Programming Research Group

A PROBABILISTIC TEMPORAL CALCULUS BASED ON EXPECTATIONS

Carroll Morgan and Annabelle McIver

PRG-TR-13-97

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

A probabilistic temporal calculus based on expectations

Carroll Morgan and Annabelle McIver*

April 24, 1997

Abstract

Generalising Boolean-valued predicates to *expectations* — functions from the state space into $[0,1]$ — allows the definition of probabilistic temporal operators that treat explicit probabilities as well as demonic nondeterminism and divergence.

The conventional operational interpretation of the temporal operators does not generalise so easily: although one may speak of "satisfying a predicate" in the standard case, it is not meaningful to "satisfy an expectation". That difficulty is avoided by giving the operational interpretation of the operators for the probabilistic case in terms of various kinds of gambling game.

Keywords: Temporal logic, probability, formal semantics, program correctness, weakest precondition.

1 Introduction

Temporal logic specialises modal logic to the case in which the "worlds" are states of a computation and the relation between worlds is determined by the state-to-state evolution of some computation; the temporal formulae express properties of (future) states which the computation might or might not reach [2].

Probability can be added to temporal logic in a variety of ways. At one extreme the computation is probabilistic but the temporal formulae do not refer to probabilities in any way; the notion of validity is altered from "true" to "holds with probablility 1" [9]. At the other extreme are logics which allow virtually complete access to explicit probabilities, introducing for example a real-valued term $\Pr\phi$ giving for any formula ϕ the probability of its holding [4, for dynamic logic]. Further variation is provided in both cases by the extent to which demonic non-determinism is allowed and, if it is, to what fairness constraints it must conform.

This report takes a middle course, generalising $\{0,1\}$-valued predicates to $[0,1]$-valued expectations (following Kozen [8]) and building a logic[1] based on formu-

*Both authors are members of the Programming Research Group at Oxford University: $\{$*carroll,anabel*$\}$*@comlab.ox.ac.uk*. McIver is supported by the EPSRC.

[1]At least we hope to do so: at present we have no "favourite" axioms, and are exploring the logic's suitability in calculations based on laws proved direct from the semantics.

lae over those: both demonic non-determinism and divergence (catastrophic non-termination, or **abort**) are included naturally. The use of $[0,1]$ means that numbers do occur explicitly in the calculations (in contrast to [9]); however the use of expectations rather than explicit probabilities may mean that the expressiveness is not as great as [4].

To recover explicit probabilities, when desired, we rely on this observation:

> If a random variable $\mathcal{A}$ is $\{0,1\}$-valued over some state space S, selecting as a characteristic function some sub-space S' of it, then the expected value of $\mathcal{A}$ over a probability distribution Pr on S is in fact the probability $\Pr(S')$ that Pr assigns to S' directly.

For example, the probability that a computation will establish some standard predicate ϕ is the expected value of the random variable *1 if* ϕ *else 0* over the distribution of final states realised by executing the program.

Following [14] we first base standard temporal logic on predicate transformers rather than explicit relations and then, by generalising those to *probabilistic* predicate transformers [13], we show how a probabilistic temporal logic arises naturally; the simplicity of the definitions in [14] is retained.

A satisfactory operational interpretation of the probabilistic logic cannot however be based on the notion of "achieving a predicate", as it is in the standard case, since that is not meaningful for formulae whose value lies strictly between 0 and 1. For the probabilistic case we regard the underlying computation as a gambling game: the player(s) seek to maximise or minimise the expectations using strategies that determine whether or not to "continue playing".[2]

2 Standard temporal logic

Standard temporal logic [2] extends classical predicate logic with modal operators based on distinguished "instants of time" called "states"; the instants are determined by an underlying computation that "steps" from one state to the next.

Conventional models for standard branching-time temporal logic extend models for standard predicate logic so that classical (non-temporal) formulae are interpreted within the states separately, and the "step" relation between states is used to interpret the temporal operators.

The conventional approach thus sees the underlying computation as a relation between initial and (possibly several, with nondeterminism) final states. In this section we review an alternative but equivalent model in which the computation is a predicate transformer rather than a relation [14]. Predicate transformers map sets of final states to sets of initial states, and we show how the standard temporal operators are interpreted in that case.

[2]This interpretation is related to Back and von Wright's use of games to motivate the interplay of angelic and demonic nondeterminism expressed by predicate transformers [1]; Stirling suggests a similar game-based interpretation of validity in the modal μ-calculus [18]. In both of those cases only standard predicates are used.

2.1 Syntax of formulae

For classical predicate logic, to which the temporal operators are added, we use the conventional syntax over variables $a, \cdots, z$, with constant, function and relation symbols taken directly from mathematics as needed.

The propositional connectives are false, true, $\neg$ (highest precedence), $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$ (lowest precedence); the quantifiers are $\forall, \exists$ with their scope always indicated by explicit parentheses, as in $(\forall x \bullet \mathcal{P})$ for formula $\mathcal{P}$.

The syntax for temporal operators is $\circ\mathcal{P}$ (next), $\diamond\mathcal{P}$ (eventually), $\Box\mathcal{P}$ (always) and $\mathcal{P} \rhd \mathcal{Q}$ (until); they have highest precedence of all and associate to the right.

2.2 Classical semantics

To interpret classical formulae we fix a universe $\mathbb{V}$ of values and a meaning over $\mathbb{V}$ for any constant, function and relation symbols used. The *state space*

$$S \quad := \quad Var \rightarrow \mathbb{V} \ ,$$

is then the set of functions from the set of variables *Var* to the values $\mathbb{V}$. (The symbol $:=$ means "is defined to be".)

Formulae denote *predicates*, which are functions from S to the Booleans, or equivalently are subsets of S; we then say that a classical formula is *true* (or *false*) in a state — similarly *holds* (or *does not hold*) in a state, or that a state *satisfies* (or *does not satisfy*) a formula — if the corresponding predicate gives the Boolean *true* (or *false*) when applied to that state.

2.3 Temporal operators informally

For the temporal operators we add to the interpretation above an *underlying computation*, represented by a *predicate transformer* over the state space — the one tranformer gives for each state the possible "single steps" that can take that state to each of its successor states. Usually we call the transformer *step*, and so we have

$$step\colon\ \mathbb{P}S \leftarrow \mathbb{P}S \ ,$$

in which we write the functional arrow backwards to emphasise that the transformer takes final predicates to initial ones. If *step* is deterministic then for each state there is just one successor; where it is nondeterministic there may be more than one.

When convenient, we assume *step* is the meaning of some (syntactic) program *Step* written in the language of guarded commands: thus we have $step = \mathrm{wp}.Step$, where "wp" is the weakest-precondition semantic function [3].

We assume that *step* satisfies the usual *healthiness conditions* for predicate transformers [3], in particular that it is "feasible" and "positively conjunctive":

$$\begin{array}{rcll} step.false & \equiv & false & \text{feasible} \\ step.(\mathcal{P} \wedge \mathcal{Q}) & \equiv & step.\mathcal{P} \wedge step.\mathcal{Q} & \text{positively conjunctive.} \end{array}$$

In this section we give an operational intuition for the temporal operators by referring to the execution of various program fragments.

