Complexity of Model Checking MDPs against LTL Specifications

Dileep Kini[*1] **and Mahesh Viswanathan**[†2]

1 **Akuna Capital LLC, Chicago, USA**
dileeprkini@gmail.com
2 **University of Illinois, Urbana-Champaign, USA**
vmahesh@illinois.edu

Abstract

Given a Markov Decision Process (MDP) $\mathcal{M}$, an LTL formula φ, and a threshold $\theta \in [0,1]$, the verification question is to determine if there is a scheduler with respect to which the executions of $\mathcal{M}$ satisfying φ have probability greater than (or $\geq$) θ. When $\theta = 0$, we call it the qualitative verification problem, and when $\theta \in (0,1]$, we call it the quantitative verification problem. In this paper we study the precise complexity of these problems when the specification is constrained to be in different fragments of LTL.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Markov Decision Processes, Linear Temporal Logic, model checking, complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.35

1 Introduction

Systems exhibiting both non-deterministic and probabilistic behaviors are semantically interpreted using Markov Decision Processes (MDPs) [9, 11, 4]. Markov Decision Processes are interpreted with respect to a scheduler who resolves the non-determinism at each step – a single step of the MDP has two phases, where the scheduler first picks a probabilistic transition out of the current state based on the sequence of states visited in the computation, and then a dice is rolled to stochastically choose the next state according to the transition chosen by the scheduler. The verification problem for MDPs with respect to specifications in LTL is as follows. Given an MDP $\mathcal{M}$, a formula φ, a threshold $\theta \in [0,1]$, determine if there is a scheduler with respect to which the measure of executions satisfying φ is greater than (or greater than or equal to) the threshold θ. A special case of this problem is when $\theta = 0$ which is called the *qualitative verification problem*. When $\theta \neq 0$, this is called the *quantitative verification problem*.

The standard approach to solving the verification problem is using the *automata theoretic method* [11, 4]. Here one translates the specification φ into a *deterministic* automaton $\mathcal{A}$, takes the cross product of $\mathcal{A}$ with the MDP $\mathcal{M}$ to construct a new MDP $\mathcal{M}'$, and then analyzes $\mathcal{M}'$ to check the desired property. The complexity of this procedure is polynomial in the size of the final MDP $\mathcal{M}'$. Since any LTL formula can be translated into a deterministic automaton of doubly exponential size, this approach shows that the verification problem

* This work was partly carried out while Dileep Kini was at the University of Illinois, Urbana-Champaign. Dileep Kini was partly supported by NSF award CNS-1314485.
† Mahesh Viswanathan was partly supported by NSF awards CNS-1329991 and CCF-1422798.

Table 1 Summary of results for quantitative and qualitative model checking of MDPs against various fragments. The upper bounds for results marked a follow from the standard translation of LTL to deterministic automata. Upper bounds for results marked b follow from the translation of LTL to limit deterministic automata in [6]. Finally, upper bounds for the result marked c follow from the translation of this fragment to deterministic automata presented in [1].

	Quantitative	Qualitative
$\mathcal{B}(L_{\Diamond,\wedge})$	**PSPACE**-complete	**NP**-complete
$\mathcal{B}(L_{\Diamond,\wedge,\vee})$	**EXPSPACE**-complete	
$\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$	**2EXPTIME**-complete^a	
$\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$	**EXPTIME**-complete^c	**EXPTIME**-complete^b
$\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee})$	**EXPSPACE**-complete	
$\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$	**2EXPTIME**-complete^a	

for MDPs is in **2EXPTIME** [11, 4]. One can prove a matching lower bound [4] which establishes the problem to be **2EXPTIME**-complete.

In a series of recent papers [5, 10, 6], the qualitative verification problem for MDPs has been investigated carefully. In particular, it has been shown that for an expressive fragment of LTL called LTL_D, the qualitative model checking problem is in **EXPTIME** (as opposed **2EXPTIME**). The basis of this result is an improved translation from LTL to a special class of automata called *limit deterministic automata* which are then used in the automata theoretic approach to verify the MDP. It is shown in [6], that the translation yields exponential sized automata for the fragment LTL_D which gives the improved upper bound.

In this paper, we continue this line of research to obtain a more complete picture about quantitative and qualitative verification of MDPs against fragments of LTL. In this endeavour, we are also inspired by [1, 7, 2] that characterize the complexity of solving 2-player games against objectives described using fragments of LTL. Consider LTL to be formulae in negation normal form built using Boolean operations $\vee, \wedge$ and temporal operators $\bigcirc$ (next), $\Diamond$ (eventually), $\Box$ (always), and $\mathcal{U}$ (until). Taking $L_{op_1,\ldots,op_k}$ to denote the LTL fragment consisting of formulae built only using the operators $op_1, \ldots, op_k$, and $\mathcal{B}(L_{op_1,\ldots,op_k})$ to be all boolean combinations of formulae in $L_{op_1,\ldots,op_k}$, our results are summarized in Table 1.

We begin by discussing our results for quantitative model checking. The upper bounds that pertain to time complexity classes (namely those marked a or c in Table 1) are obtained simply from the fact that these fragments can be translated into deterministic automata of exponential (for the result marked c) or doubly-exponential (for results marked a) size. For the other upper bounds in Table 1, we present a new space efficient algorithm to compute the probability of repeatedly visiting a set of states in Markov chains of *small diameter*; this result mimics a similar result in [1] for solving games on graphs with small diameter. The upper bounds are then obtained by translating the LTL fragment into deterministic Büchi automata of small diameter (using observations in [1]), taking the cross product with the MDP, guessing an optimal scheduler, and computing the probability of repeated reachability in the resulting Markov chain using the space efficient algorithm. The lower bounds are obtained by observing that the reductions in [1, 2] work in this case, when the universal player is replaced by a stochastic player. It is worth noting that our lower bounds apply to the special case when the threshold $\theta = 1$; thus, the difficulty in solving the quantitative verification problem does not stem from the numbers involved.

