Memoryful Branching-Time Logic

Orna Kupferman
Hebrew University*

Moshe Y. Vardi
Rice University and Microsoft Research†

Abstract

Traditional branching-time logics such as CTL* are memoryless: once a path in the computation tree is quantified at a given node, the computation that led to that node is forgotten. Recent work in planning suggests that CTL* cannot easily express temporal goals that refer to whole computations. Such goals require memoryful quantification of paths. With such a memoryful quantification, $E\psi$ holds at a node s of a computation tree if there is a path π starting at the root of the tree and going through s such that π satisfies the linear-time formula ψ. We define the memoryful branching-time logic mCTL* and study its expressive power and algorithmic properties.

We show that mCTL* is as expressive, but exponentially more succinct, than CTL*, and that the ability of mCTL* to refer to the present is essential for this equivalence. From the algorithmic point of view, while the satisfiability problem for mCTL* is 2EXPTIME-complete — not harder than that of CTL*, its model-checking problem is EXPSPACE-complete — exponentially harder than that of CTL*. The upper bounds are obtained by extending the automata-theoretic approach to handle memoryful quantification, and are much more efficient than these obtained by translating mCTL* to branching logics with past. The EXPSPACE lower bound for the model-checking problem applies already to formulas of restricted form (in particular, to $AGE\psi$, which is useful for specifying possibility properties), and implies that reasoning about a memoryful branching-time logic is harder than reasoning about the linear-time logic of its path formulas.

1 Introduction

Since the introduction of temporal logic into computer science by Pnueli in [28], the family of temporal logics has been widely accepted as an appropriate formal framework for the description of dynamic behavior, cf. [22]. Within this family, the branching-time logic CTL* has emerged as one of the more expressive logics [9], unifying the linear-time logic LTL [28] and the branching-time logic CTL [7]. The modal fixpoint logic [15] is more expressive than CTL*, but it is in some sense a low-level logic, making it an "unfriendly" logic for users, whereas CTL* can express complex properties of computation trees in a natural fashion. While the appropriateness of a branching-time logic for practical formal verification is under debate [34], CTL* is the right logic for applications that require computation-tree reasoning.

An example of an application that requires computation-tree reasoning is sanity checks for the modeling of a system: verification is done with respect to a mathematical model of the system, and it is important to check that the model, whose generation typically involves abstraction and reduction techniques, is correct. In order to check that the model does not disable essential behaviors of the system, one has to check *possibility properties* [19], which require computations of the model to be extendible to computations exhibiting the behaviors.

Another example of a setting in which computation-tree reasoning is required is *automated task planning*, a major research area in Artificial Intelligence, where given a description of a dynamic domain and of the basic actions that can be performed on it, and given a goal that defines a success condition to be achieved, one has to find a suitable plan, that is, a description of the actions to be executed on the domain in order to achieve the goal. "Classical" planning concentrates on the so called "reachability" goals, that is, on goals that define a set of final desired states to be reached (e.g., the red block is above the yellow block). Quite often, practical applications require plans that deal with goals that are more general than sets of final states. Several planning approaches have been recently proposed, where *temporal logic* formulas are used as goal languages, thus allowing

*Address: School of Computer Science and Engineering, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il. Supported in part by BSF grant 9800096, and by a grant from Minerva.

†Address: Department of Computer Science, Houston, TX 77251-1892, U.S.A. Email: vardi@cs.rice.edu. Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, and ANI-0216467, by BSF grant 9800096, and by Texas ATP grant 003604-0058-2003. Part of this work was done while the author was visiting the Isaac Newton Institute for Mathematical Science, as part of a Special Programme on Logic and Algorithm.

for goals that define conditions on the whole plan execution paths, i.e., on the sequences of states resulting from the execution of plans (see, e.g., [1]). Most of these approaches use LTL as the goal language. The linear-temporal logic LTL allows one to express reachability goals (e.g., Fq — reach q), maintainability goals (e.g., Gq — maintain q), as well as goals that combine reachability and maintainability requirements (e.g., FGq — reach a set of states where q can be maintained), and Boolean combinations of these goals.

In *planning in nondeterministic domains* [5], actions are allowed to have different outcomes, and it is not possible to know at planning time which of the different possible outcomes will actually take place. Nondeterminism in action outcome is necessary for modeling in a realistic way several practical domains (e.g., robotics, autonomous controllers, etc.). For instance, in a realistic robotic application one has to take into account that actions like "pick up object" might result in a failure (e.g., if the object slips out of the robot's hand). Planning in nondeterministic domains is essentially the same as *synthesis of open systems* [30], the only difference is that in planning there is emphasis on describing the domain using a modelling language, cf. [13].

A consequence of nondeterminism is that the execution of a plan may lead to more than one possible execution path. Therefore, one has to distinguish between the case the given goal has to be satisfied by all possible execution paths ("strong" planning) and the case where the goal has to be satisfied only by some possible execution path ("weak" planning). In the case of an LTL goal φ, strong planning corresponds to interpreting the formula in a universal way, as the CTL* formula as $A\varphi$, while weak planning corresponds to interpret it in an existential way, as the CTL* formula $E\varphi$. (Synthesis corresponds to strong planning).

Weak and strong plans are two very extreme ways of satisfiability of an LTL formula. In practical applications, it might be impossible to achieve goals in a strong way: for instance, in the robotic application it might be impossible to fulfill a given task if objects keep slipping from the robot's hand. On the other hand, weak plans are too unreliable, since they achieve the goal only under overly optimistic assumptions on the outcomes of action executions. In the case of reachability goals, *strong cyclic planning* [4] has been shown to provide a viable compromise between weak and strong planning. Formally, a plan is strong cyclic if each possible partial execution of the plan can always be extended to an execution that reaches some goal state. Strong cyclic planning allows for plans that encode iterative trial-and-error strategies, like "pick up an object until succeed". The execution of such strategies may loop forever only in the case the action "pick up object" continuously fails, and a failure in achieving the goal in the case of such an unfair execution is usually acceptable. This "always possibly" approach corresponds to Lamport's possibility properties mentioned above.

Inspired by the work on strong cyclic planning, Pistore and Vardi went on to explore the different degrees in which an LTL formula φ can be satisfied that exist between the strong goal $A\varphi$ and the weak goal $E\varphi$ [27]. They showed that a two-player game can model the spectrum between strong and weak planning. Player A chooses action outcomes in order to make goal φ fail, while player E chooses action outcomes in order to satisfy the goal φ. The degree of strength of the temporal requirement is determined by the structure of the game: who makes the first move and how many turns. Pistore and Vardi then proposed a logic based on these path games, using path quantifiers $\mathcal{A}$ and $\mathcal{E}$, which can express the spectrum between strong and weak planning. (Such *path games* have also been studied in [2].)

