

# A generalisation of stationary distributions, and probabilistic program algebra

A.K.McIver<sup>1,2</sup>

*Department of Computing  
Macquarie University  
NSW, Australia*

---

## Abstract

We generalise the classical notion of stationary distributions of Markov processes to a model of probabilistic programs which includes demonic nondeterminism. As well as removing some of the conditions normally required for stationarity, our generalisation allows the development of a complete theory linking stationary behaviour to long-term average behaviour — the latter being an important property that lies outside the expressive range of standard logics for probabilistic programs.

**Keywords:** Probabilistic program semantics, probability, demonic nondeterminism, Markov process, stationary distribution, Markov decision processes.

---

## 1 Introduction

Programs or processes which can make probabilistic choices during their execution exhibit a range of (probabilistic) behaviours outside those describable by purely *qualitative* formalisms; moreover even well-known *quantitative* adaptations of familiar program logics — the foremost being probabilistic temporal logic [18,2] — are still not expressive enough in some cases. One such is the so-called “average long-term” behaviour [3,4], which we illustrate in the context of the program presented in Fig. 1. The program *FP* represents a specification of a simple failure-repair mechanism. The system it describes is intended to execute repeatedly, and the state evolves according to the specified probabilistic statements. The average long-term behaviour of *FP* determines (for example) the proportion of time that the state is *ok*, and is always well-defined [4]. Other related terms are “availability” [17] and “the stationary probability of *ok*” [8]. In this particular case an elementary analysis reveals that *ok* holds

---

<sup>1</sup> This work was done at Oxford University, UK, and was funded by the EPSRC.

<sup>2</sup> Email: [anabel@ics.mq.edu.au](mailto:anabel@ics.mq.edu.au)

$$\begin{aligned}
FP &:= \mathbf{if} \ ok \\
&\quad \mathbf{then} \ (ok \textstyle\frac{2}{3}\oplus \neg ok) \ \square \ ok \\
&\quad \mathbf{else} \ (ok \textstyle\frac{1}{2}\oplus \neg ok) \ \square \ ok \\
&\quad \mathbf{fi}
\end{aligned}$$

$ok$  and  $\neg ok$  toggle the states corresponding to working and broken behaviour. The operator  $\textstyle\frac{2}{3}\oplus$  records a probabilistic update, whereas  $\square$  records a nondeterministic update. Used together like this, we are able to specify tolerances on failure rates — at every execution, there is *at least* a probability  $1/2$  of reestablishing  $ok$  (since the only other alternative to the probabilistic branch establishes  $ok$  with certainty).

Fig. 1. An abstract failure-repair mechanism

---

on average at least  $3/5$  of the time — yet probabilistic temporal logic cannot describe that behaviour. (de Alfaro gives a nice discussion of the issues [3].)

In elementary probability theory, long term average behaviour is, in some special cases, determined by “stationary distributions” — a property of (some) Markov processes. Though some authors [10,16] have used Markov processes as a model for probabilistic programs, more recently a generalised form [13,9,2] has been found to be more suitable, since it supports the notions of (demonic) nondeterminism (or abstraction) and the induced partial order known as refinement. That is the model we shall work with here, and we give details in Sec. 5.

Thus our main contribution (in Sec. 3) is to give an axiomatic account of stationary behaviour and convergence to it, one which extends and simplifies the classical notion. Not only is our notion of generalised convergence applicable to all Markov processes (rather than only to some special cases) but it completes the theory linking stationary behaviour to average long-term behaviour. The details are set out in Sec. 5.

We develop our theory following the algebraic style already available in theories of concurrency, where it has proved a powerful tool for analysing nondeterministic programs that execute repeatedly.

