so that T_N is the probability of termination within N iterations. The variant is then given by

$$V := (\sqcap n \mid T_n > 0) , \tag{7}$$

which is well-defined in states where $T \neq 0$ (and thus in particular where $[T] \equiv 1$); define V arbitrarily otherwise. Then $V = N$ in any state in $[T]$ means that, from that state, there is a nonzero probability of termination within N iterations.

We now show that whenever $G \sqcap [T] \sqcap [V = N]$ holds (is 1) the probability of establishing $V < N$ on the very next iteration is nonzero. When $N = 0$ the result is trivial (antecedent false); for $N > 0$ we reason

¹⁴To show that from any state satisfying the standard P the program *prog* has nonzero probability of establishing the standard Q , we simply prove $P \Rightarrow [wp.prog.Q]$.

$$\begin{array}{lll}
& [wp.body.[V < N]] & \\
\Leftarrow & [wp.body.T_{N-1}] & T_{N-1} \Rightarrow [V < N] \\
\Leftarrow & [G \sqcap wp.body.T_{N-1}] & \\
\equiv & [G \sqcap T_N] & \text{definition } T_N \\
\equiv & G \sqcap [T_N] & G \text{ standard} \\
\Leftarrow & G \sqcap [T] \sqcap [V = N], & [T] \sqcap [V = N] \Rightarrow [T_N]
\end{array}$$

establishing the desired inequality. \square

We now use the lemma to show that if we assume finiteness of the state space the expression V constructed above satisfies the conditions of Lem. 7.1, so establishing completeness.

Theorem 8.2 The termination rule of Lem. 7.1 is sound and complete for certain termination over a finite state space.

Proof: Soundness was established by Lem. 7.1 directly, even when the state space S is infinite.

For completeness, note that when S is finite the expression (7) constructed in Lem. 8.1 is trivially bounded above and below. Similarly the probability of its decrease is bounded away from zero (being a finite infimum of positive quantities); in particular, we choose p for Lem. 7.1 to be the minimum, taken over the states in the finite set $[T]$, of $[wp.body[V < N]]$ with N set to the value of the variant in that state.

All that remains therefore is to show that $[T]$ is an invariant of $loop$, so that we can take $I := [T]$ in Lem. 7.1. For that we have

$$\begin{array}{lll}
& G \sqcap [T] & \\
\equiv & G \sqcap [wp.loop.1] & \\
\equiv & [G \sqcap wp.loop.1] & G \text{ standard} \\
\Rightarrow & [wp.body.(wp.loop.1)] & \\
\Rightarrow & wp.body.[wp.loop.1] & \text{Fact B.8} \\
\equiv & wp.body.[T], &
\end{array}$$

as required. \square

9 Example: self-stabilisation

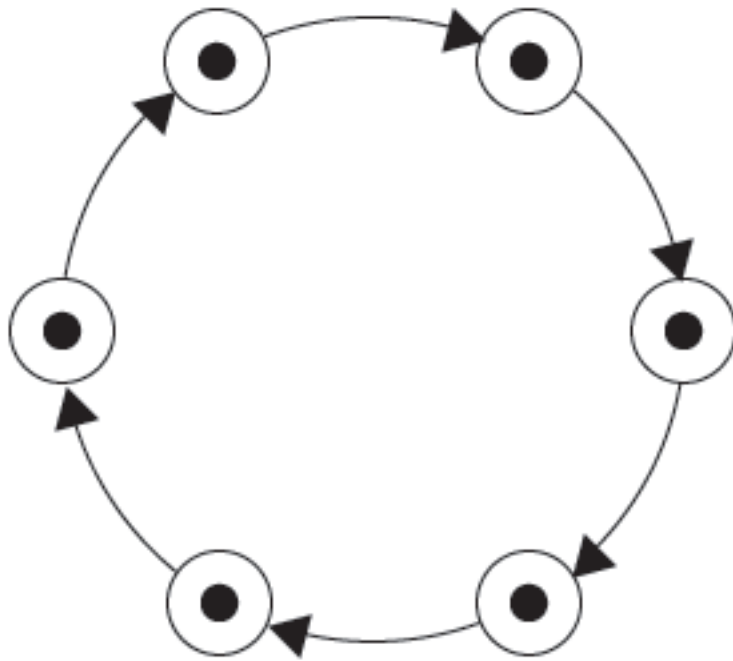
In our final example we apply Lem. 8.1 to a variation on Herman's *probabilistic self-stabilisation* [8], a distributed probabilistic algorithm that can be used for leadership election in a ring of synchronously executing processors.

Example 9.1 Consider N identical processors connected clockwise in a ring, as illustrated in Fig. 4. A single processor — a *leader* — is chosen from them in the following way.