For qualitative verification, the upper bound of **EXPTIME** (marked b) follows from the results in [6]. The improved upper bound of **NP** for the fragment $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ is obtained

by refining the translation given in [5]. The new modified translation constructs a limit deterministic automata that is a disjoint union of exponentially many, polynomial sized deterministic automata. The **NP** algorithm then guesses one of these disjoint automata and analyzes the MDP relative to the guessed deterministic automaton.

2 Preliminaries

2.1 Strings and Prefixes

Given a set S, we use S^* denote the set of all finite sequences (finite words) of elements from S, and S^+ to denote all non-empty finite sequences over S. The length of a finite word u is denoted by $|u|$. We use S^ω to denote all infinite sequences over S. Given a (finite or infinite) word, we use u_i to denote i^{th} symbol in the sequence u (we assume indices start at 0), $u_{[0,i]}$ to denote the prefix $u_0u_1 \dots u_{i-1}$ of length i, and $u_{[i,\infty]}$ to denote the suffix $u_iu_{i+1} \dots$ starting at index i. For $u \in S^+$, we use $\langle u \rangle$ to denote the last element in the sequence u. Given an infinite word u, we use $\mathsf{inf}(u)$ to denote elements of S that appear infinitely often in u. We use $S^!$ to denote words in S^+ with distinct elements. The binary relations $<, \leq$ on S^* denote the prefix relations: $u < v$ iff u is a proper prefix of v. We have $u \leq v$ iff $u < v$ or $u = v$. We use $\lessdot$ to denote the covering relation of prefixes, i.e., $u \lessdot v$ iff $v = ua$ for some $a \in S$. A set $U \subseteq S^+$ is said to be closed under prefixes iff every non-empty prefix of a word in U is also in U.

▶ **Definition 1.** A *prefix tree* on a set S is a pair (V, r), such that $V \subseteq S^+$, the set of vertices, is closed under prefixes, and r is the unique element in V of length 1. A vertex $v \in V$ is called a *leaf* if there is no $u \in V$ for which $v < u$. The set of all leaf vertices of V is denoted by $\mathsf{Leaf}(V)$ and the set of all non-leaf vertices are called inner vertices, denoted by $\mathsf{Inner}(V)$. A prefix tree is infinite if V is infinite.

2.2 Linear Temporal Logic

We recall definitions related to LTL and its fragments. Let AP be a set of atomic propositions, and let Π denote state predicates which represent boolean formulae over AP. LTL formulae are constructed using state predicates from Π, boolean connectives *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), temporal connectives *always* ($\Box$), *eventually* ($\Diamond$), *until* ($\mathcal{U}$) and *next* ($\bigcirc$).

▶ **Definition 2** (LTL Syntax)**.** Formulae in LTL are given by the following syntax:

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\,\mathcal{U}\,\varphi \qquad\qquad p \in \Pi$$

We consider the usual semantics for LTL as given by [8]. In this paper we will be interested in different fragments of LTL. We denote by $L_{op_1,\dots,op_k}$ the set of formulae built using (only) the operators $op_1, \dots, op_k$. For a collection of formulae Γ, we use $\mathcal{B}(\Gamma)$ to denote all possible boolean combinations of formulae in Γ.

2.3 Markov Chains

A Markov chain M is a tuple (Q, δ, q_0) where Q is the set of states, $q_0 \in Q$ is the initial state and $\delta : Q \times Q \to [0, 1]$ is the probabilistic transition function where $\sum_{s' \in Q} \delta(s, s') = 1$ for all $s \in Q$. A labeling of a Markov chain is a function $L : Q \to 2^{AP}$ that maps each state to an assignment over the propositions AP.

A Markov chain (Q, δ, q_0) induces an underlying graph where each state in Q is a vertex and there is an edge from s to s' iff $\delta(s, s') > 0$. A (finite/infinite) path in the Markov chain is an finite/infinite path $\pi = q_0 q_1 q_2 \dots$ in the underlying graph starting at q_0. For such a path π, its trace under labeling L is defined as the sequence of assignments $tr(\pi) = L(q_0)L(q_1)L(q_2)\cdots$. Let $\mathsf{Paths}(M)$ denote the set of all infinite paths and $\mathsf{Paths_f}(M)$ the set of all finite paths in M starting from q_0. A Markov chain M induces a probability distribution $\Pr_M$ on the infinite paths of the Markov chain. We refer the reader to [3] for detailed definitions. For an LTL formula φ over propositions AP and labeling $L : Q \to 2^{AP}$, the set of paths of M that yield a trace in $[\![\varphi]\!]$ is measurable, i.e., the quantity $\Pr_M(\{\pi \in Paths(M) \mid tr(\pi) \in [\![\varphi]\!]\})$ is well defined. We abuse notation and simply write the above quantity as $\Pr_M([\![\varphi]\!])$.

A temporal property of particular interest is what is called *repeated reachability*. For a set of states $B \subseteq Q$ we use the LTL-like notation $\Box\Diamond B$ to denote paths that visit some state in B infinitely often, i.e., $\{\pi \in Paths(M) \mid \mathsf{inf}(\pi) \cap B \neq \emptyset\}$. The computation of $\Pr_M(\Box\Diamond B)$ requires familiarity with the structure of the underlying graph of M. A set of vertices of a directed graph are called strongly connected if every pair of vertices have paths to each other. A Strongly Connected Component (SCC) is a set of vertices S that is maximally strongly connected, i.e., no superset of S is strongly connected. The SCCs of a graph induces a directed acyclic graph where the vertices are the SCCs and there is an edge from one SCC to another if there is an edge going from a vertex in the first to a vertex in the second. A SCC is called bottom (BSCC) if there is no other SCC that can be reached from it. It is well known (see Chapter 10 of [3]) that a (infinite) path of a Markov chain almost certainly (i.e. with probability 1) ends up in one of the BSCCs and visits each of the vertices in that BSCC infinitely often. In order to compute $\Pr_M(\Box\Diamond B)$ it suffices to compute the probability of reaching BSCCs that have at least one state from B. We will build upon these ideas in our proofs.

