Electronic Notes in Theoretical Computer Science 22 (1999)
URL: http://www.elsevier.nl/locate/entcs/volume22.html 19 pages

Computing Probability Bounds for Linear Time Formulas over Concurrent Probabilistic Systems

Christel Baier [a], Marta Kwiatkowska [b,⋆] and Gethin Norman [b,⋆]

[a] *Fakultät für Mathematik & Informatik*
Universität Mannheim
68131 Mannheim, Germany
baier@pi1.informatik.uni-mannheim.de

[b] *School of Computer Science*
University of Birmingham, Edgbaston
Birmingham B15 2TT, UK
{mzk,gxn}@cs.bham.ac.uk

Abstract

Probabilistic verification of concurrent probabilistic systems against linear time specifications is known to be expensive in terms of time and space: time is double exponential in the size of the formula and polynomial in the size of the state space, and space complexity is single exponential [24,7]. This paper proposes to compute for a linear time formula only a lower and upper bound on the probability measure of the set of paths satisfying it, instead of calculating the exact probability. This yields a coarser estimate, namely an interval of values in [0,1] which contains the actual probability, but the calculation is simpler and more efficient (time is single exponential and space complexity is linear), and could thus be useful as an initial check in a model checking tool.

1 Introduction

Verification of systems such as probabilistic protocols or randomized algorithms against temporal or modal specifications involves calculating the probability measure of the set of paths satisfying the given path formula. Several model checking algorithms for probabilistic extensions of temporal logics have been proposed in the literature. For the *sequential* case (essentially discrete Markov chains), the problem of computing exact probabilities can be reduced,

⋆ Research partially supported by the EPSRC project GR/M04617.

via an involved construction, to a linear equation system; an algorithm with time complexity exponential in the size of the formula and polynomial in the size of the system has been proposed in [7], and also in [24] lower bound PSPACE-hardness is shown. The case of *concurrent* probabilistic systems (similar to concurrent Markov chains of [24]) is more complex. These allow a nondeterministic choice between probability distributions on successor states, where that choice is understood to arise in the context of a distributed computation and is made by a scheduler (which we call *adversary*). Since there is *no* unique probability measure that can be defined on the set of paths in this case, we are unable to compute the exact likelihoods, and instead work with *minimum/maximum* probabilities over all adversaries. Two approaches have been introduced, qualitative and quantitative, of which we work with the latter. For the *qualitative* properties we require that the formula, say representing absence of deadlock, holds with probability 1, which may involve making suitable fairness assumptions on the nondeterministic choices. A suitable algorithm, presented in [7], has time complexity double exponential in the size of the formula and quadratic in the size of the system. For the so called *quantitative* properties, which involve establishing that the likelihood is at least some threshold value p where $p \in [0, 1]$, algorithms with time complexity double exponential in the size of the formula and polynomial in the size of the system have been proposed in [6,7] and [8] (improvement of [4]); a lower bound is double exponential time, and hence the problem is hard for single exponential space [25,7].

The above-mentioned complexity results are rather discouraging from the point of view of practical implementations of model checking tools for probabilistic systems. Without substantial efficiency gains through appropriate choice of data structures, these algorithms are unlikely to scale up to medium-size, let alone realistic, applications. The implications of this inherent complexity are rather fundamental: establishing satisfaction of atomic propositions, their conjunctions, and even path formulas, such as $p \,\mathcal{U}\, q$ where p, q are atomic propositions, is relatively straightforward (it amounts to solving a linear equation system in the latter case), but if p, q are allowed to be *path formulas* then calculating the exact probabilities involves first establishing conditional probabilities for p and q, and this is needed even for conjunctions. In the model checking algorithms of [7,4] this is reflected by the need to transform the underlying Markov chains to much larger ones which encode conditional probabilities (in the case of [7]) and (in [4]) the "history of computation" with the help of atomic propositions.

This paper proposes to calculate the *lower* and *upper bounds* on probabilities of temporal formulas being satisfied, instead of calculating the exact probabilities. Lower and upper bounds were also considered in [18,13,12,14] in the context of variants of the mu-calculus. The main advantage of such an approach is its simplicity and efficiency, particularly when space complexity is concerned (for the concurrent case, for example, the space requirement for our method is linear, as opposed to single exponential needed for exact calculations). The price to pay is the coarser nature of the probability estimates, as the outcome of the computation is an *interval* of probabilities which in general contains the minimum/maximum probability interval. In the worst case this will be [0,1], but in the best case it could coincide with the minimum/maximum probability interval. Thus, our approach would be useful as an initial check to perform on a probabilistic system, which would help identify those systems or properties that require a closer inspection.

We consider the syntax of Linear Temporal Logic (*LTL*), whose formulas we interpret in a given model as a pair of maps from its states to the [0,1] interval, denoting, for each state, the minimum and maximum *probability* of the set of forward paths from it satisfying the formula in the classical sense. To each *LTL* formula we also assign two maps (lower bound *lb* and upper bound *ub*) from states to the [0,1] interval. The idea is for the lower bound to approximate the minimum probability from below, whereas the upper bound approximates the maximum probability from above. We state the lower and upper bound calculations as optimization problems solvable via iteration; the exact probability is sandwiched between the approximations to the bounds. Our method can be viewed as a simple and space efficient heuristic approach based on a greedy algorithm, which can serve as a sound proof technique for establishing quantitative linear time properties. We also report on a case study, for which we are using a variant of the randomized dining philosophers protocol verified in [19] with the help of a proof system. The results so far are encouraging for certain classes of properties, which include all the properties verified in [19]; in particular, we can deduce the probability from the value of the lower bound.

2 Preliminaries

Let S be a finite or countable set. A *probability distribution* on S is a function $\mu : S \longrightarrow [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. A *(discrete time) Markov chain* is a tuple $(S, \mathbf{P})$ where

- S is a set of *states*,
- $\mathbf{P} : S \times S \to [0,1]$ is the *transition probability function*, i.e.

$$\sum_{t \in S} \mathbf{P}(s,t) = 1$$

for all $s \in S$. In other words, $\mu_s(t) \stackrel{\text{def}}{=} \mathbf{P}(s,t)$ is a probability distribution.