2.3.1 *Next* informally

The formula $\circ\mathcal{P}$, pronounced "next $\mathcal{P}$", holds in the current state if one execution of the underlying computation *step* is guaranteed to establish $\mathcal{P}$.

Remembering that $\mathcal{P}$ is (interpreted as) a predicate over the state space, and that $\mathit{step} = \mathrm{wp}.\mathit{Step}$, we have

$$\circ\mathcal{P} \quad \equiv \quad \mathrm{wp}.\mathit{Step}.\mathcal{P}\ . \tag{1}$$

We note the following:

1. If nondeterminism is present in *Step* it is understood *demonically*, so that $\circ\mathcal{P}$ holds only if $\mathcal{P}$ holds in *all* of the next states that *Step* can reach from the current state.

2. Divergence — or "aborting" behaviour — in *Step* is also interpreted demonically: if *Step* can diverge in the current state, then $\circ\mathcal{P}$ does not hold for any $\mathcal{P}$. (Even $\circ\mathsf{true}$ does not hold when *Step* diverges.)

2.3.2 *Eventually* informally

The formula $\diamond\mathcal{P}$, pronounced "eventually $\mathcal{P}$", holds if repeated execution of *Step* (including possibly no executions at all) is guaranteed to establish $\mathcal{P}$.

Using programming idioms[3] we have

$$\diamond\mathcal{P} \quad \equiv \quad \mathrm{wp}.(\mathbf{do}\ \neg\mathcal{P} \ \rightarrow\ \mathit{Step}\ \mathbf{od}).\mathsf{true}\ , \tag{2}$$

so that $\diamond\mathcal{P}$ holds if execution of the above loop is guaranteed to terminate (in a state satisfying $\mathcal{P}$).

Thus if Program (2) can loop forever then $\diamond\mathcal{P}$ is false; similarly $\diamond\mathcal{P}$ is false if *Step* can diverge before $\mathcal{P}$ is established.

2.3.3 *Always* informally

The formula $\Box\mathcal{P}$, pronounced "always $\mathcal{P}$", holds if repeated execution of *Step* (including no executions at all) is guaranteed to preserve the truth of $\mathcal{P}$.

We have

$$\Box\mathcal{P} \quad \equiv \quad \mathrm{wnp}.(\mathbf{do}\ \mathcal{P} \ \rightarrow\ \mathit{Step}\ \mathbf{od}).\mathsf{false}\ , \tag{3}$$

where wnp (weakest *nonterminating* precondition) differs from wp only in interpreting looping as success rather than failure,[4] so that for example[5]

$$\mathrm{wnp}.(\mathbf{do}\ \mathsf{true}\ \rightarrow\ \mathbf{skip}\ \mathbf{od}).\mathsf{false} \quad \equiv \quad \mathsf{true}\ .$$

[3] This is why the syntactic form *Step* is convenient — though of course it is not necessary, since the semantics of the loop can be given in terms of *step* directly.

[4] We see in Sec. 2.4 that wnp simply takes the greatest- rather than least fixed point for loops, acting like wp otherwise; it differs from the weakest *liberal* precondition (wlp) in that wnp applies wp (rather than wnp again "hereditarily") to the loop body. Thus divergence in the body is treated as failure.

[5] We note in passing that $\Box$, itself a predicate transformer, is therefore not feasible even though *step* is.

Thus $\Box\mathcal{P}$ holds if execution of the above loop, from the current program state, is guaranteed *not* to terminate.

Nondeterminism and divergence continue to be interpreted demonically, so that $\Box\mathcal{P}$ is false if any of the possible executions diverges, or reaches a state not satisfying $\mathcal{P}$.

2.3.4 *Unless* informally

The formula $\mathcal{P} \rhd \mathcal{Q}$, pronounced "$\mathcal{P}$ unless $\mathcal{Q}$", holds if repeated execution of *Step* is guaranteed to preserve $\mathcal{P}$ unless $\mathcal{Q}$ is established. We have

$$\mathcal{P} \rhd \mathcal{Q} \quad \equiv \quad \text{wnp}.(\mathbf{do}\ (\mathcal{P} \wedge \neg\mathcal{Q}) \ \rightarrow\ \mathit{Step}\ \mathbf{od}).\mathcal{Q}\ , \tag{4}$$

so that $\mathcal{P} \rhd \mathcal{Q}$ holds if execution of the above loop is guaranteed either to loop forever or to terminate establishing $\mathcal{Q}$.

Note that if termination can occur when $\mathcal{Q}$ does not hold then $\mathcal{P} \rhd \mathcal{Q}$ is false. Comparing the programs (3) and (4) shows that (as usual) we have the equivalence

$$\Box\mathcal{P} \quad \equiv \quad \mathcal{P} \rhd \mathsf{false}\ ,$$

so that *always* is a special case of *unless*.

2.4 Temporal semantics

Using the informal program-based descriptions of the previous section, we now give the precise meaning of the temporal operators. Once the underlying computation is given they, like classical formulae, denote predicates over the state space.

In each case the conventional wp-semantics gives the definition directly (perhaps after simplification) from Programs (1)–(4) above, except that as noted we use greatest- (ν) rather than least- (μ) fixed points for loops under wnp.

We fix the underlying computation *step*, and to avoid proliferation of "semantic brackets" we use $\mathcal{A}$, $\mathcal{B}$ on the left to stand for formulae while on the right they stand for the corresponding meanings (predicates).[6] The Boolean operators are understood to act pointwise over predicates, and the ordering with respect to which the least- and greatest fixed points are taken is the pointwise extension of *false* $\sqsubseteq$ *true*.

Definition 2.1 *next*: $\circ\mathcal{A} := \ \mathit{step}.\mathcal{A}\ .$ ■

Definition 2.2 *eventually*: $\Diamond\mathcal{A} := \ (\mu\mathcal{X} \bullet \mathcal{A} \vee \circ\mathcal{X})\ .$ ■

Definition 2.3 *always*: $\Box\mathcal{A} := \ (\nu\mathcal{X} \bullet \mathcal{A} \wedge \circ\mathcal{X})\ .$ ■

Definition 2.4 *unless*: $\mathcal{A} \rhd \mathcal{B} := \ (\nu\mathcal{X} \bullet \mathcal{B}\ \vee\ \mathcal{A} \wedge \circ\mathcal{X})\ .$ ■

The definitions of this section are based on those of Morris [14], and are related to similar definitions later given by Lukkien and van de Snepscheut [10, 7]. General properties of the temporal operators will be examined when we treat their probabilistic versions.

[6] We reserve metavariables $\mathcal{P}$, $\mathcal{Q}$ for classical formulae.

3 Probabilistic temporal logic

Our principal contribution is to generalise the previous section by using probabilistic [13] rather than standard predicate transformers, and to explore the algebra and interpretation of the resulting probabilistic temporal calculus. We generalise in three stages:

1. (Secs. 3.1, 3.2) Whereas standard formulae are expressions of type Boolean over the state space, probabilistic formulae are expressions of type real. We say that probabilistic formulae denote *expectations* (rather than predicates).

   In this presentation we restrict expectations to lie non-strictly between 0 and 1.[7] Predicates embed into expectations by taking *false* to (the constant function) 0 and *true* to 1.