While the Pistore-Vardi logic is a branching-time logic, quite close to CTL*, it is different from CTL*. In CTL*, a path quantifier turns a path formula into a pure state formula. Thus, for example, the formula $AGE\varphi$ holds at a start state s_0 if for every state s reachable from s_0 there is a path starting at s that satisfies φ; the truth of $E\varphi$ depends only on the state in which it is evaluated. In contrast, in the Pistore-Vardi logic the formula $\mathcal{AE}\varphi$ holds at a start state s_0 if for every state s reachable from s_0 there is a path starting at s_0 and continuing through s that satisfies φ; the truth of $\mathcal{E}\varphi$ in a state depends also on the path that leads to that state. In other words, while CTL* path quantifiers are *memoryless*, the path quantifiers $\mathcal{A}$ and $\mathcal{E}$ are *memoryful*[1]. The Pistore-Vardi logic, however, does not have a standard logical syntax; for example, it is not closed under conjunction and disjunction[2].

We propose in this paper a memoryful variant of CTL*, which unifies CTL* and the Pistore-Vardi logic. We name the new logic mCTL*. Semantically, mCTL* is obtained from CTL* by reinterpreting the path quantifiers of the logic to be memoryful[3]. With memoryful quantification, $E\varphi$ holds at a node s of a computation tree if there is a path π starting at the root of the tree and going through s such that π satisfies the linear-time formula φ. Syntactically, we add a special proposition $present$, which is needed to emulate the ability of CTL* to talk about the "present". One of our main results is that the addition of $present$ is needed to make mCTL* at least as expressive as CTL*. (As in [2], one can translate mCTL* to CTL* via path logic, but this translation is of nonelementary complexity.) We note that

[1]It is shown in [2] that the Pistore-Vardi logic is expressible in CTL*, but only a non-elementary translation, which goes through *path logic* [14], is known.

[2]The idea of memoryful quantification relative to the present has been studied in the context of *Ockhamist temporal logic*, cf. [3], but has not been explored in the context of CTL*, and that study does not refer to algorithmic properties.

[3]Strictly speaking, mCTL*covers only the finite-alternation fragment of the Pistore-Vardi logic, leaving out the infinite-alternation fragment.

present is not required to easily express the Pistore-Vardi logic in mCTL*; the strong cyclic goal of $\mathcal{AE}\varphi$ is expressed in mCTL* by the formula $AGE\varphi$.) We show that memoryful quantification can be expressed in CTL* extended with the ability to refer to the *linear past* [16]. We note that the proposition *present* of mCTL* is different from the "*Now*" used in [12], in the context of branching temporal logics with past. While the "*Now*" in [12] is a unary temporal operator, which chops away the past, our *present* is an atomic proposition that holds in the present.

We then examine two decision problems related to mCTL*: satisfiability and model checking. We first show that satisfiability of mCTL* is 2EXPTIME-complete, which is the same complexity as that of CTL* [10, 35]. We then show that model checking mCTL* is significantly harder than model checking CTL*. Model checking mCTL* is EXPSPACE-complete, while model checking CTL* is PSPACE-complete [11]. Establishing the two upper bounds via an automata-theoretic framework is quite nontrivial. The standard approach to mCTL*satisfiability would reduce it to emptiness of two-way alternating tree automata [33]. This approach would yield a 2EXPTIME bound for satisfiability, but not the EXPSPACE bound for model checking. Instead, we offer an extension of the automata-theoretic approach to model checking [17], and describe *hesitant alternating automata with satellites*. We show that this framework can establish both upper bounds.

Since our lower-bound proof uses an m$^-$CTL* formula corresponding to Lamport's possibility properties, it also settles an open question regarding the complexity of model checking of such properties, as well as that of the Pistore-Vardi logic. In addition, our results, together with the linear translation of mCTL* to CTL* with linear past, imply new results about the complexity of branching temporal logic with past [20, 12, 23]. Our results show that the transition from memoryless to memoryful quantifiers is significant. We can now see that the significant complexity gap between CTL* satisfiability, which is 2EXPTIME-complete, and CTL* model checking, which is PSPACE-complete, is due to quantifier memorylessness, which helps model checking [11] but not satisfiability [35]. We discuss this point more in Section 5.

Due to the lack of space, this version does not include all the proofs. A full version can be found at the authors' home pages.

2 The Temporal Logic Memoryful CTL*

The branching-time logic *memoryful CTL** (mCTL*, for short) combines both branching-time and linear-time operators [9]. A path quantifier E ("for some path") can prefix an assertion composed of an arbitrary combination of the linear-time operators X ("next time") and U ("until"). The syntax of mCTL* is similar to the syntax of CTL*: there are two types of formulas in mCTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. The only difference in the syntax between the two logics is the fact mCTL* formulas may refer to a special atomic proposition *present*, which holds only in the present. An mCTL* formula φ is *closed* if all the occurrences of *present* are in a scope of a path quantifier. The logic mCTL* consists of closed mCTL* state formulas. We define the *size* $|\varphi|$ of φ as the number of state and path subformulas that φ has.

The semantics of mCTL* is defined with respect to *computation trees*. Given a finite set D of *directions*, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot d \in T$ where $x \in D^*$ and $d \in D$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. The prefix relation induces a partial order $\leq$ between nodes of T. Thus, for two nodes x and y, we say that $x \leq y$ iff there is some $z \in D^*$ such that $y = x \cdot z$. For every $x \in T$, the nodes $x \cdot d$, for $d \in D$, are the *successors* of x. A node is a *leaf* if it has no successors. For a node $x \in T$, an *x-path* of T is a minimal set $\pi \subseteq T$ such that $x \in \pi$ and for every $y \in \pi$, either y is a leaf or there exists a unique $d \in D$ such that $y \cdot d \in \pi$. When $x = \varepsilon$, we say that π is a *path*. For a path π and a node $x \in \pi$, the *suffix* of π that starts at x is the x-path $\pi \cap \{y : x \leq y\}$.

Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau\rangle$ where T is a D-tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. Of special interest to us are Σ-labeled trees in which $\Sigma = 2^{AP}$ for some set AP of atomic propositions. We call such Σ-labeled trees *computation trees*. A computation tree is often given by means of a *Kripke structure* $K = \langle AP, W, R, w_{in}, L\rangle$, where AP is the set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is a total transition relation , w_{in} is an initial state, and $L : W \to 2^{AP}$ maps each state to the set of atomic propositions true in that state. A *path* in K is an infinite sequence of states, $\pi = w_0, w_1, \ldots$ such that $\langle w_i, w_{i+1}\rangle \in R$ for every $i \geq 0$. We define the size $\|K\|$ of K as $|W| + |R|$. The Kripke structure K induces a computation tree $\langle T_K, \tau_K\rangle$ that corresponds to the unwinding of K from w_{in}.

