The choice-coordination problem: notes for a correctness proof

Probabilistic Systems Group*

Draft of February 9, 1996

Abstract

Rabin's *Choice-coordination* algorithm [5] allows a number of identical processes to agree on a choice between two alternatives while nevertheless executing asynchronously.

The algorithm is analysed here with techniques based on *probabilistic predicate transformers* [3], in particular applying *invariants* and *variants* as set out in [2], and using a *probabilistic temporal logic*.

Keywords: Probabilistic algorithm, predicate transformer, correctness proof, temporal logic.

1 Introduction

This note applies the familiar ('standard') sequential notions of invariant and variant to a probabilistic algorithm.

Common sense suggests that the probabilistic reasoning should resemble the standard, and indeed it does: one proves a predicate invariant under execution of a loop's body; and one finds a variant that ensures the loop's eventual termination. The conclusion is that if the invariant holds initially then the invariant and the negation of the loop guard together hold finally.

Differences from the standard case are however as follows:

*Annabelle McIver, Carroll Morgan, Jeff Sanders and Karen Seidel; Programming Research Group, Oxford; {*anabel, carroll, jeff, karen*}*@comlab.ox.ac.uk.*

- The invariant can be probabilistic, in which case its operational meaning is more general than 'the computation remains within a certain set of states'.

- The variant may *have* to be probabilistically interpreted, since the usual 'must strictly decrease and is bounded below' technique is no longer adequate (though it remains sound), even for simple cases.

- When both the invariant and the termination condition are probabilistic, the combining of 'correct if terminates' and 'terminates under these conditions' to give total correctness is no longer simply Boolean conjunction.

The general theory of probabilistic predicate transformers is introduced in [3], and its application to loop correctness specifically is treated in [2], where several examples may be found of probabilstic invariants and variants, and their interaction.

In this example, Rabin's choice-coordination algorithm, the partial correctness argument is entirely standard and so is not novel (though somewhat involved and thus interesting as an exercise). In such cases one treats probabilistic choice as nondeterministic choice and proceeds with standard reasoning, since the theory shows that any *wp*-style property proved of the 'projected' nondeterministic program is valid for the original probabilistic program as well.

The termination argument is novel, since even the probabilistic variant techniques [1, 2] do not quite suffice (though they could be extended to do so). More interesting however is to use an argument based on temporal logic, since such an approach is already known to be effective in many cases (for example [6]); and even when a variant does exist a temporal argument might be simpler to construct and understand.

Our temporal system is novel also, since it is built over a (probabilistic) predicate transformer model rather than the more common Kripke-style construction of relations between states ('worlds'); unlike those models, for example, it includes catastrophic failure (**abort**) naturally. More significantly however is the access to explicit probabilities rather than the more common 'some non-zero probability': we can write statements such as 'with probability at least 1/2 the system will eventually reach a state satisfying

A'. Yet with the *0-1 Law* for termination [1, 2], we can ignore those explicit quantities if we want to.

The treatment below is very terse, especially in the standard reasoning leading to partial correctness. Section 6 discusses the temporal reasoning and how it can apply to this algorithm.[1]

2 Informal description of Rabin's algorithm

This informal description is based on that in [5].

A group of tourists are to decide between two meeting places: inside a (certain) church, or inside a museum. They may not communicate as a group.

Each tourist carries a notepad on which he will write various numbers; outside each of the two potential meeting places is a noticeboard on which various messages will be written. Initially the number 1 appears on all the notepads and on the two noticeboards.

Each tourist decides independently (nondeterministically) which place to visit first, after which he strictly alternates his visits between the two places. At each visit he looks at the noticeboard there, and if it displays "here" goes inside. If it does not display "here" it will display a number instead. The tourist compares that number K with the one on his notepad k, then taking one of the following three actions:

when $k < K$ — The tourist writes K on his notepad (erasing k), and goes to the other place.

when $k > K$ — The tourist writes "here" on the noticeboard (erasing K), and goes inside.

when $k = K$ — The tourist strictly increases K to the next higher odd number, and then flips a coin; if it comes up heads, he increases it one further. He then writes (the new) K on the noticeboard and on his notepad, and goes to the other place.

Rabin's algorithm as above terminates with probability 1; and on termination all tourists will be inside, at the same meeting place.

[1] It might be possible to begin by skimming that section, referring to earlier sections only as needed.

3 The program

The program text is shown in Fig. 1.

3.1 The program informally

Call the two places "left" and "right".