Initially each processor is given exactly 1 token; the leader is the first processor to obtain all N of them. Fix some probability p with $0 < p < 1$.

On each step (synchronously) all processors perform the following actions:

1. Make a local probabilistic decision either to *pass* (probability p) or to *keep* (probability $1 - p$) all its tokens.
2. If *pass*, then send *all* its tokens to the next-clockwise processor; if *keep*, do nothing.



Example ring topology ($N=6$),
with initial token assignment shown.

See reference **M95** at <http://www.comlab.ox.ac.uk/oucl/groups/probs/bibliography.html> for this illustration.

Figure 4: Example ring topology ($N = 6$) with initial token assignment shown.

See reference **M95** at <http://www.comlab.ox.ac.uk/oucl/groups/probs/bibliography.html> for this illustration.

Figure 5: The variant decreases with probability at least $p(1 - p)$; it may increase.

3. Receive tokens passed (if any) from the next-anticlockwise processor, adding them to the tokens currently held (if any).

We show that with probability 1 eventually a single processor will obtain all N tokens. We define

- the invariant to be that the total number of tokens is constant (at N), and
- the guard (which if true indicates that termination has not yet occurred) to be that more than one processor holds tokens and
- the variant to be the shortest length of any ring segment (contiguous sequence of arcs) containing all tokens. (See Fig. 5.)

With those definitions, for proof of termination we simply note that (refer assumptions of Lem. 7.1)

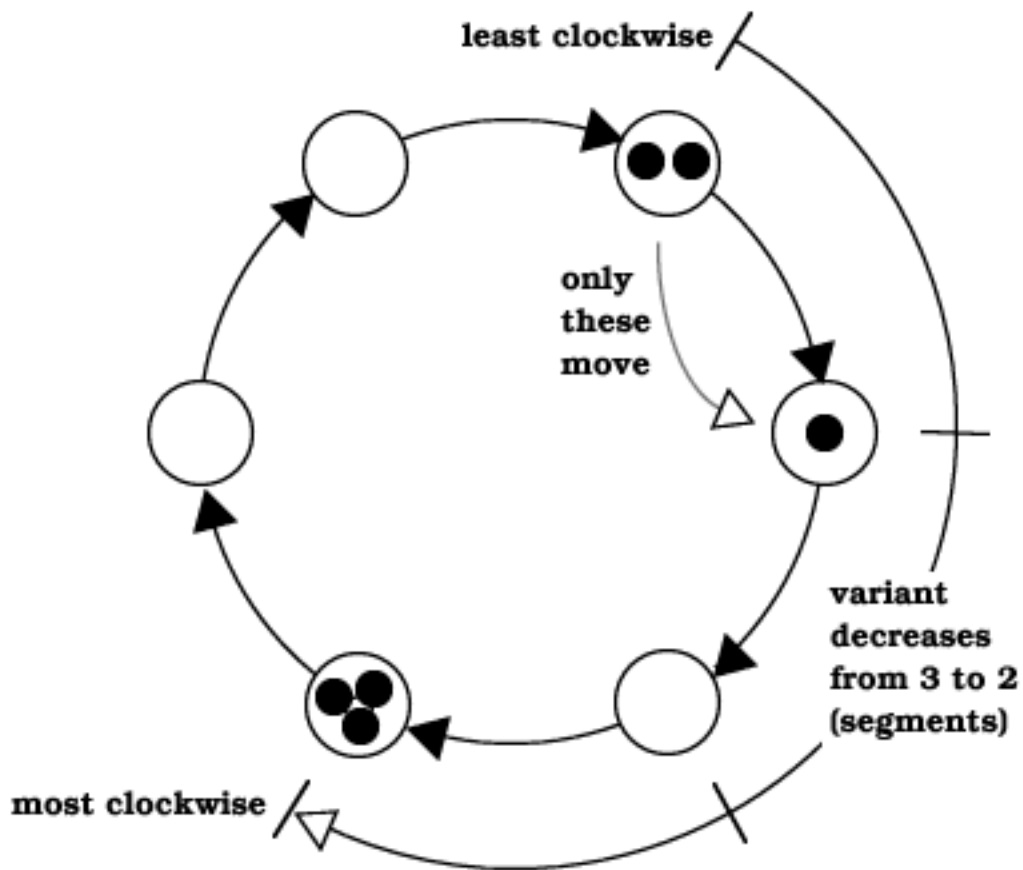
1. the guard and invariant imply that the variant is bounded below by 1 and above by N , and
2. the invariant is trivially maintained and
3. the variant decreases strictly with probability at least $p(1 - p)$, which is nonzero since $0 < p < 1$. (Let the least-clockwise processor in the shortest segment decide to *pass* while the most-clockwise processor decides to *keep*.)