2. (Sec. 3.3) The underlying computation is extended to include probabilistic behaviour. Syntactically we add a new operator ${}_p\oplus$ that selects its left (or right) operand with probability p (or $1-p$); semantically the program becomes an expectation transformer, generalising predicate transformers.

3. (Sec. 3.4) The definitions Defs. 2.1–2.4 of the temporal operators are adjusted slightly, so that they operate over expectations rather than predicates.

The above are dealt with in three successive sections further below; first, however, we review the role of expectations as a basis for probabilistic programming logic in general.

3.1 Expectations of probabilistic programs

In probability theory a *random variable* is some real-valued function of the *sample space*; *selections* may be made from the sample space according to a given *probability distribution*. The *expectation* of the random variable is then its "average" value over many selections.

For us the state space is the sample space; executing a probabilistic program determines a distribution over the (final) state space; and the postcondition is a random variable whose expectation is taken over that distribution.

To regard a (standard) postcondition as a random variable we let it take the value 1 when satisfied by a state and 0 when it is not — then the probability of the program's making the postcondition true (regarding it as a predicate) is the same as the postcondition's expected value (regarding it as a random variable over the distribution of final states).

Consider for example a coin-flipping game represented by the program

$$c{:=}\ H\ {}_{\frac{1}{2}}\oplus\ c{:=}\ T\ ,$$

and a "payoff" postcondition represented by the expectation

$$1\ \underline{\text{if}}\ c{=}H\ \underline{\text{else}}\ 0$$

[7] We use the $[0,1]$ range here for compatibility with [12], given our interest in loops. Other possibilities are $[0,\infty)$ and bounded above as in [13], or even $[0,\infty]$.

so that the player wins £1 (say) if he flips heads, and nothing if he flips tails. The expected win *per flip* is then 50p, exactly the probability (in £'s) of establishing the standard postcondition $c = H$.

In fact expectations are more general than explicit probabilities, and in a necessary way — probabilities alone do not distinguish the two programs

$$x{:=}\ A \ \sqcap\ (x{:=}\ B \ {}_{\frac{1}{2}}\!\oplus x{:=}\ C) \tag{5}$$

and

$$(x{:=}\ A \sqcap x{:=}\ B)\ {}_{\frac{1}{2}}\!\oplus\ (x{:=}\ A \sqcap x{:=}\ C)\ . \tag{6}$$

Since nondeterministic choices as above are always understood demonically — they take the alternative whose chance of establishing the postcondition is the lesser — the two programs (5) and (6) agree on their probability of establishing each of the eight possible standard postconditions locating x in the various subsets of $\{A, B, C\}$. On the basis of questions "what is the probability of establishing the standard $\mathcal{P}$?" those programs are indistinguishable, no matter which $\mathcal{P}$ is chosen.

For example, the probability of establishing postcondition $x \neq C$ is

$$\begin{array}{rcll} 1\ \sqcap\ (\frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 0) & = & \frac{1}{2} & \text{for Program (5)} \\ \text{and}\quad \frac{1}{2}(1 \sqcap 1) + \frac{1}{2}(1 \sqcap 0) & = & \frac{1}{2} & \text{for Program (6)}\ . \end{array}$$

Yet the programs *should* be distinguished, since Program (5) treats B and C equally (if the result is not A, then it is equally likely to be B or C), but Program (6) does not. (See [16] for further discussion of the example.)

If we use a probabilistic postcondition — an expectation — such as

$$\begin{array}{lll} \frac{1}{2} & \text{if} & x = A \\ 1 & \text{if} & x = B \\ 0 & \text{if} & x = C\ , \end{array} \tag{7}$$

then the programs are distinguished. For Program (5) we have pre-expectation

$$\frac{1}{2}\ \sqcap\ (\frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 0) \quad = \quad \frac{1}{2}\ ,$$

but for Program (6) we have

$$\frac{1}{2}(\frac{1}{2} \sqcap 1) + \frac{1}{2}(\frac{1}{2} \sqcap 0) \quad = \quad \frac{1}{4}\ .$$

In [13] it is shown that it is the interaction of demonic and probabilistic choice that requires proper expectations, rather than predicates and probabilities, and that expectations are exactly "general enough".

3.2 Syntax and semantics of expectations

Probabilistic formulae are written as expressions of type $\mathbb{R}$ over the variables of the program, and are restricted to the range $[0, 1]$. Standard (Boolean-valued) formulae

$$
\begin{array}{rcl}
\text{wp}.\textbf{abort}.\mathcal{P} & := & 0 \\
\text{wp}.\textbf{skip}.\mathcal{P} & := & \mathcal{P} \\
\text{wp}.(x{:=}\ E).\mathcal{P} & := & \mathcal{P}^x_E \\
\text{wp}.(\mathit{Prog}; \mathit{Prog}').\mathcal{P} & := & \text{wp}.\mathit{Prog}.(\text{wp}.\mathit{Prog}'.\mathcal{P}) \\
\text{wp}.(\textbf{if}\ B\ \textbf{then}\ \mathit{Prog}\ \textbf{else}\ \mathit{Prog}'\ \textbf{fi}).\mathcal{P} & := & \text{wp}.\mathit{Prog}.\mathcal{P}\ \underline{\text{if}}\ B\ \underline{\text{else}}\ \text{wp}.\mathit{Prog}'.\mathcal{P} \\
\text{wp}.(\mathit{Prog} \sqcap \mathit{Prog}').\mathcal{P} & := & \text{wp}.\mathit{Prog}.\mathcal{P} \wedge \text{wp}.\mathit{Prog}'.\mathcal{P} \\
\text{wp}.(\mathit{Prog}\ {}_p\!\oplus \mathit{Prog}').\mathcal{P} & := & p \cdot \text{wp}.\mathit{Prog}.\mathcal{P} + (1-p) \cdot \text{wp}.\mathit{Prog}'.\mathcal{P} \\
(\mu X \bullet \mathcal{C}) & := & \text{least fixed point of } \mathit{cntx}\text{, defined} \\
 & & \text{wp}.\mathcal{C} = \mathit{cntx}.(\text{wp}.X) \text{ for any } X.
\end{array}
$$

Figure 1: Probabilistic wp-semantics as expectation transformers [13]

are embedded using the brackets $[\cdot]$, whose effect semantically is to convert *false* to 0 and *true* to 1. Thus for example expectation (7) could be written

$$[x = A]/2 + [x = B]\ .$$

Where convenient we use the operators $\sqcup$ (maximum) and $\sqcap$ (minimum) instead of $\wedge$ and $\vee$, since for example

$$[\mathcal{P} \wedge \mathcal{Q}] \quad \equiv \quad [\mathcal{P}] \sqcap [\mathcal{Q}]\ ;$$

and for quantifiers we can use

$$
\begin{array}{ll}
(\sqcup x \bullet \mathcal{A}) & \text{distributed maximum} \\
(\sqcap x \bullet \mathcal{A}) & \text{distributed minimum}
\end{array}
$$

to generalise $\exists$ and $\forall$ over formulae in which x may be free.