We can unfold a Markov chain in the standard way into a set of execution sequences (paths). Formally, an *execution sequence* in $(S, \mathbf{P})$ is a nonempty (finite or infinite) sequence $\pi = s_0 s_1 s_2, \ldots$ where s_i are states and $\mathbf{P}(s_{i-1}, s_i) > 0$, $i = 1, 2, \ldots$. *Path* denotes the set of infinite paths, and *Path*(s) is the set of infinite paths which have s as the start state. For $s \in S$, let $\Sigma(s)$ be the smallest σ-algebra on $Path(s)$ which contains the basic cylinders $\{\pi \in Path(s) : \omega \text{ is a finite prefix of } \pi\}$. The probability measure $Prob$ on $\Sigma(s)$ is the unique measure satisfying:

$$Prob\,\{\ \pi \in Path(s)\ : \omega \text{ is a finite prefix of } \pi\ \} \;=\; \mathbf{P}(\omega)$$

where $\mathbf{P}(s_0 s_1 \ldots s_k) = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{k-1}, s_k)$.

3 Concurrent probabilistic transition systems

A *concurrent probabilistic transition system* is a pair $(S, Steps)$ where

- S is a set of states
- $Steps$ is a function which assigns to each state $s \in S$ a finite non-empty set $Steps(s)$ of distributions on S.

A system $(S, Steps)$ is called finite iff S is finite. If for each $s \in S$ the set $Steps(s)$ is a singleton set, then $(S, Steps)$ is called a *sequential* probabilistic transition system.

Intuitively, $Steps$ represents the nondeterministic alternatives in each state: given a state $s \in S$, the choice of a distribution μ from $Steps(s)$ is made by a scheduler, and then a probabilistic transition is made according to μ, i.e. some state t with positive probability ($\mu(t) > 0$) is selected and the process moves to that state.

Execution sequences of concurrent probabilistic transition systems (which we call paths) arise by alternately resolving the nondeterministic and probabilistic choices. Formally, a *path* in a concurrent probabilistic system $(S, Steps)$ is a nonempty (finite or infinite) "sequence"

$$\pi \;=\; s_0 \stackrel{\mu_1}{\to} s_1 \stackrel{\mu_2}{\to} s_2 \;\ldots$$

where s_i are states, $\mu_i \in Steps(s_{i-1})$ and $\mu_i(s_i) > 0$, $i = 1, 2, \ldots$. (The case $\pi = s_0$ is allowed.) A path π is called a *fulpath* iff it is infinite. We use the following notation for paths. The first state of a path π is denoted by $first(\pi)$. If π is finite then the last state of π is denoted by $last(\pi)$. The length $|\pi|$ of a path is defined in the usual way as follows: if $\pi = s_0 \in S$ then $|\pi| = 0$; otherwise, if $\pi \;=\; s_0 \stackrel{\mu_1}{\to} s_1 \stackrel{\mu_2}{\to} \ldots \stackrel{\mu_n}{\to} s_n$, then $|\pi| = n$. For infinite π we put $|\pi| = \infty$. Let π be a finite or infinite path as above. If $k \leq |\pi|$ then

- $\pi(k)$ denotes the k-th state of π (i.e. $\pi(k) = s_k$)
- $\pi^{(k)}$ is the k-th prefix of π (i.e. if $k < |\pi|$ then $\pi^{(k)} \;=\; s_0 \stackrel{\mu_1}{\to} s_1 \stackrel{\mu_2}{\to} \ldots \stackrel{\mu_k}{\to} s_k$, if $k \geq |\pi|$ then $\pi^{(k)} = \pi$)
- $step(\pi, k)$ denotes the k-th step in π (i.e. if $k < |\pi|$ then $step(\pi, k) = \mu_{k+1}$).

$Path(s)$ denotes the set of fulpaths π with $first(\pi) = s$. Similarly, $Path_{fin}(s)$ is the set of finite paths ω with $first(\omega) = s$.

We unfold a concurrent probabilistic system (S, Steps) into its computation trees (called "execution trees" in [10] and "maximal resolutions" in [15]), with each component described as a Markov chain. The computation trees arise by resolving the nondeterministic choices (but not the probabilistic choices). It is convenient to suppose that an *adversary* (also called *scheduler* or *policy*) decides, based of the past history of the system, which of the possible steps (probability distributions) to perform next.

Formally, an adversary of a concurrent probabilistic transition system (S, Steps) is a function A mapping every finite path ω to a distribution $A(\omega)$ on S such that $A(\omega) \in \mathit{Steps}(\mathit{last}(\omega))$. $\mathit{Path}^A(s)$ denotes the set of all paths $\pi \in \mathit{Path}(s)$ with $\mathit{step}(\pi, i) = A(\pi^{(i)})$ for all $i \geq 0$. Similarly, we define the set of finite paths $\mathit{Path}^A_{\mathit{fin}}(s)$ induced by the adversary A to be the set of all finite paths $\omega \in \mathit{Path}_{\mathit{fin}}(s)$ with $\mathit{step}(\omega, i) = A(\omega^{(i)})$ for all $i < |\omega|$. $\mathcal{A}dv$ denotes the set of all adversaries.

Thus, an adversary chooses for every finite path ω a distribution from $\mathit{Steps}(\mathit{last}(\omega))$. Note that we only consider deterministic adversaries. The notion of randomization of adversaries or probabilistic adversaries has been investigated in [10,4,22], where it is shown that the probability of a measurable set Γ w.r.t. a randomized adversary is a convex combination of the measure of Γ w.r.t. non-randomized adversaries, and hence lies between the minimal and maximal measure of Γ w.r.t. non-randomized adversaries. By Corollary 20 of [4], it suffices to only consider the deterministic adversaries, as the minimal/maximal measures over randomized adversaries coincide with those for the deterministic adversaries. Hence, our results carry over to the randomized adversaries.

