Quantitative Program Logic and Performance in Probabilistic Distributed Algorithms

Annabelle K. McIver

Programming Research Group, Oxford University, UK.
anabel@comlab.ox.ac.uk,
http://www.comlab.ox.ac.uk/oucl/groups/probs.
The work is supported by the EPSRC.

Abstract. In this paper we show how quantitative program logic [14] provides a formal framework in which to promote standard techniques of program analysis to a context where probability and nondeterminism interact, a situation common to probabilistic distributed algorithms. We show that overall performance can be formulated directly in the logic and that it can be derived from local properties of components. We illustrate the methods with an analysis of performance of the probabilistic dining philosophers [10].

1 Introduction

Distributed systems consist of a number of independent components whose interleaved behaviour typically generates much nondeterminism; the addition of probability incurs an extra layer of complexity. Our principal aims here are to illustrate how, using 'quantitative program logic' [14], familiar techniques from standard programming paradigms easily extend to the probabilistic context, and that they can be used even to evaluate performance.

Examples of all our program techniques — compositional reasoning, μ-calculus treatments of temporal logic and fairness — can be found in the general literature [2, 9]; the novel aspects of the present work lie in their smooth extension via the quantitative logic, and our exploitation of the extended type (of reals rather than Booleans) in our formulation of performance operators as explicit μ-calculus expressions in the logic. The advantages are not merely cosmetic: by making performance and correctness objects of the same kind we discover that performance can be calculated directly from local properties of components. Locality has particular significance here since in practice it reduces system-wide analysis to the analysis of a single component in isolation from the others, vastly simplifying arguments. And finally, it is worth mentioning that though our primary theme is proof, our estimated performance of the random dining philosophers [10] is still lower than some other published analyses [11].

Our presentation is in three sections. In Sec. 2 and Sec. 3 we set out the programming model, the quantitative program logic and the μ-calculus formulations for probabilistic temporal logic and performance. Some properties of the

J.-P. Katoen (Ed.): ARTS'99, LNCS 1601, pp. 19–33, 1999.

operators are also explained in those sections. In Sec. 4 we analyse the dining philosophers.

We uniformly use S for the state space and $\mathcal{D}S$ for discrete probability distributions over S. We also use ':=' for 'is defined to be' and '.' for function application. We lift ordinary arithmetic operators pointwise to operators between functions: addition ($+$); multiplication ($\times$); maximum ($\sqcup$) and minimum ($\sqcap$). Other notation is introduced as needed.

2 Program Logic and Estimating Probabilities

Operationally we model probabilistic sequential programs [14, 7] (compare also [3, 11]) as functions from (initial) state to *sets* of distributions over (final) states. Intuitively that describes a computation proceeding in two (indivisible) stages: a nondeterministic choice, immediately followed by a probabilistic choice, where the probability refers to the possible final states reachable from a given initial state. In this view the role of nondeterminism is to allow an arbitrary selection over some range of probability distributions; however the agent making that selection can only influence the weights of the probabilistic transitions, not their *actual* resolution once those weights have been picked. That behaviour is precisely the kind exhibited by an 'adversary scheduler' assumed of many distributed algorithms [11]. We shall discuss schedulers in more detail in Sec. 4.

Unlike other authors we shall not use the operational model directly for program analysis. We introduce it only as an aid to intuition and in practice we use the quantitative program logic introduced elsewhere [14, 15]: it is equivalent to the operational view and is analytically more attractive. The idea, first suggested by Kozen [8] for deterministic programs, is to extract information about probabilistic choices by considering 'expected values'. Ordinary program logic [4] identifies preconditions that guarantee post conditions, in contrast, for probabilistic programs, the *probability*, rather than the certainty of achieving a post condition is of interest, and Kozen's insight was to formulate that as the result of averaging certain real-valued functions of the state over the final distributions determined by the program. Thus, the quantitative logic we use is an extension of Kozen's (since our programs are both nondeterministic and probabilistic) and is based on *expectations* (real-valued functions of the state rather than Boolean-valued predicates). We denote the space of expectations by $\mathcal{E}S(:= S \to \mathbb{R})$, and we define the semantics of a probabilistic program r as $wp.r$, an *expectation transformer* [14].

Definition 2.1 Let $r: S \to \mathbb{P}(\mathcal{D}S)$ be a program taking initial states in S to sets of final distributions over S. Then the *least possible*[1] pre-expectation at state s of program r, with respect to post-expectation A in $\mathcal{E}S$, is defined

[1] This interpretation is the same as the *greatest guaranteed* pre-expectation used elsewhere [14].

$$wp.r.A.s \quad := \quad (\sqcap F\colon r.s \bullet \int_F A) \ ,$$

where $\int_F A$ denotes the integral of A with respect to distribution F.[2] □

In the special case that A is a $\{0,1\}$-valued — a *characteristic* expectation — we may identify a predicate which is *true* exactly at those states where A evaluates to 1, and then the above interpretation makes $wp.r.A.s$ the least possible probability that r terminates in a state satisfying that predicate. To economise on notation we often pun a characteristic expectation with its associated predicate, saying that 's satisfies A' when strictly speaking we mean $A.s = 1$. The context should dispel confusion, however. Other distinguished functions are the constants $\underline{1}$ and $\underline{0}$ evaluating everywhere to 1 and 0 respectively and thus corresponding to *true* and *false*. We also write $\overline{A}$ for the negation of A (equivalent to $\underline{1} - A$).