The logics mCTL* and CTL* differ in the semantics of path quantification. Consider a computation tree. In CTL*, path quantification ranges over paths that start in the current node. In mCTL*, path quantification ranges over paths that start at the root and visit the current node. For example, the CTL* formula $AGE\psi$, for a linear formula ψ holds in a root of a computation tree if a computation satisfying ψ starts from every node in the tree. When viewed as an mCTL* formula, it holds in a computation tree if for every node x of the tree, the path from the root to x can be extended to a path satisfying ψ. As discussed in Section 1, this corresponds to strong cyclic plans [4]. In particular, when $\varphi = Fp$, it states that p has either occurred in the past or is possible in the future. This corresponds to Lamport's possibility properties [19]. Thus, when evaluating path formulas, one cannot ignore the past and satisfaction may depend on the events that have taken place since the beginning of the execution and until the present. Below we define the semantics of mCTL* formally.

Consider a computation tree $\langle T, \tau\rangle$. For two nodes x and c in T and an mCTL* formula φ, we use $x, c \models \varphi$ to indicate that x satisfies φ with c being the present. Similarly, $\pi, x, c \models \psi$, for a path formula ψ, iff the suffix of the path π that starts at x satisfies ψ with c being the present. Formally, we have the following (with respect to given T and τ).

- $x, c \models$ **true** and $x, c \not\models$ **false**, for all x, c in T.
- $x, c \models p$, for $p \in AP$, iff $p \in \tau(x)$.
- $x, c \models \mathit{present}$ iff $x = c$.
- $x, c \models \neg\varphi_1$ iff $x, c \not\models \varphi_1$.
- $x, c \models \varphi_1 \vee \varphi_2$ iff $x, c \models \varphi_1$ or $x, c \models \varphi_2$.
- $x, c \models E\psi$ iff there exists a path π such that $x \in \pi$ and $\pi, \varepsilon, x \models \psi$.
- $\pi, x, c \models \varphi$ for a state formula φ, iff $x, c \models \varphi$.
- $\pi, x, c \models \neg\psi_1$ iff $\pi, x, c \not\models \psi_1$.
- $\pi, x, c \models \psi_1 \vee \psi_2$ iff $\pi, x, c \models \psi_1$ or $\pi, x, c \models \psi_2$.
- $\pi, x, c \models X\psi$ iff $\pi, x \cdot d, c \models \psi$, where $d \in D$ is such that $x \cdot d \in \pi$.
- $\pi, x, c \models \psi_1 U \psi_2$ iff π contains a node $y \geq x$ such that $\pi, y, c \models \psi_2$, and $\pi, z, c \models \psi_1$ for all $x \leq z < y$.

Note that while in CTL* path quantification ranges over paths that start in the current node, here path quantification ranges over paths that start at the root and visit the current node. Thus, when evaluating path formulas, one cannot ignore the past and satisfaction may depend on the events that have taken place since the beginning of the execution and until the present. Note also that formulas of the form $E\psi$ "reset the present"; thus the satisfaction of $E\psi$ with respect to x, c is independent of c, and the present is set to x. It follows the truth value of a closed mCTL* formula in node x is independent of the present. Accordingly, we use the notation $x \models \varphi$ to indicate that an mCTL* formula φ is satisfied in node x. We say that a computation tree $\langle T, \tau\rangle$ satisfies an mCTL* formula φ iff $\varepsilon \models \varphi$.

Note that by letting path quantification range over paths that start at the root, an mCTL* formula can refer to events that happen in the past. Unlike, however, branching-time logics whose syntax contain explicit past-time operators, here the ability to refer to the past is limited. Note also that the Pistore-Vardi logic can be easily expressed in mCTL*.

The logic m$^-$CTL* is a fragment of mCTL* in which the atomic proposition $\mathit{present}$ is not allowed. As we show in Section 3, m$^-$CTL* cannot emulate the ability of CTL* to refer to the present, and is strictly weaker than mCTL*.

Examples Consider the mCTL* formula $AGE(GF\mathit{grasp} \rightarrow F\mathit{pick})$ that describes a desired behavior of a robot. The formula states that each node in the computation tree belong to a computation π such that if the robot tries to grasp an object infinitely often, it eventually picks it. Note that the robot may have picked the object before the present. Likewise, the mCTL* formula $AG(\mathit{pickA} \rightarrow EF\mathit{pickB})$ states that whenever a node x is labeled by pickA, then some computation that goes through x has a node labeled pickB. Thus, whenever the robot picks A, either it has picked B in the past, or it is possible for it to pick B in the future. Note that the specification of this property in CTL* is much more complicated.

The formula $AG(\mathit{grant} \rightarrow EF(\mathit{req} \wedge F(\mathit{ack} \wedge F\mathit{present})))$ demonstrates how mCTL* formulas can use the ability to refer to the present with $\mathit{present}$ in order to refer to the past. Indeed, the formula states that whenever a node x is labeled with grant, the path from the root to x has a node labeled with req, followed by a node labeled with ack, followed by the present. Thus, grants are given only if a request was issued and then acknowledged in the past.

Finally, The formula $AG(\mathit{ack} \rightarrow EF(\mathit{req} \wedge F(\mathit{present} \wedge F\mathit{grant})))$ gives a different view of the same sequence of events and demonstrates how mCTL* formulas can refer to both the past and the future. Indeed, the formula states that whenever a node x is labeled with ack, the path from the root to x has a node labeled with req and some path that starts in x has a node labeled with grant. Thus, acknowledgments are given only if a request was issued in the past and is going to be granted in the future. $\square$

The definition of mCTL* leads to natural theoretical and practical questions: in the theoretical front, we would like to study how the transition to a memoryful path quantification influences the expressive power of the logic, its connection to logics with past, and to standard CTL*, as well as the necessity of the atomic proposition $\mathit{present}$. In the practical front, we would like to study how the transition influences the complexity of the satisfiability and the model-checking problems. The latter is of particular interest as model-checking algorithms for CTL* are based on a bottom-up reasoning, where internal state formulas are evaluated first and replaced by new atomic propositions [11]. Such reasoning cannot be applied with a memoryful path quantification.

3 Expressiveness

In this section we discuss the expressive power of mCTL* and m$^-$CTL*. We say that two branching temporal logic formulas φ_1 and φ_2 are equivalent if for every computation trees $\langle T, tau\rangle$ and node $x \in T$, we have $x \models \varphi_1$ iff $x \models \varphi_2$.

We start by comparing mCTL* with CTL*:

Theorem 3.1 *mCTL* and CTL* are equally expressive.*

As detailed in the proof, while the blow up in translating a CTL* formula to an mCTL* formula is linear (we replace each subformula of the form $E\xi$ by the formula $EF(present \wedge \xi)$), the translation in the other direction goes via monadic path logic [14, 2] and is of nonelementary complexity.

Theorem 3.2 *mCTL* is at least exponentially more succinct than CTL*.*

The formula used in the proof of Theorem 3.2 is similar to the one used in [12] for proving the succinctness of LTL+past with respect to LTL. In particular, while the formula does not use the proposition $present$, it makes use of the ability of mCTL* to refer to the past. We now show that if we extend CTL* with past temporal operators, the translation is linear. Let CTL$^\star_{lp}$ and CTL$^\star_{lp^-}$ denote the extension of CTL* with past operators, with and without $present$, respectively, where formulas are interpreted over computation trees, thus past is linear [16].