2.4 Markov Decision Processes

A Markov decision process (MDP) $\mathcal{M}$ is a tuple (Q, Act, Δ, q_0) where Q is the set of states, Act is the set of actions, and q_0 is the initial state and $\Delta : Q \times Act \times Q \to [0, 1]$ is the probabilistic transition function where $\sum_{q' \in Q} \Delta(q, a, q') = 1$ for every $q \in Q$ and $a \in Act$. A labeling of a MDP is a function $L : Q \to 2^{AP}$ that maps each state to an assignment over the propositions AP.

A MDP executes as follows: it begins at state q_0 and non-deterministically picks an action $a_0 \in Act$. This is followed by stochastically choosing a state q_1 with probability $\Delta(q_0, a_0, q_1)$. This process is now continued with q_1 which gives us an infinite run of the form $q_0 a_0 q_1 a_1 q_2 \dots$. Note that an MDP includes non-determinism in the form of the action to be picked which is absent in Markov chains. Markov chains are special cases of MDPs, which are devoid of non-determinism. In order to define the probability measure in an MDP, one requires a scheduler (or adversary) that resolves this non-determinism. A scheduler $\mathfrak{S} : Q^+ \to Dist(Act)$ is a function that maps a sequence of states (states visited until a certain point) to a distribution on actions. The action is picked stochastically according to the distribution. Pure strategies $\mathfrak{S} : Q^+ \to Act$ are those where the distribution corresponds to picking a single action with probability 1. For the problems studied here pure schedulers suffice and therefore we restrict our attention to them. A scheduler $\mathfrak{S}$ *induces* a Markov chain $\mathcal{M}_{\mathfrak{S}} = (Q^+, \delta, q_0)$ where $\delta(u, v) = \Delta(\langle u \rangle, \mathfrak{S}(u), \langle v \rangle)$ if $u \lessdot v$ and 0 otherwise. The Markov chain $\mathcal{M}_{\mathfrak{S}}$ is then used to define the measure on sets of paths of $\mathcal{M}$. The probability measure of an event E under scheduler $\mathfrak{S}$ for MDP $\mathcal{M}$, denoted by $\Pr^{\mathfrak{S}}_{\mathcal{M}}(E)$ is defined as the measure $\Pr_{\mathcal{M}_{\mathfrak{S}}}(E)$ associated with event E in Markov chain $\mathcal{M}_{\mathfrak{S}}$. A labeling $L : Q \to 2^{AP}$

for $\mathcal{M}$ can be extended to a labeling $L' : Q^+ \to 2^{AP}$ for $\mathcal{M}_{\mathfrak{S}}$ where $L'(u) = L(\langle u \rangle)$, which can then be used to define $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\varphi]\!])$ for LTL formula φ over propositions AP.

▶ **Definition 3.** The *quantitative* model checking problem for LTL is to decide if there exists a scheduler $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\varphi]\!]) \geq \theta$ given MDP $\mathcal{M}$, $\varphi \in$ LTL, and $\theta \in [0, 1]$ as inputs.

Two schedulers $\mathfrak{S}_1, \mathfrak{S}_2$ for MDP $\mathcal{M}$ are said to be equivalent, denoted by $\mathfrak{S}_1 \sim_{\mathcal{M}} \mathfrak{S}_2$, when $\mathsf{Paths}_{\mathsf{f}}(\mathcal{M}_{\mathfrak{S}_1}) = \mathsf{Paths}_{\mathsf{f}}(\mathcal{M}_{\mathfrak{S}_2})$ and $\mathfrak{S}_1(u) = \mathfrak{S}_2(u)$ for every $u \in \mathsf{Paths}_{\mathsf{f}}(\mathcal{M}_{\mathfrak{S}_1})$. Equivalent schedulers yield Markov chains whose reachable portions are isomorphic. All equivalent schedulers for a MDP can be viewed as tree which is obtained from "unfolding" the MDP under the scheduler. We define prefix trees associated with an MDP and then see how they are related to schedulers.

▶ **Definition 4.** Given a MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$, a *$\mathcal{M}$-labeled prefix tree* (V, q_0, λ), is one where (V, q_0) is a prefix tree on Q, and $\lambda : \mathsf{Inner}(V) \to Act$ is a labeling such that $\forall u \in \mathsf{Inner}(V), q \in Q : uq \in V$ iff $\Delta(\langle u \rangle, \lambda(u), q) > 0$.

Let $\mathcal{S}$ denote the set of all schedulers, and $\mathcal{S}/\!\sim_{\mathcal{M}}$ denote the equivalence classes induced by the $\sim_{\mathcal{M}}$ relation. The proposition below captures the observation that equivalent schedulers of $\mathcal{M}$ can be identified by their the infinite $\mathcal{M}$-labeled prefix tree obtained by unfolding $\mathcal{M}$ on those schedulers.

▶ **Proposition 5.** *Given MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$ there is a one to one correspondence between $\mathcal{S}/\!\sim_{\mathcal{M}}$ and infinite $\mathcal{M}$-labeled prefix trees (V, q_0, λ).*

3 Quantitative Model Checking

3.1 Upper Bounds

We will present upper bounds on the complexity of quantitative verification of MDPs against formulae from different LTL fragments. In the automata theoretic approach, if the LTL specification can be translated to a deterministic Büchi automaton, then the complexity is intrinsically tied to solving the problem of computing the optimal probability for repeatedly reaching a set of states. One of the contributions of this paper is a new space efficient algorithm for the repeated reachability problem in MDPs which is presented in Section 3.1.1. Before presenting this algorithm and using it to get the results in Table 1, we need to introduce two special classes of schedulers – *memoryless* and *depth-bounded* schedulers – and some simple observations about them.

▶ **Definition 6.** A memoryless scheduler $\mathfrak{S}$ is one where $\mathfrak{S}(u) = \mathfrak{S}(v)$ if $\langle u \rangle = \langle v \rangle$.

A memoryless scheduler uses only knowledge about the latest state to decide which action it is going to pick. For a finite execution u, $\langle u \rangle$ represents the latest state and hence the action $\mathfrak{S}(u)$ is only dependent on $\langle u \rangle$.