With each adversary we associate a discrete time (in general infinite-state) Markov chain which can be viewed as a computation tree. Let $\mathit{Path}^A_{\mathit{fin}}$ denote $\bigcup_{s \in S} \mathit{Path}^A_{\mathit{fin}}(s)$. Formally, if A is an adversary of a concurrent probabilistic system (S, Steps), then $(\mathit{Path}^A_{\mathit{fin}}, \mathbf{P}^A)$ is a Markov chain where

$$\mathbf{P}^A(\omega, \sigma) \;=\; \begin{cases} A(\omega)(\mathit{last}(\sigma)) & : \text{if } \omega = \sigma^{(k-1)} \text{ where } |\sigma| = k \\ 0 & : \text{otherwise.} \end{cases}$$

We identify each path $x = \omega_0 \omega_1 \ldots$ in $(\mathit{Path}^A_{\mathit{fin}}, \mathbf{P}^A)$ which starts in a state $s_0 \in S$ (i.e. $\omega_0 = s_0$ is a path of length 0) with the path

$$\mathit{last}(\omega_0) \xrightarrow{A(\omega_0)} \mathit{last}(\omega_1) \xrightarrow{A(\omega_1)} \ldots$$

in (S, Steps). Vice versa, if π is a fulpath in (S, Steps), $\pi \in \mathit{Path}^A$, then we identify π with the path $x = \pi^{(0)} \pi^{(1)} \pi^{(2)} \ldots$ in $(\mathit{Path}^A_{\mathit{fin}}, \mathbf{P}^A)$. This yields a one-to-one correspondence between $\mathit{Path}^A(s)$ and the fulpaths in $(\mathit{Path}^A_{\mathit{fin}}, \mathbf{P}^A)$. Hence, for each state $s \in S$ and adversary A, the probability measure on $(\mathit{Path}^A_{\mathit{fin}}, \mathbf{P}^A)$ induces a probability measure on $\mathit{Path}^A(s)$, which we denote

$Prob^A(\cdot)$.

We define a *labelled concurrent probabilistic transition system* as a tuple $(S, Steps, AP, L)$ consisting of a concurrent probabilistic transition system $(S, Steps)$ and

- a finite set AP of atomic propositions,
- a *labelling function* $L : S \to 2^{AP}$ which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that hold in s.

4 Linear time logic *LTL*

We fix a set AP of atomic propositions. The syntax of LTL (linear time logic) formulas over AP is given by the grammar:

$$f ::= \quad tt \;\mid\; a \;\mid\; \neg f \;\mid\; f_1 \wedge f_2 \;\mid\; Xf \;\mid\; f_1 \,\mathcal{U}\, f_2$$

where $a \in AP$.

We interpret LTL formulas over the states of a labelled concurrent probabilistic transition system $(S, Steps, AP, L)$ as follows. We adopt the standard satisfaction relation $\models \subseteq Path \times LTL$ [24]. By [24], for a fixed adversary and formula f, the sets of paths of the form $\{\pi \in Path^A(s) : \pi \models f\}$ are measurable, and hence $Prob^A$ for such a set is defined. Next we define, for each state s and LTL formula f, the maps $Prob_f^{inf}(s)$ and $Prob_f^{sup}(s)$ which respectively assign the *minimal* and *maximal* probability that f holds for an execution starting in s over all adversaries. The need for the minimal and maximal probability arises due to the presence of nondeterminism, which prevents us from identifying a single probability space on paths. The quantification over all adversaries is necessary to ensure that the property holds even in the worst case scenario, and agrees with the usual treatment of LTL on nonprobabilistic systems.

Let f be a LTL formula. We define maps:

$$Prob_f^{inf} : S \to [0,1], \;\; Prob_f^{inf}(s) = \inf_{A \in \mathcal{A}dv} Prob^A \left\{ \pi \in Path^A(s) : \pi \models f \right\},$$

$$Prob_f^{sup} : S \to [0,1], \;\; Prob_f^{sup}(s) = \sup_{A \in \mathcal{A}dv} Prob^A \left\{ \pi \in Path^A(s) : \pi \models f \right\}.$$

The values $Prob_f^{inf}(s)$ and $Prob_f^{sup}(s)$ induce an interval semantics for LTL formulas when interpreted over the states of a labelled concurrent probabilistic system. Note that since sequential systems admit only one adversary, for such systems these intervals collapse to single values. Clearly, this interval semantics is preserved by "maximal trace equivalence", defined in an appropriate way in the style of [21], where a probabilistic counterpart to (finite) trace inclusion is given.

5 Defining lower and upper bounds on probability for the truth-value of *LTL* formulas

We now propose the maps *lb* and *ub* from *LTL* formulas and states to the [0,1] interval which respectively define a lower and upper bound on the probability of a formula holding in a state over all the adversaries of a concurrent probabilistic transition system. Given a state s and a formula f, the pair of bounds gives an interval which in general properly contains the probability interval of f being satisfied in s.

The maps are built by induction on the structure of the formula from atomic propositions (assigned 0 or 1 as expected) and monotone operators on state vectors (functions from the set of states to the interval [0,1]) modelling conjunction, negation and "next state". The case of "until" is dealt with by a reduction to the equivalent fixed point formulation which only involves "next state" and Boolean operators, followed by conversion to a (non-linear) optimization problem which can be solved with the help of an iterative method. We omit the existence proofs of unique solutions to the optimization problems, which are verified through reduction to the existence proofs of least and greatest fixed points.

In what follows, we fix a labelled concurrent probabilistic transition system $(S, \mathit{Steps}, AP, L)$. Recall that concurrent probabilistic systems allow a *set* of next-state probability distributions given by the mapping *Steps*. We define lower and upper bounds, $lb(f, s)$ and $ub(f, s)$, respectively approximating $Prob_f^{inf}(s)$ and $Prob_f^{sup}(s)$ as follows.

5.1 *Atomic propositions and Boolean connectives*

First we define the maps $lb(f, s)$ and $ub(f, s)$ where f is an *atomic proposition*. For the usual qualitative interpretation of atomic propositions it suffices to assign the lower/upper bound to 0 or 1 depending on whether the proposition is satisfied in the given state or not. More generally, however, one could allow arbitrary probability distributions as meanings of atomic propositions.

