IDC HERZLIYA | Efi Arazi School of Computer Science

The Interdisciplinary Center, Herzlia
Efi Arazi School of Computer Science
M.Sc. program - Research Track

Automaton-Based Criteria for Membership in CTL

by
Yariv Shaulian

M.Sc. dissertation, submitted in partial fulfillment of the requirements for the M.Sc. degree, research track, School of Computer Science
The Interdisciplinary Center, Herzliya

September 2017

This work was carried out under the supervision of Dr. **Udi Boker** from the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya.

Acknowledgement

It is hard to overestimate the guidance given to me by Dr. Udi Boker in the course of preparation of this work. Udi was always ready to point out the mistakes in my arguments, propose his ideas and questions, and challenge me to achieve my goals in full. It is of no doubt that Udi has done much more than one would expect from a supervisor. For all of that, I would like to express my sincerest gratitude to him.

I would also like to thank Prof. Orna Kupferman and Dr. Dana Fisman for the careful reading of my work and their various helpful suggestions and tips.

Last but not least, I would like to thank my family and my girlfriend for supporting and believing in me throughout the whole research process.

Abstract

Computation Tree Logic (CTL) is widely used in formal verification, yet, unlike linear temporal logic (LTL), its connection to automata over words and trees is not yet fully understood. Moreover, the long sought connection between LTL and CTL is still missing; It is not known whether their common fragment is decidable, and there are very limited necessary conditions and sufficient conditions for checking whether an LTL formula is definable in CTL.

Kupferman, Vardi, and Wolper showed in 2000 that every CTL formula can be translated to a certain type of alternating tree automaton, more precisely, to a Hesitant Alternating Linear Tree Automaton (HALT). We show that HALT indeed characterizes CTL—We provide a translation from HALT to CTL, and prove that (non-hesitant) alternating linear tree automata are strictly more expressive than CTL.

Using this automaton characterization of CTL, we provide sufficient conditions and necessary conditions for LTL formulas and ω-regular languages to be expressible in CTL. Our conditions build on the fact that every ω-regular language that is expressible in CTL is also recognized by a deterministic Büchi word automaton (DBW), and relate the cycles of a given DBW to those of a potentially equivalent HALT.

Our basic necessary condition states that if a DBW $\mathcal{D}$ is equivalent to a CTL formula then there is no finite word u, such that $\mathcal{D}$ can go along u both within a part of some cycle C and from C to a state that accepts every word. The main sufficient condition narrows down the necessary condition by requiring, among other things, that the DBW leave cycles with unique words. Its correctness proof is constructive, defining an equivalent CTL formula.

Finally, using our necessary condition, we refute a conjecture by Clarke and Draghicescu from 1988, regarding a condition for a CTL* formula to be expressible in CTL.

Contents

List of Figures

1 Introduction

Background. Temporal logic plays a key role in formal verification of reactive systems, serving as the main formalism for defining the specifications to be verified. There are various types of temporal logics, classified into two main families—linear time and branching time. The most commonly used logic in the former is *Linear Temporal Logic* (LTL) [30] and in the latter is *Computation Tree Logic* (CTL) [6]. Since their introduction to computer science and formal verification in the 80s, there is a long debate on which of the two is preferable, continuing until these days. (For an overview of the debate, see for example [35].) Obviously, each of the two has its pros and cons. Roughly (and arguably) speaking, LTL is a more natural specification language, whereas CTL allows for more efficient verification algorithms.

In the linear-time setting, the automata-theoretic approach is widely used. That is, both the specification, which is given by means of an LTL formula, and the system to be verified are translated to automata on infinite words, and the verification questions are answered by resolving automata-theoretic questions, such as the language containment between two automata [32]. Furthermore, there are various *automaton characterizations* of LTL, namely automaton classes that have exactly the same expressiveness as LTL. In particular, LTL is equivalent to deterministic counter-free Muller automata [22], nondeterministic counter-free Büchi automata [7], and alternating linear (also called "very-weak" or "1-weak") automata [7].

In the branching-time setting, the automata-theoretic approach is in place, though less thoroughly used. Here, the specification, given by means of a CTL or a CTL* formula, and the system to be verified are translated to automata on infinite trees [16]. There are translations of CTL formulas to various tree-automaton classes, among which are nondeterministic Büchi (NBT) [24], alternating weak (AWT) [24], amorphous [2], and hesitant alternating linear (HALT) automata [16].

For completing the picture, we would like to also have a translation from automata to CTL. NBTs have the same expressiveness as the class of alternating Büchi tree automata (ABT) [28], where the latter is known to be as expressive as the propositional modal μ-calculus [29, 38]. Therefore both NBTs and ABTs cannot characterize CTL, since μ-calculus is strictly more expressive. Similarly, even a restricted class of ABTs, the class of alternating weak tree automata (AWT) is not suitable, as it is equivalent to alternation-free μ-calculus (AFMC) [15] which is still more expressive than CTL. The next natural candidates are the classes of alternating linear (i.e., very weak) automata (ALT) or their restriction to HALT.

Automaton characterization of CTL. It is stated in [36, p. 710, Theorem 5.11] that CTL is equivalent to ALT. Yet, a close look on the definition of alternating tree automata in [36] reveals that it is different from the standard definition, as originally given in [27].

In [36], alternating automata may use ϵ-transitions; That is, the automaton may move between states without progressing on the input tree. Moreover, while classic alternating automata have a uniform definition in the literature [27, 26, 31, 33, 34, 16, 19, 15, 18], alternating automata with ϵ-transitions have a few, slightly different, definitions; Sometimes the domain of the transition function is the set of states [38, 36] and sometimes the set of states together with the alphabet [37]; In addition, sometimes the Boolean connectives and the path quantifiers (or path directions) can be combined freely in the transition condition [37, 12] and sometimes only in a limited way [38, 12, 36].

In general, classic alternating tree automata and the various versions of alternating tree automata with ϵ-transitions are considered to be equivalent. See, for example, [37, p. 14, Proposition 1] on the equivalence of automata with and without ϵ-transitions, and [12, p. 8, Remark 9.4] for the equivalence of defining the transition condition in a general or restricted way. Yet, it turns out that for linear alternating tree automata, these subtle variations in the definitions have a significant influence.

We show that alternating linear tree automata as defined in [36] are strictly less expressive than standard alternating linear tree automata (ALT)—We prove that HALT is equivalent to CTL, and thus also to the automata of [36], while being strictly less expressive than ALT.

The translation of CTL to HALT is given in [16], and we provide the other direction. Our translation of an HALT $\mathcal{A}$ to a CTL formula φ generalizes the technique used in [19] for translating an alternating linear word automaton to an LTL formula, by handling the subtle branching possibilities of tree automata. For showing that HALT is strictly less expressive than ALT, we present an ALT $\mathcal{A}$ that allows for unboundedly many alternations between A- and E-transitions. Assuming toward contradiction an HALT $\mathcal{H}$ equivalent to $\mathcal{A}$, we construct a tree that is accepted by $\mathcal{A}$ and "exhausts" $\mathcal{H}$, showing that $\mathcal{H}$ can also accept trees not in the language of $\mathcal{A}$.

LTL∩CTL. Of special interest is the translation of the linear time setting, namely of ω-regular word languages and in particular of LTL formulas, into the branching-time setting of CTL formulas. More precisely, given an ω-regular word language L, the question is whether there exists a CTL formula φ, such that φ accepts a tree T iff all paths in T belong to L. (When such a formula φ exists, we say that it accepts the *derived language* of L [13].)

This connection between the linear and branching time logics is interesting not only from the theoretical point of view, but also from the practical one; An algorithm for translating an LTL formula into an equivalent CTL formula, when possible, may allow to use the more efficient verification algorithms of CTL formulas. Although an exponential lower bound is known for such a translation [37], it might be that in practice most of the used LTL formulas can be translated succinctly. Moreover, as claimed by Eisner and Fisman [9]: "the vast majority of properties used in practice belong to the overlap between CTL and LTL". In addition, a characterization of this class can be useful for synthesis. A related example can be found in [8], where Ehlers used an automaton characterization of LTL∩ACTL [20] to improve synthesis procedures. One may hope that a similar characterization for LTL∩CTL would help to achieve new results in that field.

Nevertheless, this highly sought connection between LTL and CTL is still missing; There is no algorithm to decide whether a given LTL formula can be translated to CTL, and it is not even known whether it is decidable. Moreover, there are currently very limited necessary conditions and sufficient conditions for an LTL formula to be expressible in CTL.

We use the HALT characterization of CTL for providing sufficient conditions and necessary conditions for LTL formulas and ω-regular languages to be expressible in CTL. Our conditions are decidable. Note, however, that there is still a gap between our necessary conditions and the sufficient conditions. Before we elaborate on it, we give a short overview on what is currently known in the area.

Expressibility comparison between LTL and CTL was first made by Lamport in 1980, showing that they are incomparable [17]. In 1983, Emerson and Halpern presented CTL* [10], which subsumes both CTL and LTL and simplifies the technical difficulties in the comparison between the linear-time and branching-time settings. In the CTL* formalism, an LTL formula φ is interpreted as the branching-time formula $A\varphi$, meaning that φ holds in all possible paths. They further studied in this paper the expressive power of different temporal logics, and in particular showed that the LTL formula $F(p \wedge Xp)$, stating that there are eventually two consequent p's in all paths, is not expressible in CTL.

In 1988, Clarke and Draghicescu (now, Browne) presented an algorithm to determine, given a CTL formula, whether it has an equivalent LTL formula [5]. They left the other direction open, while providing a necessary condition for an LTL formula to have an equivalent CTL formula; The equivalence in this condition, however, is not the standard one. Rather then defining it with respect to standard Kripke structures, as is usually done, they defined it with respect to Kripke structures with fairness constraints. They conjectured that this necessary condition is also sufficient, leaving it as another open question.

In 1994, Grumberg and Kurshan showed that if a CTL* formula defines a property over all paths then it is definable in LTL [11]. Hence, an ω-regular language is expressible in CTL (over all paths) iff it is in LTL$\cap$CTL.

In 1996, Kupferman, Safra, and Vardi showed that for every ω-regular language L, if its derived language is recognized by a nondeterministic Büchi tree automaton (NBT), then L is recognized by a deterministic Büchi word automaton (DBW) [13]. Hence, as CTL can be translated to an NBT [24], it follows that LTL $\cap$ CTL $\subseteq$ DBW.

In 2000, Maidl provided a necessary and sufficient condition for an LTL formula to have an equivalent ACTL formula, namely a formula in the universal fragment of CTL: An LTL formula is ACTL-definable iff it has an equivalent nondeterministic linear word automaton [20]. Yet, there was no algorithm to decide whether a given LTL formula satisfies the condition. Moreover, it was not clear whether LTL$\cap$ACTL is equivalent to LTL$\cap$CTL.