Bag *lout* (*rout*) is the bag of numbers held by processes waiting to look at the left (right) noticeboard; bag *lin* (*rin*) is the bag of numbers held by processes who have already decided on the left (right) alternative; number L (R) is the number on the left (right) noticeboard.

Initially there are M (N) processes on the left (right), all holding the number 1; no process has yet made a decision. Both noticeboards show 1.

Execution is informally as follows. If there are undecided processes (so that *lout* (*rout*) is not yet empty), select one: the number it holds is l (r). If some process has (already) decided on this alternative (so that *lin* (*rin*) is not empty), this process does the same; otherwise there are three further possibilities.

If this process' number l (r) is greater than the noticeboard value L (R), then decide on this alternative (joining *lin* (*rin*)).

If this process' number equals the noticeboard value, increase the noticeboard value, copy that value and go to the other alternative (*rout* (*lout*)).

If this process' number is less than the noticeboard value, copy that value and go to the other alternative.

3.2 Notation

- $[\![\cdots]\!]$ — Bag (multiset) brackets.
- $\square$ — The empty bag.
- $[\![n]\!]^N$ — A bag containing N elements all of value n.
- n **from** b — Choose an element nondeterministically from non-empty bag b, assign it to n and remove it from b.
- n **to** b — Add element n to bag b.

```
lout, rout := ⟦1⟧^M, ⟦1⟧^N;
lin, rin := □, □;
L, R := 1, 1;

do lout ≠ □ →
     l from lout;
     if lin ≠ □ then  l to lin  else
          l > L   →  l to lin
     ▯    l = L   →  L := L + 2 ½⊕ (L + 2)*;    L to rout
     ▯    l < L   →  L to rout
     fi

▯    rout ≠ □ →
     r from rout;
     if rin ≠ □ then  r to rin  else
          r > R   →  r to rin
     ▯    r = R   →  R := R + 2 ½⊕ (R + 2)*;    R to lout
     ▯    r < R   →  R to lout
     fi
od
```

Figure 1: Rabin's choice-coordination algorithm

- **if** B **then** *prog* **else** $\cdots$ **fi** — Execute *prog* if B holds, otherwise treat $\cdots$ as normal.
- n^* — The value $n+1$ if n is odd, and $n-1$ if n is even.
- $\overline{n}$ — The value $n \sqcup n^*$.
- $\#b$ — The number of elements in bag b.
- $x{:=}\ m\ {}_p\!\oplus\ n$ — Assign m to x with probability p, and n to x with probability $1{-}p$.

3.3 Correctness criteria

The program is guaranteed with probability 1 to terminate, and on termination satisfies

$$\#lin = M{+}N \ \wedge \ rin = \Box \quad \vee \quad lin = \Box \ \wedge \ \#rin = M{+}N \ .$$

That is, the processes end up either all on the left or all on the right.

4 Partial correctness

The arguments for partial correctness are entirely standard, involving no probabilistic reasoning.

4.1 Three invariants

This is our first invariant.

$$\#lout + \#lin + \#rout + \#rin \quad = \quad M + N \tag{1}$$

It holds initially, and is trivially maintained.

This is our second invariant.

$$\begin{array}{l} lin, lout \leq R \\ rin, rout \leq L \end{array} \tag{2}$$

To show its invariance we reason as follows.

- It holds initially.
- Since L, R never decrease, it can be falsified only by adding elements to the bags.
- Adding elements to lin, rin cannot falsify it, since those elements come from $lout, rout$.
- The only commands adding elements to $lout, rout$ are

  $$L \ \textbf{to} \ rout \quad \text{and} \quad R \ \textbf{to} \ lout \ ,$$

  and they maintain it trivially.

This is our third invariant.

$$\begin{array}{lll} \sqcup lin > L & \text{if} & lin \neq \Box \\ \sqcup rin > R & \text{if} & rin \neq \Box \end{array} \tag{3}$$

By symmetry we can consider only the left (lin) case. The invariant holds on initialisation (when $lin = \Box$); and inspection of the program shows that it is trivially established when the first value is added to lin since the command concerned,

$$l > L \;\rightarrow\; l \;\textbf{to}\; lin \;,$$

is executed when $lin = \Box$ to establish $lin = [\![l]\!]$ for $l > L$.

Since elements never leave lin, it remains non-empty and $\sqcup lin$ can only increase; finally L cannot change when lin is non-empty.