Next, we consider *conjunctions*. Let A be an adversary. We aim to define an operator $\wedge$ on functions from the set of states to the interval [0,1] such that $lb(f_1, s) \wedge lb(f_2, s)$ yields an approximation to $Prob^A_{f_1 \wedge f_2}(s)$ from below (the case of the upper bound *ub* is dual). Note that f_1 and f_2 could be path formulas, and hence in order to compute the exact probability we would require knowledge of dependence/independence of the underlying events, together with the corresponding conditional probabilities. Our idea is instead to assume the *worst case* outcome and compute only the greatest lower bound on the probability measures computed for the conjuncts (which could be 0) [1]. Dually, we assume the *best case* outcome when calculating the upper bound.

[1] A similar map for conjunctions was used in [23], where we refer the reader for a more detailed discussion.

Unfortunately, we are unable to estimate the error, as in the worst case the values obtained could be 0 for the lower bound and 1 the for upper bound.

The soundness of our approach follows from the following observation, which can be thought of as yielding expressions defining lower and upper bounds for conjunctions and disjunctions:

Lemma 5.1 *Let f_1, f_2 be LTL formulas. Then, for all $A \in \mathcal{A}dv$ and $s \in S$:*

$$Prob^A_{f_1}(s) \ + \ Prob^A_{f_2}(s) \ - \ 1 \leq Prob^A_{f_1 \wedge f_2}(s) \leq \min\left\{Prob^A_{f_1}(s), Prob^A_{f_2}(s)\right\},$$
$$\max\left\{Prob^A_{f_1}(s), Prob^A_{f_2}(s)\right\} \leq Prob^A_{f_1 \vee f_2}(s) \leq Prob^A_{f_1}(s) + Prob^A_{f_2}(s).$$

Proof. Easy verification. We simply use the fact that for any $A \in \mathcal{A}dv$:

$$Prob^A_{f_1 \vee f_2}(s) + Prob^A_{f_1 \wedge f_2}(s) \ = \ Prob^A_{f_1}(s) + Prob^A_{f_2}(s).$$

□

The treatment of *negation* is rather delicate, as it changes the nature of the bound: if we complement a lower bound to 1 we get an *upper* bound, and vice-versa [12]. Altogether, we arrive at the definition given below.

Definition 5.2 [Atomic propositions, conjunction, negation] Let a be an atomic proposition and f_1, f_2 and f LTL formulas. The functions lb, $ub : LTL \times S \to [0,1]$ are defined as follows.

- $lb(tt, s) = ub(tt, s) = 1$.
- Atomic propositions:

$$lb(a, s) = ub(a, s) = \begin{cases} 1 : \text{if } a \in L(s) \\ 0 : \text{otherwise.} \end{cases}$$

- Conjunction:

$$\begin{aligned} lb(f_1 \wedge f_2, s) &= \max\{0, lb(f_1, s) + lb(f_2, s) - 1\} \\ ub(f_1 \wedge f_2, s) &= \min\{ub(f_1, s), ub(f_2, s)\}. \end{aligned}$$

- Negation:

$$\begin{aligned} lb(\neg f, s) &= 1 - ub(f, s) \\ ub(\neg f, s) &= 1 - lb(f, s). \end{aligned}$$

We next consider the lower and upper bounds of the derived Boolean operators, namely disjunction $\vee$ and implication $\to$. These can be obtained by expanding the classical identities $f_1 \vee f_2 = \neg(\neg f_1 \wedge \neg f_2)$ and $f_1 \to f_2 = \neg f_1 \vee f_2$ with operators for $\wedge$ and $\neg$ given in Definition 5.2:

$$\begin{aligned} lb(f_1 \vee f_2, s) &= \max\{lb(f_1, s), lb(f_2, s)\} \\ ub(f_1 \vee f_2, s) &= \min\{1, ub(f_1, s) + ub(f_2, s)\} \\ lb(f_1 \to f_2, s) &= \max\{1 - ub(f_1, s), lb(f_2, s)\} \\ ub(f_1 \to f_2, s) &= \min\{1, 1 - lb(f_1, s) + ub(f_2, s)\}. \end{aligned}$$

It should be noted that the Boolean operators defined with respect to lower and upper bounds are no longer idempotent. Thus, the results of the functions *lb* and *ub* are sensitive to the way a formula is written. For example, for any *LTL* formula f, $f \wedge f \equiv f$ and $f \vee f \equiv f$, where $\equiv$ stands for logical equivalence. However, if $lb(f,s) \notin \{0,1\}$ and $ub(f,s) \notin \{0,1\}$ then:

$$lb(f \wedge f, s) < lb(f,s) \text{ and } ub(f \vee f, s) > ub(f,s).$$

5.2 The temporal operators

We now consider the "next state" operator Xf. When defining the lower bound, we assume the *worst case* scenario, that is, an adversary that chooses the step that results in the *minimum* probability value. Clearly, for a fixed next state probability distribution $\mu \in Steps(s)$ from state s, the weighted sum $\sum_{t \in S} \mu(t) \cdot lb(f,t)$ is a lower bound. Therefore, taking the minimum of such values over *all* $\mu \in Steps(s)$ yields a lower bound for Xf holding in s. Dually, to calculate the upper bound (*best case* scenario) it suffices to take the maximum of the weighted sums $\sum_{t \in S} \mu(t) \cdot ub(f,t)$.

Definition 5.3 [**Next state**] Let f be a *LTL* formula. The functions *lb*, $ub : LTL \times S \to [0,1]$ are defined as follows.