In 2008, Bojańczyk provided an algorithm to decide Maidl's condition, namely to decide whether a given LTL formula has an equivalent nondeterministic linear word automaton [3]. Furthermore, he proved that LTL$\cap$ACTL is a strict fragment of LTL$\cap$CTL. This result is somewhat surprising, as LTL formulas apply to all paths, while a formula in CTL$\setminus$ACTL must also quantify existentially over paths. In contrast, for the case of LTL$\cap$AFMC, universality does not limit the expressive power [15]. In other words, the intersections of LTL with the universal fragment of AFMC ($\forall$AFMC) and with AFMC are equal (LTL$\cap$AFMC = LTL$\cap\forall$AFMC).

Our conditions for LTL$\cap$CTL. We turn to elaborate on the sufficient conditions and necessary conditions that we provide for checking whether an LTL formula is definable in CTL. As LTL $\cap$ CTL $\subseteq$ DBW [13], one can concentrate on LTL formulas that are recognized by DBWs. Notice that this preliminary check is indeed decidable [15].

Our approach for correlating the linear-time and branching-time formulations is to relate the cycles of a given DBW to those of a potentially equivalent HALT. If the DBW is linear, namely has only cycles of size one, then by Maidl's condition [20], it obviously has an equivalent CTL, and even ACTL, formula. Intuitively, the core limitation of CTL in expressing a derived language of a DBW $\mathcal{D}$, stems from trees in which one path stays in some cycle of $\mathcal{D}$ while another path leaves it. The CTL formula must "decide" at the splitting node how to proceed, either with only one path or with all paths, and cannot properly handle the different paths.

Our basic necessary condition states that in order for a DBW to be CTL-recognizable, it cannot have a cycle C, such that there is a finite word u on which $\mathcal{D}$ can stay in C from some state, while also being able to proceed with u from some other state of C to a forever-accepting state q_{good}. Notice that the cycle C need not be simple, and the states from which $\mathcal{D}$ stays in C and proceeds from C need not, and obviously cannot, be the same. The condition can be decided by checking for each maximal strongly connected component X of $\mathcal{D}$, whether the intersection between the following two nondeterministic finite automata is empty: Both automata are defined over the structure of $\mathcal{D}$ and have all states of X as initial states; In the first automaton, all states of X are accepting, while in the second automaton, the forever-accepting states are accepting.

We strengthen the necessary condition, by showing that the state q_{good} need not accept every word, but can rather accept some CTL-recognizable language, provided that it satisfies some additional constraints. The strengthened condition, combined with sufficient conditions for a DBW to be CTL-recognizable, allows to inductively construct DBWs that can and DBWs that cannot be expressed in CTL. The construction's might yield an exponential sized formula w.r.t. the number of states in the DBW.

We prove the necessary condition by assuming toward contradiction that a DBW $\mathcal{D}$ that does not satisfy the necessary condition has an equivalent HALT $\mathcal{H}$. Metaphorically, every state s of $\mathcal{H}$ can be thought of as a "guard" that rejects subtrees not in the language. A run r of $\mathcal{H}$ can nondeterministically proceed from a state s to a set of states S. Since $\mathcal{H}$ is linear, the set S can only contain s and states that appear after s in the ordering of states. Now, we define a tree T that belongs to the derived language of $\mathcal{D}$, and in which there are sufficiently many "splitting" nodes. In a splitting node, the run of $\mathcal{D}$ on the tree (when $\mathcal{D}$ is considered as a deterministic tree automaton) stays in the relevant cycle for some paths and leaves the cycle for other paths. We then show that there is an accepting run r of $\mathcal{H}$ on T that "abandons" the so-far minimal state on every splitting node. That is, if r assigns to a splitting node n the set of states S, then it assigns to the next splitting node a set of states S', such that the minimal state of S does not appear in S'. Thus, there is eventually some splitting node n' that is not assigned any state. As a result, we are able to change the tree T into a tree T' that is not in the language, hanging on n' a "bad" subtree, such that a variant of the run r will nevertheless accept it—there are no longer "guards" in the node n' to reject the bad subtree.

We continue with the sufficient condition for a DBW $\mathcal{D}$ to be expressible in CTL. It also considers the cycles of $\mathcal{D}$, narrowing down the necessary condition.

It roughly requires that:

I. There is no finite word u and two distinct states q and q' of $\mathcal{D}$, such that:
   (a) the outdegree of q is bigger than one,
   (b) the run of $\mathcal{D}$ on u from q goes out of a simple cycle, and
   (c) the run of $\mathcal{D}$ on u from q' is a prefix of some accepting run.

II. There is a special "delimiting" letter that is eventually read in every accepting run, and after which $\mathcal{D}$ reaches a state whose residual language is CTL-recognizable.

Observe that given a DBW $\mathcal{D}$, the condition, except for constraint II, can be decided by examining all of $\mathcal{D}$'s simple cycles. Constraint II is obviously not known to be decidable. However, it allows the inductive construction of involved CTL-expressible DBWs—Starting with obvious languages that are known to be in CTL, such as `true`, one can inductively apply the condition, as well as other sufficient conditions, for getting a CTL-expressible DBW.

The proof of the sufficient condition is constructive, defining a CTL formula that is equivalent to a given DBW $\mathcal{D}$ that satisfies the condition. The basic idea behind its construction is the following. As long as *all* paths of the input tree correspond to the same simple cycle C of $\mathcal{D}$, a universal CTL formula, denoted by $\texttt{Cycle}(C)$, requires all next children to also follow C. Now, once *some* path goes out of C, it must be, due to the sufficient condition, via a unique "escaping word". Thus an *existential* CTL formula can "trigger" the relevant `Cycle` formulas, whenever an escaping words occurs in some path. This triggering is done until the delimiting letter appears.

Observe that the formula constructed for a DBW that satisfies the sufficient condition is, at least syntactically, in CTL\ACTL. Indeed, it is also semantically in CTL\ACTL; A simple DBW for Bojańczyk's language (see Figure 11 in Section 5.5), which is in LTL∩CTL\ACTL, satisfies the sufficient condition.

Returning to the necessary condition, we demonstrate that it easily captures some LTL formulas that are known not to be expressible in CTL, such as $F(p \wedge Xp)$ (see Figure 5). Moreover, it allows us to refute a conjecture by Clarke and Draghicescu from 1988, regarding a sufficient condition for a CTL* formula to be expressible in CTL. The conjecture roughly states that a CTL* formula is expressible in CTL (with respect to Kripke structures with fairness constraints) if it cannot distinguish between any two Kripke structures with fairness constraints that satisfy some specific properties. We refute the conjuncture by showing that the CTL* formula $E(p \vee Xp)Uq$ is not expressible in CTL (already with respect to standard Kripke structures, and thus also with respect to Kripke structures with fairness constrains), while it cannot distinguish between any two Kripke structures with fairness constraints that satisfy the conjecture's conditions.

2 Preliminaries

Given a finite alphabet Σ, a *word* over Σ is a (possibly infinite) sequence $w = w_0 \cdot w_1 \cdots$ of letters in Σ.

2.1 Trees

We consider a *tree* to be a directed unordered infinite rooted tree in the graph-theoretic sense, that is, a triple $T = \langle N, E, \epsilon \rangle$ where N is a set of nodes, E is a set of node transitions (a subset of $N \times N$) and ϵ is the root node. Each node is of indegree one (except for the root, which is of indegree zero) and of outdegree at least one. Given a tree T, we denote its set of nodes by $N(T)$. For a node n of T, we use the notation of $Succ(n)$ to describe the set of nodes that are transitioned to from n.

A *path* π in T is a finite or infinite sequence of nodes from $N(T)$ with transitions from each node to its successor in the sequence. If not said otherwise, a path always starts at the root of the tree. When we write π_i we refer to the ith member of the sequence π, starting with $i = 0$. Given a tree T and a node $n \in N(T)$, we denote by $T|_n$ the subtree of T rooted at n.

Given an alphabet Σ, a Σ-*labeled tree* is a pair $T' = \langle T, V \rangle$, where T is a tree and $V : N(T) \to \Sigma$ maps each node of T to a letter in Σ. With the notation $T'|_n$ we refer to $\langle T|_n, V \rangle$.

2.2 Kripke Structures

Let Σ be an alphabet. A *Kripke structure* over Σ is a tuple $M = \langle S, R, L \rangle$ consisting of

- a finite set S of states.
- a transition relation $R \subseteq S \times S$ such that R is left-total, i.e., $\forall s \in S\, \exists s' \in S$ such that $\langle s, s' \rangle \in R$.
- a labeling (or interpretation) function $L : S \to \Sigma$.

In a way, Kripke structures are a compact method to describe sets of trees. That is, given a state $s \in S$, the pair $\langle M, s \rangle$ stands for the *computation tree of M from s*. It is defined to be a Σ-labeled tree $\langle T, V \rangle$ with a root labeled $L(s)$. For a state s' transitioned from s, namely for s' s.t. $\langle s, s' \rangle \in R$, there is a child $n_{s'}$ of the root, that is labeled $L(s')$. For each state s'' transitioned from s', the node $n_{s'}$ has a child $n_{s''}$ labeled $L(s'')$, etc..

A *Kripke structure with fairness constraints* [5, 1] over Σ is a tuple $\langle S, R, L, \mathcal{F} \rangle$ where

- $\langle S, R, L \rangle$ is a Kripke structure over Σ.
- $\mathcal{F} \subseteq 2^S$ is a set of fairness constraints.

Let $M = \langle S, R, L, \mathcal{F} \rangle$ be a Kripke structure with fairness constraints and $\pi = s_0 s_1 \ldots$ a path in M. Let $inf(\pi)$ denote the set of states occurring infinitely often on π. Then π is *fair* iff $inf(\pi) \in \mathcal{F}$.

For two sets $\mathcal{F}$ and $\mathcal{F}'$ of fairness constraints, we say that $\mathcal{F}'$ *extends* $\mathcal{F}$ if $\mathcal{F}' = \mathcal{F} \cup F'$, where F' is a superset of some set $F \in \mathcal{F}$.

2.3 Automata

2.3.1 Word Automata

A *nondeterministic Büchi word automaton* (NBW) is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $\alpha \subseteq Q$ is the set of accepting states. In the case where $|Q_0| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\delta(q, \sigma)| \leq 1$, we say that $\mathcal{A}$ is a *deterministic Büchi word automaton* (DBW). We then use q_0 instead of Q_0 to denote the single initial state, and often refer to $\delta(q, \sigma)$ as a state rather than as a singleton set.