By taking the minimum over a set of distributions in Def. 2.1, we are adopting the demonic interpretation for nondeterministic choice, and for many applications it is the most useful, since it generalises the 'for all' modality of transition semantics [1]. Thus if $wp.r.A.s = p$ say for some real p then *all* (probabilistic) transitions in $r.s$ ensure a probability of at least p of achieving A. For our application to performance, however, upper bounds have more significance: thus we define also the dual of $wp.r$, generalising the 'exists' modality [1]. (Compare also upper and lower probability estimates [3].)

Definition 2.2 Let $r\colon S \to \mathbb{P}(\mathcal{D}S)$ be a program, taking initial states in S to sets of final distributions over S. Then the *greatest possible* pre-expectation at state s of program r, with respect to post-expectation A in $\mathcal{E}S$, is defined[3]

$$\widetilde{wp}.r.A.s \quad := \quad \underline{1} - wp.r.(\underline{1} - A).s \ .$$

□

The semantics for a (restricted) programming language, sufficient for the applications of later sections, is set out in Fig. 1. It is essentially the same as for ordinary predicate transformers [4] except for probabilistic choice which is defined as the weighted average of the pre-expectations of its operands.

To illustrate the logic we consider the program set out in Fig. 2, for which we calculate the least possible probability that the variable b is set to *true* after a single execution. From Def. 2.1 that is given by wp.Chooser.$\{b = true\}$, where $\{e = v\}$ denotes the predicate (i.e. characteristic expectation) 'e is equal to v'. From Fig. 1 we see that in order to evaluate the conditional choice in Chooser, we need to consider each of the options separately.

We calculate the '$b = false$' case first. Denoting equivalence of expectations by $\equiv$, we reason:

[2] In fact $\int_F A$ is just $\sum_{s:S} A.s \times F.s$ because S is finite and and F is discrete [5]. We use the $\int$-notation because it is less cluttered, and to be consistent with the more general case.

[3] This is equivalent to interpreting nondeterministic choice as maximum, and so $\widetilde{wp}.r.A.s = (\sqcup F\colon r.s \bullet \int_F A)$. It is similar to an angelic interpretation for nondeterminism.

assignment	$wp.(x := E).A \;:=\; A[E/x]$
probabilistic choice	$wp.(r\ {}_p\!\oplus\ r').A \;:=\; p(wp.r.A) + (1-p)(wp.r'.A)$
nondeterministic choice	$wp.(r [\!] r').A \;:=\; wp.r.A \sqcap wp.r'.A$
sequential composition	$wp.(r; r').A \;:=\; wp.r.(wp.r'.A)$
conditional choice	$wp.(r \textbf{ if } B \textbf{ else } r').A \;:=\; B{\times}wp.r.A \;+ \overline{B}{\times}wp.r'.A$

A is in $\mathcal{E}S$ and E is an expression in the program variables. The expression $A[E/x]$ denotes replacement of variable x by E in A. The real p satisfies $0 \leq p \leq 1$, and pA means 'expectation A scaled by p'. Finally B is Boolean-valued when it appears in a program statement but is interpreted as a $\{0,1\}$-valued expectation in the semantics.

Fig. 1. Probabilistic *wp* semantics. Nondeterminism is interpreted demonically.

Chooser $:=$ $(b := true)$ **if** $\{b = true\}$ **else** $(b := true) [\!] (b := true\ {}_{1/2}\!\oplus\ b := false)$

If b is *false* initially then it can be either set to *true* unconditionally, or only with probability 1/2: the choice between those two options is resolved nondeterministically.

Fig. 2. Randomised Chooser with a Boolean-valued variable b.

$$
\begin{array}{lll}
 & wp.((b := true) [\!] (b := true\ {}_{1/2}\!\oplus\ b := false)).\{b = true\} & \\
 & & \text{nondeterministic choice} \\
\equiv & wp.(b := true).\{b = true\} \sqcap wp.(b := true\ {}_{1/2}\!\oplus\ b := false).\{b = true\} & \\
 & & \text{assignment} \\
\equiv & \{true = true\} \sqcap wp.(b := true\ {}_{1/2}\!\oplus\ b := false).\{b = true\} & \\
 & & \text{see below} \\
\equiv & \underline{1} \sqcap (wp.(b := true\ {}_{1/2}\!\oplus\ b := false).\{b = true\}) & \\
 & & \text{probabilistic choice} \\
\equiv & \underline{1} \sqcap (1/2(wp.(b := true).\{b = true\}) + 1/2(wp.(b := false).\{b = true\})) & \\
 & & \text{assignment} \\
\equiv & \underline{1} \sqcap (\{true = true\}/2 + \{false = true\}/2) & \\
 & & \text{see below} \\
\equiv & \underline{1} \sqcap (1/2{\times}\underline{1} + 1/2{\times}\underline{0}) & \\
 & & \\
\equiv & \underline{1/2}\ . & \text{arithmetic}
\end{array}
$$

For the deferred justifications, we use the equivalences $\{true = true\} \equiv \underline{1}$ and $\{false = true\} \equiv \underline{0}$.