We use “.” for function application;  $\leq$ ,  $+$  and  $\sqcap$  denote respectively “is no more than”, addition and minimum applied pointwise to real-valued functions. Throughout  $S$  is a finite state space and  $\overline{S}$  is  $\{F \mid S \rightarrow [0, 1] \cdot \sum_{s:S} F.s = 1\}$ , the set of (discrete) probability distributions over  $S$ . For real  $k$ , we write  $\underline{k}$  for the constant real-valued function with range  $\{k\}$ . If  $\alpha$  is a real-valued function over  $S$  then  $(\sqcup\alpha)$  and  $(\sqcap\alpha)$  denote respectively the maximum and minimum value taken by  $\alpha$  as the state varies over  $S$ ; and  $(k\alpha)$  or  $k(\alpha)$  represents the the function  $\alpha$  pointwise multiplied by the real  $k$ . We introduce other notation as we need it.

## 2 Probabilistic sequential programs

We summarise two equivalent models for probabilistic programs; more details are given elsewhere [13,9]. The semantics for probabilistic sequential programs supports the interpretation of traditional programming control structures together with a binary probabilistic choice operator  $_p\oplus$ , where the operational meaning of the expression  $A \ _p\oplus \ B$  is that either  $A$  or  $B$  is executed, with probability respectively  $p$  or  $1-p$ . Since there is no determined output, that behaviour is sometimes called “probabilistic nondeterminism”. Probabilistic nondeterminism is however very different from “demonic nondeterminism”, denoted by “[ $\square$ ]”, already present in standard guarded commands [5], and which can model underspecification or demonic scheduling in distributed systems.

And the two operators are modelled very differently — as usual probabilistic information is described by (output) probability distributions over final states, whereas demonic behaviour is described by subsets of possible outputs. Putting those two ideas together leads to a model in which programs correspond to functions from initial state to sets of distributions over final states, where the multiplicity of the result set represents a degree of nondeterminism and the distribution records the probabilistic information after that nondeterminism has been resolved. We have the following definition for the probabilistic program space  $\mathcal{HS}$  [9,13] for programs operating over the abstract state space  $S$ ,<sup>3</sup> and its treatment of nondeterminism is similar to that of other models [2,15,4]:

$$\mathcal{HS} := S \rightarrow \overline{\mathbb{P}S} .$$

More generally, like Markov processes, every program in  $\mathcal{HS}$  can be considered to be a function from probability distributions over initial states, but in this case to *sets* of probability distributions over final states [9].

We order programs using program refinement, which compares the extent of nondeterminism — programs higher up the refinement order exhibit less nondeterminism than those lower down:

$$Q \sqsubseteq P \text{ iff } (\forall s: S \cdot P.s \subseteq Q.s) .$$

Classical Markov processes can be identified with the subclass of “deterministic”, or purely probabilistic programs in  $\mathcal{HS}$ , and as such are maximal with respect to  $\sqsubseteq$ . For instance the (demonically deterministic) program  $ok_{1/2} \oplus \neg ok$  has no proper refinements at all.

One consequence of  $\sqsubseteq$  above is that (worst case) quantitative properties improve as programs become more refined. If  $Q$  guarantees to establish a predicate  $\phi$  with probability at least  $p$  (irrespective of the nondeterminism), then  $P$  must also establish  $\phi$  with probability at least that same  $p$ .

That observational view of probabilistic systems (in which the frequency of outputs is recorded) is captured more generally with the idea of “expected values”. Kozen was the first to exploit this fact in his probabilistic program logic

<sup>3</sup> This basic model can also be enhanced to include nontermination [13] and miracles [14].

(but for deterministic programs). His insight was to regard programs as operators which transform real-valued functions in a goal-directed fashion, in the same way that standard programs can be modelled as predicate transformers [5]. The use of real-valued functions instead of predicates allows expressions to incorporate quantitative (as well as qualitative) information. The idea has been extended by others [13] to include demonic nondeterminism as well as probability. We write  $\mathcal{E}S$  for the space of real-valued functions (expectations) over  $S$ , and  $\mathcal{T}S$  for the associated space of “expectation transformers”, defined next.