Finally, by analogy with $\equiv$ (equal in all states) for comparing standard formulae, we generalise semantic entailment by defining

$$
\begin{array}{lcl}
\mathcal{A} \Rrightarrow \mathcal{B} & := & \text{in all states } \mathcal{A} \text{ is no more than } \mathcal{B}\text{, and} \\
\mathcal{A} \Lleftarrow \mathcal{B} & := & \text{in all states } \mathcal{A} \text{ is no less than } \mathcal{B}.
\end{array}
$$

Thus $\equiv$, $\Rrightarrow$ and $\Lleftarrow$ are relations between probabilistic formulae, while $=$, $\leq$ and $\geq$ retain their usual (Boolean-valued) meanings *within* formulae.

This table illustrates our terminology for formulae:

	classical	temporal
standard	$x = 1$	$\Diamond(x = 1)$
probabilistic	$[x = 1]/2$	$\Diamond[x = 1]/2$

3.3 Expectation transformers

The probabilistic semantics of a simple imperative language is given in Fig. 1, including explicit operators for nondeterministic ($\sqcap$) and probabilistic (${}_p\!\oplus$) choice: such programs denote *expectation transformers*.

Standard semantics — using predicate transformers — is preserved by the extension in the sense that, for standard program *Prog* and standard formula $\mathcal{P}$, we have

$$[\mathrm{wp}_s.Prog.\mathcal{P}] \quad \equiv \quad \mathrm{wp}_p.Prog.[\mathcal{P}] \ ,$$

where (here only) we write wp_s, wp_p explicitly for the standard, probabilistic weakest precondition functions respectively.

The healthiness conditions for probabilistic programs — extending the feasibility and positive conjunctivity conditions for standard programs — are given in [13], and will be introduced as required below. We note here for well-definedness that $\mathrm{wp}.Prog.\mathcal{A}$ is an expectation (lies in $[0,1]$) if $\mathcal{A}$ is.

Lemma 3.1 *(probabilistic) feasibility* For any probabilistic program *Prog* and expectation $\mathcal{A}$ we have

$$0 \quad \Rrightarrow \quad \mathrm{wp}.Prog.\mathcal{A} \quad \Rrightarrow \quad \sqcup \mathcal{A} \ .$$

Proof: Use either structural induction over the semantics in Fig. 1 or direct reference to the healthiness conditions for expectation transformers [13]. ■

We conclude this section with three examples of probabilistic reasoning, based on simple coin-flipping programs. We write $c{:=}\ (H\ {}_p{\oplus}\ T)$ to abbreviate

$$c{:=}\ H\ \ {}_p{\oplus}\ \ c{:=}\ T\ ,$$

and write $c{:=}\ (H \sqcap T)$ similarly; we use $\overline{p}$ for $1{-}p$.

3.3.1 Example: two successive probabilistic choices

Two coins c, d are flipped in succession, the first giving heads H with probability p (giving tails T with probability $\overline{p}$) and the second giving heads with probability q. The probability that after both flips the coins will show the same face is the probability that the program

$$c{:=}\ (H\ {}_p{\oplus}\ T);\ d{:=}\ (H\ {}_q{\oplus}\ T)$$

will establish the standard postcondition $c = d$; using expectations, we calculate from Fig. 1

$$\begin{array}{lll}
 & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T));\ d{:=}\ (H\ {}_q{\oplus}\ T)).[c=d] & \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)). & \text{sequential composition} \\
 & \quad (\mathrm{wp}.(d{:=}\ (H\ {}_q{\oplus}\ T)).[c=d]) & \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)). & \text{probabilistic choice} \\
 & \quad (q\cdot \mathrm{wp}.(d{:=}\ H).[c=d] + \overline{q}\cdot \mathrm{wp}.(d{:=}\ T).[c=d]) & \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)).(q[c=d]^d_H + \overline{q}[c=d]^d_T) & \text{assignments} \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)).(q[c=H] + \overline{q}[c=T]) & \text{substitutions} \\
\equiv & p(q[H=H] + \overline{q}[H=T]) + \overline{p}(q[T=H] + \overline{q}[T=T]) & \text{as above} \\
\equiv & p(q\cdot 1 + \overline{q}\cdot 0) + \overline{p}(q\cdot 0 + \overline{q}\cdot 1) & [\mathit{false}] \equiv 0;\ [\mathit{true}] \equiv 1 \\
\equiv & pq + \overline{p}\,\overline{q}\ . &
\end{array}$$

Since the "post-expectation" $[c = d]$ is standard[8], we recognise the "pre-expectation" $pq + \overline{p}\,\overline{q}$ as indeed the probability of the program's establishing the standard post-condition $c = d$. Note however that the intermediate expectation (between the two statements)

$$q[c = H] + \overline{q}[c = T]$$

is *not* standard.

3.3.2 Example: nondeterministic then probabilistic choice

In this case the first coin is "placed" by a demon striving to make the probability of establishing $c = d$ as low as possible; the second coin is flipped, as before. We have therefore the program

$$c{:=}\ (H \sqcap T);\ d{:=}\ (H\ {}_q{\oplus}\ T)\ ,$$

and calculate

$$\begin{array}{lll}
 & \mathrm{wp}.(c{:=}\ (H \sqcap T);\ d{:=}\ (H\ {}_q{\oplus}\ T)).[c = d] & \\
\equiv & \mathrm{wp}.(c{:=}\ (H \sqcap T)).(q[c = H] + \overline{q}[c = T]) & \text{as above} \\
\equiv & \quad (q[H = H] + \overline{q}[H = T]) & \text{nondeterministic choice} \\
 & \sqcap \quad (q[T = H] + \overline{q}[T = T]) & \\
\equiv & q \sqcap \overline{q}\ . &
\end{array}$$

Here the demon — given first choice — selects for c the face that is *least* likely to result from the subsequent flip of d, thus minimising the probability that $c = d$ will be established by the second flip.

3.3.3 Example: probabilistic then nondeterministic choice

In this case the first coin is flipped and the second is placed. The program is

$$c{:=}\ (H\ {}_p{\oplus}\ T);\ d{:=}\ (H \sqcap T)\ ,$$

and we calculate

$$\begin{array}{lll}
 & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T);\ d{:=}\ (H \sqcap T)).[c = d] & \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)). & \text{nondeterministic choice; assignment} \\
 & \quad ([c = H] \sqcap [c = T]) & \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)).[c = H \wedge c = T] & \sqcap\text{-}\wedge \text{ correspondence} \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)).[\mathit{false}] & \\
\equiv & \mathrm{wp}.(c{:=}\ (H\ {}_p{\oplus}\ T)).0 & \\
\equiv & 0\ . &
\end{array}$$

In this case the demon — given second choice — is able on every occasion to make d differ from c, no matter how the flip of c resolved. Thus the least *guaranteed* chance of the program's establishing $c = d$ is just 0 — there is no guarantee at all, since we must always assume the demon might act against us.

[8] It is the embedding of a standard predicate.