A similar (though easier) calculation follows for the '$b = true$' case, resulting in $wp.(b := true).\{b = true\} \equiv \underline{1}$, and putting the two together with the rule for conditional choice we find

$$wp.\text{Chooser}.\{b = true\} \;\equiv\; \{b = true\} + \{b = false\}/2\ , \tag{1}$$

implying that there is a probability of at *least* 1/2 of achieving $\{b = true\}$ if execution of Chooser begins at $\{b = false\}$ and of (at least) 1 if execution begins at $\{b = true\}$.

In contrast, we can calculate the *greatest possible* probability of reaching $\{b = false\}$ using Def. 2.2:

$$\left.\begin{array}{lll} & \widetilde{wp}.\text{Chooser}.\{b = false\} & \\ \equiv & \underline{1} - wp.\text{Chooser}.(\underline{1} - \{b = false\}) & \text{Def. 2.2} \\ \equiv & \underline{1} - wp.\text{Chooser}.\{b = true\} & b \text{ is Boolean-valued} \\ \equiv & \underline{1} - (\{b = true\} + \{b = false\}/2) & (1) \\ \equiv & \{b = false\}/2\ , & \end{array}\right\} \quad (2)$$

yielding a probability of at *most* 1/2 if execution begins at $\{b = false\}$ and 0 if it begins at $\{b = true\}$ — there is no execution from $\{b = true\}$ to $\{b = false\}$.

In this section we have considered maximum and minimum probabilities using wp and $\widetilde{wp}$ for a single execution of a program. In the next section we extend the ideas to temporal-style semantics.

3 Computation Trees and Fixed Points

In this section we consider arbitrarily many executions of a fixed program denoted $\bigcirc$. Later we shall interpret it as wp.prog for some program prog, but for the moment we adopt an abstract view. (We shall also use $\widetilde{\bigcirc}$ to be interpreted as $\widetilde{wp}$.prog.) Ordinary program semantics of such systems are computation trees [1], with each arc of the tree representing a transition determined by $\bigcirc$. A *path* in the tree represents the effect of some number of executions of $\bigcirc$, and is defined by a sequence whose entries are (labelled by) the states connecting contiguous arcs. When $\bigcirc$ contains probabilistic choices, the probabilistic transitions in $\bigcirc$ generate (sets of) probability distributions over paths of the computation tree: probabilities over finite paths may be calculated directly, and they determine a well defined probability distribution over all paths.[4] Our aim for this section is, as for the state-to-state transition model, to extract probabilistic information about the paths by interpreting 'path operators' (defined with expectation transformers) over the computation trees, again avoiding direct reference to the underlying path-distributions.

Standard treatments of tree-based semantics often use μ-calculus expressions in the program logic [9], and it turns out [16] that such formulations for the temporal properties 'eventually' and 'always' have straightforward generalisations in the quantitative logic by replacing $\vee$ and $\wedge$ in those expressions by $\sqcup$ and $\sqcap$ respectively. The resulting operators, $\Diamond$ and $\Box$ (set out in Fig. 3), when applied to a characteristic expectation A return the probability (rather than the certainty) that eventually or always A holds of the paths in the computation tree.

[4] It is usually called the Borel measure over cones [5].

But in the more general setting we can do more: for the reals support a wider range of operators (than do Booleans), promising greater expressivity; indeed as well as correctness we can also express *performance* directly within the logic. Our first performance measure denoted ΔA, (also set out in Fig. 3) expresses the expected length of the path in the computation tree until predicate A holds. In the context of program analysis it corresponds to the expected *number* of (repeated) executions of $\bigcirc$ required until A holds.

least possible eventually	$\Diamond A := (\mu X \bullet A \sqcup \bigcirc X)$
greatest possible eventually	$\widetilde{\Diamond} A := (\mu X \bullet A \sqcup \widetilde{\bigcirc} X)$
least possible always	$\Box A := (\nu X \bullet A \sqcap \bigcirc X)$
greatest possible always	$\widetilde{\Box}\, A := (\nu X \bullet A \sqcap \widetilde{\bigcirc} X)$
least possible time to A	$\Delta A := (\mu X \bullet \underline{0} \ \textbf{if}\ A\ \textbf{else}\ (\underline{1} + \bigcirc X))$
greatest possible time to A	$\widetilde{\Delta} A := (\mu X \bullet \underline{0} \ \textbf{if}\ A\ \textbf{else}\ (\underline{1} + \widetilde{\bigcirc} X))$

A is $\{0,1\}$-valued.

Fig. 3. Expectation operators with respect to a distribution over the computation tree generated by $\bigcirc$.