Next we define *depth-bounded schedulers* that generalize memoryless schedulers. Depth-bounded schedulers can make decisions based on the current history. However, they only consider the portion of history from which "loops" have been removed. For example, consider a history $u = q_1, q_2, q_3, q_4, q_2, q_5$. The sequence q_2, q_3, q_4, q_2 is a loop, and removing it from u gives the history $v = q_1, q_2, q_5$. The action chosen by a depth-bounded scheduler on history u is the same as the one chosen on history v. This is formally defined next.

▶ **Definition 7.** A depth-bounded scheduler $\mathfrak{S} : Q^+ \to Act$ is one such that

$$\forall v \in Q^*, u \in Q^!, m \in \{1, \ldots, |u|\} : \mathfrak{S}(uu_m v) = \mathfrak{S}(u_{[0,m]} v).$$

A depth-bounded scheduler removes loops from the current history as soon as they form, and makes its decision based on the truncated history. Note that the process of loop removal leaves the last state in the history unchanged. From this it follows that a memoryless scheduler is a special case of the depth-bounded scheduler where the decision depends only on $\langle u \rangle$.

▶ **Proposition 8.** *Every memoryless scheduler is a depth-bounded scheduler.*

Next, we define special kinds of prefix trees that correspond to depth-bounded schedulers. Let $\mathcal{D}$ denote all depth-bounded schedulers.

▶ **Definition 9.** A prefix tree (V, r) on S is called *depth-bounded* if V is finite, $\mathsf{Inner}(V) \subseteq S^!$ and $\mathsf{Leaf}(V) \cap S^! = \emptyset$.

Next, analogous to Proposition 5, we observe that there is a 1-to-1 onto correspondence between equivalence classes of depth-bounded scheduler for $\mathcal{M}$ and $\mathcal{M}$-labeled depth-bounded prefix trees. Let $\mathcal{D}$ denote all depth-bounded schedulers, and $\mathcal{D}/\!\sim_{\mathcal{M}}$ denote the equivalence classes induced by the $\sim_{\mathcal{M}}$ relation.

▶ **Proposition 10.** *Given MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$ there is a one to one correspondence between $\mathcal{D}/\!\sim_{\mathcal{M}}$ and $\mathcal{M}$-labeled depth-bounded prefix trees.*

3.1.1 Space efficient algorithm for repeated reachability

In the section, we present one of our core technical results, that gives a space-bounded algorithm for solving the quantitative repeated reachability problem for MDPs. The salient feature of the algorithm is that its space requirements are polynomial in the *diameter* of the underlying graph of the MDP and logarithmic in its size; here, by diameter we refer to the length of the longest simple path in the graph. Thus, this algorithm is space efficient for MDPs whose diameter is small when compared to its size.

▶ **Theorem 11.** *Consider MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$ with diameter d, graph size n, and for any $q, q' \in Q$ and $a \in Act$, $\Delta(q, a, q')$ is a rational number of size at most k. Given a set of states $B \subseteq Q$, the problem of deciding if $\exists \mathfrak{S} : \Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B) \geq \theta$ can be solved in non-deterministic space $O(d^2 \cdot (\log(n) + k))$.*

The proof of the theorem relies on an algorithm that guesses a scheduler $\mathfrak{S}$, computes $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B)$ and compares it to θ. Recall that memoryless schedulers suffice for attaining the maximum probability of repeatedly reaching a set of states. Guessing a memoryless scheduler requires n bits of space, which does not meet our space requirements. But we know every memoryless scheduler is also a depth-bounded scheduler (Proposition 8), so it is suffices to look for a depth-bounded scheduler $\mathfrak{S}$. We use Proposition 10 to guess the $\mathcal{M}$-labeled depth-bounded prefix tree $T(\mathfrak{S}) = (V, q_0, \lambda)$. Storing the entire tree would use too much space; instead we guess the tree in a path-by-path manner using a depth first strategy (DFS). In this approach, at any given time, we only store a single path in the tree $T(\mathfrak{S})$. Observe that any single path in the depth-bounded tree has to be a path in $\mathcal{M}$ and hence bounded by the diameter d. Therefore storing a path and its labels requires only $d \cdot \log(n)$ bits of space. To complete the proof of Theorem 11, we need to describe how to compute the probability $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B)$ as we guess and explore the tree. The Markov chain $\mathcal{M}_{\mathfrak{S}}$ induced by a depth-bounded scheduler $\mathfrak{S}$ can be shown to be (probabilistic) bisimulation [3] equivalent to Markov chain $\mathcal{M}_{T(\mathfrak{S})}$, which is obtained from $T(\mathfrak{S})$ as follows: $\mathcal{M}_{T(\mathfrak{S})} = (\mathsf{Inner}(V), \delta, q_0)$

where

$$\delta(u,v) \stackrel{\text{def}}{=} \begin{cases} \Delta(\langle u\rangle, \lambda(u), \langle v\rangle) & \text{if } (u \lessdot v) \text{ OR } (v \leq u \text{ and } u\langle v\rangle \in \mathsf{Leaf}(V)) \\ 0 & \text{otherwise.} \end{cases}$$

We classify the edges of the above Markov chain as: (i) *forward edges* (u,v) where $u \lessdot v$, (ii) *back edges* (u,v) where $v \leq u$.

The probability of repeatedly reaching B in $\mathcal{M}_{\mathfrak{S}}$ equals the probability of repeatedly reaching B' in $\mathcal{M}_{T(\mathfrak{S})}$ where $B' = \{v \in \mathsf{Inner}(V) \mid \langle v\rangle \in B\}$. Computing repeated reachability in Markov chains boils down to computing probability of reaching BSCCs that contain at least one of the states in B'. In the remainder of this section we see how to do this during the DFS that explores the tree $T(\mathfrak{S})$.