Theorem 3.3 *Every mCTL* (resp. m$^-$CTL*) formula has an equivalent CTL$^\star_{lp}$ (resp. CTL$^\star_{lp^-}$) formula of linear length.*

We note that the complexity of the model-checking and the satisfiability problems for CTL$^\star_{lp}$ (with or without $present$) is open [16] (see also [20]), thus Theorem 3.3 does not lead to a model-checking or decidability procedure. In fact, the translation, together with the lower bounds we describe in Section 4 for mCTL* and mCTL*improves the known lower bounds known for CTL$^\star_{lp}$ and CTL$^\star_{lp}$-.

Recall that the proposition $present$ emulates the ability of CTL* to talk about the present. Indeed, the translation we described, of CTL* formulas to mCTL* formulas, uses $present$. In the full version, we show that the CTL formula $\psi = EF(EXp \wedge EX\neg p)$ has no equivalent m$^-$CTL* formula. Thus, the use of $present$ is essential, and m$^-$CTL* is strictly weaker than mCTL* (and thus, also CTL*).

Theorem 3.4 *There is a CTL formula with no equivalent m$^-$CTL* formula.*

The fact that m$^-$CTL* loses track of the present whenever path quantification is applied weakens its branching nature. This is reflected in the fact that the CTL formula we have used in the proof of Theorem 3.4 is similar to the formula with which Milner shows the differences between trace equivalence and bisimulation [24]. This weakness of m$^-$CTL* motivates us to focus, in the rest of this paper, on the logic mCTL*. In addition, as we show in Section 4.4, the EXPSPACE lower bound for mCTL* model checking holds also for m$^-$CTL*.

4 Decision Procedures

In this section we study the model-checking and satisfiability problems for mCTL*. We show that while the satisfiability problem is not harder than the one for CTL*, the model-checking problem is exponentially harder. For the upper bounds, we need to develop an automata-theoretic approach for memoryful branching temporal logic. For that, we start with definitions of alternating automata and introduce *alternating automata with satellites*. We then translate mCTL* formulas to such automata.

4.1 Alternating automata

Automata over infinite trees (tree automata) run over Σ-labeled D-trees that have no leaves [32]. *Alternating tree automata* generalize nondeterministic tree automata and were first introduced in [26]. Here we define *symmetric* alternating automata, which cannot distinguish between the different successors of a node, and send copies to the successors in either a universal or an existential manner [37].

Let $\Omega = \{\Box, \Diamond\}$, and let $\mathcal{B}^+(\Omega \times Q)$ be the set of positive Boolean formulas over $\Omega \times Q$; i.e., Boolean formulas built from elements in $\Omega \times Q$ using $\wedge$ and $\vee$, where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. For a set $S \subseteq \Omega \times Q$ and a formula $\theta \in \mathcal{B}^+(\Omega \times Q)$, we say that S *satisfies* θ iff assigning **true** to elements in S and assigning **false** to elements in $(\Omega \times Q) \setminus S$ makes θ true.

In a symmetric alternating automaton, the transition function δ maps q and σ to a formula in $\mathcal{B}^+(\Omega \times Q)$. Intuitively, an atom $\langle \Box, q\rangle$ corresponds to copies of the automaton in state q, sent to all the successors of the current node. An atom $\langle \Diamond, q\rangle$ corresponds to a copy of the automaton in state q, sent to some successor of the current node.

Formally, a symmetric automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_{in}, \alpha\rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(\Omega \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω). Let $T = D^*$. A *run* of a symmetric alternating automaton $\mathcal{A}$ on an input Σ-labeled D-tree $\langle T, \tau\rangle$ is a $(D^* \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r\rangle$ that satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_{in})$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S \subseteq \Omega \times Q$, such that S satisfies θ, and for all $(c, s) \in S$, the following hold:
    - If $c = \Box$, then for each $d \in D$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$.
    - If $c = \Diamond$, then for some $d \in D$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$.

Each infinite path ρ in $\langle T_r, r\rangle$ is labeled by a word in Q^ω. Let $\mathit{inf}(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r\rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In *Büchi* automata, $\alpha \subseteq Q$, and an infinite path ρ satisfies α iff $\mathit{inf}(\rho) \cap \alpha \neq \emptyset$. In *co-Büchi* automata, $\alpha \subseteq Q$, and an infinite path ρ satisfies α iff $\mathit{inf}(\rho) \cap \alpha = \emptyset$.

An automaton accepts a tree iff there exists an accepting run on it. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of all labeled trees that $\mathcal{A}$ accepts. The *nonemptiness problem* is to decide, given $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) \neq \emptyset$. When $|\Sigma| = 1$, we refer to the problem as *1-letter nonemptiness*. The *complement* of an automaton $\mathcal{A}$ is an automaton $\tilde{\mathcal{A}}$ that accepts exactly all the tress that $\mathcal{A}$ rejects.

Symmetric automata are powerful enough for deciding satisfiability of CTL* formulas. Given a CTL* formula φ, we translate it to a symmetric automaton $\mathcal{A}_\phi$ that accepts tree models of φ. Then to decide satisfiability we have to check that $\mathcal{A}_\varphi$ is non-empty, cf. [17]. As the translation is exponential and the emptiness test is exponential, we get a doubly exponential upper bound, which is asymptotically optimal [35]. To decide satisfiability of mCTL*, we need an extension of symmetric automata to *two-way* automata [33]. This approach would yield a 2EXPTIME bound for satisfiability, but not the EXPSPACE bound for model checking. The reason is that mCTL* is interpreted over unfolding of programs. This requires us to convert two-way automata to one-way automata, involving another exponential blow-up. Thus, just the construction of automata is doubly exponential. Instead, we offer an extension of the automata-theoretic approach to model checking [17], and describe *hesitant alternating automata with satellites*. We show that this framework can establish upper bounds both for satisfiability and model checking.

In [17], the authors introduce *hesitant alternating automata* (HAAs, for short) and show that CTL* formulas can be translated to such automata. An HAA is an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha\rangle$, where $\alpha = \langle G, B\rangle$ with $G \subseteq Q$ and $B \subseteq Q$. That is, the acceptance condition of HAAs consists of a pair of sets of states. As in weak alternating automata [25], there exists a partition of Q into disjoint sets $Q_1, \ldots, Q_m$ and a partial order $\leq$ on these sets such that transitions lead to sets that are lower in the partial order. Formally, for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$, for some $\sigma \in \Sigma$, we have $Q_j \leq Q_i$. In addition, each set Q_i is classified as either *transient* ($\delta(q, \sigma)$ contains no states of Q_i), *existential* ($\delta(q, \sigma)$ only contains disjunctively related states of Q_i), or *universal* ($\delta(q, \sigma)$ only contains conjunctively related elements of Q_i). It follows that every infinite path π of a run $\langle T_r, r\rangle$ gets trapped within some existential or universal set Q_i. The path then satisfies an acceptance condition $\langle G, B\rangle$ if and only if either Q_i is an existential set and $\mathit{inf}(\pi) \cap G \neq \emptyset$, or Q_i is a universal set and $\mathit{inf}(\pi) \cap B = \emptyset$.