To make the link between the fixed-point expression for ΔA in Fig. 3 and expected length of the computation path to reach A we unfold the fixed point once: if A holds it returns 0 — no more steps are required to achieve A along the path; otherwise we obtain a '1+' — at least one more step is required to reach A[5]. A formal justification is given elsewhere [13].

In general the μ-calculus expressions correspond to the 'for all' or 'exists' fragments of temporal logic [1] according to whether they are defined with $\bigcirc$ or $\widetilde{\bigcirc}$. For example $\widetilde{\Diamond} A$ defined with $\widetilde{\bigcirc}$ returns the maximum possible probability that a path satisfies eventually A. Also $\widetilde{\Delta} A$ gives an upper bound on the number of steps required to reach A.

The introduction of fixed points requires a notion of partial order on $\mathcal{E}S$, and here we use one induced by *probabilistic implication* $\Rrightarrow$ (defined next with its variants) extending ordinary Boolean implication:

$$\begin{array}{l} \Rrightarrow \text{ 'everywhere no more than'} \\ \equiv \text{ 'everywhere equal to'} \\ \Lleftarrow \text{ 'everywhere no less than' .} \end{array}$$

[5] An equivalent formulation for $\widetilde{\Delta} A$ is $(\mu X \bullet \overline{A} \times (\underline{1} + \widetilde{\bigcirc} X))$. We shall use this more succinct form in our calculations rather than that set out in Fig. 3, which is helpful only in that it is close to the informal explanation.

In fact we define fixed points within certain subsets of $\mathcal{E}S$ because the existence of the fixed points are assured in spaces that have a least or greatest element and the whole of $\mathcal{E}S$ has neither.[6] We are careful however to choose subsets that suit our interpretation. Thus for least fixed points (μ) we take the non-negative expectations: the least element is $\underline{0}$, and our applications — average times and probabilities of events — are all the result of averaging over non-negative expectations. For greatest fixed points (ν) we restrict to expectations bounded above by $\underline{1}$: the greatest element is $\underline{1}$ and we use greatest fixed points only to express probabilities which involve averaging over characteristic expectations, themselves bounded above by $\underline{1}$. We set out the full technical details elsewhere [16].

feasibility	$\widetilde{\Diamond} A \Rrightarrow \underline{1}$
duality	If $A \Rrightarrow \underline{1}$ then $\widetilde{\Diamond} A \equiv \underline{1} - \Box(\underline{1} - A)$
invariants	If $I \Rrightarrow \bigcirc I$ and $I \Rrightarrow A$ then $I \Rrightarrow \Box A$

A is $\{0,1\}$-valued and I are in $\mathcal{E}S$.

Fig. 4. Some properties of the path operators

The first property of Fig. 4 are general to expectation operators whereas the latter two apply only to fixed points. In particular the invariant law extends the notion of ordinary program invariants: an expectation I in $\mathcal{E}S$ is said to be an *invariant* of $\bigcirc$ provided $\bigcirc I \Lleftarrow I$. When I takes arbitrary values, the invariant law says that the probability that A always holds along the paths with initial state s is at least $I.s$. This property is fundamental to our treatment of 'rounds' in the next section.

We end this section with a small example illustrating $\widetilde{\triangle}$. We use the program Chooser set out in Fig. 2 above, (hence $\widetilde{\bigcirc}$ is interpreted as $\widetilde{wp}$.Chooser), and we wish to calculate $\widetilde{\triangle}\{b = true\}$ an upper bound on the expected number of times Chooser must be executed until b is set to *true*. In a simple case where there is a probability of success on each execution (specifically here if Chooser sets b to *true*) elementary probability theory implies that the expected time to success is the result of summing over a geometric distribution; in contrast the calculation below shows how to find that time using our program logic. (In fact since Chooser is nondeterministic, probability theory is not applicable, and the analysis requires a more general framework such as this one.) We note first that $\widetilde{\triangle}\{b = true\}$ evaluated at '$b = true$' is 0 (for b is already set to *true*). Thus we

[6] We also assume continuity of the operators concerned [14].

know that $\widetilde{\triangle}\{b = true\} \equiv q\{b = false\}$, for some non-negative real, q which we must determine (where recall that we use qA to mean 'A scaled by q'). With that in mind, we reason

$$
\begin{array}{lll}
 & \widetilde{\triangle}\{b = true\} & \\
 & & \text{definition } \widetilde{\triangle}\text{; Fig. 3 and footnote 5} \\
\equiv & \overline{\{b = true\}}\times(\underline{1} + \widetilde{wp}.\text{Chooser}.(\widetilde{\triangle}\{b = true\})) & \\
 & & \widetilde{\triangle}\{b = true\} \equiv q\{b = false\} \\
\equiv & \{b = false\}\times(\underline{1} + \widetilde{wp}.\text{Chooser}.(q\{b = false\})) & \\
 & & \text{see below} \\
\equiv & \{b = false\}\times(\underline{1} + q(\widetilde{wp}.\text{Chooser}.\{b = false\})) & \\
 & & \text{from (2)} \\
\equiv & \{b = false\}\times(\underline{1} + q\{b = false\}/2)\ . &
\end{array}
$$

For the deferred justification we are using the scaling property of $\widetilde{wp}$.Chooser which allows us to distribute the scalar q. [7]

Now evaluating at '$b = false$' we deduce from the above equality that

$$q = 1 + q/2\ ,$$

giving $q = 2$, and (unsurprisingly) an *upper bound* of 2 on the number of executions of Chooser required to achieve success.