Index Vertex of a SCC. For a SCC of $\mathcal{M}_{T(\mathfrak{S})}$ define an *index* vertex w as one for which there is no other vertex w' in the SCC such that $w' < w$. The first observation is that there is a unique index vertex for a SCC. For contradiction assume there are two indices $u \neq v$ for a SCC. By definition of SCC, there is a simple path from u to v. If this path does not contain any back edges then clearly $u < v$ contradicting the fact that v is an index. If the path contains back edges, consider the first back edge in the path, say (u', v'). Now v' cannot be on the path from u to u', due to the fact that nodes cannot be repeated on a simple path (otherwise v' would be visited twice from u to v). Now, $v' < u'$ (because back edges lead to a prefix/ancestor), and $u \not< v'$. This means $v' < u$ because two ancestors of any node in a tree are always directly related. Since there is a path from u to v' (as observed), and a path from v' to u owing to the fact that $v' < u$, we get that v' is included in the SCC. This contradicts u being an index. This proves the uniqueness of an index node. In our algorithm we guess which nodes in the tree are index nodes and compute the probability of reaching every index node. The probability of reaching a BSCC is simply the probability of reaching the index vertex of that BSCC.

Parent-Child relationship between index vertices. In order to compute the probability of reaching an index node in an inductive fashion, we identify the parent-child relationship between index vertices. An index node v is called the *child* of an index node u if $u < v$ and there is no index node w such that $u < w < v$. Similarly u is called the *parent* of v, if v is the child of u. Note that every index has a unique parent except for the root which has no parent. For an index node u let $C(u)$ denote all the children index nodes of u. Given a node u, let p_u denote the probability of reaching u. Given a node uv with $u, v \neq \epsilon$, let $q_{u,v}$ denote the probability of moving from u to uv along the unique path of forward edges from u to uv in $\mathcal{M}_{T(\mathfrak{S})}$. $q_{u,v}$ is given by

$$q_{u,v} \stackrel{\text{def}}{=} \prod_{i=0}^{|v|-1} \delta(u.v_{[0,i]}, u.v_{[0,i+1]}) \tag{1}$$

The Proposition below formulates how we can calculate the probability of reaching an index by using the reachability probability of its parent.

▶ **Proposition 12.** *For a node $uv \in C(u)$, the probability p_{uv} of reaching uv is given by $(p_u \cdot q_{u,v})/s_u$, where $s_u \stackrel{\text{def}}{=} \sum_{w \in C(u)} q_{u,w}$ is called the normalizing factor of u.*

In order to use the above formula for the computation of p_{uv} we need to know the normalizing factor s_u of the parent node u. We guess this quantity s_u associated with each

node u that is an index, and store it along with u on the depth-first stack. Now, let us see how these can be used to compute reachability probabilities. For an index node uv whose parent is u, assume p_u is already computed and stored. The parent of a node can be identified by looking at the latest node before u in the stack that is an index node. For the root node r, $p_r = 1$. Now p_{uv} can be computed according to Proposition 12 using:

- p_u which is already computed and stored on the stack when u was first encountered
- s_u which is guessed and stored on the stack when u was first encountered
- $q_{u,v}$ which can be computed by looking at the path from u to uv on the depth-first stack.

Next, to compute the probabilities of reaching the BSCCs that have a state from B' in them, we observe that an index node u corresponds to a BSCC iff the normalizing factor $s_u = 0$, implying that it has no children. For such a state u we mark it as *final* as soon as a descendant uv is encountered where $\langle uv \rangle \in B$. Once all the descendants of u are explored we check if it is final, and if so we add the probability of reaching it, p_u, to a running total. The total value at the end of the DFS exploration is the required probability $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B)$.

Confirming guesses. In the computation described above we have guessed two things for every node u: (a) if u is an index or not; (b) the normalizing factor s_u, whenever u is an index. In order to check that our guess regarding u being an index is correct we use the following:

▶ **Proposition 13.** *For $T(\mathfrak{S}) = (V, q_0, \lambda)$, a node $u \in \mathsf{Inner}(V)$ is an index node of some SCC in $\mathcal{M}_{T(\mathfrak{S})}$ iff every $uv \in \mathsf{Leaf}(V)$ is such that $\langle uv \rangle = \langle uv' \rangle$ for some $uv' \in \mathsf{Inner}(V)$.*

So, when u is guessed as an index node we make sure that every leaf descendant of u points to a repetition of a state that is no earlier than u, and when u is guessed as non-index we ensure there is a leaf descendant of u pointing to a repetition of a state earlier than u. In order to check that the guess for s_u is correct, we maintain a running sum for each index node u on the stack. When a $uv \in C(u)$ is encountered we add the computed quantity $q_{u,v}$ to the running sum associated with u. When the DFS exploration for u is complete we check that the running sum equals the guess s_u.

Size of numerical quantities. So far we have not accounted for the space requirements of the quantities we calculate. Let us begin by looking at $q_{u,v}$ for parent-child indices u, v. Equation 1 tells us that $q_{u,v}$ is a product of transition probabilities from u to v of which there are at most d. Therefore $q_{u,v}$ requires $d{\cdot}k$ bits to store since each transition probability has no more than k bits. Next, the normalizing factor s_u for an index u is the sum of $q_{u,v}$ where uv is a child of u. Note that the number of children for any index is bounded by the total number of nodes in the tree which is at most n^d. So each s_u, the sum of n^d quantities $(q_{u,v})$ each of size $d{\cdot}k$ requires only $d{\cdot}(\log(n) + k)$ bits. By Proposition 12, p_u is $p_{u'} \cdot q_{u',v'}/s_{u'}$, where u' is the parent of u and $u = u'v'$. By induction on the number of ancestors of u, we can argue that p_u has at most $O(d^2 \cdot (\log(n) + k))$ bits, since the maximum number of ancestors for any index node is d. The sum of p_u for u that are final BSCCs of which there are no more than n^d, will increase the bits required by $d{\cdot}\log(n)$. So the total space required remains $O(d^2 \cdot (\log(n) + k))$.

3.1.2 Upper Bounds for LTL-fragments

We are now ready to present all our upper bounds for the quantitative verification problem for different LTL fragments. Our results rely on the automata theoretic approach that solves

the quantitative verification problem by constructing a deterministic automaton for the given LTL specification. We, therefore, begin by recalling results on translations of fragments of LTL to deterministic automata.