Given a transition function δ, let $\tilde{\delta}$ denote the dual function of δ. That is, for every q and σ with $\delta(q, \sigma) = \theta$, let $\tilde{\delta}(q, \sigma) = \tilde{\theta}$, where $\tilde{\theta}$ is obtained from θ by switching $\Box$ and $\Diamond$, switching $\vee$ and $\wedge$, and switching **true** and **false**. If, for example, $\theta = \Box p \vee (\mathbf{true} \wedge \Diamond q)$ then $\tilde{\theta} = \Diamond p \wedge (\mathbf{false} \vee \Box q)$,

Lemma 4.1 [25, 17] *Given an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, q_{in}, \langle G, B\rangle\rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_{in}, \langle B, G\rangle\rangle$ is an HAA that complements $\mathcal{A}$.*

A *satellite* for an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, q_{in}, \alpha\rangle$ is a deterministic word automaton $\mathcal{U} = \langle \Sigma, Q', \delta', q'_{in}\rangle$ with no acceptance condition. For a node $x = d_0 \cdots d_k$ of a Σ-labeled D-tree $\langle T, \tau\rangle$, let $\mathit{word\_to}(x) = \tau(\varepsilon) \cdot \tau(d_0) \cdot \tau(d_0 \cdot d_1) \cdot \tau(d_0 \cdot d_1 \cdots d_{k-1})$ be the word that labels the path from the root to x. When the HAA reads a node x of the input tree, its transitions may depend on $\delta'(q_{in}, \mathit{word\_to}(x))$, namely on the state in which $\mathcal{U}$ would have been if we had run it along the paths of the tree. Formally, the transition function of $\mathcal{A}$ is $\delta : Q \times \Sigma \times Q' \to \mathcal{B}^+(\Omega \times Q)$, and is such that when $\mathcal{A}$ is in state q as it reads the node x, it proceeds according to $\delta(q, \tau(x), \delta'(q'_{in}, \mathit{word\_to}(x)))$. Technically, an HAA with a satellite is equivalent to the HAA obtained by taking the product of $\mathcal{A}$ and $\mathcal{U}$: the new state space is $Q \times Q'$, and whenever the product is in state $\langle q, q'\rangle$ and reads a node x, the transition from $\langle q, q'\rangle$ is obtained from $\delta(q, \tau(x), q')$ by replacing each atom $\Box s$ or $\Diamond s$ by $\Box(s, \delta'(q', \tau(x)))$ or $\Diamond(s, \delta'(q', \tau(x)))$, respectively. The acceptance condition of the product HAA is induced by F. The partition of the state space to sets, the partial order on the sets, and their classification into transient, universal, and existential are induced by these in $\mathcal{A}$.

Note that satellites are only a convenient way to describe HAA in which the state space can be partitioned to two components, one of which is deterministic, independent of the other, has no influence on the acceptance, and runs on all the branches of the tree. In particular, Lemma 4.1 holds also for HAA with satellites. It is sometimes convenient to describe the HAA and its satellite separately. In addition to clarity, the separation to $\mathcal{A}$ and $\mathcal{U}$ enables a tighter analysis of the complexity of the nonemptiness problem. Recall that the solution of the emptiness problem for alternating automata involves alternation removal, which results in a nondeterministic automaton with exponentially many states. While the product of an HAA with n states and a satellite with n' states has nn' states, there is a need to pay the exponential price of alternation removal in the process of the nonemptiness check only for $\mathcal{A}$. Formally, we have the following.

Theorem 4.2 *The nonemptiness problem for an HAA with n states and a satellite with n' states can be solved in time $2^{O(n \log n' + n^2 \log n)}$.*

Proof: Let $\mathcal{A}$ be an HAA with n states, and let $\mathcal{U}$ be its satellite with n' states. We first claim that $\mathcal{A}$ is not empty iff it accepts a tree of branching degree n. The proof of the claim is similar to the linear branching degree property of μ-calculus [7, 31]. Now, we can translate $\mathcal{A}$ to a nondeterministic parity tree automaton $\mathcal{A}'$ over trees of branching degree n, with $n^{O(n)}$ states, index $O(n)$, and with the same satellite $\mathcal{U}$. By taking the product of $\mathcal{A}'$ and $\mathcal{U}$, we get a nondeterministic parity automaton with $n'n^{O(n)}$ states, index $O(n)$, and no satellite. Checking the nonemptiness of such an NPT requires time $n'^{O(n)}n^{O(n^2)} = 2^{O(n \log n' + n^2 \log n)}$ [10, 30]. □

Theorem 4.3 *The 1-letter nonemptiness problem for an HAA with n states, depth m, and a satellite with n' states can be solved in space $O(m \log^2(nn'))$.*

Proof: Let $\mathcal{A}$ be an HAA with n states and depth m, and let $\mathcal{U}$ be its satellite with n' states. We can translate $\mathcal{A}$ to an HAA with nn' states, depth m, and no satellite. By [17], checking the 1-letter nonemptiness of such an HAA requires space $O(m \log^2(nn'))$. □

4.2 From mCTL* to HAA

In this section we describe a translation of mCTL* formulas to HAA. In the sequel, we use the translation in order to obtain satisfiability and model-checking decision procedures for mCTL*.

We first need some definitions. For an mCTL* formula ψ, let $sf(\psi)$ be the set of state subformulas of ψ. For two formulas θ and φ of ψ, we say that θ is *maximal* in φ if θ is a strict state subformula of φ and there exists no state formula "between them", namely, there exists no strict subformula ξ of φ such that θ is a strict subformula of ξ. We denote by $max(\varphi)$ the set of all formulas maximal in φ. For example, $max(A((X\neg p)U(EXq))) = \{\neg p, EXq\}$. Consider an mCTL* formula ψ and a computation tree $\langle T, \tau\rangle$. We say that a $2^{sf(\psi)}$-labeled tree $\langle T, g\rangle$ is *sound* for ψ if for all $x \in T$ and $\theta \in sf(\psi)$, we have that $x, \epsilon \models \theta$ iff $\theta \in g(x)$. Thus, a node x is labeled by exactly all the formulas in $sf(\psi)$ that are satisfied in x, with ϵ being the present (note that the fact ϵ is the present is important only for $\theta = present$).