4 Fair Adversary Schedulers and Counting Rounds

We now illustrate our operators above by considering Rabin and Lehmann's randomised solution [10] to the well-known problem of the dining philosophers. The problem is usually presented as a number of philosophers $P_1, .., P_N$ seated around a table, who variously think (T) or eat (E). In order to eat they must pick up two forks, each shared between neighbouring philosophers, where the i'th philosopher has left, right neighbours respectively P_{i-1} and P_{i+1} (with subscripts numbered modulo N). The problem is to find a distributed protocol guaranteeing that some philosopher will eventually eat (in the case that some philosopher is continuously hungry). Randomisation is used here to obtain a symmetrical solution in the sense that philosophers execute identical code — any non-random solution cannot both guarantee eating and be symmetrical [10].

The aim for this section is to calculate upper bounds on the expected time until some philosopher eats, and since we are only interested in the time to eat we have excluded the details following that event. The algorithm set out in Fig. 5 represents the behaviour of the i'th philosopher, where each atomic step is numbered. A philosopher is only able to execute a step provided he is scheduled and when he is, he executes exactly one of the steps, without interference from

[7] Scaling is a standard property of expectation operators from probability theory [20] which also holds here. Others are monotonicity and continuity. In fact only distribution of addition fails: nondeterminism forces a weakening of that axiom; compare suplinearity of Fig. 7.

the other philosophers. Fig. 5 then describes a philosopher as follows: initially he decides randomly which fork to pick up first; next he persists with his decision until he finally picks it up, only putting it down later if he finds that his other fork is already taken by his neighbour. We have omitted the details relating to the shared fork variables, and for ease of presentation we use labels T, E, l, r etc. to denote a philospher's state, rather than the explicit variable assignments they imply. Thus, for example, if P_i is in state L_i or P_{i-1} is in state R_{i-1}, it means the variable representing the shared fork (between P_i and P_{i-1}) has been set to a value that means 'taken'. The distributed system can now be defined as repeated executions of the program $[\!]_{1\leq i\leq N} P_i$, together with a fairness condition, discussed next.

$$
\begin{array}{lll}
1.\ \textbf{if}\ T_i & \to l_i \;{}_{1/2}\!\oplus r_i & \\
2.\ [\!]\ (l_i \vee r_i) & \to \textbf{if}\ (l_i \wedge \neg R_{i-1}) & \to L_i \\
 & \quad [\!]\ (l_i \wedge R_{i-1}) & \to l_i \\
 & \quad [\!]\ (r_i \wedge \neg L_{i+1}) & \to R_i \\
 & \quad [\!]\ (r_i \wedge L_{i+1}) & \to r_i \\
 & \quad \textbf{fi} & \\
3.\ [\!]\ (L_i \vee R_i) & \to \textbf{if}\ (L_i \wedge \neg R_{i-1}) & \to E_i \\
 & \quad [\!]\ (L_i \wedge R_{i-1}) & \to \mathbb{L}_i \\
 & \quad [\!]\ (R_i \wedge \neg L_{i+1}) & \to E_i \\
 & \quad [\!]\ (R_i \wedge L_{i+1}) & \to \mathbb{R}_i \\
 & \quad \textbf{fi} & \\
4.\ [\!]\ (\mathbb{L}_i \vee \mathbb{R}_i) & \to T_i & \\
\textbf{fi} & &
\end{array}
$$

The state T_i represents thinking, l_i (r_i) that a philosopher will attempt to pick up the left (right) fork next time he is scheduled, L_i (R_i) that he is holding only the left (right) fork, $\mathbb{L}_i$ ($\mathbb{R}_i$) that he will put down the left (right) fork next time he is scheduled and E_i that he eats. The use of state as a Boolean means 'is in that state'; as a statement it means 'is set to that state'.

Fig. 5. The i'th philosopher's algorithm [10].

A fundamental assumption of distributed systems is that of a scheduler. Roughly speaking it is the mechanism that manages the nondeterminism in $[\!]_{1\leq i\leq N} P_i$, and its only constraint here is fairness: if a philosopher is continuously hungry (or 'enabled') then he must eventually be scheduled (and for simplicity we assume that philosophers are either enabled or eating). That assumption, of course, means that counting atomic steps is pointless — any particular philosopher may be ignored for an arbitrary long interval whilst other philosophers are

scheduled. Instead we count *rounds*, an interval in which each philosopher has been scheduled at least once — and fairness guarantees that rounds exist.