**Definition 2.1** Let  $r: S \rightarrow \mathbb{P}(\overline{S})$  be a program taking initial states in  $S$  to sets of final distributions over  $S$ . Then the *greatest guaranteed* pre-expectation at state  $s$  of program  $r$ , with respect to post-expectation  $\alpha$  in  $\mathcal{E}S$ , is defined

$$wp.r.\alpha.s \quad := \quad (\sqcap F: r.s \cdot \int_F \alpha),$$

where  $\int_F \alpha$  denotes the expected value of  $\alpha$  with respect to distribution  $F$ .<sup>4</sup> We say that  $wp.r$  is an *expectation transformer* corresponding to  $r$ , and we define  $\mathcal{T}S$  to be  $wp.\mathcal{H}S$ .

Programs are ordered by comparing the results of qualitative observations: thus

$$t \sqsubseteq t' \quad \text{iff} \quad (\forall \alpha : \mathcal{E}^+S \cdot t.\alpha \leq t'.\alpha),$$

where  $\mathcal{E}^+S$  denote the non-negative expectations. There is no conflict in using “ $\sqsubseteq$ ” to denote the order in both  $\mathcal{H}S$  and  $\mathcal{T}S$ , since the definitions correspond [13].

In the special case that the post-expectation takes values in  $\{0, 1\}$  and thus represents a predicate, the pre-expectation represents the greatest guaranteed probability of the program establishing that predicate. Nondeterminism, as for predicate transformers, is interpreted demomonically.

Although the two views are equivalent [13], we usually use  $\mathcal{T}S$  because its arithmetic properties make it more convenient for proof than  $\mathcal{H}S$ . Transformers in  $\mathcal{T}S$  are continuous (in the sense of real-valued functions) and subadditive, that is

$$t.(k\alpha + k'\beta - \underline{k}'') \quad \geq \quad k(t.\alpha) + k'(t.\beta) - \underline{k}'',$$

which can be strengthened to additivity in the case of deterministic programs (classical Markov processes). We interpret basic program constructs as operations on transformers: thus  $(t; t').\alpha := t.(t'.\alpha)$ ;  $(t \sqcap t').\alpha := t.\alpha \sqcap t'.\alpha$  and  $(t \text{ }_p\oplus\text{ } t').\alpha := p(t.\alpha) + (1-p)(t'.\alpha)$ , from which we see that determinism is preserved by  $\text{ }_p\oplus$  and  $;$ , but not by  $\sqcap$ .

The next lemma can be proved very simply using the notions of  $\mathcal{T}S$ . Define the norm  $\|\cdot\|$  on expectations as  $\|\alpha\| := (\sqcup \alpha) - (\sqcap \alpha)$ . Our definitions imply

<sup>4</sup> In fact  $\int_F \alpha$  is just  $\sum_{s:S} \alpha.s \times F.s$  because  $S$  is finite and  $F$  is discrete [6]. We use the  $\int$ -notation because it is less cluttered, and to be consistent with the more general case.

that if  $\|\alpha\| = 0$  then  $\alpha$  is constant on  $S$ .

**Lemma 2.2** *Let  $t, t'$  be an expectation transformers in  $\mathcal{TS}$ . If  $t$  is deterministic, and  $t'; t = t'$ ; and furthermore if there is some  $0 \leq c < 1$  such that for any  $\alpha$  we have  $\|t.\alpha\| \leq c\|\alpha\|$ , then  $t'$  is deterministic.*

**Proof:** *The above discussion suggests that we just need to show that  $t'$  is additive, which follows by continuity of transformers in  $\mathcal{TS}$ .*

Even though Lem. 2.2 is more generally true for any programs in  $\mathcal{TS}$  satisfying the conditions, it actually characterises the property which underlies whether a Markov process converges to its so-called stationary distribution or not, namely that it acts like a contraction with respect to  $\|\cdot\|$ . The term “contraction” however is more general and can be applied to the whole of  $\mathcal{TS}$ , not just to its deterministic portion:  $FP$  in Fig. 1 is a contraction for instance, though it is not a Markov process.

Conversely, if  $t^n$  is not a contraction for any power of  $t$  then it can be shown that there is some proper subset of states that is left invariant by  $t^n$ , for some  $n$ . Such programs are also called “periodic”, and we shall return to them later.