▶ **Theorem 14** (Alur-LaTorre [1]). *The following fragments of* LTL *can be translated into deterministic Büchi automata with the following space and diameter bounds.*

- $L_{\Diamond,\wedge}$ *has automata of exponential size and linear diameter.*
- $L_{\Diamond,\bigcirc,\wedge}$ *has automata of exponential size and exponential diameter.*
- $L_{\Diamond,\wedge,\vee}$ *has automata of double exponential size and exponential diameter.*
- $L_{\Diamond,\bigcirc,\wedge,\vee}$ *has automata of double exponential size and exponential diameter.*
- $L_{\Diamond,\Box,\wedge,\vee}$ *has automata of double exponential size and double exponential diameter.*

These bounds on size and diameter are also tight.

We now present our upper bound results for quantitative verification shown in Table 1.

▶ **Theorem 15.** *The quantitative verification problem for MDPs against* LTL *specifications has the following complexity bounds – for* $\mathcal{B}(L_{\Diamond,\wedge})$ *it is in* **PSPACE***; for* $\mathcal{B}(L_{\Diamond,\wedge,\vee})$ *it is in* **EXPSPACE***; for* $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ *it is in* **2EXPTIME***; for* $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ *it is in* **EXPTIME***; for* $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee})$ *it is in* **EXPSPACE***; for* $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ *it is in* **2EXPTIME**.

Proof Sketch. Recall that in the automata theoretic approach to quantitative verification of MDPs, the LTL specification φ is translated into a deterministic automaton $\mathcal{A}$, and then the cross product of $\mathcal{A}$ with the MDP $\mathcal{M}$ is analyzed. When the automaton $\mathcal{A}$ is Büchi , the analysis involves solving the repeated reachability problem on the cross product MDP. The algorithm of [11, 4] runs in time that is polynomial in the size of the cross product. Given the results on the size of deterministic Büchi automata mentioned in Theorem 14, we immediately get the complexity bounds for $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$, $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$, and $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$.

For the other upper bounds, we follow a similar approach, but we construct the product of $\mathcal{M}$ and $\mathcal{A}$ on the fly. We exploit the fact that the Büchi automata constructions for LTL formulae have a representation that allows one to guess its states and check the transition relation just from knowing the formula. This allows us to apply Theorem 11 to the implicit product whose diameter is the product of the diameters of $\mathcal{M}$ and $\mathcal{A}$. Given the bounds on the diameter of the deterministic Büchi automata mentioned in Theorem 14, and using Theorem 11, we obtain the complexity bounds for the remaining fragments. ◀

3.2 Lower Bounds

In this section we prove matching lower bounds for the upper bounds established in Theorem 15. The lower bounds essentially follow from lower bounds established in [1, 2] for 2-player games. The reason for this observation is that games constructed in the lower bound reductions in [1, 2] have a special property that enable their lifting to the quantitative verification problem for MDPs. We begin this section by identifying this property, and showing how it helps transfer complexity bounds to the quantitative verification case.

Recall that a two player game is played on a graph $G = (V, E)$, where the set of vertices V is partitioned into two sets – $V_\exists$ which belong to $\exists$-player, and $V_\forall$ which belong to $\forall$-player. At any given time, the play is at some vertex u of graph G. Player P ($P \in \{\exists, \forall\}$) plays from u if $u \in V_P$, by picking the target of some outgoing edge from u. Starting from an initial vertex u_0, a *play* is the infinite sequence of vertices visited as the players choose edges on their turn. Given an objective described by LTL formula φ, we say a play π is winning for $\exists$-player if π satisfies φ; otherwise the play is said to be winning for the $\forall$-player. We now identify a special class of games that we call *finitely winnable*.

▶ **Definition 16.** A game (G, u_0, φ) is said to be *finitely-winnable* for a player P ($P \in \{\exists, \forall\}$) iff for any play π of (G, φ) that P wins, there is a prefix of π, say π', such that every play (according to game graph G) that is an extension of π' is also winning for P.

The main observation about games that are finitely winnable for the $\forall$-player is that if the $\forall$-player is replaced by a stochastic player that uniformly chooses among the available choices, then in the resulting MDP, there is a scheduler that meets objective φ with probability 1 if and only if the $\exists$-player has a winning strategy in the game.

▶ **Proposition 17.** *Given a game (G, u_0, φ) which is finitely-winnable for the $\forall$-player, the MDP $\mathcal{M}_G$ obtained by replacing the $\forall$-player with stochastic choices is such that the $\exists$-player has a winning strategy for (G, u_0, φ) if and only if there exists a scheduler $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) = 1$.*

Proof. If the $\exists$-player has a winning strategy for (G, u_0, φ), then the strategy interpreted as scheduler for $\mathcal{M}_G$ is going to be such that all runs of that scheduler are going to satisfy φ, which implies $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) = 1$. Now consider the case the where the $\forall$-player has a winning strategy for (G, u_0, φ). Here, for any strategy for the $\exists$-player, there is going to be a play that is won by the $\forall$-player. Since the game is finitely-winnable for the $\forall$-player, we know there is a prefix of the play whose every extension is winning for the $\forall$-player. What this means in the MDP setting, is that for any strategy there is a finite run ρ whose every extension results in φ not being met. Since the measure associated with all extensions of ρ is non-zero (since ρ is finite), we get that any strategy loses with non-zero probability, i.e., $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) < 1$ for any scheduler $\mathfrak{S}$. ◀

We use the above observations to obtain matching lower bounds for the quantitative verification problem.

▶ **Theorem 18.** *The quantitative verification problem for MDPs against* LTL *specification has the following complexity lower bounds – for $\mathcal{B}(L_{\Diamond,\wedge})$ it is* **PSPACE***-hard; for $\mathcal{B}(L_{\Diamond,\wedge,\vee})$ it is* **EXPSPACE***-hard; for $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ it is* **2EXPTIME***-hard; for $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ it is* **EXPTIME***-hard; for $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee})$ it is* **EXPSPACE***-hard; for $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ it is* **2EXPTIME***-hard.*

Proof Sketch. All the lower bounds follow from similar lower bounds established for solving 2-player games for the same LTL fragments in [1, 2]. All reductions for games essentially reduce the membership problem of a space/time bounded Alternating Turing machine (ATM) – given an ATM $\mathcal{A}$ and an input w they construct a game graph G, initial state u_0, and a specification φ such that $w \in L(\mathcal{A})$ iff there exists a winning strategy for the $\exists$-player in the game (G, u_0, φ). In each of these reductions, the game (G, u_0, φ) is finitely winnable for the $\forall$-player. Thus, we can use the same reduction and Proposition 17 to obtain a lower bound for the quantitative verification problem for MDPs when the threshold is $\theta = 1$. ◀

4 Qualitative Model Checking

The qualitative model checking problem is the following: given MDP $\mathcal{M}$ and LTL formula φ, check if there exists a scheduler $\mathfrak{S}$ under which the probability of paths satisfying φ is non-zero. In this section, we present results that refine our understanding of the complexity of qualitative verification for LTL fragments.