3.4 Probabilistic temporal operators

The final step for defining probabilistic versions of the temporal operators is to adjust Defs. 2.1–2.4 from Sec. 2.4 so that they act over expectations rather than predicates. We replace $\wedge,\vee$ by $\sqcap,\sqcup$ respectively and take fixed points of real- rather than Boolean-valued functions:

Definition 3.2 probabilistic *next*: $\circ\mathcal{A} := \ \mathit{step}.\mathcal{A}$. ■

Definition 3.3 probabilistic *eventually*: $\Diamond\mathcal{A} := \ (\mu\mathcal{X} \bullet \mathcal{A} \sqcup \circ\mathcal{X})$. ■

Definition 3.4 probabilistic *always*: $\Box\mathcal{A} := \ (\nu\mathcal{X} \bullet \mathcal{A} \sqcap \circ\mathcal{X})$. ■

Definition 3.5 probabilistic *unless*: $\mathcal{A} \rhd \mathcal{B} := \ (\nu\mathcal{X} \bullet \mathcal{B} \ \sqcup \ \mathcal{A} \sqcap \circ\mathcal{X})$. ■

To establish the credibility of the definitions, in subsequent sections we give both an operational interpretation of the probabilistic temporal operators and an investigation of the laws they satisfy.

As a start, however, we recall here the programs from Sec. 2.3, allowing the computation *Step* now to be probabilistic but stipulating for simplicity that the formula $\mathcal{P}$ remain standard. Then in particular Program (2) becomes

$$\mathrm{wp}.(\textbf{do}\ \mathcal{P}{=}0 \ \rightarrow \ \mathit{Step}\ \textbf{od}).\mathcal{P}\ , \tag{8}$$

with the equality $\mathcal{P}{=}0$ "converting" the expectation $\mathcal{P}$ to a Boolean, as required syntactically for the guard, and the explicit postcondition $\mathcal{P}$ replacing true, simply to emphasise that the "aim" of the loop is to establish $\mathcal{P}$.[9]

When $\mathcal{P}$ is standard, the expression (8) gives $\Diamond\mathcal{P}$ in agreement with Definition 3.3 — and so we can as usual interpret (8) as a probability: in a given state the value of $\Diamond\mathcal{P}$ is indeed the probability that, starting from that state, repeated execution of *Step* will establish $\mathcal{P}$ eventually.

Similar reasoning applies to $\circ P$, $\Box\mathcal{P}$ and $\mathcal{P} \rhd \mathcal{Q}$, based on Programs (1), (3) and (4) when as above $\mathcal{P}$ and $\mathcal{Q}$ are standard (and classical), showing that standard temporal formulae over probabilistic computations may be interpreted as probabilities directly, as one would hope. That interpretation is sufficient for programs containing no demonic nondeterminism.

As noted at (5) and (6) however, standard formulae have insufficient resolution when probabilistic and demonic nondeterminism interact: one must use expectations. And the above analogies fail — for all four programs — in the more general case when the formulae are not standard, because it does not make sense to speak of "establishing" a proper (not standard) expectation.

To interpret probabilistic temporal formulae, therefore, we rely on the shift of viewpoint described in the next section.

[9]By analogy with predicates, we say that a standard expectation is *established* in a state where its value is 1.

4 Temporal operators as games

If one considers a program's establishing a postcondition to be "winning a game", then our generalisation to expectations simply adds the extra information of how much is won: the standard case embeds as "win 1 if the postcondition is established; win 0 if it is not".

For an operational interpretation of probabilistic temporal formulae we consider *poker machines*[10]; they have a window in which various symbols are visible; pulling a handle changes the symbols probabilistically; and each configuration of visible symbols is associated with a certain reward (possibly 0). In computational terms, each configuration of symbols shown in the window is a state; the underlying computation is the effect of one pull of the handle; and the post-expectation is the "pay function", giving the worth to the player of each symbol configuration should he press *pay* at that point.

Each of the temporal operators corresponds to a different way of gambling with the machine, and we consider them in turn.

4.1 The *next* game

The simplest game is *next* where, direct from Definition 3.2, we can see that the game consists of pulling the handle exactly once and then pressing *pay*: the expectation $\circ\mathcal{A}$ evaluated over the initial configuration (showing *before* the handle is pulled) gives the win determined jointly by the expectation $\mathcal{A}$ and the probabilistic distribution of final configurations (that could show *after* the handle is pulled).

Suppose for example the configurations are just natural numbers $\mathbb{N}$, the effect of a handle pull is the computation $n{:=}\ \ (n+1 \;{}_p\!\oplus\, 0)$, and the pay function is $n/(n+1)$. Then the expected win from a single handle pull is

$$\begin{array}{lll} & \circ(n/(n+1)) & \\ \equiv & \mathrm{wp}.(n{:=}\ \ (n+1 \;{}_p\!\oplus\, 0)).(n/(n+1)) & \text{Definition 3.2} \\ \equiv & p\cdot(n+1)/(n+2) \ + \ \overline{p}\cdot 0 & \text{probabilistic choice; assignment} \\ \equiv & p(n+1)/(n+2)\ , & \end{array}$$

from which it is evident that the expected win does depend on the initial configuration: a gambler starting at 0 every time would expect an average win per game of $p/2$ in the long run; but the larger the initial configuration, the larger the expected win from a single handle pull would be (approaching p in the limit).

4.2 The *eventually* game

For *eventually* the player pulls the handle repeatedly, and decides on the basis of the current configuration whether to stop and press *pay*, or to pull again. Thus for $\Diamond\mathcal{A}$ we consider an expectation

$$\mathrm{wp}.(\mathbf{do}\ G \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A}\ , \tag{9}$$

[10] *slot* machines (US); *fruit* machines (UK).

in which the guard G — a predicate — represents the gambler's *strategy*: if it is true (in a configuration) then he plays again; if it is false, he presses *pay* and stops playing.[11]

For the gambler some strategies G are clearly worse than others. The strategy $G := \mathsf{true}$ continues the game forever, which we interpret as losing (as winning 0 no matter what $\mathcal{A}$ is) since *pay* is never pressed; the strategy false represents pressing *pay* without pulling the handle at all, in which case the amount won is determined by the current configuration.[12]

For standard pay function $\mathcal{P}$, which pays 1 for some configurations and 0 for all others, an optimal strategy is given by the predicate $\mathcal{P}{=}0$ that continues play as long as the current configuration pays nothing — since things can only get better. Whenever $\mathcal{P}{=}1$, however, the gambler should stop immediately — he can't win more than 1, and could well win less if he continued.

Indeed (recall Program (8)) the expected win above is $\Diamond\mathcal{P}$ and remarkably, as the following lemma shows, even for proper expectations $\mathcal{A}$ the expectation $\Diamond\mathcal{A}$, applied to the initial state, gives the best that can be obtained for *any* strategy.

Lemma 4.1 For underlying computation *Step*, expectation $\mathcal{A}$ and strategy G we have

$$\mathrm{wp}.(\mathbf{do}\ G \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A} \quad \Rrightarrow \quad \Diamond\mathcal{A}\ ,$$

showing that $\Diamond\mathcal{A}$ is an upper bound over all strategies G for the eventually game.
Proof: Compare Definition 3.3 with the usual least-fixed-point definition of $\mathbf{do} \cdots \mathbf{od}$, and use monotonicity of μ. ■

More significant is the complementary result, that for any computation *Step* and expectation $\mathcal{A}$ there is a strategy G whose expected win approaches $\Diamond\mathcal{A}$ arbitrarily closely — in fact if the state space is finite, there is a G that realises $\Diamond\mathcal{A}$ exactly.