- Next state:

$$lb(Xf,s) = \min\left\{ \sum_{t \in S} \mu(t) \cdot lb(f,t) \;:\; \mu \in Steps(s) \right\},$$

$$ub(Xf,s) = \max\left\{ \sum_{t \in S} \mu(t) \cdot ub(f,t) \;:\; \mu \in Steps(s) \right\}.$$

To deal with the "until" operator $f_1 \,\mathcal{U}\, f_2$ we first reduce it to the equivalent fixed point representation $f_2 \vee (f_1 \wedge X(f_1 \,\mathcal{U}\, f_2))$. Since the fixed point formulation only uses Boolean connectives and "next state", we can expand it using the operators for $\vee$, $\wedge$ and Xf defined in Definitions 5.2 and 5.3. More specifically, in the case of the lower bound we need to solve the set of recursive equations: for all $s \in S$

$$\begin{aligned}
&lb(f_1 \,\mathcal{U}\, f_2, s) = lb(f_2 \vee (f_1 \wedge X(f_1 \,\mathcal{U}\, f_2)), s)\\
&= \max\{lb(f_2,s), lb(f_1 \wedge X(f_1 \,\mathcal{U}\, f_2), s)\}\\
&= \max\{lb(f_2,s), \max\{0, lb(f_1,s) + lb(X(f_1 \,\mathcal{U}\, f_2), s) - 1\}\\
&= \max\left\{ lb(f_2,s), lb(f_1,s) + \min\left\{ \sum_{t \in S} \mathbf{P}(s,t) \cdot x_t : \mu \in Steps(s) \right\} - 1 \right\}.
\end{aligned}$$

What we have obtained is an optimization problem (non-linear because of nested minimum). Observe that in the restricted case of sequential systems the nested minimum is taken over a singleton set, resulting in a *linear* optimization problem. The case of the upper bound is dual.

This motivates the definition below.

Definition 5.4 [Until] Let f_1 and f_2 be LTL formulas. The functions lb, $ub : LTL \times S \to [0,1]$ are defined as follows.

- The "until" operator:
  - The vector $(lb(f_1 \,\mathcal{U}\, f_2, s))_{s \in S}$ is the unique solution of the following optimization problem:

$$\begin{aligned} 0 \;\leq\; x_s \;&\leq 1 \\ x_s \;&\geq\; lb(f_2, s) \\ x_s \;&\geq\; lb(f_1, s) - 1 + \min\left\{ \sum_{t \in S} \mu(t) \cdot x_t : \mu \in \mathit{Steps}(s) \right\} \end{aligned}$$

    where $\sum_{s \in S} x_s$ is minimal.
  - The vector $(ub(f_1 \,\mathcal{U}\, f_2, s))_{s \in S}$ is the unique solution of the following optimization problem:

$$\begin{aligned} 0 \;\leq y_s \;&\leq\; 1 \\ y_s \;&\leq\; ub(f_1, s) \;+\; ub(f_2, s) \\ y_s \;&\leq\; ub(f_2, s) + \max\left\{ \sum_{t \in S} \mu(t) \cdot y_t : \mu \in \mathit{Steps}(s) \right\} \end{aligned}$$

    where $\sum_{s \in S} y_s$ is maximal.

The existence of unique solutions of the optimization problems given in the definition above follows from fixed point arguments for maps on state vectors [3].

Similarly, we can unwind the operators $\Diamond f = (tt \,\mathcal{U}\, f)$ and $\Box f = (\neg \Diamond \neg f)$. Consider, for example, $\Diamond f$:

- The vector $(lb(\Diamond f, s))_{s \in S}$ is the unique solution of the following optimization problem:

$$\begin{aligned} 0 \;\leq\; x_s \;&\leq 1 \\ x_s \;&\geq\; lb(f, s) \\ x_s \;&\geq\; \min\left\{ \sum_{t \in S} \mu(t) \cdot x_t : \mu \in \mathit{Steps}(s) \right\} \end{aligned}$$

  where $\sum_{s \in S} x_s$ is minimal.
- The vector $(ub(\Diamond f, s))_{s \in S}$ is the unique solution of the following optimization problem:

$$\begin{aligned} 0 \;\leq y_s \;&\leq\; 1 \\ y_s \;&\leq\; ub(f, s) \;+\; \max\left\{ \sum_{t \in S} \mu(t) \cdot y_t : \mu \in \mathit{Steps}(s) \right\} \end{aligned}$$

  where $\sum_{s \in S} y_s$ is maximal.

We note that, for all $s \in S$ and $f \in LTL$, the upper bound for $\Diamond f$ is constant and equal to 1, and dually $lb(\Box f, s) = 0$.

The next lemma and theorem state the correctness of our model checking method: for a given adversary, the lower and upper bounds as defined in Definition 5.4 respectively approximate from below and above the exact probability measure of the set of paths satisfying "until".

Lemma 5.5 *Let f_1, f_2 be LTL formulas. Then, for all $A \in \mathcal{A}dv$ and $s \in S$:*

$$Prob^A_{f_1 \mathcal{U} f_2}(s) \geq \max\left\{Prob^A_{f_2}(s),\ Prob^A_{f_1}(s) \ + \ Z^A_s \ - \ 1 \right\},$$

$$Prob^A_{f_1 \mathcal{U} f_2}(s) \leq \min\left\{1, Prob^A_{f_1}(s) + Prob^A_{f_2}(s),\ Prob^A_{f_2}(s) + \ W^A_s \ \right\}$$

where

$$Z^A_s = \min \left\{ \sum_{t \in S} \mu(t) \cdot Prob^A_{f_1 \mathcal{U} f_2}(t) \ : \ \mu \in Steps(s) \right\},$$

$$W^A_S = \max \left\{ \sum_{t \in S} \mu(t) \cdot Prob^A_{f_1 \mathcal{U} f_2}(t) \ : \ \mu \in Steps(s) \right\}.$$

Proof. Uses Lemma 5.1 and the fact that $f_1 \mathcal{U} f_2$ is logically equivalent to $f_2 \vee (f_1 \wedge X(f_1 \mathcal{U} f_2))$. □

Theorem 5.6 *For all LTL formulas f, $A \in \mathcal{A}dv$ and states $s \in S$:*

$$lb(f,s) \ \leq \ Prob^A_f(s) \ \leq \ ub(f,s).$$

Hence, $lb(f,s) \ \leq \ Prob^{inf}_f(s) \ \leq \ Prob^{sup}_f(s) \ \leq \ ub(f,s)$.