A *run* of $\mathcal{A}$ on a word $w = w_0 \cdot w_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $r = r_0, r_1, \cdots$ such that $r_0 \in Q_0$, and for every $i \geq 0$, we have $r_i \in \delta(r_i, w_i)$. A run r is accepting if it visits the accepting states infinitely often. Formally, $\mathit{inf}(r) = \{q \in Q \mid \text{ for infinitely many } i \in \mathbb{N}, \text{we have } r_i = q\}$, and r is accepting iff $\mathit{inf}(r) \cap \alpha \neq \emptyset$.

An automaton accepts a word if it has an accepting run on it. The language of an automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We also say that $\mathcal{A}$ *recognizes* the language $L(\mathcal{A})$. Two automata, $\mathcal{A}$ and $\mathcal{A}'$, are *equivalent* iff $L(\mathcal{A}) = L(\mathcal{A}')$.

For a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by changing the set of initial states to $\{q\}$. For a subset $Q' \subseteq Q$, where $Q' \cap Q_0 \neq \emptyset$, the *restriction of $\mathcal{A}$ to Q'*, denoted by $\mathcal{A}|_{Q'}$, is the NBW $\langle \Sigma, Q', \delta|_{Q'}, Q_0 \cap Q', \alpha \cap Q' \rangle$ where $\delta|_{Q'}$ is the restriction of δ on the domain Q'.

We often think of DBWs as graphs. A DBW $\mathcal{D}$ can be considered as a directed graph whose vertices are the states of $\mathcal{D}$, and every two vertices (states) are connected by an edge if there is a transition from one to another over some letter. Note that this graph may contain self loops, but no multiple edges. A cycle in a graph is, as usual, a finite list of vertices, each connected by an edge to its successor, where the first vertex in the list is also the last one.

Next, we give two definitions that we will use in the course of analyzing the cycles of DBWs. Let q be a state in a DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, and consider a letter $\sigma \in \Sigma$. The state that $\mathcal{D}$ reaches upon reading σ from q may have a path back to q through none, one, or several simple cycles. We formally define

$$\mathit{Cycles}(q) = \{C \mid C \text{ is a simple cycle that includes } q\}, \text{ and}$$

$$\mathit{Cycles}(q, \sigma) = \{C \mid C \text{ is a simple cycle that includes both } q \text{ and } \delta(q, \sigma) \text{ as adjacent states}\}.$$

Finally, we bring the definition of a counter-free automaton. For two states $p, q \in Q$ of an automaton $\mathcal{A}$, let $L_{p,q}$ be the set of labels of finite paths from p to q. A Büchi automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$ is called *counter-free*, if $w^m \in L_{p,p}$ implies $w \in L_{p,p}$ for all states $p \in Q$, words $w \in L_{p,p}$ and $m \geq 1$.

2.3.2 Alternating Tree Automata

We consider automata that in each step of the run can either ensure that all children move to the same state or can ensure the existence of such a child. In this regard they are symmetric, which makes sense since the trees themselves are unordered. Such symmetric automata were presented, for example, in [15].

Formally, an *alternating Büchi tree automaton* (ABT) is a tuple $\langle \Sigma, Q, q_0, \delta, \alpha \rangle$ where, as usual, Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, δ is the transition function that we define below, and $\alpha \subseteq Q$ is a set of accepting states.
We design the automaton to operate on trees. The transition function is $\delta : Q \times \Sigma \to B^+(\{E, A\} \times Q)$; Given a state $q \in Q$ and a letter $\sigma \in \Sigma$, the transition function returns a positive boolean formula that defines to which states the

automaton should send a copy of itself to, and whether it is enough to choose only one child of the processed tree and send it there (E transition), or all of the children are required (A transition). The output of δ, as every other positive boolean formula over $\{E, A\} \times Q$, is called a *transition condition.*

A *run* of an ABT $\mathcal{A}$ over a Σ-labeled tree $\langle T, V\rangle$ is a $(N(T) \times Q)$-labeled tree $R = \langle T_r, r\rangle$. Each node of T_r stands for a node of T and a state of $\mathcal{A}$. The linkage is done by the labeling function r: a node of T_r, labeled by (n, q), describes a new computation of $\mathcal{A}$ staring from its state q and operating on $T|_n$. A run should satisfy conditions described further below.

To explain these conditions we need some more definitions. For simplicity in notations, for (n', q') in $(N(T) \times Q)$, we will write $(n', q')_n$ for the node component (n'), and $(n', q')_q$ for the state component (q'). For every node m in R, we define what it means for a transition condition θ over Q to hold in m, denoted by $m \vDash \theta$. This definition is by induction on the structure of θ, where the boolean connectives, `true`, and `false` are dealt in the usual way. Further:

- $m \vDash (E, q)$ if the corresponded node $r(m)_n$ of m in T has a successor n' and there exists some successor m' of m, such that $r(m') = (n', q)$.
- $m \vDash (A, q)$ if for every successor n of $r(m)_n$, there is some successor m' of m, such that $r(m') = (n', q)$.

We can now define the conditions that a run $\langle T_r, r\rangle$ with root ϵ_r over a tree $\langle T, V\rangle$ with root ϵ should satisfy:

1. *Initial condition.* $r(\epsilon_r) = (\epsilon, q_0)$
2. *Local consistency.* Let m be a node in T_r with $r(m) = (n, q)$. Then $m \vDash \delta(q, V(n))$.

Note that by definition, a run cannot encounter a `false` transition-condition since there are no such trees that satisfy the local consistency condition. Furthermore if $\delta(q, V(n)) =$ `true` for some state q and a node n, then the local consistency condition allows the run to move to any state. We will think of reaching a `true` transition condition in some path π of the run as making the path accepting, which can be formally considered as reaching an accepting state $q_{\texttt{true}}$ with a self loop over all alphabet letters.

A run is *accepting* if all its infinite paths satisfy the Büchi condition w.r.t. α, namely, each run-path has infinitely many nodes which are labeled by a state from α.

An automaton accepts a tree if and only if it has an accepting run on it. The language of an automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of trees that $\mathcal{A}$ accepts.

For a run $\langle T_r, r\rangle$ of an ABT $\mathcal{A}$ over a labeled-tree $\langle T, V\rangle$, we say that a state q of $\mathcal{A}$ is *assigned* to a node n of T by an E statement if there is a node m in T_r that is labeled by (n, q), and it is assigned also by an A statement if in addition the parent of m satisfies the (A, q) transition condition, that is, if there are nodes m and m' of T_r such that $m \in \mathit{Succ}(m')$, $r(m) = (n, q)$, and $m' \vDash (A, q)$.

For an ABT $\mathcal{A}$ and a transition condition ρ, we denote by $\mathcal{A}^\rho$ the ABT that is obtained from $\mathcal{A}$ by changing the transition condition of the initial state q_0 to ρ, namely setting for every $\sigma \in \Sigma, \delta(q_0, \sigma) = \rho$. For a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the ABT that is obtained from $\mathcal{A}$ by setting the initial state of $\mathcal{A}$ to be q.

2.3.3 Hesitant Alternating Linear Tree Automata

Hesitant alternating linear tree automata are restricted alternating linear tree automata, which are in turn restricted alternating weak automata. We define them below in their expressiveness order.

Definition 1. An *alternating weak tree automaton* (AWT) is an ABT, in which every strongly connected component in the transition graph consists of either only accepting states or only rejecting states.

AWTs are known to have the same expressiveness as NBTs [25, 26], ABTs [28], and alternation-free μ-calculus (AFMC) [28].

Definition 2. An *alternating linear tree automaton* (ALT) is an ABT all of whose cycles in the transition graph are of size one.

Notice that in a run of an ALT, every path eventually gets stuck in some state. Therefore, the set of recurrent states of a path boils down to a singleton, implying that all acceptance conditions (Büchi, parity, Muller, etc.) provide the same expressiveness. Linear automata are also called in the literature "very weak" and "1-weak".

We further consider a restricted version of ALTs, in which the states are of three specific types, along the lines of hesitant alternating automata[1], presented in [16].

Definition 3. An ALT is *hesitant*, denoted by HALT, if every state q is either

- *transient*, where for every $\sigma \in \Sigma$, q does not appear in $\delta(q, \sigma)$; Or
- *existential*, where for every $\sigma \in \Sigma$, every appearance of q in $\delta(q, \sigma)$ is in the form of (E, q); Or
- *universal*, where for every $\sigma \in \Sigma$, every appearance of q in $\delta(q, \sigma)$ is in the form of (A, q).

2.4 Temporal Logic

2.4.1 CTL*

Formulas of CTL* are built from atomic propositions using the boolean connectives $\neg$ and $\wedge$, the linear next-time (X) and until (U) operators, and the existential path-quantifier E. The dual of E is the universal path-quantifier A defined by $A\varphi := \neg E \neg \varphi$. Additional temporal-operators can be defined using the U operator. In particular, the F ("eventually" or "finally") , G ("always" or "globally"), R ("release") and W ("weak until") temporal-operators are defined as follows: $F\varphi := trueU\varphi$, $G\varphi := \neg F \neg \varphi$, $\varphi R \psi := \neg(\neg\varphi U \neg\psi)$ and $\varphi W \psi := \varphi U \psi \vee G\varphi$.

[1]A hesitant alternating automaton (HAA) [16] need not be linear, and its acceptance condition combines the Büchi and co-Büchi conditions. Yet, restricting attention to symmetric linear HAAs, one gets our definition of an HALT.

The syntax of CTL* over a set of atomic propositions AP, defines two types of formulas: path formulas and state formulas. A formula φ is called a *state formula* if $\varphi =$

- `true`, `false`
- an atomic proposition $p \in AP$,
- $\neg\varphi'$, for a state formula φ',
- $\varphi_1 \vee \varphi_2$, for two state formulas φ_1 and φ_2,
- $A\varphi'$ or $E\varphi'$ for a path formula φ'.

In addition, a formula φ is called a *path formula* if $\varphi =$

- φ', for a state formula φ',
- $\neg\varphi'$, for a path formula φ',
- $\varphi_1 \vee \varphi_2$, for two path formulas φ_1 and φ_2,
- $X\varphi'$, for a path formula φ',
- $\varphi_1 U \varphi_2$, for two path formula φ_1 and φ_2.

Proper CTL* formulas are state formulas.

The semantics of CTL* is defined with respect to labeled trees [10]. Consider a 2^{AP}-labeled tree $\langle T, V\rangle$, and a path π of it. We say that π *satisfies* φ, denoted by $\pi \models \varphi$, when the following hold, where p stands for an atomic proposition and φ, φ_1, and φ_2 for CTL* formulas.