We sketch a proof of the above, relying on informal operational arguments rather that the wp-definitions, for the case in which *Step* is deterministic and non-divergent; we assume at first that the state space is finite.[13]

The optimal strategy will be the predicate $\mathcal{A} < \Diamond\mathcal{A}$, and our first lemmas establish that it guarantees termination of the game.

Lemma 4.2 For any (probabilistic) loop with deterministic and non-divergent body, the (standard) predicate "has probability 0 of termination" is invariant.
Proof: Let $T.s$ give the probability of termination from state s; and consider a single execution of the loop body that takes some initial state s to a final state s'. We show the contrapositive, that $T.s' > 0$ implies $T.s > 0$ also.

[11] Note we consider only strategies based on the configuration: the predicate G cannot express strategies like "play 5 times then stop".

[12] On a real poker machine the handle must be pulled at least once after inserting your money: the normal game would thus be $\circ\Diamond\mathcal{A}$.

[13] The full proof, using wp for rigour and allowing both nondeterminism and divergence, has the same structure as the following but depends substantially on technical results from [13, 12].
None of the technical results elsewhere in the paper depend on this section.

Since the body is deterministic and non-divergent, the transition from s to s' has some non-zero probability p — otherwise it could never be taken.[14] That gives

$$T.s \;\geq\; p \cdot T.s' \;>\; 0$$

as required. ∎

Lemma 4.3 If a non-empty (standard) predicate I is preserved by the computation *Step*, and the state space is finite, then $\mathcal{A}.s = (\Diamond\mathcal{A}).s$ holds for some state s satisfying I.
Proof: Suppose first that I is true. From Definition 3.3 we have $\mathcal{A} \Rrightarrow \Diamond\mathcal{A}$ for any $\mathcal{A}$ and, with Lemma 3.1 as well, we have also that $\Diamond\mathcal{A} \Rrightarrow \sqcup\mathcal{A}$. Thus because the state space is finite we can choose state s with $\mathcal{A}.s = \sqcup\mathcal{A}$, and have then

$$\sqcup\mathcal{A} \;=\; \mathcal{A}.s \;\leq\; (\Diamond\mathcal{A}).s \;\leq\; \sqcup\mathcal{A}\;,$$

so that $\mathcal{A}.s = (\Diamond\mathcal{A}).s$ — and any s satisfies I when I is true.

Now if I is not true, but still is non-empty and invariant, we simply restrict the whole system to I; the above argument repeated "within I", in that sense, gives the s required as before. ∎

Lemma 4.4 For deterministic and non-divergent *Step*, the loop

$$\mathbf{do}\ \mathcal{A}{<}\Diamond\mathcal{A} \rightarrow \mathit{Step}\ \mathbf{od} \tag{10}$$

terminates with probability 1 from every initial state.
Proof: Let Z be the predicate denoting the states at which the probability of termination is 0. From Lemma 4.2 we have invariance of Z; from Lemma 4.3 we have that $\mathcal{A} = \Diamond\mathcal{A}$ is attained at some s in Z if Z is non-empty. But that would be a contradiction, since when $\mathcal{A}.s = \Diamond\mathcal{A}.s$ termination from s is immediate, and so s could not lie in Z.

Hence Z is empty: the probability of (10)'s termination is everywhere non-zero.

Since the state space is finite, we conclude from the *0-1 law*[15] [6, 12] that the probability of termination is in fact everywhere 1. ∎

We have shown that the strategy $\mathcal{A} < \Diamond\mathcal{A}$, the very reasonable "keep going as long as taking $\mathcal{A}$ now is strictly worse than waiting for $\mathcal{A}$ later", is terminating at least. We finish off as follows.

Lemma 4.5 For deterministic and non-divergent *Step*, and finite state space, we have

$$\Diamond\mathcal{A} \quad\Rrightarrow\quad \mathrm{wp}.(\mathbf{do}\ \mathcal{A}{<}\Diamond\mathcal{A} \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A} \tag{11}$$

Proof: We use the probabilistic analogue of "loop invariants": for general loop

$$\mathbf{do}\ G \rightarrow \mathit{Body}\ \mathbf{od}\;,$$

[14] This is where determinism is used. If nondeterminism were present, a transition could be taken even though its minimum *guaranteed* probability was zero.

[15] The *0-1 law* states that if the probability of a process' eventual escape from a system is bounded away from 0, then in fact that probability is 1.

if expectation $\mathcal{B}$ is *invariant* — if $\mathcal{B} \sqcap [G] \Rrightarrow \mathrm{wp}.\mathit{Body}.\mathcal{B}$ holds — and the loop terminates with probability 1, then

$$\mathcal{B} \quad \Rrightarrow \quad \mathrm{wp}.(\mathbf{do}\ G \rightarrow \mathit{Body}\ \mathbf{od}).(\mathcal{B} \sqcap [\neg G]) \ . \tag{12}$$

(See [17], [12, Theorem A.3].)

To use that result, first we show that $\Diamond\mathcal{A}$ is invariant:

$$\begin{array}{lll}
 & \Diamond\mathcal{A} \sqcap [\mathcal{A} < \Diamond\mathcal{A}] & \\
\Rrightarrow & \Diamond\mathcal{A} \sqcap [\Diamond\mathcal{A} = \mathrm{wp}.\mathit{Step}.\Diamond\mathcal{A}] & \text{Definition 3.3} \\
\Rrightarrow & \mathrm{wp}.\mathit{Step}.\Diamond\mathcal{A} \ . &
\end{array}$$

But we know from Lemma 4.4 that the loop terminates, so we use (12) to conclude with

$$\begin{array}{lll}
 & \Diamond\mathcal{A} & \\
\Rrightarrow & \mathrm{wp}.(\mathbf{do}\ \mathcal{A} < \Diamond\mathcal{A} \rightarrow \mathit{Step}\ \mathbf{od}). & \text{termination; loop rule above} \\
 & \quad (\Diamond\mathcal{A} \sqcap [\mathcal{A} \geq \Diamond\mathcal{A}]) & \\
\Rrightarrow & \mathrm{wp}.(\mathbf{do}\ \mathcal{A} {<} \Diamond\mathcal{A} \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A} & [\mathcal{A} \geq \Diamond\mathcal{A}] \sqcap \Diamond\mathcal{A} \Rrightarrow \mathcal{A}
\end{array}$$

as required. ■

Putting Lemmas 4.1 and 4.5 together gives

Lemma 4.6 For deterministic and non-divergent *Step*, and finite state space, we have

$$\Diamond\mathcal{A} \quad \equiv \quad \mathrm{wp}.(\mathbf{do}\ \mathcal{A} {<} \Diamond\mathcal{A} \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A} \ ,$$

so showing that $\mathcal{A} < \Diamond\mathcal{A}$ is the optimal strategy. ■

Before moving to infinite state spaces, we give an example showing that finiteness is indeed necessary for Lemma 4.5. Let *Step* be $n{:=}\ n+1$ over the infinite state space $\mathbb{N}$, and define $\mathcal{A} {:=}\ n/(n+1)$. From Definition 3.3 we have $\Diamond\mathcal{A} \equiv (\sqcup N \bullet (\Diamond\mathcal{A})_N)$, where