Proof. By structural induction on the syntax of f. For conjunction we use Lemma 5.1. We consider the "until" operator. Let $f = f_1 \mathcal{U} f_2$ and $A \in \mathcal{A}dv$. By induction hypothesis:

$$lb(f_i, s) \ \leq \ Prob^A_{f_i}(s), \quad i = 1,2.$$

By Lemma 5.5:

$$Prob^A_{f_1 \mathcal{U} f_2}(s) \geq \max\left\{ \ lb(f_2,s),\ lb(f_1,s) \ + \ Z^A_s - 1 \ \right\}.$$

It can be shown by induction that $lb(f_1 \mathcal{U} f_2, s) \leq Prob^A_{f_1 \mathcal{U} f_2}(s)$ for all $s \in S$. An essentially dual argument yields the claim for $ub(f_1 \mathcal{U} f_2, \cdot)$. □

5.3 Complexity of the method

The above definitions and statements result in an algorithm which inputs a concurrent probabilistic system and an *LTL* formula f, and outputs two state vectors (mappings from states to [0,1]) yielding for each state s a probability interval $[lb(f,s), ub(f,s)]$. The algorithm proceeds as follows. First, it builds the parse tree of the formula, and then traverses it in a bottom-up fashion, successively computing the lower/upper bounds for subformulas. The case of atomic propositions and Boolean operators is straightforward.

The "next state" operator involves scalar vector product and taking minimum/maximum. For the "until" operator we invoke an iterative method described below.

Lemma 5.7 *If f_1 and f_2 are LTL formulas and $s \in S$, then*

$$\lim_{l\to\infty} x_s^l \;=\; lb(f_1\,\mathcal{U}\,f_2, s), \quad \lim_{l\to\infty} y_s^l \;=\; ub(f_1\,\mathcal{U}\,f_2, s).$$

where

- $x_s^0 = z_s^0 = 0$ *and* $y_s^0 = w_s^0 = 1$
- *For* $l = 0, 1, 2, \ldots$*:*

$$z_s^l = \min\left\{ \sum_{t\in S} \mu(t)\cdot x_t^l \;:\; \mu \in \mathit{Steps}(s) \right\},$$
$$x_s^{l+1} = \max\left\{\, lb(f_2, s),\; lb(f_1, s) \;+\; z_s^l - 1 \right\},$$

$$w_s^l = \max\left\{ \sum_{t\in S} \mu(t)\cdot y_t^l \;:\; \mu \in \mathit{Steps}(s) \right\},$$
$$y_s^{l+1} = \min\left\{\, 1,\; ub(f_1, s) + ub(f_2, s),\; ub(f_2, s) \;+\; w_s^{l-1} \right\}.$$

Proof. The proof follows from the fixed point arguments used in the proof of Theorem 5.6. □

The chains $(x_s^l)_l$ and $(y_s^l)_l$ are respectively increasing and decreasing in [0,1], and in particular,

$$x_s^l \;\le\; x_s^{l+1} \;\le\; \mathit{Prob}_{f_1\,\mathcal{U}\,f_2}(s) \;\le y_s^{l+1} \;\le\; y_s^l.$$

for all $l = 0, 1, 2, \ldots$. Hence, *all* values x_s^l are lower bounds for $\mathit{Prob}^{\mathit{inf}}_{f_1\,\mathcal{U}\,f_2}(s)$, and thus premature termination will still result in sound estimates from below. Likewise, values y_s^l approximate $\mathit{Prob}^{\mathit{sup}}_{f_1\,\mathcal{U}\,f_2}(s)$ from above.

The complexity of the algorithm outlined above is summarised in the next theorem.

Theorem 5.8 *Let (S, Steps) be a concurrent probabilistic system. For each $f \in$ LTL, $lb(f,\cdot)$ and $ub(f,\cdot)$ can be approximated in space linear in the size of the formula and the number of states.*

Proof. Straightforward. □

Note that we cannot estimate the number of steps required to compute the bounds via the iterative method of Lemma 5.7 as it depends on the speed of convergence and the desired accuracy. For sequential systems, however, we can show that time complexity is polynomial.

Theorem 5.9 *Let $(S, Steps)$ be a sequential probabilistic system. For each $f \in LTL$, $lb(f, \cdot)$ and $ub(f, \cdot)$ can be approximated in time polynomial in the size of the formula and the system.*

Proof. By structural induction on the syntax of the formulas. We build the parse tree and traverse it bottom-up, successively computing the lower/upper bounds for the subformulas (nodes in the parse tree). For subformulas where the outermost operator is

- atomic proposition, negation or conjunction/disjunction: constant time
- "until": polynomial time (via a standard linear optimization algorithm, e.g. the ellipsoid method [16])
- "next state": linear time

Summing up the above, we obtain polynomial time. □

The approximation method for the lower/upper bounds on probability derived in this paper can be viewed as a sound proof technique for establishing *quantitative* linear time properties. Given an *LTL* formula f and e.g. a threshold p for the "acceptable" probabilities, we compute (in linear space) an approximation x_s^l for $lb(s, f)$. In case where we have $x_s^l \geq p$ we can deduce that, for any adversary A,

$$Prob_f^A(s) \geq Prob_f^{inf}(s) \geq lb(s, f) \geq p.$$

(Otherwise the outcome has not been determined, and an exact method may have to be invoked.) On the other hand, any exact verification algorithm requires at least single exponential space. This follows from the results by [24,7] where the verification problem for probabilistic systems with nondeterminism against (qualitative) *LTL* formulas was shown to be complete for double exponential time. Thus, the problem is hard for single exponential space. In this sense, the approximate method proposed here (which works in linear space) overcomes the limitations of exact verification methods (e.g. [4]) that might not be feasible because of space restrictions. For sequential systems, our method does not offer a reduction in space complexity, but it improves time.

6 Case Study

We use the randomized dining philosophers example verified in [19] as a case study. The verification argument of [19] is for qualitative *LTL* formulas (with probability 1) assuming fairness. Although the properties verified there involve only simple path formulas with atomic propositions, for which the calculation of exact probabilities is not any more complex than the methods proposed here, the dining philosophers protocol is nevertheless a flexible test case: by varying the number of philosophers we were able to investigate quite large systems, and we could draw comparisons with exact probability calculations.