However there is a problem: recall that the tree semantics for a distributed system is composed of paths (denoted by t) in which arcs represent the effect of atomic steps, therefore interpreting the performance measure $\widetilde{\triangle}$ directly over that tree would not estimate rounds. We must do something else: we construct a new tree, one in which $\widetilde{\bigcirc}$ is associated with the effect of a round rather than an atomic step, and we interpret $\widetilde{\triangle}$ over that. We map each path t (a sequence of states with successors determined by atomic steps) to t' (a sequence of states with successors determined by rounds) as follows: first t is divided into contiguous subsequences, each one containing a round (we have omitted the precise details of how this can be done); t' is the projection of t onto the states at the delimiters.[8] Now interpreting $\widetilde{\triangle}$ over the resulting tree will correctly provide an estimate of numbers of rounds rather than atomic steps; however we require more, namely a bound that dominates all possible such interpretations.

We assume some program Round which represents the overall effect of a round — it is the sequential composition, in some order, of some number of atomic steps (where each step is determined by some P_i in this case). To abstract from that order and how rounds are delimited, we assume no more about Round's behaviour except the following: we require Round to terminate and that each P_i appears somewhere in the sequential composition. The trick now is to specify those requirements (i.e. to define 'round') in a way that allows promotion of atomic-step properties to round properties: we use the technique of invariants.

$$\left.\begin{array}{ll} \textit{local invariants} & \text{If } I \text{ is in } \mathcal{E}S \text{ and } wp.(\llbracket_{1\le i\le N} P_i).I \Leftarrow I \text{ then} \\ & \quad wp.\text{Round}.I \Leftarrow I \ . \\ \textit{fair progress} & \text{If } I, I' \text{ in } \mathcal{E}S \text{ are both local invariants and} \\ & \text{there is some } i \text{ such that } wp.P_i.I \Leftarrow I' \text{ then} \\ & \quad wp.\text{Round}.I \Leftarrow I' \ , \end{array}\right\} \quad (3)$$

where an invariant I is said to be *local* if $wp.P_i.I \Leftarrow I$ for all i, equivalently if $wp.(\llbracket_{1\le i\le N} P_i).I \Leftarrow I$. This technique is very powerful, for although local invariants may be difficult to find, they are easy to check.

The fair progress property is specific to computations in the context of fair execution. It states that if an invariant I' holds initially, and if from within that invariant some (helpful) P_i establishes a second invariant I, then I must hold at the end of the round — fairness must guarantee that P_i executes at some stage in the round, and no matter how the round begins if I' holds initially, invariance ensures that it holds at that stage; after which the now established invariant I continues to hold, no matter how the round is completed. The local invariant property is a special case of that, and in fact implies termination of Round. Termination is specified by the property $wp.\text{Round}.\underline{1} \Leftarrow \underline{1}$ which follows from the local invariant rule with I taken to be $\underline{1}$.

[8] This is effectively sequentially composing the atomic steps.

The problem of counting rounds now becomes one of interpreting $\widetilde{\bigcirc}$ as $\widetilde{wp}$.Round in the definition of $\widetilde{\triangle}$, but then only using properties (3) to deduce properties of wp.Round (hence of $\widetilde{wp}$.Round, Def. 2.2). With those conventions we can specify the expected time to the first eating event as $\widetilde{\triangle}(\exists i \bullet E_i)$, and our next task is to consider how in general to estimate upper bounds on $\widetilde{\triangle}A$, for some A, using only local invariants.

We introduce a second performance operator set out in Fig. 6 which counts steps in a different way. Informally $\widetilde{\#}A$ counts the expected number of times that A holds ever along paths in the computation tree. If A holds in the current state on a path, it is deemed to be one more visit to A; similarly unfolding the fixed point once reveals a corresponding '1+' in that case [9].The new performance operator is related to $\widetilde{\triangle}\overline{A}$ by observing that for characteristic A the number of times A holds on the path is at least as great as the length of the path until $\overline{A}$ holds. Other properties of # are set out in Fig. 7. (Note that with this notation we have returned briefly to the abstract notions of Sec. 3.)

The connection between performance and local invariants is to be found in the visit-eventually rule. It generalises a result from Markov processes [5] which says that the expected number of visits to A is the probability that A is ever reached ($\widetilde{\Diamond}A$) conditioned on the event that it is never *re*visited (probability $1-p$). Its relevance here is that an upper bound on $\widetilde{\Diamond}A/(1-p)$ (and hence on $\widetilde{\#}A$) may be calculated from upper bounds on both $\widetilde{\Diamond}A$ and $\widetilde{\bigcirc}\widetilde{\Diamond}A$, both of which are implied by *lower bounds* on local invariants, and the next theorem sets out how to do that.

least possible visits to A	$\#A := (\mu X \bullet (\bigcirc X) \ \textbf{if} \ \overline{A} \ \textbf{else} \ A + \bigcirc X)$
greatest possible visits to A	$\widetilde{\#}A := (\mu X \bullet (\widetilde{\bigcirc} X) \ \textbf{if} \ \overline{A} \ \textbf{else} \ A + \widetilde{\bigcirc} X)$

A is $\{0,1\}$-valued.

Fig. 6. The expected number of visits.