$$\begin{array}{rcl}
(\Diamond\mathcal{A})_0 & \equiv & 0 \\
(\Diamond\mathcal{A})_1 & \equiv & n/(n+1) \sqcup \mathrm{wp}.(n{:=}\ n+1).0 \\
 & \equiv & n/(n+1) \\
(\Diamond\mathcal{A})_2 & \equiv & n/(n+1) \sqcup \mathrm{wp}.(n{:=}\ n+1).(n/(n+1)) \\
 & \equiv & (n+1)/(n+2) \\
 & \vdots & \\
(\Diamond\mathcal{A})_N & \equiv & n/(n+1) \sqcup \mathrm{wp}.(n{:=}\ n+1).(\Diamond\mathcal{A})_{N-1} \\
 & \equiv & (n+N-1)/(n+N) \ ,
\end{array}$$

and hence — taking the limit — we see that $\Diamond\mathcal{A} \equiv 1$. But then $\mathcal{A} < \Diamond\mathcal{A}$ is identically true, and with that strategy the expectation

$$\mathrm{wp}.(\mathbf{do}\ \mathsf{true} \rightarrow n{:=}\ n+1\ \mathbf{od}).(n/(n+1))$$

is 0 everywhere, due to non-termination.

Thus for infinite state spaces we must reconsider Lemmas 4.3 and 4.4, where the finiteness assumption is used. Let some predicate F over the state space be true for only finitely many states, and define

$$\begin{array}{rcl} \mathit{Step}_F & := & \textbf{if } F \textbf{ then } \mathit{Step} \textbf{ fi} \\ \Diamond_F \mathcal{A} & := & \text{“}\Diamond\mathcal{A} \text{ interpreted over computation } \mathit{Step}_F\text{”} \end{array}$$

Then it is immediate from Definition 3.3 that $(\Diamond_F\mathcal{A}).s = \mathcal{A}.s$ for any state s not satisfying F, and that suffices to recover both lemmas: for Lemma 4.3 note that if $I \not\subseteq F$ then $(\Diamond_F\mathcal{A}).s = \mathcal{A}.s$ is attained for $s \in I{-}F$, and that otherwise I is finite; for Lemma 4.4 we have that the probability of termination is bounded away from 0, since it is 1 except in finitely many states and non-zero for the rest.

Thus from Lemma 4.6 we have

$$\begin{array}{ll} & \Diamond_F\mathcal{A} \\ \equiv & \text{wp.}(\textbf{do } \mathcal{A}{<}\Diamond_F\mathcal{A} \rightarrow \mathit{Step}_F \textbf{ od}).\mathcal{A} \\ \equiv & \text{wp.}(\textbf{do } \mathcal{A}{<}\Diamond_F\mathcal{A} \rightarrow \mathit{Step} \textbf{ od}).\mathcal{A}\ , \end{array} \tag{13}$$

where we can replace Step_F in the body by *Step* because the body is executed only within F, where they do not differ. That gives our principal theorem for *eventually*:

Theorem 4.7 For deterministic, non-divergent and continuous computation *Step*, and expectation $\mathcal{A}$, the expectation $\Diamond\mathcal{A}$ is the supremum over all standard strategies G of

$$\text{wp.}(\textbf{do } G \rightarrow \mathit{Step} \textbf{ od}).\mathcal{A}\ . \tag{14}$$

Proof: From (13) we have an explicit set of strategies $G_F{:=}\ \mathcal{A}{<}\Diamond_F\mathcal{A}$ whose limit in (14) attains $(\sqcup F \bullet \Diamond_F\mathcal{A})$; from Definition 3.3 and the assumed continuity of *Step*, however, it is routine to show that

$$(\sqcup F \bullet \Diamond_F\mathcal{A}) \quad\equiv\quad \Diamond A\ .$$

■

It is instructive to specialise Lemma 4.6 to the standard case: we then have

$$\begin{array}{lll} & \Diamond\mathcal{P} & \\ \equiv & \text{wp.}(\textbf{do } \mathcal{P}{<}\Diamond\mathcal{P} \rightarrow \mathit{Step} \textbf{ od}).\mathcal{P} & \\ \equiv & \text{wp.}(\textbf{do } (\mathcal{P}{=}0) \wedge (\Diamond\mathcal{P}{=}1) \rightarrow \mathit{Step} \textbf{ od}).\mathcal{P}\ . & \mathcal{P} \text{ is standard} \end{array}$$

Thus for standard $\mathcal{P}$ the “recommended” strategy is to continue playing as long as $\mathcal{P}$ does not hold ($\mathcal{P}{=}0$) but only *provided* it will hold eventually ($\Diamond\mathcal{P}{=}1$) — since otherwise there is no point in continuing, and one might as well give up now. The earlier operational interpretation of $\Diamond\mathcal{P}$ in Program (8) remains valid, but its strategy is missing the second conjunct: if that gambler is ignorant of $\Diamond\mathcal{P}{\neq}1$ he continues to play, but can never win.

4.3 The *always* game

It can be shown [11] that for deterministic and non-diverging *step* we have

$$step.(1-\mathcal{A}) \quad \equiv \quad 1 - step.\mathcal{A} \ ,$$

where the subtraction $1-$ acts pointwise over $\mathcal{A}$; then straightforward comparison of Definitions 3.3 and 3.4 gives us the following lemma.

Lemma 4.8 If the underlying computation is deterministic and non-diverging, then

$$\Box\mathcal{A} \quad \equiv \quad 1 - \Diamond(1-\mathcal{A})$$

for any expectation $\mathcal{A}$. ■

With that lemma and Theorem 4.7 we can calculate the strategy for an *always* game, as follows: writing $\overline{\mathcal{A}}$ for $1-\mathcal{A}$ when convenient, we have

$$\begin{array}{lll}
 & \Box\mathcal{A} & \\
\equiv & 1 - \Diamond\overline{\mathcal{A}} & \text{Lemma 4.8} \\
\equiv & 1 - (\sqcup F \bullet \mathrm{wp}.(\mathbf{do}\ \overline{\mathcal{A}} {<} \Diamond_F \overline{\mathcal{A}} \rightarrow \mathit{Step}\ \mathbf{od}).\overline{\mathcal{A}}) & \text{Theorem 4.7} \\
\equiv & (\sqcap F \bullet 1 - \mathrm{wp}.(\mathbf{do}\ \overline{\mathcal{A}} {<} \Diamond_F \overline{\mathcal{A}} \rightarrow \mathit{Step}\ \mathbf{od}).\overline{\mathcal{A}}) & \\
\equiv & & \mathit{Step}\text{ deterministic and non-divergent: see below} \\
 & (\sqcap F \bullet \mathrm{wnp}.(\mathbf{do}\ \overline{\mathcal{A}} {<} \Diamond_F \overline{\mathcal{A}} \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A}) & \\
\equiv & (\sqcap F \bullet \mathrm{wnp}.(\mathbf{do}\ \mathcal{A} {>} \Box_F \mathcal{A} \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A}) \ , & \text{Lemma 4.8}
\end{array}$$

where by analogy with $\Diamond_F$ we define $\Box_F\mathcal{A}$ to be $\Box\mathcal{A}$ interpreted over Step_F instead of *Step*. For the deferred justification note that the $(1-)$ converts the least fixed point to a greatest fixed point, while at the same time replacing $step.\mathcal{X}$ by $1 - step.\overline{\mathcal{X}}$ in the fixed-point equation: but since *step* is deterministic and non-divergent those two terms are equal.