4.2 Correctness argument

On termination we have $lout = rout = \Box$, and so with invariant (1) we need only

$$lin = \Box \quad \vee \quad rin = \Box \;.$$

Assuming for a contradiction that both lin and rin are non-empty, we then have from invariants (2) and (3) the impossible

$$L \;\geq\; \sqcup rin \;>\; R \;\geq\; \sqcup lin \;>\; L \;,$$

as required.

5 Termination via variants: an attempt

5.1 Transitions of L, R

We first consider this integer-valued function of the state.

$$sum \quad := \quad \sum lout + \sum rout + 2\sum lin + 2\sum rin \tag{4}$$

Its strict increase is trivial; from invariants (2, 1) it is bounded above by $2N(L \sqcup R)$.

We have this further invariant.

$$\overline{L} - \overline{R} \quad \in \quad \{-2, 0, 2\} \tag{5}$$

It holds initially; from invariant (2), the statement

$$L := \ L + 2 \ {}_{\frac{1}{2}}\!\oplus (L+2)^*$$

is executed only when $L \leq R$, thus only when $\overline{L} \leq \overline{R}$, and has the effect

$$\overline{L} := \ \overline{L} + 2 \ .$$

Thus we can classify L, R into three sets of states:

- $\overline{L} = \overline{R} - 2 \ \vee \ \overline{L} = \overline{R} + 2$ — write $\overline{L} \neq \overline{R}$ for those states.
- $L^* = R$ — equivalently $L = R^*$.
- $L = R$.

And the underlying iteration of the loop induces state transitions as follows. (We write $[L = R]$ for the set of states satisfying $L = R$, and so on, and we indicate nondeterministic choice by $\sqcap$.)

$$\begin{array}{rcl} [\overline{L} \neq \overline{R}] & \rightarrow & [\overline{L} \neq \overline{R}] \ \sqcap \ [L = R] \ {}_{\frac{1}{2}}\!\oplus [L^* = R] \\ [L = R] & \rightarrow & [L = R] \ \sqcap \ [\overline{L} \neq \overline{R}] \\ [L^* = R] & \rightarrow & [L^* = R] \end{array}$$

(See Fig. 2 for a diagram of the state transitions.)

To explain the absence of a transition leaving $[L^* = R]$ we use this invariant.

$$L^* \notin \mathit{rout} \quad \wedge \quad R^* \notin \mathit{lout} \tag{6}$$

It holds initially, and cannot be falsified by the statement

$$L \ \mathbf{to} \ \mathit{rout} \ ,$$

because $L^* \neq L$. That leaves the statement

$$L := \ L + 2 \ {}_{\frac{1}{2}}\!\oplus (L+2)^* \ ;$$

but in that case from (2) we have

$$rout \ \leq \ L \ < \ L+2, (L+2)^* \ = \ (L+2)^{**}, (L+2)^* \ ,$$

so that in neither case does the statement set L to the conjugate of a value already in *rout*.

Thus with (6) we see that execution of the only alternatives that change L, R is impossible when $L^* = R$, since in that case l canot be L and r cannot be R.

5.2 The variant functions

The first variant (recall (4)) is

$$2N(L \sqcup R) - sum \ ,$$

which is guaranteed strictly to decrease if L, R don't change, and is bounded below by 0.

The second variant is the following.

$$L \oplus R \quad := \quad \begin{array}{ll} 2 & \text{, if } L = R \\ 1 & \text{, if } \overline{L} \neq \overline{R} \\ 0 & \text{, if } L^* = R \end{array} \tag{7}$$

Thus an overall vararariant might be the lexicographically-ordered pair (refer (4, 1))

$$(L \oplus R, \ 2N(L \sqcup R) - sum) \ ,$$

which has probability at least 1/2 of strict decrease on *every* iteration of the algorithm.

Unfortunately, even that variant does not satisfy (either of) the two properties each of which separately would be sufficient for termination with probability 1 [1, 2]: it is not non-increasing; and it is not bounded above.

6 Termination via temporal operators

6.1 Introduction to the operators

The essentials of the termination argument we seek are captured in Fig. 3, in which the arrows represent *eventual* state-change: rigorously those eventualities are established in the three separate cases by a simple variant argument based on (the strict and bounded increase of) *sum* alone.