Theorem 4.1 Consider a distributed system defined by processes $P_1, ..., P_N$ arbitrated by a fair scheduler. Given an expectation A and local invariants I, I' such that $A \Rrightarrow \underline{1} - I$ and $I' \Rrightarrow wp.P_i.I$ for some i, then $\widetilde{\#}A$, the maximal possible number of times that A holds (after executions of Round) is given by

$$\widetilde{\#}A \Rrightarrow \underline{1/(1-q)} \ ,$$

[9] A more succinct form for $\widetilde{\#}A$ is given by $(\mu X \bullet \ A + \widetilde{\bigcirc} X)$.

suplinearity	$\widetilde{\#}(A+B) \Rrightarrow \widetilde{\#}A + \widetilde{\#}B$
visit-reach	$\widetilde{\triangle}\overline{A} \Rrightarrow \widetilde{\#}A$ with equality if $\overline{A} \Rrightarrow \bigcirc\overline{A}$
visit-eventually	$\widetilde{\#}A \Rrightarrow \widetilde{\Diamond}A/(1-p)$, where $p := (\sqcup s\colon A \bullet \widetilde{\bigcirc}(\widetilde{\Diamond}A).s)$.

A is $\{0,1\}$-valued. We write $(\sqcup s\colon A \bullet f.s)$ for the maximum value of $f.s$ when s ranges over the (predicate) A. In the visit-eventually rule, p is the greatest possible probability that A is ever revisited; if $p = 1$ that upper bound is formally infinite.

Fig. 7. Some properties of expected visits

where $q := 1 - (\sqcap s : A \cdot I'.s)$ and $\widetilde{\bigcirc}$ is defined to be $\widetilde{wp}$.Round in the definition of $\widetilde{\#}$.

Proof: Using the notation of the visit-eventually property of Fig. 7 we see that an upper bound on $\widetilde{\#}A$ is given by upper bounds on both $\widetilde{\Diamond}A$ and p. By appealing to feasibility (Fig. 4) and arithmetic we deduce immediately that $\widetilde{\#}A \Rrightarrow 1/(1-q)$ for any $q \geq p$. All that remains is to calculate the condition on q. We begin by estimating an upper bound for $\widetilde{wp}$.Round.$(\widetilde{\Diamond}A)$.

$$
\left.
\begin{array}{lll}
 & \widetilde{wp}.\mathrm{Round}.(\widetilde{\Diamond}A) & \text{Fig. 7} \\
 & & \text{monotonicity, footnote 7; } A \Rrightarrow \underline{1} - I \\
\Rrightarrow & \widetilde{wp}.\mathrm{Round}.(\widetilde{\Diamond}(\underline{1} - I)) & \\
\equiv & \widetilde{wp}.\mathrm{Round}.(\underline{1} - \Box(\underline{1} - (\underline{1} - I))) & \text{duality, Fig. 4} \\
\equiv & \widetilde{wp}.\mathrm{Round}.(\underline{1} - \Box I) & \text{arithmetic} \\
\Rrightarrow & \widetilde{wp}.\mathrm{Round}.(\underline{1} - I) & I \Rrightarrow \Box I\text{; invariants, Fig. 4} \\
\equiv & \underline{1} - wp.\mathrm{Round}.I & \text{Def. 2.2} \\
\Rrightarrow & \underline{1} - I' \ , & \text{fair progress, (3)}
\end{array}
\right\} \quad (4)
$$

where in the last step we are using our assumption that there is some philosopher such that $wp.P_i.I \Leftarrow I'$ for local invariant I'. Next we bound q:

	$q \geq p$	
if	$q \geq (\sqcup s\colon A \cdot \widetilde{wp}.\mathrm{Round}(\widetilde{\Diamond}A).s)$	definition p, Fig. 7
if	$q \geq (\sqcup s\colon A \cdot (\underline{1} - I').s)$	(4)
if	$q \geq 1 - (\sqcap s\colon A \cdot I'.s)$,	arithmetic

as required. □

Finally we are in a position to tackle the dining philosophers: our principal tools will be Thm. 4.1 and suplinearity of Fig. 7 — the latter property allows

the problem to be decomposed, making application of Thm. 4.1 easier (since it is then applied to fragments of the state space rather than to the whole state space). Deciding how to choose an appropriate decomposition often depends on the *probabilistic invariant* properties, and we discuss them next.

Ordinary program invariants are predicates that are maintained; probabilistic invariants, on the other hand, describe predicates that are maintained *only with some probability*, and we end this section by illustrating one such for the dining philosophers.

Consider first the case first when two philosophers (P_i and P_{i+1}) are in the configuration $(l_i \vee L_i) \wedge (r_{i+1} \vee R_{i+1})$. Informally we reason that the configuration is maintained unless one of P_i or P_{i+1} eats: other philosophers cannot disturb it, and inspection of P_i shows that l_i can only evolve to L_i, and that L_i can only evolve to E_i (since P_{i+1} is in state $r_{i+1} \vee R_{i+1}$). More generally we reason that $A{:=}\,(\exists i \cdot (l_i \vee L_i) \wedge (r_{i+1} \vee R_{i+1}) \vee E_i)$ is always maintained, if it is ever established.