Thus $\Box\mathcal{A}$ is given as an *infimum* of strategies strategies of the gambling game. The analogue of Lemma 4.1 shows that it is in fact the infimum over *all* strategies D of

$$\mathrm{wnp}.(\mathbf{do}\ D \rightarrow \mathit{Step}\ \mathbf{od}).\mathcal{A} \ ,$$

in which the decision to stop or continue is taken by a *demon* with the gambler's worst interests at heart. The demon will not force the gambler to play forever, however, since wnp interprets non-termination as success and the demon strives for (the gambler's) failure.

4.4 The *unless* game

The game for $\mathcal{A} \rhd \mathcal{B}$ combines the eventually and always games: the gambler tries to maximise the expectation, while the demon tries to minimise it. Let the gambler's strategy be G and the demon's D; the game is then

$$\begin{array}{l}
(\nu X \bullet \quad \mathbf{if}\ \neg P\ \mathbf{then\ resultis}\ \mathcal{B} \\
\quad \mathbf{elsif}\ \neg D\ \mathbf{then\ resultis}\ \mathcal{A} \\
\quad\quad \mathbf{else}\ \mathit{Step}; X) \ .
\end{array} \tag{15}$$

On each step the gambler decides whether to stop and accept $\mathcal{B}$ in the current state; if the gambler did not stop then the demon decides whether to force him to stop, and accept $\mathcal{A}$ instead; if neither stop, one step is taken and the game continues (with non-termination interpreted as success for the gambler).

For strategies G, D and expectations $\mathcal{A}, \mathcal{B}$ denote by $U_{G,D}.(\mathcal{A}, \mathcal{B})$ the result of playing the game (15) above. It can be shown that

$$(\sqcap D \bullet U_{G,D}.(\mathcal{A}, \mathcal{B})) \quad \Rrightarrow \quad \mathcal{A} \rhd \mathcal{B} \quad \Rrightarrow \quad (\sqcup G \bullet U_{G,D}.(\mathcal{A}, \mathcal{B})) \ ,$$

which states (on the left) that for all gambler's strategies G the demon can ensure that he wins no more than $\mathcal{A} \rhd \mathcal{B}$, and (on the right) that for all demon's strategies D the gambler can win at least $\mathcal{A} \rhd \mathcal{B}$. In the limit we have

$$(\sqcup P \bullet (\sqcap D \bullet U_{G,D}.(\mathcal{A}, \mathcal{B}))) \quad \equiv \quad \mathcal{A} \rhd \mathcal{B} \quad \equiv \quad (\sqcap D \bullet (\sqcup P \bullet U_{G,D}.(\mathcal{A}, \mathcal{B}))) \ ,$$

showing that if both players use their best strategy, the result is $\mathcal{A} \rhd \mathcal{B}$.

5 Conclusion

The generalisation from predicates to expectations is suggested by Kozen for imperative (but deterministic) programs [8]; in [13] we extend that to include nondeterminism. Morris' formulation of temporal properties as predicate transformers [14] then gives access to a probabilistic temporal logic.

Our principal technical contributions are two. The first is demostrating the agreement between the simple fixed-point definitions in Sec. 3.4 and the operational interpretations as games in Sec. 4 — that for example $\diamond\mathcal{A}$ is the least upper bound of all "seek to maximise" game strategies based on $\mathcal{A}$. Such an agreement is needed to justify our fixed-point definitions, in spite of their simplicity: we must be sure they correspond to some actual behaviour. Back and von Wright [1] and Stirling [18] suggest similar uses for games, but over predicates (Boolean-valued) rather than the more general expectations (real-valued).

Our second substantial contribution (not reported here) is the identification of *sub-linearity* as the key property of the *next* operator. It is based on the healthiness conditions for probabilistic predicate transformers [13]: the property is that

$$\circ(a\mathcal{A} + b\mathcal{B} \ominus c) \quad \Lleftarrow \quad a(\circ\mathcal{A}) + b(\circ\mathcal{B}) \ominus c \ ,$$

for arbitrary expectations $\mathcal{A}$, $\mathcal{B}$ and non-negative reals a,b and c with $a\mathcal{A}$ denoting the pointwise multiplication of expectation $\mathcal{A}$ by scalar a, and $x \ominus y$ denoting $(x - y) \sqcup 0$.

When $\mathcal{A}$ and $\mathcal{B}$ are standard the property specialises to distributivity of conjunction, as in a modal algebra.

Using elementary fixed-point properties of the remaining operators *eventually*, *always* and *unless*, we are able with sub-linearity to establish probabilistic analogues for many if not most of the axioms for standard branching-time temporal logic.

References

[1] R.-J.R. Back and J. von Wright. Interpreting nondeterminism in the refinement calculus. In He Jifeng, John Cooke, and Peter Wallis, editors, *Proceedings of the BCS-FACS 7th Refinement Workshop*, Workshops in Computing. Springer Verlag, July 1996. Invited lecture. See *http://www.springer.co.uk/ewic/workshops/7RW.*

[2] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[3] E.W. Dijkstra. *A Discipline of Programming.* Prentice Hall International, Englewood Cliffs, N.J., 1976.

[4] Yishai A. Feldman and David Harel. A probabilistic dynamic logic. *J. Computing and System Sciences*, 28:193–215, 1984.

[5] Probabilistic Systems Group. Notes on the *random walk*: an example of probabilistic temporal reasoning. Available via *http* [15].

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

[7] Wim Hesselink. Safety and progress of recursive procedures. *Formal Aspects of Computing*, 7:389–411, 1995.

[8] D. Kozen. A probabilistic PDL. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, New York, 1983. ACM.

[9] Daniel Lehmann and Saharon Shelah. Reasoning with time and chance. *Information and Control*, 53:165–198, 1982.

[10] J.J. Lukkien and J.L.A. van de Snepscheut. Weakest preconditions for progress. *Formal Aspects of Computing*, 4:195–236, 1992.

[11] A. K. McIver and C. C. Morgan. Probabilistic predicate transformers: part 2. Technical Report PRG-TR-5-96, Programming Research Group, March 1996. Also available via *http* [15].

[12] C. C. Morgan. Proof rules for probabilistic loops. In He Jifeng, John Cooke, and Peter Wallis, editors, *Proceedings of the BCS-FACS 7th Refinement Workshop*, Workshops in Computing. Springer Verlag, July 1996. See [15] and *http://www.springer.co.uk/ewic/workshops/7RW.*

[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. Also available via *http* [15].

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

[15] PSG. Probabilistic Systems Group: Collected reports. *http://www.comlab.ox.ac.uk/oucl/groups/probs/bibliography.html.*

[16] K. Seidel, C. C. Morgan, and A. K. McIver. An introduction to probabilistic predicate transformers. Technical Report PRG-TR-6-96, Programming Research Group, February 1996. Also available via *http* [15].

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

[18] Colin Stirling. Local model checking games. In *CONCUR 95*, number 962 in LNCS, pages 1–11. Springer Verlag, 1995. Extended abstract.