Theorem 4.4 *Given an mCTL* formula ψ, we can construct an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{A}_\psi$ runs on $2^{sf(\psi)}$-labeled trees and accepts exactly all trees that are sound for ψ and satisfy ψ.*

Proof: We first define the satellite $\mathcal{U}$ for $\mathcal{A}_\psi$. Consider a subformula of ψ of the form $E\xi$. Let $\mathcal{U}_\xi = \langle 2^{sf(\psi)}, Q_\xi, M_\xi, Q^{in}_\xi, \alpha_\xi\rangle$ be a nondeterministic Büchi automaton on infinite words such that $\mathcal{U}_\xi$ accepts exactly all the computations satisfying ξ [36]. Note that $\mathcal{U}_\xi$ regards the formulas maximal in φ as atomic propositions ($\mathcal{U}_\xi$ ignores the other formulas in $sf(\psi)$). Let $\mathcal{U}^d_\xi = \langle 2^{sf(\psi)}, 2^{Q_\xi}, M^d_\xi, \{Q^{in}_\xi\}\rangle$ be the deterministic automaton with no acceptance condition obtained by applying the subset construction to $\mathcal{U}_\xi$. Thus, for all $S \in 2^{Q_\xi}$ and $\sigma \in 2^{sf(\psi)}$, we have that $M^d_\xi(S, \sigma) = \bigcup_{s \in S} M_\xi(s, \sigma)$. Now, the satellite $\mathcal{U} = \langle 2^{sf(\psi)}, Q', \delta', q'_{in}\rangle$ is the crossproduct of all the automata $\mathcal{U}^d_\xi$ above (for all the subformulas $E\xi$ of ψ). Intuitively, $\mathcal{U}$ supplies to $\mathcal{A}_\psi$ the information required in order to evaluate path formulas on paths that start in the root of the tree and visit the current node. When there is a need to check that $E\xi$ holds in some node x, the automaton $\mathcal{A}_\psi$ guesses a state q that is a member of the current state $S \in 2^{Q_\xi}$ of $\mathcal{U}^d_\xi$ (recall that 2^{Q_ξ} is a component in the state space of the satellite) and it executes $\mathcal{U}^q_\xi$ along some x-path. The position in which $\mathcal{A}_\psi$ starts the execution of $\mathcal{U}^q_\xi$ is the only position in which the atomic proposition $present$ holds. Note that a correct guess of q and a successful run of $\mathcal{U}^q_\xi$ along some x-path are possible iff there is a path that visits x and satisfies ξ, which corresponds to the semantics of $E\xi$.

As in the case of the HAA for CTL* [17], we construct $\mathcal{A}_\psi$ by induction on the structure of ψ. With each subformula φ of ψ, we associate an HAA $\mathcal{A}'_\varphi$ composed from HAAs associated with formulas maximal in φ. We assume that the state sets of composed HAAs are disjoint (otherwise, we rename states). The HAA $\mathcal{A}'_\varphi$ assumes that the tree is sound for the formulas in $max(\varphi)$ and only checks the satisfaction of φ under this assumption. We then define $\mathcal{A}_\psi$ as the intersection of $\mathcal{A}'_\psi$ with an automaton that checks, by sending copies to the different $\mathcal{A}'_\varphi$ automata, that the input tree is indeed sound with respect to ψ.

- If $\varphi = p$ for $p \in AP$ or $p = present$, then $\mathcal{A}'_\varphi$ is the one-state HAA that goes to **true** when it reads σ with $p \in \sigma$ and goes to **false** otherwise.
- If $\varphi = \neg\varphi_1$, then $\mathcal{A}'_\varphi$ is $\tilde{\mathcal{A}}_{\varphi_1}$ — the HAA obtained by dualizing the HAA $\mathcal{A}_{\varphi_1}$ for φ_1. If $\varphi = \varphi_1 \vee \varphi_2$, then $\mathcal{A}'_\varphi$ has an initial state that sends all the copies sent by $\mathcal{A}'_{\varphi_1}$ or all the copies sent by $\mathcal{A}'_{\varphi_2}$.
- If $\varphi = E\xi$, where ξ is an mCTL* path formula, we proceed as follows. Let $\mathcal{U}_\xi = \langle 2^{sf(\psi)}, Q_\xi, M_\xi, Q^{in}_\xi, \alpha_\xi\rangle$. Recall that the state space Q' of the satellite $\mathcal{U}$ is the product of the state spaces of the deterministic automata for the path formulas. Thus, each state $q' \in Q'$ has a component for each of these automata, and in particular for $\mathcal{U}^d_\xi$. Consider a state $q' \in Q'$. Let $q'_{|\xi}$ be the state of $\mathcal{U}^d_\xi$ in q'. Note that $q'_{|\xi} \in 2^{Q_\xi}$. Then, $\mathcal{A}'_\varphi =$

$\langle 2^{sf(\psi)}, Q_\xi, \delta', q_{in}, \langle \alpha_\xi, \emptyset \rangle \rangle$ is defined so that from its initial state q_{in}, it consults the satellite's state q' and executes $\mathcal{U}_\xi$ along a single path, starting from some state in q'_ξ. The proposition $present$ holds exactly at the node in which the execution of $\mathcal{U}_\xi$ starts, thus its first transition assumes that $present$ holds. Formally, for all $\sigma \in 2^{sf(\psi)}$ and $q' \in Q'$, we have $\delta'(q_{in}, \sigma, q') = \bigvee_{s \in q'_{|\xi}} \bigvee_{s' \in M_\xi(s, \sigma \cup \{present\})} \Diamond s'$. Also, for all $q \in Q_\xi$, we have $\delta'(q, \sigma, q') = \bigvee_{q_i \in M_\xi(q,\sigma)} \Diamond q_i$. If $M_\xi(q, \sigma) = \emptyset$, then $\delta(q, \sigma, q') = \textbf{false}$. Note that the only transition in which the input from the satellite is taken into an account is the transition from q_{in}, where $\mathcal{A}'_\varphi$ chooses a state from $q_{|\xi}$ to proceed with. Note also that Q_ξ constitutes a single existential set. The HAA $\mathcal{A}'_\varphi$ accepts a $2^{sf()}$-labeled tree from node x iff x satisfies φ, assuming the input tree is sound for the formulas in $max(\varphi)$. The states in Q_ξ constitute an existential set of the HAA.

We now add to $\mathcal{A}'_\psi$ transitions that check that the input tree is indeed sound for ψ, thus the letter read at node x describes the set of formulas satisfied in x. For this purpose, we add a new state q_{check}. Whenever $\mathcal{A}_\psi$ is in state q_{check} and reads a letter σ, it sends copies sent from the initial states of the HAA $\mathcal{A}'_{\varphi_i} = \langle 2^{sf(\psi)}, Q^i, \delta^i, q^i_{in}, \langle G^i, B^i \rangle \rangle$, for all $\varphi_i \in \sigma$, sends copies sent from the initial states of the HAA $\tilde{\mathcal{A}}'_{\varphi_i} = \langle 2^{sf(\psi)}, \tilde{Q}^i, \tilde{\delta}^i, q^i_{in}, \langle B^i, G^i \rangle \rangle$, for all $\varphi_i \notin \sigma$, (and also sends a copy that stays in q_{check} to all the successors). Formally, for all $\sigma \in 2^{sf(\psi)}$ and $q' \in Q'$, we have $\delta(q_{check}, \sigma, q') = \Box q_{check} \wedge \bigwedge_{\varphi_i \in \sigma} \delta^i(q^i_{in}, \sigma, q') \wedge \bigwedge_{\varphi_i \notin \sigma} \tilde{\delta}^i(q^i_{in}, \sigma, q')$.