1. $\pi \models p$ iff $p \in V(\pi_0)$.
2. $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$.
3. $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$.
4. $\pi \models X\varphi$ iff $\pi^1 \models \varphi$.
5. $\pi \models \varphi_1 U \varphi_2$ iff there is $i \in \mathbb{N}$ s.t. $\pi^i \models \varphi_2$ and for every $j \in [0..i-1]$, $\pi^j \models \varphi_1$.
6. $\pi \models E\varphi$ iff there is a path π' starting from π_0 s.t. $\pi' \models \varphi$.

The semantics of CTL* with respect to Kripke structures relates to their computation trees. That is, a state s of a Kripke structure M satisfies a CTL* formula φ, denoted by $\langle M, s\rangle \models \varphi$, if their computation tree satisfies φ.

The semantics of CTL* with respect to a Kripke structure with fairness constraints is defined using only the fair paths of the structure. Thus, the satisfaction relation, denoted by $\models_f$, is defined inductively for all states s and fair paths π of M using the same clauses as in the case of ordinary trees except for replacing clause 6 by the following clause.

6'. $\langle M, s\rangle \models_f E\varphi$ iff there is a fair path π' starting from s s.t. $\pi' \models \varphi$.

2.4.2 LTL

An LTL formula φ is a CTL* path-formula that does not contain path quantifiers, thus, we can specify in LTL requirements on the paths themselves without an inner branching along the path. In the context of CTL*, we treat φ as the state formula $A\varphi$. For example, the LTL formula FGp is interpreted as the CTL* formula $AFGp$ and specifies that in every path, there will eventually be p in every state.

2.4.3 CTL

A CTL formula is a CTL* state formula such that each path-quantifier is followed immediately by a temporal-operator. For example, the formula $AFEGp$ specifies that on each path of the tree, eventually there will be a node that starts a path of ps.

ACTL is a fragment of CTL, in which only the A path quantifier is allowed, and negations are only allowed on atomic propositions.

2.5 Connecting Automata and Temporal Logic

In temporal logic, formulas are interpreted over a set AP of atomic propositions. For instance, the formula $ApUq$ is over the atomic propositions $\{p, q\}$. On the other hand, ABTs operate on Σ-labeled trees. Since we want to relate between ABTs and temporal-logic formulas, we take Σ to be 2^{AP}. In other words, the alphabet of the discussed ABT is the power set of the atomic propositions of the discussed formula.

We say that a tree automaton $\mathcal{A}$ and a CTL formula φ are *equivalent* if the set of trees accepted by $\mathcal{A}$ is equal to the set of trees that satisfy φ. In other words, if for every Σ-labeled tree $\langle T, V\rangle$, it holds that $\langle T, V\rangle \in L(\mathcal{A})$ iff $\langle T, V\rangle \models \varphi$.

2.5.1 Derived and Dual-Derived Languages

For an ω-regular language L, the *derived language* of L, denoted by $L\Delta$, is the set of trees all of whose paths belong to L [14]. Analogously, we define the *dual derived language of* L, denoted by $L\nabla$, to be the set of trees in which there is at least one path in L.

It turns out that for every ω-regular language L, there is an ABT for its dual derived language $L\nabla$. Such an ABT is achieved by "upgrading" an NBW that accepts L, as described in the following proposition.

Proposition 4 ([16])**.** *Let $\mathcal{B}$ be an NBW. Then we can extend $\mathcal{B}$ to an ABT $\mathcal{A}$, s.t. $L(\mathcal{A}) = L(\mathcal{B})\nabla$. Further, the automaton $\mathcal{A}$ has the same structure as $\mathcal{B}$, only adding E over the transitions.*

Note that the ABT guaranteed from Proposition 4 does not use its full alternation power; A transition condition of a state q over a letter σ is of the form $\delta_{\mathcal{A}}(q, \sigma) = \bigvee_{q' \in \delta_{\mathcal{B}}(q,\sigma)}(E, q')$, using only $\vee$s and Es.

Upfront, it may seem that adding As over the transitions of an NBW $\mathcal{B}$ results in an ABT for $L(\mathcal{A})\Delta$. This is, however, not the case. It does hold for DBWs, as shown in [14], but not for general NBWs.

Proposition 5 ([14])**.** *Let $\mathcal{D}$ be a DBW. Then we can extend $\mathcal{D}$ to an ABT $\mathcal{A}$, s.t. $L(\mathcal{A}) = L(\mathcal{D})\Delta$. Further, the automaton $\mathcal{A}$ has the same structure as $\mathcal{D}$, only adding A over the transitions.*

2.5.2 ω-regular $\cap$ CTL = LTL $\cap$ CTL $\subseteq$ DBW

It was shown in [11] that the language of a CTL* formula is derivable iff it is expressible in LTL. Therefore, if an ω-regular language is expressible in CTL (hence in CTL*) it is also expressible in LTL. That is, ω-regular $\cap$ CTL $=$ LTL $\cap$ CTL.

Kupferman and Vardi showed in [15] that an ω-regular language L can be characterized by a DBW iff $L\Delta$ can be characterized by an alternation-free mu-calculus (AFMC). As CTL is subsumed by AFMC [21], we have the following.

Corollary 6 ([15])**.** *Let φ be an LTL formula of a language L, equivalent to some CTL formula. Then, there is a DBW that recognizes L.*

In other words, we know that LTL $\cap$ CTL $\subseteq$ DBW. Notice that the sets are not equal. Moreover, we have LTL $\cap$ CTL $\subsetneq$ LTL $\cap$ DBW. For example, the LTL formula $F\,(p \wedge Xp)$ is not expressible in CTL [10] (as well as Corollary 15), while expressible by a DBW (Figure 5).

3 CTL is Equivalent to HALT

As elaborated on in the introduction, it is stated in [36, p. 710, Theorem 5.11] that CTL is equivalent to ALT. Yet, a close look on the definition of alternating tree automata in [36] reveals that it is different from the standard definition, as originally given in [27]. We show that alternating linear tree automata as defined in [36], are strictly less expressive than standard alternating linear tree automata (ALT)—We prove that HALT is equivalent to CTL, and thus also to the corresponding automata of [36], and that they are strictly less expressive than ALT.

The translation of CTL to HALT is established in [16], and is given in Section 3.1 for the sake of completeness. We provide in Section 3.2 the other direction, getting their equivalence. In Section 3.3, we show that HALT indeed tightly characterizes CTL; Relaxing either the linearity or the hesitant obligations of the automata results in strictly more expressive automata.

Theorem 7. *CTL formulas and HALTs have the same expressiveness.*

Proof. Lemmas 8 and 11 below give the two directions of the claimed equivalence. □

The proofs of both Lemma 8 and Lemma 11 are constructive, and generate CTL formulas and HALTs whose size is linear in each other.

3.1 CTL to HALT

In [16], a construction was presented for translating a CTL formula ψ to an automaton that runs on ordered trees. Moreover, the construction is also suitable, almost as is, for translating ψ to an HALT, which runs on unordered trees.

Lemma 8 ([16])**.** *Every CTL formula can be translated to an equivalent HALT.*

First, we assume w.l.o.g. that ψ is in negation normal form. Moreover, we can ignore the case in which $\psi \in \{\texttt{true}, \texttt{false}\}$ because then the construction is trivial. Let $cl(\psi)$ be the set of all subformulas of ψ that are state-formulas. We define the ALT $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \psi, \delta, \alpha\rangle$, where α is the set of all formulas in $cl(\psi)$ whose outermost temporal operator is AR or ER, and for every state in $cl(\psi)$ and letter $\sigma \in 2^{AP}$, δ is defined as follows.

- $\delta(q, \sigma) = \texttt{true}$ if $q \in \sigma$ and $\texttt{false}$ otherwise.
- $\delta(\neg q, \sigma) = \texttt{true}$ if $q \notin \sigma$ and $\texttt{false}$ otherwise.
- $\delta(\varphi_1 * \varphi_2, \sigma) = \delta(\varphi_1, \sigma) * \delta(\varphi_2, \sigma)$, for $* \in \{\vee, \wedge\}$
- $\delta(EX\varphi, \sigma) = (E, \varphi)$
- $\delta(AX\varphi, \sigma) = (A, \varphi)$
- $\delta(E\varphi_1 U\varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee (\delta(\varphi_1, \sigma) \wedge (E, E\varphi_1 U\varphi_2))$
- $\delta(A\varphi_1 U\varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee (\delta(\varphi_1, \sigma) \wedge (A, A\varphi_1 U\varphi_2))$
- $\delta(E\varphi_1 R\varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee (E, E\varphi_1 R\varphi_2))$
- $\delta(A\varphi_1 R\varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee (A, A\varphi_1 R\varphi_2))$

Note that the constructed automaton is indeed an HALT.

3.2 HALT to CTL

We show that every HALT can be translated to an equivalent CTL formula, adapting the technique used in [19] for translating a linear alternating word automaton into an LTL formula. The challenge in the adaptation is how to properly generalize the technique from word automata to tree automata. Indeed, as explained in the Introduction and will be demonstrated in Section 3.3, small variations in the definition of alternating linear tree automata determine whether or not they are equivalent to CTL.

Construction. Consider an HALT $\mathcal{A} = \langle \Sigma, Q = \{q_0, q_1, \ldots, q_n\}, q_0, \delta, \alpha\rangle$. Since $\mathcal{A}$ is linear, we can assume w.l.o.g. that the transitions are ascending, namely that for every $\sigma \in \Sigma$ and $i \in [0..n]$, $\delta(q_i, \sigma)$ includes q_j only if $i \le j$, and that $q_n = q_{\texttt{true}}$. For every $\sigma \in \Sigma$, we define the CTL formula $\psi_\sigma = (\bigwedge_{p\in\sigma} p) \wedge (\bigwedge_{p\notin\sigma} \neg p)$, intuitively meaning that a σ-labeled node is read.