Eventuality in the logic is represented by the operator $\diamond$ on predicates, so that $\diamond P$ is pronounced 'eventually P' and — at least for standard P, Q — the implication $P \Rrightarrow \diamond Q$ means 'from any state satisfying P eventually a state satisfying Q will be reached'. Such statements are relative to some fixed underlying system

$$\textbf{do } G \to \mathit{body} \textbf{ od}$$

and, following [4] (but not exactly), we define

$$\diamond P \quad := \quad (\mu X \cdot P \sqcup (G \sqcap \mathit{wp}.\mathit{body}.X)) \ .$$

When P is standard that reduces to the usual definition, in which case one can show as expected that

$$\diamond P \quad \equiv \quad \mathit{wp}.(\textbf{do } G \wedge \neg P \to \mathit{body} \textbf{ od}).P \ .$$

There is a similar interpretation when P is properly probabilistic; but in that case the (still standard) loop guard expresses a termination strategy necessarily more sophisticated than 'check for truth of P'.

In the calculations below our use of probability is limited to statements of the form

$$p(P) \Rrightarrow \diamond Q$$

for standard P and Q, in which the antecedent is multiplied by some scalar p with $0 \leq p \leq 1$; it means 'from any state satisfying P there is probability at least p of reaching a state satisfying Q'.

It is also convenient sometimes to reason in terms of eventualities conditional on the system's continuing to run: we use $\diamond' P$ to mean 'eventually P holds unless the system terminates first', where termination occurs when the guard G is false. Thus we define

$$\diamond' P \quad := \quad \diamond(P \sqcup \neg G) \ ,$$

so that by contrast (the ordinary) $\diamond P$ can be understood as 'P is established eventually *even if the system terminates*'.

Note that $\diamond 0$ (eventually false) is identically 0, whereas for example $p(P) \Rrightarrow \diamond' 0$ for standard P means that termination occurs with probability at least p from P.

6.2 Proving basic eventualities

Given a system $G \rightarrow body$, a general method for proving eventualities $P \Rrightarrow \diamond Q$ by reasoning over the program text — 'from scratch' — is this: pick a guard G', representing a 'strategy' for reaching Q, and consider the sufficient

$$P \quad \Rrightarrow \quad wp.(\mathbf{do}\ G \wedge G' \rightarrow body\ \mathbf{od}).Q \ .$$

The invariant/variant techniques of [2] are then available for showing the implication.

Three examples of the above are sketched in the next section.

6.3 The termination argument

In our case the underlying system is the one of Fig. 1; and we claim that with the techniques of Sec. 6.2 the following basic eventualities can be shown, so that we can argue at the level of Fig. 3.

$B \Rrightarrow \diamond' A$ — Eventual transition from B to A is guaranteed, unless the system terminates first. That is established as in Sec. 6.2 by choosing $G' := L = L_0 \wedge R = R_0$ for constants L_0, R_0 representing the initial values of L, R — the strategy 'terminate as soon as L or R changes' — and then using the (increasing but bounded) variant *sum* to show termination.

$C \Rrightarrow \diamond' 0$ — Eventual termination of the whole system is guaranteed from state C. (That is established as above.)

$A/2 \Rrightarrow \diamond' C$ — Eventual transition from A to C is guaranteed with probability at least $1/2$, unless the system terminates first.

Eventual departure from A is shown as above; simultaneous arrival in C with probability at least $1/2$ is shown using the probabilistic invariant $A/2 + C$, maintained by each iteration under strategy $G' := A$.

(App. B contains a more detailed example.)

We now present the argument in steps, introducing the temporal laws as they are needed.

1. From $B \Rrightarrow \diamond' A$ and $A/2 \Rrightarrow \diamond' C$ we conclude $B/2 \Rrightarrow \diamond' C$, using Facts A.1, A.2. Hence $(A \sqcup B)/2 \Rrightarrow \diamond' C$.

2. From the partition A, B, C of the state space we have $A \sqcup B \equiv \neg C$, thus from Step 1 we conclude $(\neg C)/2 \Rrightarrow \diamond' C$.

3. From Step 2, the fact the system is non-aborting and Fact A.3 we have $1 \equiv \diamond' C$.

4. From Step 3, Fact A.2 and $C \Rrightarrow \diamond' 0$ we conclude finally $1 \Rrightarrow \diamond' 0$ — in fact that $1 \equiv \diamond' 0$ — meaning the system terminates from every initial state.