### 3 A program-algebraic treatment of ‘stationary behaviour’

In this section we study some algebraic properties of programs or systems that execute repeatedly. Algebraic approaches have proved to be very powerful in the development of concurrency theory [1]; we find them to be extremely effective in this context as well.

Our basic language (in Fig. 2) consists of two binary operators (“;”, sequential composition and “[ ]”, demonic nondeterministic choice), one constant (1, “do nothing”) and a unary operator (“\*”, the “Kleene star”). Both ; and [ ] are associative and [ ] is commutative; 1 is the identity of ;. Observe that for probabilistic models [ ] fails to distribute to the left. (Other nonprobabilistic interpretations would allow full distributivity [1].) We interpret  $x^*$  in  $\mathcal{TS}$  as the transformer  $x^*.\alpha := (\nu Y \cdot \alpha \sqcap x; Y)$ ,<sup>5</sup> which corresponds to the program that from initial state  $s$  outputs the strongest set of invariant states containing  $s$ . We shall also use the special program **chaos** which denotes a nondeterministic selection over all the states in  $S$ . A program  $t$  which can reach all states from all initial states (with probability 1) has no proper invariants, and thus satisfies  $t^* = \mathbf{chaos}$ .

Next we introduce our first generalisation — a probabilistic operator  ${}_p\oplus$ ; its properties [9] also appear in Fig. 2. Observe that the sub-distribution of  ${}_p\oplus$  corresponds to subadditivity of  $\mathcal{TS}$ .

We say that a probability distribution  $F$  in  $\overline{S}$  is *stationary* with respect to a Markov process  $t$  if whenever the input states are distributed as  $F$ , the

<sup>5</sup>  $\nu$  forms the greatest fixed point with respect to  $\leq$  on  $\mathcal{ES}$ .

$$\begin{array}{ll}
 x \sqsubseteq y \Leftrightarrow x \sqcap y = x & x^* = 1 \sqcap x \sqcap x^*; x^* \\
 x; (y \sqcap z) \sqsubseteq x; y \sqcap x; z & x; (y \sqcap 1) \sqsupseteq x \Rightarrow x; y^* = x \\
 (y \sqcap z); x = y; x \sqcap z; x & x; y \sqsupseteq y \Rightarrow x^*; y = y \\
 \\
 x \oplus_p y = y \oplus_{1-p} x & x \sqcap y \sqsubseteq x \oplus_p y \\
 x; y \oplus_p x; z \sqsubseteq x; (y \oplus_p z) & (y \oplus_p z); x = y; x \oplus_p z; x \\
 \\
 x \oplus_p (y \oplus_q z) = (x \oplus_{\frac{p}{p+q-pq}} y) \oplus_{p+q-pq} z
 \end{array}$$

$x, y, z$  are interpreted as programs in  $\mathcal{TS}$ , and  $0 < p < 1$ . The axioms without  $\oplus_p$  are similar to Kozen's axiomatisation of Kleene's language for regular expressions [11].

Fig. 2. Basic axioms

output states are also distributed exactly according to  $F$ . In this section we generalise this idea to all programs in  $\mathcal{TS}$ .

Observe first that any  $F$  in  $\overline{\mathcal{S}}$  can be modelled as the program that outputs  $F$  — we call such programs deterministic assignments. Writing  $\hat{F}$  for the deterministic assignment that outputs  $F$  for any initial state, we can see that the definition of stationarity above is the same as saying that  $\hat{F}; t = \hat{F}$  holds as an equality in  $\mathcal{TS}$ .

Our crucial generalising step is now to consider *any* program  $t'$  satisfying  $t'; t = t'$  to represent stationary behaviour (rather than only those programs  $\hat{F}$  generated from distributions  $F$  as above); that takes us beyond the classical treatment.

To fill in the details, we begin with the idea of weakest stationary program, as follows. We make use of  $x^*$  to encode “all invariants of  $x$ ”, noted above.