Consider a state q_i and a letter σ, and let $\theta = \delta(q_i, \sigma)$ be the transition condition of q_i on σ. Notice that since $\mathcal{A}$ is hesitant, q_i is either transient, existential, or universal. It is thus possible to present θ as follows, where $\theta_{i,\sigma}$ and $\theta'_{i,\sigma}$ are transition conditions that contain states only from $\{q_{i+1}, q_{i+2}, \ldots, q_n\}$.

$$\theta = \begin{cases} \theta'_{i,\sigma} & q_i \text{ is transient} \\ ((E, q_i) \wedge \theta_{i,\sigma}) \vee \theta'_{i,\sigma} & q_i \text{ is existential} \\ ((A, q_i) \wedge \theta_{i,\sigma}) \vee \theta'_{i,\sigma} & q_i \text{ is universal} \end{cases}$$

For every state q_i, we define below the two CTL formulas $\varphi_{i,stay}$ and $\varphi_{i,leave}$, intuitively meaning that the run stays in q_i or leaves it, respectively. They are based on the above formulas $\theta_{i,\sigma}$ and $\theta'_{i,\sigma}$, respectively, after their recursive translation from transition conditions into CTL formulas (ToCTL). The latter is formally defined afterwards.

$$\varphi_{i,stay} = \bigvee_{\sigma\in\Sigma} \psi_\sigma \wedge \texttt{ToCTL}(\theta_{i,\sigma}) \qquad \varphi_{i,leave} = \bigvee_{\sigma\in\Sigma} \psi_\sigma \wedge \texttt{ToCTL}(\theta'_{i,\sigma}).$$

The ToCTL function, translating a transition condition θ of $\mathcal{A}$ into a CTL formula, is defined in the expected way by induction on the structure of θ. The non-trivial translation is of the atomic subformula q_i, for $i \in [0..n]$. Let ρ_1 and ρ_2 be subformulas of θ.

- $\texttt{ToCTL}(\texttt{true}) = \texttt{true}$.
- $\texttt{ToCTL}(\texttt{false}) = \texttt{false}$.
- $\texttt{ToCTL}(\rho_1 \vee \rho_2) = \texttt{ToCTL}(\rho_1) \vee \texttt{ToCTL}(\rho_2)$.
- $\texttt{ToCTL}(\rho_1 \wedge \rho_2) = \texttt{ToCTL}(\rho_1) \wedge \texttt{ToCTL}(\rho_2)$.
- $\texttt{ToCTL}((E, q_i)) = EX\,\texttt{ToCTL}(q_i)$.
- $\texttt{ToCTL}((A, q_i)) = AX\,\texttt{ToCTL}(q_i)$.
- $\texttt{ToCTL}(q_i) = \begin{cases} \texttt{true} & i = n \\ \varphi_{i,leave} & q_i \text{ is transient} \\ E\varphi_{i,stay}U\varphi_{i,leave} & q_i \text{ is existential and } q_i \notin \alpha \\ E\varphi_{i,stay}W\varphi_{i,leave} & q_i \text{ is existential and } q_i \in \alpha \\ A\varphi_{i,stay}U\varphi_{i,leave} & q_i \text{ is universal and } q_i \notin \alpha \\ A\varphi_{i,stay}W\varphi_{i,leave} & q_i \text{ is universal and } q_i \in \alpha \end{cases}$

Notice that the above definitions are circular, defining the ToCTL function on top of the $\varphi_{i,stay}$ and $\varphi_{i,leave}$ formulas, and vice versa. Yet, this is exactly the recursion in the definition, presenting no problem—The translation of a state q_i is defined via $\varphi_{i,stay}$ and $\varphi_{i,leave}$ on top of states q_j, for $j > i$. The recursion ends with q_n, which is translated to true.

An Example. Consider the three-states HALT $\mathcal{A} = \langle\{\{p\}, \emptyset\}, \{q_0, q_1, q_{\texttt{true}}\}, q_0, \delta, \{q_{\texttt{true}}\}\rangle$ with the following definition of δ: $\delta(q_0, \{p\}) = (A, q_0) \vee (E, q_2)$; $\delta(q_1, \{p\}) = (E, q_{\texttt{true}})$; and $\delta(q_{\texttt{true}}, \{p\}) = \delta(q_{\texttt{true}}, \emptyset) = \texttt{true}$. As always, any unspecified transition is false.

We start by expressing the CTL formula for q_1. Since it is transient $\texttt{ToCTL}(q_1) = \varphi_{1,leave} = \psi_{\{p\}} \wedge \texttt{ToCTL}((E, q_{\texttt{true}})) = \psi_{\{p\}} \wedge EX\texttt{ToCTL}(q_{\texttt{true}}) = \psi_{\{p\}} \wedge EX\texttt{true} = \psi_{\{p\}}$. We proceed to the universal non-accepting state q_0, for which $\texttt{ToCTL}(q_0) = A\varphi_{0,stay}U\varphi_{0,leave} = A\psi_{\{p\}} \wedge \texttt{ToCTL}(\texttt{true})\,U\,(\psi_{\{p\}} \wedge \texttt{ToCTL}((E, q_2))) = A\psi_{\{p\}} \wedge \texttt{true}\,U\,(\psi_{\{p\}} \wedge EX\texttt{ToCTL}(q_2)) = A\psi_{\{p\}}U(\psi_{\{p\}} \wedge EX\psi_{\{p\}})$.

Correctness. We continue with showing that the construction defined above indeed produces for every HALT $\mathcal{A}$ an equivalent CTL formula. For $i \in [0..n]$, let $Q_i = \{q_i, q_{i+1}, \ldots, q_n\}$ and let $\mathcal{A}_i$ denote the ALT that is derived from $\mathcal{A}$ by changing the initial state to q_i. We will show by induction on i, starting with $i = n$ and proceeding toward $i = 0$, that $\mathcal{A}_i$ is equivalent to $\texttt{ToCTL}(q_i)$. The induction step, going from i to $i-1$, will be shown by induction on structure of the transition condition θ of the state q_i. We start with a lemma on the correctness of this structural induction.

Lemma 9. *Let $i \in [0..n-1]$, and assume that for every $k > i$, the HALT $\mathcal{A}_k$ is equivalent to* $\texttt{ToCTL}(q_k)$*. Then, for every transition condition $\rho \in B^+(\{A, E\} \times Q_{i+1})$ and labeled tree T, we have that $\mathcal{A}_i^\rho$ accepts T iff $T \models \rho$.*

Proof. By induction on the structure of ρ.

Base cases:

- $\rho = \texttt{true}$: By definition, T satisfies $\texttt{true}$, and $\mathcal{A}_i^{\texttt{true}}$ accepts every tree.
- $\rho = \texttt{false}$: By definition, T does not satisfy $\texttt{false}$, and $\mathcal{A}_i^{\texttt{false}}$ does not accept any tree.
- $\rho = (A, q_k)$, for $k > i$: Then $\texttt{ToCTL}(\rho) = AX\texttt{ToCTL}(q_k)$.

  Assume $T \models AX\texttt{ToCTL}(q_k)$. We will build an accepting run $R = \langle T_r, r\rangle$ of $\mathcal{A}_i^\rho$ on T. Let ϵ be the root of T and ϵ_r the root of R. We define the labeling of ϵ_r to be the root of T and the initial state of $\mathcal{A}_i$, namely $r(\epsilon_r) = (\epsilon, q_i)$, and the successors of ϵ_r to be a copy of the successors of ϵ, namely $Succ(\epsilon_r) = \{m_l | l \in Succ(\epsilon)\}$.

  We label every successor of ϵ_r with its corresponded node in T, along with the state q_k, namely $r(m_l) = (l, q_k)$. Trivially, the ρ-local consistency is achieved, namely $\epsilon_r \models (A, q_k)$. By the assumption that $T \models AX\texttt{ToCTL}(q_k)$ and the CTL semantics, we conclude that every successor of ϵ satisfies $\texttt{ToCTL}(q_k)$. Hence, by the assumption that $\mathcal{A}_k$ is equivalent to $\texttt{ToCTL}(q_k)$, for every successor l of ϵ, $\mathcal{A}_k$ accepts $T|_l$. Therefore, for each such l there is an accepting run R_l of $\mathcal{A}_k$ on $T|_l$. Recall that for each such l, there is a corresponded m_l in the run. We concatenate R_l under m_l, and get an accepting run of $\mathcal{A}_i^\rho$ on T, as required.

  For the other direction, suppose the existence of an accepting run R of $\mathcal{A}_i^\rho$ on T. Let ϵ be the root of T, l a successor of ϵ, and ϵ_r the root of R. By the local consistency of ϵ_r, we have $\epsilon_r \models (A, q_k)$, implying that there is a successor m_l of ϵ_r such that $r(m_l) = (l, q_k)$. As before, we denote by R_l the run induced by m_l. Notice that R_l is an accepting run of $\mathcal{A}_k$ on $T|_l$. Hence, by the assumption that $\mathcal{A}_k$ is equivalent to $\texttt{ToCTL}(q_k)$, we have $T|_l \models \texttt{ToCTL}(q_k)$. Therefore, $T \models AX\texttt{ToCTL}(q_k)$, as required.
- $\rho = (E, q_k)$: Analogous to the previous case.

Induction step:

- $\rho = \rho_1 \vee \rho_2$:
  $T \models \texttt{ToCTL}(\rho_1 \vee \rho_2)$ iff
  $T \models \texttt{ToCTL}(\rho_1) \vee \texttt{ToCTL}(\rho_2)$ iff
  $T \models \texttt{ToCTL}(\rho_1)$ or $T \models \texttt{ToCTL}(\rho_2)$ iff, by the induction assumption,
  $A_i^{\rho_1}$ accepts T or $A_i^{\rho_2}$ accepts T iff
  $A_i^{\rho_1 \vee \rho_2}$ accepts T.
- $\rho = \rho_1 \wedge \rho_2$:
  It is analogous to the previous case, yet the last implication, namely the

claim that ($A_i^{\rho_1}$ accepts T and $A_i^{\rho_2}$ accepts T) iff ($A_i^{\rho_1 \wedge \rho_2}$ accepts T), deserves an explanation.

The right-to-left implication is straightforward, as the accepting run R of $A_i^{\rho_1 \wedge \rho_2}$ is also an accepting run of $A_i^{\rho_1}$ and of $A_i^{\rho_2}$.

As for the left-to-right implication, let R_1 and R_2 be accepting runs of $A_i^{\rho_1}$ and $A_i^{\rho_2}$, respectively. We assume that R_1 and R_2 do not contain common nodes, as otherwise we can duplicate them. We define an accepting run R of $A_i^{\rho_1 \wedge \rho_2}$ by merging R_1 and R_2: the root of R has as successors all the successors of R_1's root and of R_2's root.

□

We continue with the main correctness lemma.

Lemma 10. *For every $i \in [0..n]$, the CTL formula* $\texttt{ToCTL}(q_i)$ *is equivalent to* $\mathcal{A}_i$.

Proof. We prove the lemma by induction on i, starting with $i = n$, and proceeding toward $i = 0$.

The base case is trivial: $L(\texttt{ToCTL}(q_n)) = L(\texttt{true}) = L(\mathcal{A}_n)$.