The conclusion of Lem. 7.1 gives us certain termination: that eventually only one processor contains tokens (negated guard), and that it has all N of them (invariant).¹⁵ \square

¹⁵Converting the processors' arithmetic to **mod** 2 yields a scheme very close to Hermans' original; in that case the number of processors must be odd so that $N \bmod 2$ and $0 \bmod 2$ can be distinguished on termination.

Note that the use of a ring is not essential for correctness: in fact if each processor chooses probabilistically from all others where to pass its tokens (with a nonzero probability for each possible recipient), then termination is still certain — and in fact is easier to show than with a ring. The variant is just the number of processors holding tokens, and cannot increase; it decreases with nonzero probability $p(1 - p)r$, where the extra factor r is the minimum probability over all processor pairs P, P' that P will choose P' as its recipient.

That 'chaotic' scheme remains correct even if the processors execute asynchronously, provided their scheduling is starvation-free.



The variant decreases with probability at least $p(1-p)^{n-1}$; it may increase.

10 Conclusion

Our main results are Thm. 4.6 for total correctness of iterations when the termination condition is known, and Thm. 8.2 for termination with probability 1. With the examples of Sections 5 and 9 we have shown that probabilistic reasoning for partial correctness — on this scale at least — is not much more complex than standard reasoning.

For total correctness it seems harder however to achieve simplification using grossly pessimistic variants (a familiar technique in the standard case). Our experience so far suggests that it is often necessary to use accurate bounds on the number of iterations remaining, and that can require intricate calculation.

We do not have general rules for determining the termination condition when it is not 1; at this stage it seems those situations have to be handled by using the *wp* semantics to extract a recurrence relation to which standard probabilistic methods can then be applied. A promising approach however is to use (probabilistic) data refinement to extract not a recurrence relation but a simple(r) program, involving only the variant captured by a single variable. That program's termination condition is equal to the original, but could perhaps be taken straight from the literature; one would then have access to a collection of termination 'paradigms'.

A longer term approach to probabilistic termination is to build a temporal logic over the probabilistic predicate transformers [17], generalising a similar construction by Morris [20] over standard transformers. The resulting properties are then very like those of Ben-Ari, Pnueli and Manna [1], and allow termination conditions to be determined for quite complicated programs using structured arguments in the style for example of UNITY [2].

Acknowledgements

This work was carried out in collaboration with Annabelle McIver, Jeff Sanders and Karen Seidel; we are grateful for the support of the EPSRC.

I benefited from the intellectual and social stimulation provided by six months' stay at the Software Verification Research Center and the Department of Computer Science of the University of Queensland.

The work has been improved by the attentions of IFIP Working Groups 2.1 and 2.3.

References

- [1] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.
- [2] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, Mass., 1988.
- [3] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall International, Englewood Cliffs, N.J., 1976.
- [4] R.W. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Mathematical Aspects of Computer Science*. American Mathematical Society, 1967.

- [5] G. Grimmett and D. Welsh. *Probability: an Introduction*. Oxford Science Publications, 1986.
- [6] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5:356–380, 1983.
- [7] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.
- [8] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, October 1969.
- [9] C. Jones. Probabilistic nondeterminism. Monograph ECS-LFCS-90-105, Edinburgh University, 1990. (Ph D Thesis).
- [10] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.
- [11] D. Kozen. A probabilistic PDL. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, New York, 1983. ACM.
- [12] A.K. McIver and C.C. Morgan. Partial correctness for probabilistic programs. Submitted to *Theoretical Computing Science*; available at [22].
- [13] Carroll Morgan and Annabelle McIver. An expectation-based model for probabilistic temporal logic. Technical Report PRG-TR-20-97, Programming Research Group, 1997. To appear in *IGPL*; available at [22].
- [14] Carroll Morgan and Annabelle McIver. A probabilistic temporal calculus based on expectations. In Lindsay Groves and Steve Reeves, editors, *Proc. Formal Methods Pacific '97*. Springer Verlag Singapore, July 1997. Available at [22].
- [15] C.C. Morgan. *Programming from Specifications*. Prentice-Hall, second edition, 1994.
- [16] C.C. Morgan and A.K. McIver. A temporal logic for probabilistic predicate transformers. See [15, 14].
- [17] C.C. Morgan, A.K. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
- [18] C.C. Morgan, A.K. McIver, K. Seidel, and J.W. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996.
- [19] J.M. Morris. Temporal predicate transformers and fair termination. *Acta Informatica*, 27:287–313, 1990.
- [20] Probabilistic Systems Group. Probabilistic weakest liberal preconditions. See [13].
- [21] PSG. Probabilistic Systems Group: Collected reports.
<http://web.comlab.ox.ac.uk/oucl/research/areas/probs/bibliography.html>.
- [22] K. Seidel, C.C. Morgan, and A.K. McIver. An introduction to probabilistic predicate transformers. Technical Report PRG-TR-6-96, Programming Research Group, February 1996. Available at [22].