The HAA $\mathcal{A}_\psi = \langle 2^{sf(\psi)}, Q, \delta, q_{in}, \langle G, B \rangle \rangle$ is such that Q is the union of $\{q_{in}, q_{check}\}$ with the union of the state spaces of $\mathcal{A}'_\theta$, for $\theta \in sf(\psi)$. The initial state q_{in} checks for the satisfaction of ψ and for the soundness with respect to ψ, thus $\delta(q_{in}, \sigma, q') = \delta^\psi(q^\psi_{in}, \sigma, q') \wedge \delta(q_{check}, \sigma, q')$, where δ^ψ and q^ψ_{in} are the transition function and initial state of $\mathcal{A}'_\psi$. Finally, $G = \bigcup_{\varphi_i \in sf(\psi)} G^i$ and $B = \bigcup_{\varphi_i \in sf(\psi)} B^i$. The state q_{in} is transient and the state q_{check} constitute a singleton universal set of the HAA.

The arguments about the correctness of the construction, its size, and its depth, are similar to these in [17], and are given in the full version. ☐

4.3 Satisfiability

Theorem 4.5 *The satisfiability problem for mCTL* is 2EXPTIME-complete.*

Proof: We start with the upper bound. Consider an mCTL* formula ψ. By Theorem 4.4, there is an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of $2^{sf(\psi)}$-labeled trees that are sound for ψ and satisfy ψ. The HAA $\mathcal{A}_\psi$ is nonempty iff ψ is satisfiable. By Theorem 4.2, the nonemptiness of $\mathcal{A}_\psi$ can be decided in time $2^{2^{O(|\psi|)}}$, so we are done.

It is left to prove the lower bound. By Theorem 3.1, each CTL* formula can be linearly translated to an equivalent mCTL* formula. The lower bound then follows from the fact CTL* satisfiability is 2EXPTIME-hard [35]. ☐

Note that the application of the 2EXPTIME lower bound of CTL* satisfiability is not possible for m$^-$CTL*, and the exact complexity of the satisfiability problem for this logic is left open.

4.4 Model checking

For CTL*, the automata-theoretic approach is similar for satisfiability and model checking: the HAA for a CTL* formula ψ accepts exactly all 2^{AP}-labeled trees that satisfy ψ, so satisfiability is reduced to emptiness and model checking to 1-letter emptiness of the product of the HAA with the system [17]. In the case of mCTL*, the need to execute the word automata for the path formulas from the root of the tree, has forced us to define the HAA with respect to $2^{sf(\psi)}$-labeled trees. While this was not a problem for satisfiability, it is a problem for model checking, where we need to take the product of the HAA with a Kripke structure that is labeled by 2^{AP}. Fortunately, this is not a real problem, as it is possible to guess an extension of the 2^{AP}-labeling to a $2^{sf(\psi)}$-labeling, and then let the HAA check the soundness of the guess.

Theorem 4.6 *Given an mCTL* formula ψ, we can construct an HAA $\mathcal{A}^{AP}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{A}^{AP}_\psi$ runs on 2^{AP}-labeled trees and accepts exactly all trees that satisfy ψ.*

Proof: The construction is similar to the one described in Theorem 4.4, only that we have to adjust the HAA and its satellite to the alphabet 2^{AP}. Intuitively, instead of having the input tree labeled by subsets of $sf(\psi)$, we let the satellite guess the richer labels, and then let the HAA check the guess. Thus, the satellite is nondeterministic — on top of its deterministic transition we add a guess of the subset of $sf(\psi)$ to be read in the successor node. Yet, running the HAA on a 2^{AP}-labeled tree, and letting it check the guesses, guarantees that $word\_to(x)$ can be viewed as a word in $(2^{sf(\psi)})^*$ rather than a word in $(2^{AP})^*$. Accordingly, $\delta'(q_{in}, word\_to(x))$ is a singleton, as in the case of a deterministic satellite.

Formally, if in the construction in Theorem 4.4 we ended up with a satellite $\mathcal{U} = \langle 2^{sf(\psi)}, Q', \delta', q'_{in} \rangle$ and

HAA $\mathcal{A}_\psi = \langle 2^{sf(\psi)}, Q, \delta, q_{in}, \langle G, B\rangle\rangle$, now we have a satellite $\mathcal{U}_{AP} = \langle 2^{AP}, Q' \times 2^{sf(\psi)}, \delta'_{AP}, q'_{in}\rangle$ where for all $\langle q', \sigma\rangle \in Q' \times 2^{sf(\psi)}$ and $\sigma' \in 2^{AP}$, we have $\delta'_{AP}(\langle q', \sigma\rangle, \sigma') = \{\delta'(q', \sigma)\} \times 2^{sf(\psi)}$, and $\mathcal{A}^{AP}_\psi = \langle 2^{AP}, Q, \delta^{AP}, q_{in}, \langle G, B\rangle\rangle$, where for all $q \in Q$, $\sigma' \in 2^{AP}$, and $\langle q', \sigma\rangle \in Q' \times 2^{sf(\psi)}$, we have $\delta^{AP}(q, \sigma', \langle q', \sigma\rangle) = \delta(q, \sigma, q') \wedge \bigwedge_{\varphi_i \in \sigma} \delta^i(q^i_{in}, \sigma, q') \wedge \bigwedge_{\varphi_i \notin \sigma} \tilde{\delta}^i(q^i_{in}, \sigma, q')$. □

Theorem 4.7 *The model-checking problem for mCTL* is EXPSPACE-complete, even for formulas of the form $AGE\xi$, where ξ is an LTL formula.*

Proof: We start with the upper bound. Consider an mCTL* formula ψ. By Theorem 4.4, there is an HAA $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states, such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of computation trees satisfying ψ. Consider a Kripke structure K. By [17], K satisfies ψ iff the 1-letter HAA obtained by taking the product of K with $\mathcal{A}_\psi$ is not empty. The product has $|K|2^{O(|\psi|)}$ states, depth $O(|\psi|)$, and a satellite with $2^{2^{O(|\psi|)}}$ states. Thus, by Theorem 4.3, its 1-letter nonemptiness problem can be solved in space $|\psi| \log^2(|K|2^{2^{O(|\psi|)}}) = |\psi|(\log^2|K| + 2^{O(|\psi|)} \log|K| + 2^{O(\psi)})$. Note that the complexity is only logarithmic in the size of the structure, and the exponential dependency is in the length of the formula, which is usually much smaller.