The proof rules of [2] now allow certain termination to be combined with the partial correctness established in Sec. 4 to conclude that Rabin's algorithm is guaranteed to terminate in a state satisfying

$$\begin{array}{lll} & \#lin = M+N \ \wedge \ rin = \Box & \text{— all tourists inside the museum} \\ \vee & lin = \Box \ \wedge \ \#rin = M+N & \text{— all tourists inside the church.} \end{array}$$

References

[1] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5:356–380, 1983.

[2] Carroll Morgan. Proof rules for probabilistic loops. Technical Report PRG-TR-25-95, Programming Research Group, November 1995. *http://www.comlab.ox.ac.uk/oucl/publications/tr/TR-25-95.html.*

[3] Carroll Morgan, Annabelle McIver, Karen Seidel, and JW Sanders. Probabilistic predicate transformers. Technical Report PRG-TR-4-95, Programming Research Group, February 1995. *http://www.comlab.ox.ac.uk/oucl/publications/tr/TR-4-95.html.*

[4] J.M. Morris. Temporal predicate transformers and fair termination. *Acta Informatica*, 27:287–313, 1990.

[5] M.O. Rabin. The choice-coordination problem. *Acta Informatica*, 17(2):121–134, June 1982.

[6] J.R. Rao. Reasoning about probabilistic parallel programs. *ACM Transactions on Programming Languages and Systems*, 16(3), May 1994.

A Temporal facts

The following facts are proved using probabilistic weakest preconditions over the underlying system [3, 2].

Fact A.1 *(sub-) scaling* For scalar p with $0 \leq p \leq 1$ and predicate P we have

$$p(\diamond P) \ \equiv \ \diamond(pP) \quad \text{and} \quad p(\diamond' P) \ \Rrightarrow \ \diamond'(pP),$$

where juxtaposition denotes multiplication. □

Fact A.2 *transitivity* For predicates P, Q, R we have

$$P \Rrightarrow \diamond Q \text{ and } Q \Rrightarrow \diamond R \quad \text{implies} \quad P \Rrightarrow \diamond R\ .$$

The same holds for $\diamond'$. □

Fact A.3 *The 0-1 law* [1, 2] If for some *standard* predicate P and scalar p with $0 < p \leq 1$ we have $p(\neg P) \Rrightarrow \diamond P$, then provided the system is non-aborting we have in fact $1 \equiv \diamond P$.

The same holds for $\diamond'$. □

B An example eventuality proof

Consider the following program over variable s taking values A, B, C and natural number n:

$$\textbf{do } n \neq 0 \ \ \rightarrow \ \ n{:=}\ n-1 \ \sqcap \ (s{:=}\ B \ {}_{\frac{1}{2}}{\oplus}\ s{:=}\ C) \textbf{ od}$$

We use the techniques of Sec. 6.2 and [2] to show

$$[s = A]/2 \quad \Rrightarrow \quad \diamond'[s = C] \tag{8}$$

for that system.

Recalling Sec. 6.2 we choose strategy $s = A$ and consider the iteration

$$\textbf{do } n \neq 0 \wedge s = A \ \ \rightarrow \ \ n{:=}\ n-1 \ \sqcap \ (s{:=}\ B \ {}_{\frac{1}{2}}{\oplus}\ s{:=}\ C) \textbf{ od}$$

with respect to postcondition $[s = C] \sqcup [n = 0]$ (allowing $[n = 0]$ because $\diamond'$ includes 'or terminates').

Termination is guaranteed trivially by the lexicographic variant

$$([s = A], n) \ ,$$

guaranteed to decrease on every iteration and bounded below by $(1, 0)$.

For invariant take $[s = A]/2 + [s = C]$. Straightforward calculation shows it is maintained by execution of the body (assuming truth of the guard), since

$$
\begin{array}{ll}
 & [s = A]/2 + [s = C] \\
\equiv & \qquad\qquad \text{applying } wp.(n{:=}\ n-1 \ \sqcap \ (s{:=}\ B \ {}_{\frac{1}{2}}\oplus \ s{:=}\ C)) \\
 & ([s = A]/2 + [s = C]) \sqcap 1/2 \\
\Leftarrow & [n \neq 0] \sqcap [s = A] \sqcap ([s = A]/2 + [s = C]) \ .
\end{array}
$$

Thus on termination we have the invariant and the negated guard

$$([s = A]/2 + [s = C]) \sqcap ([n = 0] \sqcup [s \neq A]) \ ,$$

provided the invariant holds initially. That implies $[s = C] \sqcup [n = 0]$ as required.

We now note that the precondition $[s = A]/2$ implies the invariant, and (8) is proved.