We have built three models of the system, for three, four and five philosophers, with the help of the TIPPtool [17]. The specification of the system for n philosophers, based on [19], in terms of the process algebra TIPP [11] is given below.

$$\Big(phil(1)|||phil(2)|||\cdots|||phil(n)\Big)\|_{\{get,release,nf\}}\Big(fork(1)|||fork(2)|||\cdots|||fork(n)\Big)$$

where

$$\begin{aligned}
fork(i) &:= get!i.nofork(i)\\
nofork(i) &:= release!i.fork(i) + nf!i.nofork(i)\\
phil(i) &:= hungry.h(i) + think.phil(i)\\
h(i) &:= (left, 0.5).wl(i) + (right, 0.5).wr(i)\\
wl(i) &:= \texttt{if } i < n \texttt{ then } get!(i+1).l(i) + nf!(i+1).wl(i)\\
&\quad\ \texttt{else } get!1.l(i) + nf!1.wl(i)\\
l(i) &:= get!i.g(i) + nf!i.nr(i)\\
nr(i) &:= \texttt{if } i < n \texttt{ then } release!(i+1).h(i)\\
&\quad\ \texttt{else } release!1.h(i)\\
wr(i) &:= get!i.r(i) + nf!i.wr(i)\\
r(i) &:= \texttt{if } i < n \texttt{ then } get!(i+1).g(i) + nf!(i+1).nl(i)\\
&\quad\ \texttt{else } get!1.g(i) + nf!1.nl(i)\\
nl(i) &:= release!i.h(i)\\
g(i) &:= eat.e(i)\\
e(i) &:= \texttt{if } i < n \texttt{ then } release!(i+1).p(i)\\
&\quad\ \texttt{else } release!1.p(i)\\
p(i) &:= release!i.phil(i)
\end{aligned}$$

The properties given in [19] only hold if a suitable fairness constraint is imposed. As fairness constraints would be difficult to program in Matlab, instead we consider a fully probabilistic variant of this system. This corresponds to modelling the system where the scheduling of the nondetermistic choices is made by the randomized adversary that makes choices according to the uniform probability distribution. In the cases of three, four and five philosophers, the system has 770, 7070 and 64858 states and gives rise to sparse probability matrices $\mathbf{P}$ with 2845, 34125 and 384621 nonzero entries respectively.

Adopting the notation of [19], we have defined atomic propositions h and e such that h (e) is true in a state if, and only if, a philosopher is hungry (eating). Using the above atomic propositions we can now express **Liveness** (Theorem 2 in [19]) by: $h \to \Diamond e$. To find the lower and upper bounds for the above formula, we first calculate the lower and upper bounds for the formula $\Diamond e$. As noted above the upper bound for $\Diamond e$ will be constant and equal to 1, and therefore it suffices to only calculate the lower bound. Since the model we use is fully probabilistic, there exists only one adversary, and hence the lower bound can be computed by the following approximation (adapted from

Lemma 5.7):

- $x_s^0 = lb(e, s)$
- For $l = 0, 1, \ldots$:

$$x_s^{l+1} = \max\left\{ lb(e, s), \sum_{t \in S} \mathbf{P}(s, t) \cdot x_t^l \right\},$$

then $\lim_{l \to \infty} x_s^l = lb(\Diamond e, s)$.

We have implemented the calculation of the lower bound and exact probability [2] in Matlab using sparse matrices for each of the three systems. We consider **Liveness** together with three more properties given in [19], all of which are of the form $f_1 \to \Diamond f_2$ where both f_1 and f_2 are atomic. These are:

Property 1 If philosopher P_i is waiting for a fork (while no philosopher is eating), then eventually either some philosopher eats, or P_i will get hold of the fork ([19, Lemma 1]).

Property 2 If no philosopher is eating and some philosophers are hungry, then eventually some philosopher eats, or all the philosophers become hungry ([19, Lemma 3]).

Property 3 If a thinking philosopher has a hungry neighbour and no philosopher is eating, then eventually either some philosopher will eat, or the philosopher will become hungry ([19, Lemma 2]).

In addition, we have implemented the above (lower bound and exact) probability calculations in terms of MTBDDs (Multi-Terminal Binary Decision Diagrams [5]) using the CUDD package (The Colorado University Decision Diagram package of Fabio Somenzi) [1].

The table below summarises the results of our experiments for $\varepsilon = 10^{-6}$, first for the three, then four and finally five philosopher system. Both the lower bound and exact probability converge to 1; they are calculated by terminating the iterations when the difference between successive approximations is less than ε. Since the upper bound is also 1, in this particular case we can deduce the exact probability from the value of the lower bound (with error given by ε) for the chosen randomized adversary.

The lower bound calculation is implemented as the iterative method proposed in Section 5.3 solving the (linear) optimization problem derived for the given property (an “until” formula with atomic propositions in each case), both in Matlab and MTBDDs. The value of time *excludes* the construction of the model (which is substantial for a Matlab sparse matrix, and considerably faster for MTBDDs), and only concerns the probability calculations once a memory representation of the model has been built.

The exact probability calculation reduces to a system of linear equations (a

[2] Exact probability calculation is feasible since all properties are of the form of a single “until” with atomic propositions.

subsystem of the linear equations involving the full matrix which corresponds to the unknown probability values [2,9]) which we solve via Jacobi iteration, again both in the case of Matlab and MTBDDs. The value of time *includes* the construction of the filter which identifies the linear subsystem.

Property	BOUNDS			EXACT		
	iters	time		iters	time	
		Matlab	MTBDDS		Matlab	MTBDDS
Liveness	49	0.2397	3.34	48	0.2081	2.29
Property 1	44	0.2123	3.65	43	0.1869	2.25
Property 2	23	0.1154	1.55	20	0.0800	0.42
Property 3	22	0.1096	1.64	19	0.0665	0.23
Liveness	50	0.9832	39.19	49	1.5129	30.90
Property 1	46	0.7936	38.46	45	1.3088	28.41
Property 2	26	0.4761	13.26	25	0.5537	7.90
Property 3	26	0.3601	4.79	25	0.2490	2.53
Liveness	51	14.1584	501.57	50	112.0660	503.37
Property 1	48	13.4373	474.35	47	91.5086	529.46
Property 2	30	8.2874	202.24	29	44.0190	157.37
Property 3	30	8.1520	122.53	29	10.2395	38.32

Analysing the experimental results, we note that, as anticipated [1], sparse matrices are faster than MTBDDs on *pure calculations* involving matrix-by-vector multiplication (approx. by a factor of 10 on the models considered here). However, when it is necessary to *manipulate* the models, such as filtering out the submatrix corresponding to the subsystem of linear equations, then MTBDDs begin to have an advantage over sparse matrices, which is reflected in the reduction of the factor from 10 to 5. For comparable model size, the lower bound calculation is faster than the exact probability (recall that the exact probability calculation generally uses a *smaller* system of equations, which is why the timing for the exact calculations in the case of the three philosophers model is comparable to the lower bound calculations).