**Definition 3.1** Define  $x^\infty$  to be the the least program that is stationary with respect to  $x$  (that is, which satisfies  $x^\infty; x = x^\infty$ ) and which preserves all invariants of  $x$  (that is  $x^* \sqsubseteq x^\infty; x^*$ ). We have

$$x^\infty := (\sqcap y : \mathcal{HS} \cdot y; x \sqsubseteq y \wedge x^* \sqsubseteq y; x^*).$$

Note that an important intuitive property of  $x^\infty$  is that it preserves all invariants of  $x$  — an alternative definition that only considers stationarity (the first conjunct in Def. 3.1) gives the incorrect

$$(\sqcap y : \mathcal{HS} \cdot y; 1 \sqsubseteq y) = \mathbf{chaos} \neq 1^\infty = 1$$

for the case  $x = 1$ .

Program  $t^\infty$  can be thought of as delivering from initial state  $s$  the strongest invariant reachable from  $s$ , whilst preserving the probabilistic stationary be-

$$\begin{array}{ll}
 x^*; x^\infty = x^\infty & x^{\infty\infty} = x^\infty \\
 x^\infty \sqsubseteq x^{n\infty} & x^* \sqsubseteq x^{n*} \\
 x^* \sqsubseteq x^\infty & x^{*\infty} = x^* \\
 x; (y; x)^\infty \sqsubseteq (x; y)^\infty; x & x; x^\infty = x^\infty = x^\infty; x
 \end{array}$$

$$\begin{array}{l}
 (p > 0) \Rightarrow (x_{p\oplus 1})^\infty \sqsubseteq x^\infty \quad x^*; x = x^* \Rightarrow x^* = x^\infty \\
 x; y \sqsubseteq z; x \Rightarrow x; y^\infty \sqsubseteq z^\infty; y \quad x^{n*} = x^* \Rightarrow x^\infty = x^{n\infty}
 \end{array}$$

To avoid clutter, we write  $x^{n\infty}$  etc. instead of  $(x^n)^\infty$ .

Fig. 3. A selection of basic theorems

haviour. In fact  $t^\infty$  in  $\mathcal{TS}$  is the the limit of the increasing chain of programs  $t^* \sqsubseteq t^*; t \sqsubseteq t^*; t^2 \sqsubseteq \dots \sqsubseteq t^*; t^n \sqsubseteq \dots$ . That limit is well-defined since  $\mathcal{TS}$  is directed-complete, and hence we have the additional fact

$$(1) \quad (\forall n > 0 \cdot x^*; x^n \sqsubseteq y) \Rightarrow x^\infty \sqsubseteq y .$$

In Fig. 3 we set out some general theorems about  $^\infty$  and  $^*$ , all implied by the axioms of Fig. 2 and the properties of  $^\infty$  set out in Def. 3.1 and (1).

To see the difference between  $^*$  and  $^\infty$  we reconsider  $FP$  from Fig. 1. The only nontrivial invariant set of states is  $\{ok, \neg ok\}$ , hence  $FP^* = ok \sqcap \neg ok$ ; but this program is not stationary with respect to  $FP$ , and so  $FP^\infty \neq FP^*$ . In fact  $FP^\infty = (ok_{3/5} \oplus \neg ok) \sqcap ok$ , the generalised distribution in which the probability of  $ok$  is at least  $3/5$ .

## 4 Extended Markov theory

From (1) it is easy to see that in the general setting, any program  $t$  (if executed for long enough) achieves some notion of stationary behaviour encapsulated by the program  $t^\infty$ . But that is not the view taken by classical Markov process theory. To see where the general and the classical theories diverge, consider the program  $b := 1-b$ , where the variable  $b$  can only take values in  $\{0, 1\}$ . The classical theory says that this program does not converge (because it oscillates between  $b$ 's two values). On the other hand  $(b := 1-b)^\infty = (b := 1-b)^* = (b := 0 \sqcap b := 1)$ , which says that the long term stationary behaviour is a program that assigns to  $b$  nondeterministically from its type. That behaviour is disqualified by the classical theory because it is not deterministic and so does not represent a distribution. We discuss the ‘‘observational’’ intuition behind this solution in the next section.