In the induction step, we assume the claimed equivalence for every $j \in [i{+}1..n]$, and prove it for i. There are five cases to consider:

- q_i is universal and $q_i \in \alpha$.
- q_i is universal and $q_i \notin \alpha$.
- q_i is existential and $q_i \in \alpha$.
- q_i is existential and $q_i \notin \alpha$.
- q_i is transient.

We show only the first case, where q_i is a universal accepting state. The other cases are proven analogously.

Recall that $\texttt{ToCTL}(q_i) = A\varphi_{i,stay} W \varphi_{i,leave}$.

I. $L(\mathcal{A}_i) \subseteq L(\texttt{ToCTL}(q_i))$: Let T be a Σ-labeled tree s.t. $\mathcal{A}_i$ accepts T via a run $R = \langle T_r, r \rangle$ with root ϵ_r, and let π be an infinite path in T. We need to show that either i) there is $k \in \mathbb{N}$ such that $T|_{\pi_k} \models \varphi_{i,leave}$ and for every $j < k$, $T|_{\pi_j} \models \varphi_{i,stay}$, or ii) for every $k \in \mathbb{N}$, $T|_{\pi_k} \models \varphi_{i,stay}$.

Let σ be the labeling of T's root, namely of π_0. Notice that $T \models \psi_\sigma$, while for every $\sigma' \neq \sigma$, $T \not\models \psi_{\sigma'}$. By the local consistency of R in its root, we have $\epsilon_r \models \delta(q_i, \sigma) = ((A, q_i) \wedge \theta_{i,\sigma}) \vee \theta'_{i,\sigma}$. In the case that $\epsilon_r \models \theta'_{i,\sigma}$, R is also an accepting run of $\mathcal{A}_i^{\theta'_{i,\sigma}}$ on T. Then, by Lemma 9, $T \models \texttt{ToCTL}(\theta'_{i,\sigma})$, implying that $T \models \varphi_{i,leave}$, and we are done.

Otherwise, $\epsilon_r \models (A, q_i) \wedge \theta_{i,\sigma}$. Similarly, since $\epsilon_r \models \theta_{i,\sigma}$ it holds that $T \models \varphi_{i,stay}$. Moreover, since $\epsilon_r \models (A, q_i)$, we learn that there is some m in $Succ(\epsilon_r)$ such that $r(m) = (\pi_1, q_i)$. Note that $R|_m$ is an accepting run of $\mathcal{A}_i$ on $T|_{\pi_1}$, so we can repeat using the above argument and learn that $\varphi_{i,stay}$ always holds along π, unless "interrupted" by $\varphi_{i,leave}$, as required.

II. $L(\texttt{ToCTL}(q_i)) \subseteq L(\mathcal{A}_i)$: Let T be a Σ-labeled tree s.t. $T \vDash \texttt{ToCTL}(q_i)$. We will present an accepting run of $\mathcal{A}_i$ on T. As $T \vDash \texttt{ToCTL}(q_i)$, in particular $T \vDash \varphi_{i,stay} \vee \varphi_{i,leave}$. That is, there is $\sigma \in \Sigma$ s.t. $T \vDash \psi_\sigma$, and either $T \vDash \texttt{ToCTL}(\theta_{i,\sigma})$ or $T \vDash \texttt{ToCTL}(\theta'_{i,\sigma})$.

If $T \vDash \texttt{ToCTL}(\theta'_{i,\sigma})$ then by Lemma 9, $\mathcal{A}_i^{\theta'_{i,\sigma}}$ accepts T by some run R. By the definition of $\delta(q_i, \sigma)$, which is $((A, q_i) \wedge \theta_{i,\sigma}) \vee \theta'_{i,\sigma}$, we have that R is also an accepting run of $\mathcal{A}_i$ on T, and we are done.

Otherwise, $T \vDash \texttt{ToCTL}(\theta_{i,\sigma})$, and by Lemma 9, $\mathcal{A}_i^{\theta_{i,\sigma}}$ accepts T via some run R_ϵ. We will extend R_ϵ to an accepting run R of $\mathcal{A}_i$ on T. To do so, we hang additional successors under the root of R_ϵ for satisfying (A, q_i).

Let l be a successor of the root of T. Recall that T satisfies the weak until condition for all paths, and $\varphi_{i,leave}$ has not been fulfilled yet. Therefore, $T|_l \vDash \varphi_{i,stay} \vee \varphi_{i,leave}$. Using again the above argument, we have $\mathcal{A}_i^{\theta_{i,\sigma}}$ accepts $T|_l$ via some run R_l. We hang R_l as a successor of the root of R_ϵ. We proceed in this way, handling every successor l of T's root, then handling the successors of each such node l, etc.. For every path of T, the procedure continues indefinitely or until a node satisfies the $\texttt{ToCTL}(\theta'_{i,\sigma})$ condition.

We claim that R is an accepting run of $\mathcal{A}_i$ on T. Let π_r be a path of R. Then π_r either eventually becomes a path of an accepting run, in the case that the above procedure reached a node of T that satisfies $\texttt{ToCTL}(\theta'_{i,\sigma})$, or it remains for ever in q_i. As q_i is an accepting state, in both cases π_r satisfies the Büchi condition, and therefore R is accepting.

□

As a special case of Lemma 10, considering the initial state q_0, we get the correction of our construction.

Lemma 11. *Every HALT can be translated to an equivalent CTL formula.*

3.3 Tightness

We show that both the linearity and the hesitant properties of an HALT are indeed essential for the equivalence with CTL. That is, we prove that non-linear hesitant AWT (HAWT) and non-hesitant ALT are more expressive than CTL. (In Section 2.3.3, we only defined the hesitant property w.r.t. an ALT. Its definition w.r.t an AWT is the same.)

The inequality HALT<HAWT is straightforward. Consider the DBW $\mathcal{D}$, presented in Figure 5. By Proposition 5, adding As to the transitions on the $\mathcal{D}$, one gets an hesitant AWT accepting the derived language of D. As shown in Corollary 15, the derived language of $\mathcal{D}$ has no equivalent CTL formula, and thus by Lemma 11, has no equivalent HALT.

For showing that HALT<ALT, we provide an ALT $\mathcal{A}$, as depicted in Figure 1, and prove that it does not have an equivalent HALT. Intuitively, an HALT cannot follow the unboundedly many alternations between A- and E-transitions that $\mathcal{A}$ allows. Assuming toward contradiction an equivalent HALT $\mathcal{H}$, we define a tree T in the language of $\mathcal{A}$ that "exhausts" $\mathcal{H}$, in the sense that we can change some subtree of T and get a tree T' that is not in the language of $\mathcal{A}$, while it is nevertheless accepted by $\mathcal{H}$.

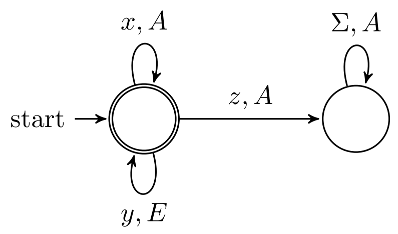


Figure 1: An ALT $\mathcal{A}$ that has no equivalent HALT.

Theorem 12. *Hesitant alternating linear tree automata (HALT) are strictly less expressive than alternating linear tree automata (ALT).*

Proof. Consider the ALT $\mathcal{A}$ appearing in Figure 1, and let $L = L(\mathcal{A})$. Assume toward contradiction that there exists an HALT $\mathcal{H}$ that recognizes L. Let $Q = \{s_0, s_1, \ldots, s_m\}$ be the set of $\mathcal{H}$'s states. We build a tree T that is accepted by $\mathcal{A}$, and then show how an accepting run of $\mathcal{H}$ on T can be changed into an accepting run of $\mathcal{H}$ on some tree T' that is not accepted by $\mathcal{A}$.

We assume w.l.o.g. that the last state with respect to the linear order of $\mathcal{H}$, namely s_m, accepts every tree, and that it is the only state that accepts every tree. Thus, for a state $s \in Q \setminus \{s_m\}$, it holds that $\overline{L(\mathcal{H}^s)} \neq \emptyset$. There are two options, either $\overline{L(\mathcal{H}^s)} \cap \overline{L} = \emptyset$ or $\overline{L(\mathcal{H}^s)} \cap \overline{L} \neq \emptyset$. We define the set X (respectively, Y) to contain all the states in Q that satisfy the first option (respectively, the second option). Then, for every state $s \in X$, there exists a tree $T^x_s \in \overline{L(\mathcal{H}^s)} \cap L$, and for every state $s \in Y$, there exists a tree $T^y_s \in \overline{L(\mathcal{H}^s)} \cap \overline{L}$.

The tree T (see Figure 2): T starts with a node denoted by n_0 that is labeled x. For every state s in X, there is under n_0 the subtree T^x_s. There are also two additional identical subtrees of n_0, denoted by n^l_0 and n^r_0. Since they are identical, it is sufficient to describe the former. n^l_0 is labeled y and has for every $s \in Y$, the subtree T^y_s. In addition, it has another subtree, starting with a node denoted by n_1 that is labeled x. The subtrees of n_1 are similar to those of n_0—for every state s in X, there is under n_1 a subtree T^x_s, and two identical subtrees, denoted by n^l_1 and n^r_1. n^l_1 is labeled y etc... there are m such similar levels of T, until having the node n_m, which is labeled x. Then, under n_m there is a singled-path tree labeled x in all its nodes.

Notice that T is indeed in L—It can be shown that $T|_{n_i} \in L$ using induction on i, staring from m towards 0. Obviously, $T|_{n_m}$ is in L. For the induction step, suppose $T|_{n_{i+1}}$ is in L. Therefore $T|_{n^l_i}$ is also in L, since it is labeled y and has a subtree in L. Recall that by definition, $T|_{n^r_i}$ is identical to $T|_{n^l_i}$, therefore also in L. It is left to show that the rest of the children of n_i are in L. Namely, that T^x_s is in L for every $s \in X$, which indeed holds by the definition of T^x_s. Therefore, $T \in L$.

The tree T': T' is identical to T, except for having in n_m the single path z^ω as a subtree. Notice that T' is not in L—We use induction again, showing that $T|_{n_i} \notin L$ for i staring from m towards 0. Obviously, $T|_{n_m} \notin L$. Assume that $T|_{n_{i+1}}$ is not in L. Note that all of its siblings are T^y_s for $s \in Y$, which by definition are not in L. Therefore n^l_i has no child in L, and thus $T|_{n^l_i} \notin L$.

Then, n_i is labeled x but has a child that is not in L, implying that $T|_{n_i} \notin L$. Hence, $T' \notin L$.