In order to prove the EXPSPACE lower bound, we do a reduction from the *exponential tiling* problem [21]. In this problem we are given a fixed set T of tiles, two relations $H, V \subseteq T \times T$, an integer n, a tuple of n tiles $\langle t_0, \ldots, t_{n-1}\rangle \in T^n$, and a tile $t_{fin} \in T$. The problem is to decide whether there is $m \geq 0$ such that it is possible to tile a $(2^n \times m)$-square so that horizontal neighbors belong to H, vertical neighbors belong to V, the first n tiles in the first row are $t_0, \ldots, t_{n-1}$, and the first tile in the last row is t_{fin}. Given an instance $\mathcal{T}$ of the tiling problem, we construct an mCTL* formula φ over a fixed set AP of atomic propositions such that a legal tiling for $\mathcal{T}$ exists iff the universal model M_{AP} for AP (that is, a clique in which each node is labeled by a different subset of AP) satisfies φ. The formula φ is of length polynomial in $\mathcal{T}$, and the size of M_{AP} is fixed, so we actually prove that the EXPSPACE lower bound holds for structures of a fixed size. Essentially, the formula φ describes a path that encodes a legal tiling. It is easy to describe all the conditions for such a tiling, except for the condition about vertical neighbors, which are exponentially far. This is where the memoryfulness of φ comes into the picture. By pointing to a tile in the present (using $present$), we can relate it with the tile below it. For the detailed proof, see the full version. Finally, in order to prove the lower bound for formulas of the form $AGE\xi$, we use the fact that pointing to the present with $present$ can be emulated by pointing to a location in the computation in which some special event occurs. For the detailed proof, see the full version. □

Note that the lower-bound proof applies already for an mCTL* formula of the form $AGE\xi$, for an LTL formula ξ. Thus, the EXPSPACE lower bound applies also for model checking of the less expressive m$^-$CTL*, the Pistore-Vardi logic, and Lamport's possibility properties. In addition, together with Theorem 3.3, it implies an EXPSPACE lower bound for the model-checking problem of CTL$^\star_{lp}$, a problem that was left open in [16] (see also [20]).

5 Discussion

We introduced and studied mCTL* — a variant of CTL* in which path quantification ranges over paths that start in the beginning of the execution of the system and go through the present. We argued for the appropriateness of mCTL* for sanity checks and for planning in nondeterministic domains, and showed that memoryful path quantification can lead to formulas that are exponentially more succinct. Studying the algorithmic properties of memoryful path quantification, we showed that while the transition from memoryless to memoryful path quantification does not make the satisfiability problem harder, it does make the model-checking problem exponentially harder.

Our complexity results shed some new light on the old debate on the relative merits of linear vs. branching time, cf. [18, 29, 34]. One position in favor of branching time argues that a branching-time logic such as CTL* subsumes the expressive power of linear-time logic, without increasing the complexity of model checking [11]. The analysis in [19, 27] show that true computation-tree reasoning requires memoryful path quantification. As we showed here, this results in an exponential increase in the complexity of model checking. Thus the expressiveness of branching-time logic does not come for free.

The fragment CTL of CTL* has received a lot of attention, and its model-checking problem can be solved in linear time [8]. One can also define the logics mCTL and m$^-$CTL, which is the memoryful counterparts of CTL. Unfortunately, mCTL and m$^-$CTL are not of much interest. As we show in the full version, their expressive power is very limited (essentially, the fact its path formulas cannot contain a Boolean assertion containing $present$ makes mCTL as expressive as m$^-$CTL, where present is lost with each application of a temporal operator), and still, their model-checking problem is at least NP-hard.

Acknowledgement We thank Philippe Schnoebelen for helpful discussions regarding the succinctness of CTL$^\star_{lp}$.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[2] D. Berwanger, E. Grädel, and S. Kreutzer. Once upon a time in a west - determinacy, definability, and complexity of path games. In *Proc. 10th LPR*, LNCS 2850, pages 229–243, 2003.

[3] M. Brown and V. Goranko. An extended branching-time ockhamist temporal logic. *Journal of Logic, Language and Information*, 8(2):143–166, 1999.

[4] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. 15th AAAI*, pages 875–881, 1998.

[5] A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In *Proc. of 4th AIPS*, pages 36–43, 1998.

[6] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. LNCS 354, pages 428–437, 1988.

[7] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71, 1981.

[8] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2):244–263, 1986.

[9] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *JACM*, 33(1):151–178, 1986.

[10] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th FOCS*, pages 328–337, 1988.

[11] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 20th POPL*, pages 84–96, 1985.

[12] N. Markey F. Laroussinie and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proc. 17th LICS*, pages 383–392, 2002.

[13] M. Fox, D. Long, and K. Halsey. An investigation into the expressive power of PDDL2.1. In *Proc. 16th ECAI*, pages 328–342, 2004.

[14] T. Hafer and W. Thomas. Computation tree logic $\mathrm{CTL}^\star$ and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th ICALP*, LNCS 267, pages 269–279, 1987.

[15] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

[16] O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th LICS*, pages 25–35, 1995.

[17] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *JACM*, 47(2):312–360, 2000.

[18] L. Lamport. Sometimes is sometimes “not never” - on the temporal logic of programs. In *Proc. 7th POPL*, pages 174–185, 1980.

[19] L. Lamport. Proving possibility properties. *TCS*, 206(1–2):341–352, 1998.

[20] F. Laroussinie and Ph. Schnoebelen. Specification in CTL+past for verification in CTL. *I& C*, 156(1-2):236–263, 2000.

[21] H.R. Lewis. Complexity of solvable cases of the decision problem for the predicate calculus. In *Foundations of Computer Science*, volume 19, pages 35–47, 1978.

[22] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[23] N. Markey. Temporal logic with past is exponentially more succinct. *EATCS Bulletin*, 79:122–128, 2003.

[24] R. Milner. *A Calculus of Communicating Systems*, LNCS 92, 1980.

[25] D.E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd LICS*, pages 422–427, 1988.

[26] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *TCS*, 54:267–276, 1987.

[27] M. Pistore and M. Vardi. The planning spectrum – one, two, three, infinity. In *Proc. 18th LICS*, pages 234–243, 2003.

[28] A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57, 1977.

[29] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proc. 12th ICALP*, LNCS 194, pages 15–32, 1985.

[30] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

[31] R.S. Streett and E.A. Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *I& C*, 81(3):249–264, 1989.

[32] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 133–191, 1990.

[33] M.Y. Vardi. Reasoning about the past with two-way automata. In *25th ICALP*, LNCS 1443, pages 628–641, 1998.

[34] M.Y. Vardi. Branching vs. linear time: Final showdown. In *7th TACAS*, LNCS 2031, pages 1–22, 2001.

[35] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th STOC*, pages 240–251, 1985.

[36] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I&C*, 115(1):1–37, November 1994.

[37] T. Wilke. CTL^+ is exponentially more succinct than CTL. In *19th FST& TCS*, LNCS 1738, pages 110–121, 1999.

[38] P. Wolper. Temporal logic can be more expressive. In *Proc. 22nd FOCS*, pages 340–348, 1981.