4.1 Upper Bounds

The qualitative problem for MDPs against LTL can be solved using an automata-theoretic approach described by [11, 4] in which the LTL formula is translated into a *limit-deterministic* automata. An automaton is said to be limit-deterministic (or deterministic in the limit) if every state reachable from a final state is deterministic. The result proved by [4] is as follows:

▶ **Proposition 19.** *Given an MDP $\mathcal{M}$ and a limit-deterministic Büchi automaton $\mathcal{A}$, the problem of checking if there exists $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\mathcal{A}]\!]) > 0$, can be solved by taking a cross-product of $\mathcal{M}$ and $\mathcal{A}$ and checking if this product has a reachable BSCC containing a state (s, q) where q is a final state of $\mathcal{A}$.*

Checking for the existence of such a *final* BSCC boils down to analyzing the product graph which runs in linear time. We have recently shown how to transform LTL to limit-deterministic Büchi automata (LDBA) such that the construction is of exponential size for a large class of properties, namely LTL_D [6]. This allows us to prove an **EXPTIME** upper bound for qualitative model checking against that fragment using the result above. In this section we see that the problem is in **NP** for the fragment $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$. The construction in [5] for $\varphi \in \mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ produces an exponential sized LDBA $\mathcal{A}_\varphi$, so it would seem Proposition 19 is not useful for proving our desired upper bound. The key idea we introduce here is the following: the automaton $\mathcal{A}_\varphi$ can be *split* into a disjoint union of exponentially many LDBAs each of which is polynomially large in the size of φ. In Proposition 19, if $\mathcal{A}$ is a disjoint union of multiple LDBAs, then the product of $\mathcal{M}$ and $\mathcal{A}$ has reachable final BSCC if and only if the product of $\mathcal{M}$ and some individual component of $\mathcal{A}$ has a final BSCC. The **NP**-algorithm guesses this individual component of $\mathcal{A}$ (of polynomial size) and analyzes the product graph of $\mathcal{M}$ and the individual component in time polynomial in both $\mathcal{M}$ and φ.

In order to understand the *splitting* of $\mathcal{A}_\varphi$ we recall the core idea behind the construction of $\mathcal{A}_\varphi$ for $\varphi \in \mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$. For each $\Diamond$ or $\Box$ subformula ψ of φ, the automaton keeps track of *how often ψ is true*, which is one of three things: (a) ψ is always true; (b) ψ is true at some point but not always; (c) ψ is never true. This yields a tri-partition, $\pi = \langle\, \alpha \,|\, \beta \,|\, \gamma \,\rangle$, of all the $\Diamond, \Box$ subformulae of φ. If a $\Diamond$ subformula is in α, β or γ we take it to mean that it is never true, true at some point but not always, or always true, respectively. Dually, if $\Box$ subformula is in α, β or γ we take it to mean that it is always true, true at some point but not always, or never true, respectively. With this semantics in mind we see that a subformula in α or γ should remain in α or γ respectively in the future, and a subformula in β can remain in β only for a finite time before moving to α. It turns out that, for a given input word w, correctly guessing (a) the triple π at the beginning of w and (b) the points along w at which subformulae move from β to α, enables us to check if w satisfies the original formula φ. A key observation here is that a triple at a certain point not only tells us how often a $\Diamond, \Box$ subformula is true from that point onwards in the future, but also whether or not the subformula is true at that point. In other words, a triple refines the truth of $\Diamond, \Box$ formulae. This observation is used to inductively check that the guessed triple is correct at every point. We encourage the reader to refer to [5, 6] for a detailed account of the construction. For the purposes of this paper it suffices to know that the state of the automaton for φ is of the form (π, k) where:

- π is a triple reflecting how often the $\Diamond, \Box$ subformulae are true on the remaining input.
- k is an integer counter no larger than $|\varphi|$, which is updated deterministically.

The transitions of the automaton allow moving from a state with $\pi = \langle\, \alpha \,|\, \beta \,|\, \gamma \,\rangle$ to a state with $\pi' = \langle\, \alpha' \,|\, \beta' \,|\, \gamma' \,\rangle$ only if $\alpha \subseteq \alpha'$, $\beta' \subseteq \beta$ and $\gamma = \gamma'$ ($\pi \sqsubseteq \pi'$ for short) in accordance with the semantics we associate with the triple. That is $\pi \sqsubseteq \pi'$ is a necessary condition for