From an accepting run of $\mathcal{H}$ on T to an accepting run of $\mathcal{H}$ on T': Consider a run r of $\mathcal{H}$ that accepts T, and let S' be the set of states that r assigns to n_0^l. For each state s in S', we check whether it is assigned to n_0^l by an E or by an A statement. If s is assigned to n_0^l only by an E statement, there is another accepting run r' of $\mathcal{H}$ on T that is identical to r, except for not assigning s to n_0^l, while assigning s to n_0^r. This is indeed the case, since n_0^l and n_0^r start identical subtrees. Thus, we may assume that in r, all states that are assigned to n_0^l are assigned by an A statement.

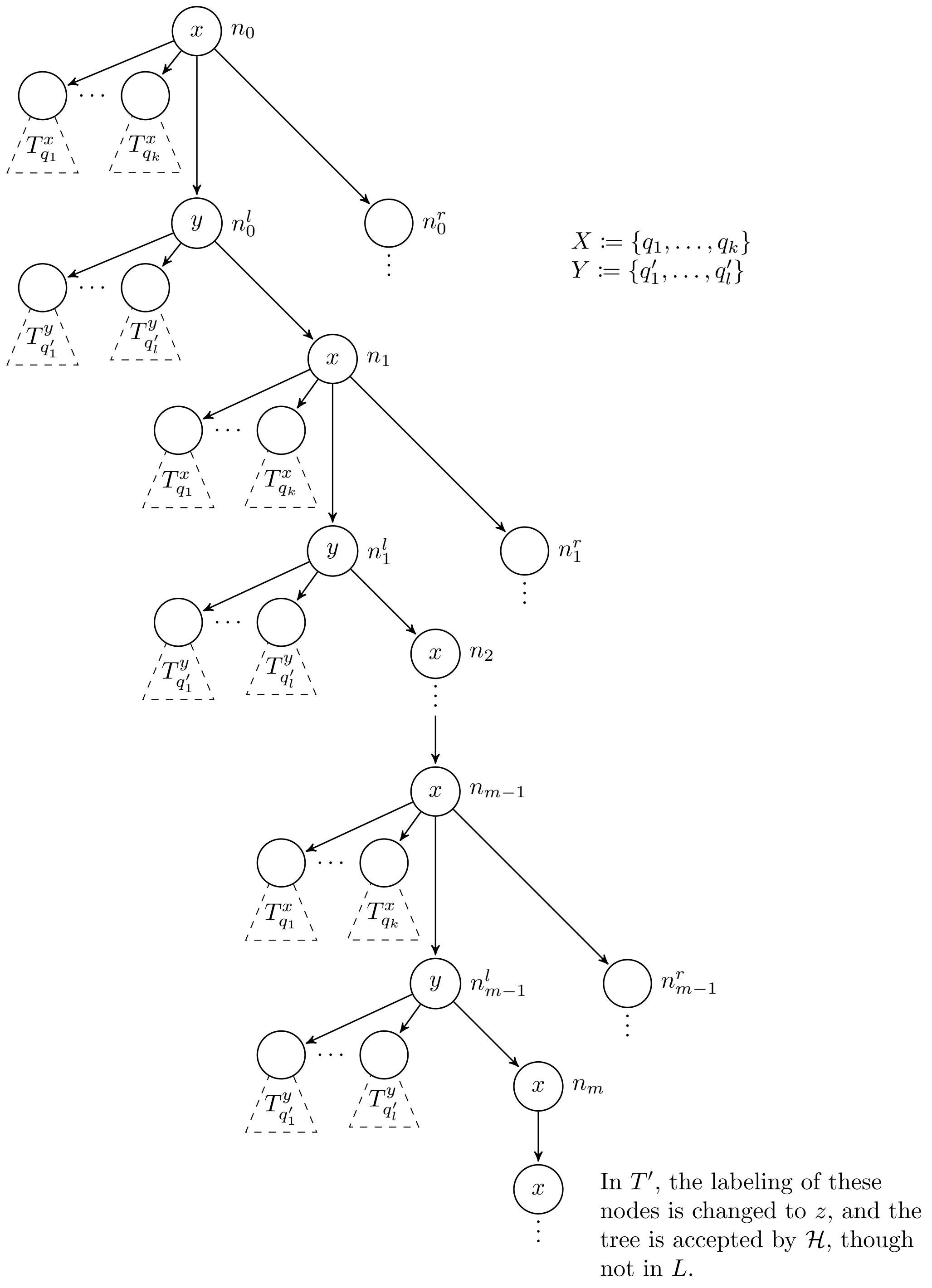
Consider a state s that is assigned to n_0^l by an A statement. Then by definition, s is assigned to all other siblings of n_0^l. Hence, $s \notin X$, as otherwise, there would have been a sibling of n_0^l that is the root of a tree T_s^x that is rejected by $\mathcal{H}^s$, which would have implied that the run r is rejecting.

By the same argument, we get that if a state s is assigned to n_1 by an A statement, it implies that $s \notin Y$.

Let s be the minimal state assigned by r to n_0. Since $\mathcal{H}$ is hesitant, s is either transient, existential or universal. As we assumed that s is assigned to n_0^l by an A statement it implies, by the above argument, that $s \notin X$. Further, it also implies that s is universal, since only s has a transition to s ($\mathcal{H}$ is linear and s is the minimal state assigned to n_0). Therefore, if s is assigned to n_1, it is assigned by an A statement (by a transition from s, which is the minimal state assigned to n_0^l), implying that $s \notin Y$. Hence, since $Q = X \cup Y \cup \{s_m\}$ and $s \notin X \cup Y$, it follows that $s = s_m$. In other words, the minimal state of n_0 is not assigned to n_1.

Applying the above argument by induction on i, we get an accepting run r of $\mathcal{H}$ on T, such that for every $i \in [0..m-1]$, the node n_{i+1} is not assigned any of the states in $\{s_0, s_1, \ldots, s_i\}$. In particular, the node n_m is not assigned any of $Q \setminus \{s_m\}$.

Thus, since $\mathcal{H}^{s_m}$ accepts every tree, we arrived to an accepting run of $\mathcal{H}$ on T', which does not belong to L. $\Box$



n_0
n_0^l
n_0^r
n_1
n_1^l
n_1^r
n_2
n_{m-1}
n_{m-1}^l
n_{m-1}^r
n_m
$T_{q_1}^x$
$T_{q_k}^x$
$T_{q_1'}^y$
$T_{q_l'}^y$
$X \coloneqq \{q_1, \ldots, q_k\}$
$Y \coloneqq \{q_1', \ldots, q_l'\}$
In T', the labeling of these nodes is changed to z, and the tree is accepted by $\mathcal{H}$, though not in L.

Figure 2: The tree T used in Theorem 12

4 Necessary Conditions for LTL $\cap$ CTL

There are currently very limited techniques for showing that an LTL formula cannot be expressed in CTL. Emerson and Halpern showed in [10] that the LTL formula $F(p \wedge Xp)$ is not expressible in CTL. Their proof is quite long and tedious and uses a complicated inductive argument that required about 2 journal pages to present. Furthermore, the used technique is not easily generalized to other examples [4].

Later on in [5], Clarke and Dragahicescu (now Browne) presented some necessary condition for an LTL formula to be expressible in CTL, yet not with respect to standard Kripke structures, but with respect to Kripke structure with fairness constraints. (We cite their condition as Theorem 29). They conjectured that their necessary condition is also sufficient, however we refute it in Section 6.

We provide in this section general techniques for showing that an LTL formula is not expressible in CTL, using the HALT characterization of CTL. The techniques can be used for easily showing that the LTL formula $F(p \wedge Xp)$ is not expressible in CTL, as well as for refuting the aforementioned conjecture of Clarke and Dragahicescu.

We start with a basic necessary condition, which we will strengthen in Section 4.2. Recall that LTL $\cap$ CTL $\subseteq$ DBW [24, 13]. Hence, it is enough to check the CTL-expressibility of a given DBW.

4.1 The Basic Condition

Our basic necessary condition states that in order for a DBW $\mathcal{D}$ to be CTL-recognizable, it cannot have a cycle C, such that there is a finite word u on which $\mathcal{D}$ can stay in C from a state q_1 and also proceed to a forever-accepting state from some other state q_2 of C.

Theorem 13. *Let $\mathcal{D}$ be a DBW, and L the derived language of $\mathcal{D}$. If there is a state q of $\mathcal{D}$ s.t. the following hold, then L is not expressible in CTL.*

- *There is a finite word $y \in \Sigma^+$ that is an infix of the labels on the path from q back to itself (see Figure 3).*
- *$\mathcal{D}^q$ accepts every word that starts with y.*
- *$L(\mathcal{D}^q) \subsetneq \Sigma^\omega$.*

Proof. Assume toward contradiction that L is expressible in CTL. Then by Theorem 7, there is an HALT $\mathcal{A}$ that recognizes L.

We shall describe a tree T that belongs to L, as depicted in Figure 4, and via which we will show that $\mathcal{A}$ also accepts some tree T' not in L, reaching a contradiction. We will do that by analyzing an accepting run of $\mathcal{A}$ on T, and show how it can be altered to be an accepting run of $\mathcal{A}$ on T'. Intuitively speaking, an HALT has a hard time following simultaneously two paths where one stays in a cycle while the other leaves it, especially when the paths share a word y, which "confuses" the HALT.

Let $v \in \Sigma^*$ be the finite word s.t. $\mathcal{D}$ reaches q upon reading it. Let x and z be two finite words such that when reading xyz, $\mathcal{D}$ completes a cycle from q back to itself , and let $w \in \Sigma^\omega$ a word rejected by $\mathcal{D}^q$ (See Figure 3 for a sketch of $\mathcal{D}$).

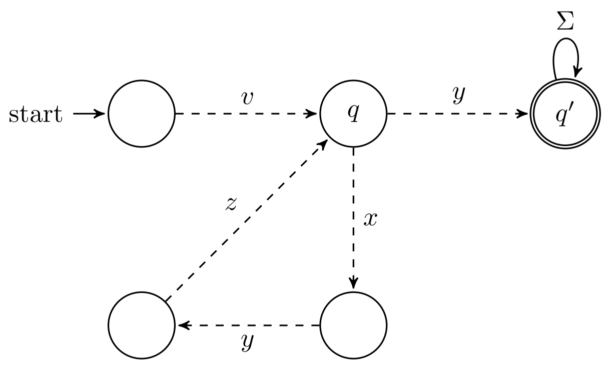


Figure 3: A schematic drawing of DBWs that cannot be expressed in CTL.