[23] M. Sharir, A. Pnueli, and S. Hart. Verification of probabilistic programs. *SIAM Journal on Computing*, 13(2):292–314, May 1984.

A Nondeterministic loops

Here we show that the result of Thm. 4.6 extends to nondeterministic loops. The approach is to use Fact B.5 to replace the loop body by an appropriate deterministic refinement of it, for which we use the following lemma.

Notation A.1 The program $dloop$ is defined

$$dloop := \mathbf{do} \ G \rightarrow \mathbf{det} \ \mathbf{od} ,$$

for standard predicate G (the loop guard) and deterministic program det (the loop body). \square

Lemma A.2 For any $loop$ and postcondition Q there is a $dloop$ such that $body \sqsubseteq det$ and

$$wp.dloop.Q \equiv wp.loop.Q .$$

Proof: Define $P := wp.loop.Q$, and use Fact B.5 to choose det so that $body \sqsubseteq det$ and

$$wp.body.P \equiv wp.det.P . \tag{8}$$

Then we have

$$\begin{aligned} & \overline{G} \sqcap Q \sqcup G \sqcap wp.det.P \\ \equiv & \overline{G} \sqcap Q \sqcup G \sqcap wp.body.P && \text{by construction (8)} \\ \equiv & P , && \text{definition } P; \text{ refolding of iteration} \end{aligned}$$

so that P satisfies the (least) fixed-point equation for $wp.dloop.Q$. Hence $wp.dloop.Q \Rightarrow P$ and, from $loop \sqsubseteq dloop$ and monotonicity, we have

$$wp.dloop.Q \equiv wp.loop.Q$$

as required. \square

With Lem. A.2 we have our theorem easily.

Theorem A.3 If I is a wp -invariant of $loop$ and $I \Rightarrow T$, then

$$I \Rightarrow wp.loop.(\overline{G} \sqcap I) .$$

Proof: Use Lem. A.2 to choose deterministic refinement det of $body$ so that

$$wp.dloop.(\overline{G} \sqcap I) \equiv wp.loop.(\overline{G} \sqcap I) ,$$

and observe that since $body \sqsubseteq det$ we have I a wp -invariant of $dloop$ also.

The result is then immediate from Thm. 4.6. \square

B Facts about probabilistic wp and wlp

Proofs of these facts are to be found in other publications of the Group [22].

Fact B.1 For standard program $prog$ and standard postcondition Q we have

$$wlp.prog.Q \sqcap wp.prog.1 \Rightarrow wp.prog.Q .$$

□

Fact B.2 *sub-distributivity of $\&$* For program $prog$ and postconditions Q_0, Q_1 we have

$$wlp.prog.Q_0 \& wp.prog.Q_1 \Rightarrow wp.prog.(Q_0 \& Q_1) .$$

We observe as a special case that

$$wp.prog.Q_0 \& wp.prog.Q_1 \Rightarrow wp.prog.(Q_0 \& Q_1) ,$$

since for any $prog$ and Q we have $wp.prog.Q \Rightarrow wlp.prog.Q$. If $wlog$ both Q_0 and $wp.prog.Q_0$ are standard, the above reduces further to sub-distributivity of \sqcap . □

Fact B.3 *sub-distributivity of $+$* For any program $prog$ and postconditions Q_0, Q_1 we have

$$wp.prog.Q_0 + wlp.prog.Q_1 \Rightarrow wlp.prog.(Q_0 + Q_1) ,$$

with equality when $prog$ is deterministic. □

Fact B.4 For any program $prog$ we have $wlp.prog.1 \equiv 1$. □

Fact B.5 For any program $prog$ and postcondition Q there is a deterministic refinement of it — a deterministic det with $prog \sqsubseteq det$ — such that

$$wp.prog.Q \equiv wp.det.Q .$$

□

Fact B.6 *scaling* For any program $prog$, postcondition Q and constant c with $0 \leq c \leq 1$, we have

$$c(wp.prog.Q) \equiv wp.prog.(cQ) .$$

□

Fact B.7 *monotonicity* For any program $prog$ and postconditions Q, Q' with $Q \Rightarrow Q'$ we have

$$\begin{aligned} wp.prog.Q &\Rightarrow wp.prog.Q' \\ \text{and } wlp.prog.Q &\Rightarrow wlp.prog.Q' . \end{aligned}$$

□

Fact B.8 For any program $prog$ and postcondition Q we have

$$\lfloor wp.prog.Q \rfloor \Rightarrow wp.prog.\lfloor Q \rfloor .$$

□