For now we end this section by demonstrating that our generalised notion of convergence really supersedes the classical theory. We present a new proof of the important result about convergence to a stationary distribution of ‘‘ape-

riodic” Markov processes; the proof relies crucially on the ability to postulate the existence of  $t^\infty$  for all Markov processes, and not just those permitted by the classical theory.

Recall that a distribution is modelled as a deterministic assignment which is independent of the initial state. A transformer  $t$  which corresponds to such an assignment is additive and, for any  $\alpha$ , the expectation  $t.\alpha$  is a constant function. For example  $wp.(ok_{2/3} \oplus \neg ok).\alpha$  returns the expected (final) value of  $\alpha$ , which is constant at  $2(\alpha.ok)/3 + \alpha.(\neg ok)/3$ , whatever the initial value.

Hence in our terms all we need do is show that  $^\infty$  maps the aperiodic deterministic programs to transformers that correspond to deterministic assignments.

Aperiodicity is a property of  $t$  provided that all states are eventually reachable from all other states, and the probability of returning to the original state with a definite period is strictly less than 1 [8]. The first property is the same as saying that  $t^* = \mathbf{chaos}$ , and the second is the same as saying that  $t^{n*} = t^*$  for all  $n > 1$  — in the case that the equality fails for some  $n$ , we are saying that  $t$  exhibits a period of  $n$ . The general theorem about convergence of Markov processes is then as follows.

**Theorem 4.1** *If  $t$  in  $\mathcal{TS}$  is deterministic and aperiodic then  $t^\infty$  is a deterministic assignment.*

**Proof:** *The comment after Lem. 2.2 implies that  $t^n$  must be a contraction for some  $n > 0$ , and hence  $t^{n^\infty}$  must be a deterministic assignment (also by Lem. 2.2). The result follows from Fig. 3 since  $t^{n*} = t^*$ .*

## 5 Applications to long-term average behaviour

The properties of systems that execute indefinitely are usually investigated using an adaptation of temporal logic — in our case *probabilistic* temporal logic. Formulae are interpreted over trees of execution paths — in our case *probabilistic distributions* over execution paths [15,2]. The interpretation of a typical formula  $\phi$  over a path-distribution yields the proportion of paths satisfying  $\phi$ . As de Alfaro points out [3] however, this kind of “probabilistic satisfaction” refers to the aggregate path-distribution; put another way it measures the chance of a single event occurring *among* paths, and ignores the frequency with which events occur *along* paths. But this is precisely what is called for in *availability* or long-term average analyses of failing systems. In this section we show that both are determined by  $t^\infty$  — even for systems that include nondeterminism, such as *FP* in Fig. 1.

We define long-term average behaviour as de Alfaro [3] does. Given a sequence  $seq$  of expectations, let  $seq_i$  be the  $i$ 'th element, and define the partial sum  $\sum_k seq = seq_1 + seq_2 + \dots + seq_k$ .

**Definition 5.1** Let  $t$  in  $\mathcal{TS}$  execute indefinitely, and let  $\alpha$  be a predicate. The long-term average number of occurrences of  $\alpha$  observed to hold as  $t$  executes

is given by  $V_t.\alpha$  in

$$V_t.\alpha := \liminf_{k \rightarrow \infty} \frac{\sum_k seq}{k},$$

where in this case  $seq_k := t^*; t^k.\alpha$ .

Def. 5.1 corresponds to the average result after sampling the state of the system at arbitrary intervals of time as  $t$  executes repeatedly. Here we assume that at the  $k$ 'th sample point, the system has executed *at least*  $k$  times — and in that case the chance that  $\alpha$  holds at the time of the test is  $t^*; t^k.\alpha$ . When  $t$  corresponds to a Markov process that converges classically, that average is determined by the stationary distribution. We have a corresponding result here, but it is valid for *all* programs.