Finally, we note that, as yet, we have not fully addressed the issue of efficiency of our MTBDD implementation. Several improvements, such as heuristics for the variable ordering, Gauss-Seidel and SOR iterative methods, remain to be investigated.

7 Conclusions

We have proposed a sound, space-efficient, heuristic approach for the verification of quantitative linear time properties of concurrent probabilistic systems. The method is based on a greedy algorithm for computing the lower and upper bounds on the probability measure of the set of paths of the system satisfying an *LTL* formula, instead of the exact minimum/maximum probability. The bounds are computed via an iteration process, with each iterate respectively approximating the exact minimum/maximum probability from below/above. Hence, they can be used to verify quantitative *LTL* properties, albeit at a cost of accuracy. Our method also adapts to the sequential systems, where it leads to time improvement over the exact probability calculations. The proposed algorithm behaves well in the case of the randomized dining philosophers protocol proved in [19], but this is not surprising, since the formulas considered there are relatively simple.

We have performed initial experiments using sparse matrices and also an MTBDD-based implementation. The experiments so far are encouraging, and point to the efficiency of model construction and manipulation in terms of MTBDDs, but poorer performance compared to sparse matrices (see also [1]).

One point to note about the lower and upper bounds is that in some cases, such as all the examples discussed here, they can provide a faster and equally accurate method for computing the exact probability. In other cases, however, they may give rise to a larger interval of probabilities ([0,1] in the worst case). So far we have been unable to estimate the error associated with the lower/upper bound calculations. This seems impossible in the general case (it is relatively easy to construct an example which yields [0,1]), but perhaps one could characterise subclasses of formulas for which the error is small. Another possible direction is to investigate ways of improving the bounds by locally estimating conditional probabilities, and using those to derive more accurate estimates.

Acknowledgements

We would like to acknowledge Markus Siegle for his help in creating the philosophers example with the TIPPtool, and Dave Parker for the MTBDD-based implementation. All authors are members of the British Council/DAAD ARC project 1031 "Stochastic Modelling and Verification".

References

[1] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo and F. Somenzi. Algebraic decision diagrams and their applications. *Journal of Formal Methods in Systems Design*, 10(2/3):171–206, 1997.

[2] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Proc. ICALP'97*, volume 1256 of Lecture Notes in Computer Science, pp 430–440, Springer, 1997.

[3] C. Baier. On algorithmic verification methods for probabilistic systems. Habilitation thesis, University of Mannheim, 1999.

[4] A. Bianco, L. de Alfaro: Model checking of probabilistic and nondeterministic Systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of Lecture Notes in Computer Science, pp 499–513, Springer, 1995.

[5] E.M. Clarke, M. Fujita, P.C. McGeer, J.Yang, and X. Zhao. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. In *IWLS'93: International Workshop on Logic Synthesis, Tahoe City*, May 1993.

[6] C. Courcoubetis, M. Yannakakis: Markov decision processes and regular events. In *Proc. ICALP'90*, volume 443 of Lecture Notes in Computer Science, pages 336–349, Springer, 1990.

[7] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[8] L. de Alfaro. Formal verification of performance and reliability of real-time Systems, Technical Report STAN-CS-TR-96-1571, Stanford, 1996.

[9] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, vol. 6, pages 512–535, 1994.

[10] S. Hart, M. Sharir, A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages*, Vol. 5, pages 356–380, 1983.

[11] H. Hermanns and M. Rettelbach. Syntax, semantics, equivalences and axioms for MTIPP. In *Proc. PAPM'94*, pages 71–88, Erlangen-Regensberg, 1994.

[12] M. Huth. The interval domain: a matchmaker for aCTL and aPCTL. In *US-Brasil Joint Workshop on the Formal Foundations of Software Systems.* Electronic Notes in Computer Science, vol. 14, 1999.

[13] M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *Proc. LICS'97*, pages 111–122. IEEE Computer Society Press, 1997.

[14] M. Huth and M. Kwiatkowska. Comparing CTL and PCTL on labeled Markov chains. In *Proc. PROCOMET'98*, IFIP. Chapman & Hall, 1998. Also available as Technical Report CIS-97-16.

[15] B. Jonsson, K.G. Larsen: Specification and Refinement of Probabilistic Processes, *Proc. LICS'91*, pages 266–277, IEEE Computer Society Press, 1991.

[16] H. Karloff. *Linear Programming.* Birkhauser, 1991.

[17] U. Klehmet, V. Mertsiotakis: TIPPtool: timed processes and performability evaluation, Technical Report IMMD VII-3/98, University of Erlangen Nurnberg, 1998.

[18] A. McIver and C. Morgan. A probabilistic temporal calculus based on expectations. In *Proc. Formal Methods Pacific'97.* Springer-Verlag, 1997. Also available as Technical Report PRG-TR-13-97.

[19] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.

[20] A. Schrijver. Theory of linear and integer programming. J.Wiley & Sons, 1987.

[21] R. Segala. Modelling and verification of randomized distributed real-time systems. Ph.D. Thesis, Department of Mathematics, Massachusetts Institute of Technology, 1995.

[22] R. Segala, N. Lynch: Probabilistic simulations for probabilistic processes. in *Proc. CONCUR'94*, volume 836 of Lecture Notes in Computer Science 836, pages 481–496, 1994.

[23] K. Seidel, A. McIver and C. Morgan. An introduction to probabilistic predicate transformers. Technical Report PRG-TR-6-96, Oxford Computing Labratory, 1996.

[24] M. Vardi. Automatic verification of probabilistic concurrent finite-state Programs. Proc. In *Proc. FOCS'85*, pages 327–338, 1985.

[25] M. Vardi, P. Wolper. An automata-theoretic approach to automatic program verification, In *Proc. LICS'86*, pages 332–344, 1986.