a transition to move from π to π', i.e., subformulae in β are allowed to move α while the remaining stay put. In order to *split* this automaton into smaller components as anticipated earlier, we add restrictions to the order in which the formulae in β are moved to α. First, let us fix an initial triple $\pi = \langle\, \alpha_0 \,|\, \beta_0 \,|\, \gamma_0 \,\rangle$. Given π, consider a ranking function $\rho : \beta_0 \to \mathbb{N}$ whose range is allowed to be any consecutive set of positive integers starting from 1, i.e. $\{1, \ldots, n\}$. Given π and ρ we are going to define a component $\mathcal{A}_{(\pi,\rho)}$ of the original automaton $\mathcal{A}_\varphi$. We define the space of possible triples $\pi_i = \langle\, \alpha_i \,|\, \beta_i \,|\, \gamma_i \,\rangle$ for $i \in \{0, 1, \ldots, n\}$ as follows: $\alpha_i = \{\psi \in \beta_0 \mid f(\psi) \leq i\} \cup \alpha_0$; $\beta_i = \{\psi \in \beta_0 \mid f(\psi) > i\}$; $\gamma_i = \gamma_0$ The states of $\mathcal{A}_{(\pi,\rho)}$ are those states of $\mathcal{A}_\varphi$ where the triple is restricted to be some π_i as defined above. A transition τ, say $(\pi_i, m) \xrightarrow{\sigma} (\pi_j, n)$, is allowed in $\mathcal{A}_{(\pi,\rho)}$ iff τ is a valid transition in $\mathcal{A}_\varphi$ and either $j = i$ or $j = i + 1$. In $\mathcal{A}_{(\pi,\rho)}$ a transition is allowed to either keep the triple unchanged (when $j = i$), or move only the formulae mapped to $i + 1$ from β to α (when $j = i + 1$). Thus the ranking function ρ restricts the order in which the subformulae move from β to α. A subformula with smaller rank is moved earlier compared to one with a larger rank. Note that two or more formulae can be mapped to the same number, which means those formulae are moved simultaneously. The initial state of $\mathcal{A}_{(\pi,\rho)}$ is defined to be $(\pi_0, 0)$ and the state $(\pi_n, 0)$ is marked as the only final state. Note that the size of the automaton $\mathcal{A}_{(\pi,\rho)}$ is $n + |\gamma_0|$ which is linear in $|\varphi|$. The number of different (π, ρ) is exponential in φ, hence there can be exponentially many different individual components.

What remains to be seen is that the disjoint union of these components $\biguplus \mathcal{A}_{(\pi,\rho)}$ accepts exactly the same language as $\mathcal{A}_\varphi$. Since $\mathcal{A}_{(\pi,\rho)}$ is a projection of $\mathcal{A}_\varphi$ it is the case that $[\![\mathcal{A}_{(\pi,\rho)}]\!] \subseteq [\![\mathcal{A}_\varphi]\!]$ and so $[\![\uplus \mathcal{A}_{(\pi,\rho)}]\!] \subseteq [\![\mathcal{A}_\varphi]\!]$. To see the other direction consider any word w accepted by $\mathcal{A}_\varphi$, and let $(\pi_0, k_0), (\pi_1, k_1), \ldots$ be an accepting run for w on $\mathcal{A}_\varphi$ with $\pi_i = \langle\, \alpha_i \,|\, \beta_i \,|\, \gamma_i \,\rangle$. From the construction of $\mathcal{A}_\varphi$ we know that $\pi_0 \sqsubseteq \pi_1 \sqsubseteq \pi_2 \cdots$. Identify all the positions $j_1 < j_2 < \cdots < j_n$ where the triple changes, i.e.,

$$(\pi_0 = \pi_1 \cdots = \pi_{j_1}) \sqsubset (\pi_{j_1+1} = \cdots = \pi_{j_2}) \sqsubset (\pi_{j_2+1} = \cdots = \pi_{j_3}) \sqsubset (\cdots) \sqsubset (\pi_{j_n} = \cdots$$

Here j_i is the i^{th} time the triple changes, n being the last. Now we consider the automaton $\mathcal{A}_{(\pi_0,\rho)}$ where $\rho(\psi) \stackrel{\text{def}}{=} i$ if ψ moves from β to α at position j_i, i.e., $\psi \in \beta_{j_i}$ and $\psi \in \alpha_{j_i+1}$. Observe that the above accepting run is also an accepting run of $\mathcal{A}_{(\pi,\rho)}$ on the word w. This gives us $[\![\mathcal{A}_\varphi]\!] \subseteq [\![\uplus \mathcal{A}_{(\pi,\rho)}]\!]$.

Thus we have successfully split $\mathcal{A}_\varphi$ into exponentially many individual components of linear size. The index (π, ρ) for any component requires only polynomially many bits to represent. This combined with our earlier observation of using Proposition 19 for the disjoint union gives us the **NP**-algorithm for qualitative model checking against $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$. We end this section by summarizing the upper bound results for qualitative model checking.

▶ **Theorem 20.** *The qualitative verification problem for MDPs against specifications in* $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ *is in* **NP** *and against specifications in* $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ *is in* **EXPTIME**.

Proof. The argument for qualitative model checking of $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ being in **NP** has been spelt out above. The qualitative model checking problem of $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ is in **EXPTIME** because $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$-formulae can be translated into exponential sized limit deterministic automata [5, 6] and the observation in Proposition 19. ◀

4.2 Lower Bounds

In this section, we show that the upper bounds proved in Theorem 20 are tight.

▶ **Theorem 21.** *The qualitative verification problem for MDPs against specifications in* $\mathcal{B}(L_{\Diamond,\wedge})$ *is* **NP***-hard and against specifications in* $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ *is in* **EXPTIME***-hard.*

We note that the **EXPTIME**-hardness for the fragment $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ strengthens the result in [5] that establishes the hardness for the larger fragment $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$.

5 Conclusions

In this paper, we presented results for the quantitative and qualitative verification problems for MDPs against fragments of LTL studied in [1, 2]. In doing so we refined the upper and lower bounds for qualitative verification that were obtained in [5, 10, 6].

Acknowledgements. We'd like to thank annonymous referees for their comments which improved the draft.

References

1 R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Logic*, 5(1):1–25, 2004.
2 R. Alur, S. La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *Proceedings of the International Conference on Concurrency Theory*, pages 128–143, 2003.
3 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.
5 Dileep Kini and Mahesh Viswanathan. Limit deterministic and probabilistic automata for LTL \GU. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 628–642, 2015.
6 Dileep Kini and Mahesh Viswanathan. Optimal translation of LTL to limit deterministic automata. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 113–129, 2017.
7 J. Marcinkowski and T. Truderung. Optimal complexity bounds for positive LTL games. In *Proceedings of the International Conference on Computer Science Logic*, pages 262–275, 2002.
8 A. Pnueli. The temporal logic of programs. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.
9 M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.
10 Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínskỳ. Limit-deterministic Büchi automata for linear temporal logic. In *Proceedings of the International Conference on Computer-Aided Verification*, pages 312–332, 2016.
11 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE Computer Society, 1985.