**Lemma 5.2** *Let  $t$  be a program in  $\mathcal{HS}$  and  $\alpha$  an expectation in  $\mathcal{ES}$ . Then we have  $t^\infty.\alpha = V_P.\alpha$ .*

To illustrate the above, recall the program  $b := 1-b$ , and let  $[b = 0]$  represent the expectation that evaluates to 1 at states where  $b$  is 0 and to 0 elsewhere. To calculate  $V_{b:=1-b}.[b = 0]$  we consider

$$wp.(b := 1-b)^*; (b := 1-b)^n.[b = 0] = \underline{0},$$

hence  $V_{b:=1-b}.[b = 0] = \underline{0}$  as well.

Alternatively,  $(b := 1-b)^\infty = (b := 0 \sqcup b := 1)$ , hence  $wp.(b := 1-b)^\infty.[b = 0] = 0$  also.

These results can be understood operationally in the context of a tester who is allowed to choose when to sample the state of the program. Clearly if the tester only observes the state after an even number of executions of  $b := 1-b$  then he will deduce that  $b$  is never 0 on average (or even at all). The point about aperiodic programs in the classical theory is that the average measurement is to an extent robust against such accidental testing bias. And the same applies here: whatever the proposed testing regime, the proportion of time that  $FP$  is  $ok$  will be found to be at least  $3/5$ , since  $FP^\infty = (ok_{3/5} \oplus \neg ok) \sqcup ok$ .

## 6 Conclusion

Our main contribution is to extend the notion of stationary behaviour of Markov processes to a model that includes demonic nondeterminism, setting it on a par with other programming concepts. The main insight was to model stationary behaviour explicitly as a distribution-generating program in  $\mathcal{TS}$ ; that allows access to the techniques of program algebra and probabilistic models [1,13]. The generalisation proposed here allows the completion of the theory linking long-term average behaviour and stationary behaviour — both are now always defined, and they determine each other. Moreover our generalisation provides a striking simplification to classical theory of convergence.

The operator  $t^*$  presented here is unable to express many of the experiments offered by the much more elaborate framework due to de Alfaro [3]. The main difference is that results are assigned to states rather than transitions. Nevertheless many useful performance measures are covered by this simpler framework. Examples include average waiting times and availability measures.

Further work is needed to incorporate other programming notions such as coercions [12], which significantly increase the power of algebraic reasoning.

An important consequence is that stationary behaviour is now susceptible to other programming techniques such as refinement and data abstraction [7].

## References

- [1] Ernie Cohen. Separation and reduction. In *Mathematics of Program Construction, 5th International Conference, Portugal, July 2000*, number 1837 in LNCS, pages 45–59. Springer Verlag, 2000.
- [2] L. de Alfaro. Temporal logics for the specification of performance and reliability. *Proceedings of STACS '97*, LNCS volume 1200, 1997.
- [3] L. de Alfaro. How to specify and verify the long-run average behavior of parobabilistic systems. In *Proceedings of 'LICS '98, 23-24 June, Indianapolis, 1998*.
- [4] C. Derman. *Finite State Markov Decision Processes*. Academic Press, 1970.
- [5] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall International, Englewood Cliffs, N.J., 1976.
- [6] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. Wiley, second edition, 1971.
- [7] P. H. B. Gardiner and C. C. Morgan. Data refinement of predicate transformers. *Theoretical Computer Science*, 87:143–162, 1991.
- [8] G. Grimmett and D. Welsh. *Probability: an Introduction*. Oxford Science Publications, 1986.
- [9] 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.
- [10] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.
- [11] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110:336–390, 1994.
- [12] C. C. Morgan. *Programming from Specifications*. Prentice-Hall, second edition, 1994.

- [13] 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.
- [14] C.C. Morgan. Private communication. 1995.
- [15] R. Segala. Modeling and verification of randomized distributed real-time systems. PhD Thesis, 1995.
- [16] M. Sharir, A. Pnueli, and S. Hart. Verification of probabilistic programs. *SIAM Journal on Computing*, 13(2):292–314, May 1984.
- [17] N. Storey. *Safety-critical computer systems*. Addison-Wesley, 1996.
- [18] M. Vardi. Automatic verification of probabilistic concurrent finite-state systems. *Proceedings of 26th IEEE Symposium on Found. of Comp. Sci.*, pages 327–338, 1985.