Next, to factor in probability, we look for a property that is only maintained with some probability. Again we consider the neighbours P_i and P_{i+1} but this time for the the configuration $(l_i \vee L_i) \wedge T_{i+1}$. As before the configuration can only evolve to eating, unless P_{i+1} is scheduled, and when it is the state r_{i+1} (and thus A) is established with probability $1/2$; hence defining A' to be

$$(\exists i \cdot (l_i \vee L_i) \wedge T_{i+1}) \wedge \overline{A}\ ,$$

we argue that if $A \vee A'$ ever holds, then it must *continue to hold* with probability at least $1/2$. Expressed in the logic, we say that that $A + A'/2$ is a local invariant. Checking formally that $wp.P_i.(A + A'/2) \Leftarrow A + A'/2$ confirms that fact. (Notice that the invariants are expressions over the whole state space, however we only need check invariance with respect to a *single* philosopher.)

Finally if we define $I{:=}\,A$ and $I'{:=}\,A + A'/2$ the remarks above imply that Thm. 4.1 provides an upper bound of $\underline{2}$ on $\widetilde{\#}A'$. Intuition tells us that the bound must be finite since from probability theory [5] A' cannot be visited infinitely often if there is always a probability of $1/2$ of reaching an ordinary invariant A, disjoint from A'.

Calculations such as the above are required to find individual upper bounds on the return visits to a collection of predicates whose union implies the whole space. The particular invariant properties of the algorithm provide the best guide for choosing a decomposition. For example, if I is a (standard) local invariant and J is a predicate disjoint from I, and if $\bigcirc.I.s = p$ for all s in J, where $0 < p < 1$ then $I + pJ$ is also a (probabilistic) invariant. Thus I and J might form part of a suitable decomposition. (We would put $I'{:=}\,I + pJ$ to deduce a maximum of $1/(1{-}p)$ visits to J.) But whatever the decomposition, the most important feature of this method is that once invariants are discovered, verifying that they satisfy the properties of Thm. 4.1 is straightforward in the logic.

The precise decomposition used for the analysis of the dining philosophers follows that of the correctness proof (appearing elsewhere [17]), and it gives a

total expected time to first eating of no more than 33 (compare 63 of [11]). The details of those calculations are set out elsewhere [12].

5 Conclusion

In this paper we have shown how ordinary correctness techniques of distributed algorithms can be applied to probabilistic programs by using quantitative program logic, and that the methods apply even in the evaluation of performance. This treatment differs from other approaches to performance analysis of probabilistic algorithms [11, 3, 6] in that we do not refer explicitly to the distribution over computation paths; neither do we factor out the nondeterminism as a first step nor do we analyse the behaviour of the composed system: instead we use compositionality of local properties thus simplifying our formal reasoning. Other approaches using expectations [19, 8] do not treat nondeterminism and thus are not applicable to distributed algorithms like this at all.

Acknowledgements

This paper reports work carried out at Oxford within a project supported by the *EPSRC* — Carroll Morgan and Jeff Sanders are also members of that project.

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] L. de Alfaro. Temporal logics for the specification of performance and reliability. *Proceedings of STACS '97*, LNCS volume 1200, 1997.
[4] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall International, Englewood Cliffs, N.J., 1976.
[5] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. Wiley, second edition, 1971.
[6] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
[7] Jifeng He, K.Seidel, and A. K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2,3):171–192, January 1997.
[8] D. Kozen. A probabilistic PDL. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, New York, 1983. ACM.
[9] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
[10] D. Lehmann and M. O. Rabin. On the advantages of free choice: a symmetric and fully-distributed solution to the Dining Philosophers Problem. In *Proceedings of the 8th Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, New York, 1981. ACM.

[11] N. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. *Proceedings of 13th Annual Symposium on Principles of Distributed Algorithms*, pages 314–323, 1994.
[12] A.K. McIver. Quantitative program logic and efficiency in probabilistic distributed algorithms. Technical report. See QLE98 at *http* [18].
[13] A.K. McIver. Reasoning about efficiency within a probabilistic mu-calculus. 1998. Submitted to pre-LICS98 workshop on Probabilistic Methods in Verification.
[14] 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.
[15] Carroll Morgan. The generalised substitution language extended to probabilistic programs. In *Proceedings of B'98: the Second International B Conference. See B98 at* http *[18]*, number 1397 in LNCS. Springer Verlag, April 1998.
[16] 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 [18].
[17] C.C. Morgan and A.K. McIver. Correctness proof for the randomised dining philosophers. See RDP96 at *http* [18].
[18] PSG. Probabilistic Systems Group: Collected reports. *http://www.comlab.ox.ac.uk/oucl/groups/probs/bibliography.html*.
[19] M. Sharir, A. Pnueli, and S. Hart. Verification of probabilistic programs. *SIAM Journal on Computing*, 13(2):292–314, May 1984.
[20] P. Whittle. *Probability via expectations*. Wiley, second edition, 1980.