The tree T (see Figure 4): We define Y to be the set of tress whose $|y|$ first labels along all paths form the word y. Let Q be the set of states of $\mathcal{A}$ and let $m = |Q|$. Let $B \subseteq Q$ be the states of $\mathcal{A}$ that reject some tree in Y. That is, $B = \{s \in Q \mid \text{ there exists a tree } T_s \in Y \setminus L(\mathcal{A}^s)\}$. Note that B is not empty, as otherwise $\mathcal{A}$ would have accepted the singled-path tree $vxyzw$, which is not in L. An accepting run can be accomplished by changing an accepting run on a word that starts with vx; the prefix of such a run can be easily continued since every state should accept trees that start with y along all of their paths.

T starts with a path labeled $vxyz$. We denote by n_0 the last node of that path. For every state s in B, there is under n_0 a subtree T_s that starts with y along all of its paths and is rejected by $\mathcal{A}^s$. There are also two additional identical subtrees of n_0, denoted by n_0^l and n_0^r. Since they are identical, it is sufficient to describe the former. n_0^l starts with a path labeled xyz that ends in a node denoted by n_1. The subtrees of n_1 are similar to those of n_0—for every state s in B, there is under n_1 a subtree T_s that starts with y and is rejected by $\mathcal{A}^s$, and two identical subtrees, denoted by n_1^l and n_1^r. n_1^l starts with a path labeled xyz that ends at n_2 etc... there are m such similar levels of T, until having the node n_m. Then, under n_m there is a member of Y as a subtree, for example the single path that begins with y. Notice that T is indeed in L.

The tree T': T' is identical to T, except for having in n_m the single path w as a subtree. Notice that T' is not in L, since it has a path labeled $v(xyz)^{m+1}w$, which is not accepted by $\mathcal{D}$.

Analyzing accepting runs of $\mathcal{A}$ on T: Consider a run r of $\mathcal{A}$ that accepts T, and let S' be the set of states that r assigns to n_0^l. For every state s in S', we check whether it is assigned to n_0^l by an E or by an A statement. If s is assigned to n_0^l only by an E statement, there is another accepting run r' of $\mathcal{A}$ on T that is identical to r, except for not assigning s to n_0^l, while assigning s to n_0^r. This is indeed the case, since n_0^l and n_0^r start identical subtrees. Thus, we may assume that in r, all states that are assigned to n_0^l are assigned by an A statement.

Consider a state s that is assigned to n_0^l by an A statement. Then by definition, s is assigned to all other siblings of n_0^l. Hence, $s \notin B$, as otherwise, there would have been a sibling of n^l_0 that is the root of a tree T_s that is rejected by $\mathcal{A}^s$, which would have implied that the run r is rejecting. Thus, $\mathcal{A}^s$ accepts every tree in Y. Therefore we can assume that after reading y, a run of $\mathcal{A}^s$ can accept every tree (otherwise, we change $\mathcal{A}$ to an HALT $\mathcal{A}'$ that extends $\mathcal{A}$ with $|y|-1$ new states, namely $\{s'_1, \ldots, s'_{|y|-1}\}$, having the transitions $s \xrightarrow[A]{y_1} s'_1 \xrightarrow[A]{y_2} \ldots \xrightarrow[A]{y_{|y|-1}} s'_{|y|-1} \xrightarrow[A]{y_{|y|}} \texttt{true}$. Notice that $\mathcal{A}$ and $\mathcal{A}'$ recognize the same language).

Deducing an accepting run of $\mathcal{A}$ on T': We now describe how to change the accepting run r of $\mathcal{A}$ on T to an accepting run of $\mathcal{A}$ on T'. Let s_0 be the minimal state assigned by r to n_0. We claim that there is an accepting run r' of $\mathcal{A}$ on T that is identical to r, except for not assigning s_0 to n^l_0. Indeed, if r does not assign s_0 to n^l_0 then r' is simply r. If r assigns s_0 to n^l_0 only by an E statement, r' assigns s_0 to n^r_0 instead. Otherwise, by the above argument, s_0 does not belong to B and has a transition to $\texttt{true}$ after reading the word y. Hence, the run r' can lead s_0 to $\texttt{true}$ when reading y, before reaching n_0, implying that no state that is assigned to n_0 can assign s_0 to n^l_0. (Recall that the automaton is linear and s_0 is the minimal state.)

Applying the above argument by induction on i, we get an accepting run r of $\mathcal{A}$ on T, such that for every $i \in [1..m-1]$, the node n^l_i is not assigned any of the states in $\{s_0, s_1, \ldots, s_i\}$. In particular, the node n^l_{m-1}, and therefore also the node n_m, is not assigned any state!

Thus, we can have an accepting run of $\mathcal{A}$ on T', which does not belong to L. □

Notice that the condition provided in Theorem 13 can be decided by checking for each maximal strongly connected component X of the given DBW $\mathcal{D}$, whether the intersection between the following two nondeterministic finite automata is empty: Both automata are defined over the structure of $\mathcal{D}$ and have all states of X as initial states; In the first automaton, all states of X are accepting, while in the second automaton, the forever-accepting states are accepting.

4.2 A Stronger Condition

We narrow down the necessary condition, by extending the families of DBWs that are shown not to be expressible in CTL. Recall that the basic condition, as defined in Theorem 13, handled DBWs in which some finite word y appears both in a cycle that includes a state q, and as a path from q to a state q', s.t $\mathcal{D}^{q'}$ accepts every word.

We strengthen Theorem 13, by allowing $\mathcal{D}^{q'}$ to recognize a richer variety of languages. In particular, $\mathcal{D}^{q'}$ can recognize any CTL-recognizable language, provided that it satisfies some constraints, as defined in the following theorem.

The motivation for considering states whose residual language is CTL-expressible, is to allow the combination of the necessary condition and sufficient conditions, such as the ones described in Section 5. Then, one can inductively construct DBWs that cannot be expressed in CTL. See, for example, Corollary 18.

Theorem 14 (Theorem 13 Extended)**.** *Let $\mathcal{D}$ be a DBW and L the derived language of $\mathcal{D}$. If there is a state q of $\mathcal{D}$ s.t. the following hold, then L is not expressible in CTL.*

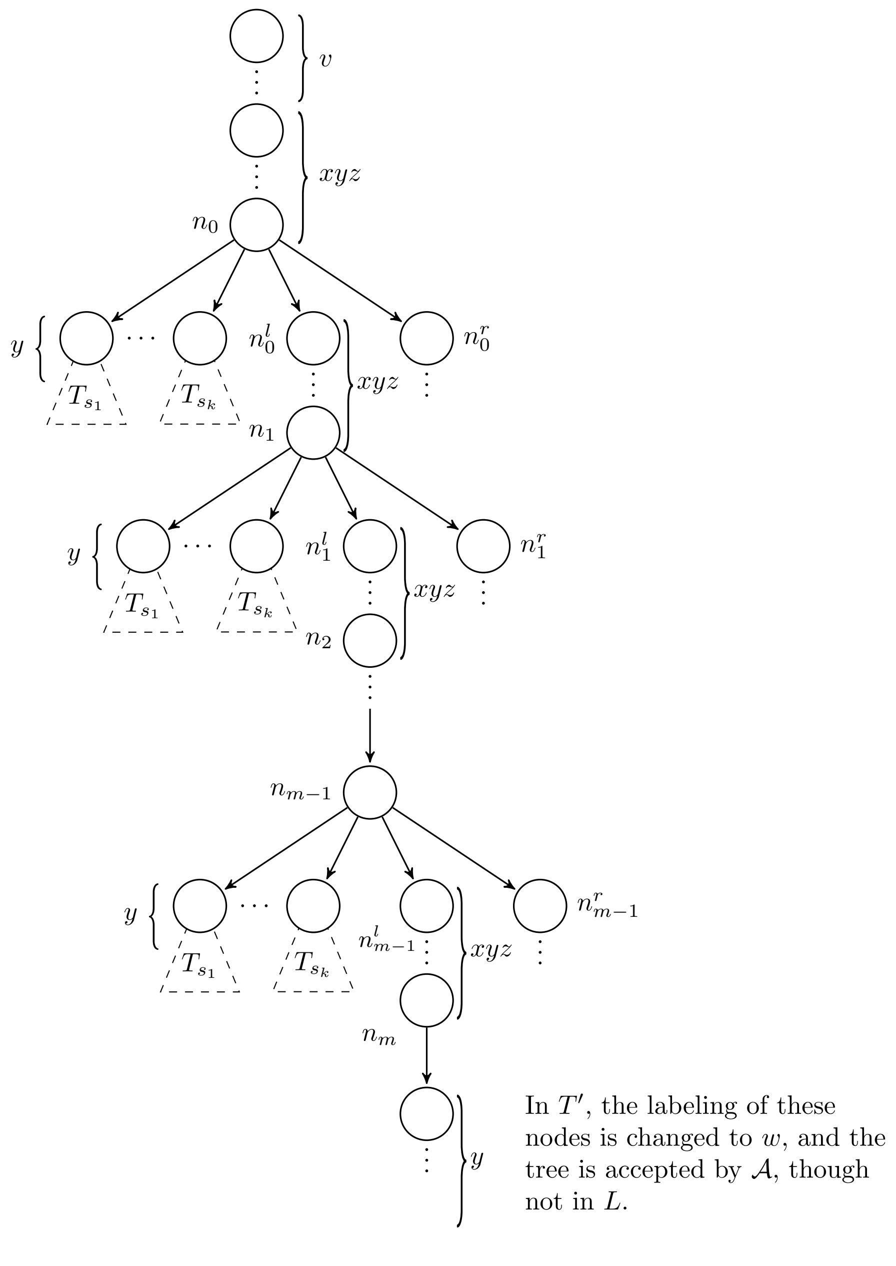


Figure 4: The tree T used in Theorem 13

- *There is a cycle from q back to itself labeled xyz, for finite words $x, z \in \Sigma^*$ and $y \in \Sigma^+$.*
- *The run of $\mathcal{D}^q$ on y reaches a state q', s.t. $\mathcal{D}^{q'}$ has an equivalent CTL formula.*
- *There exists a word $w \notin L(D^q)$, such that for every $i \in \mathbb{N}$, $z(xyz)^i w \in L(D^{q'})$.*
- *For every word $y' \in L(\mathcal{D}^{q'})$ and every $i \in \mathbb{N}$, $z(xyz)^i yy' \in L(D^{q'})$.*

Notice that Theorem 13 is a special case of Theorem 14, taking q' to accept every word in Σ^ω. Then, $\mathcal{D}^{q'}$ is equivalent to the CTL formula `true`, the third condition falls back to be $L(\mathcal{D}^q) \subsetneq \Sigma^\omega$, and the fourth condition obviously holds.

Proof. We explain below how the proof of Theorem 13 is changed in order to suit Theorem 14.

Similarly to the proof of Theorem 13, we assume toward contradiction that L is expressible in CTL, and therefore there is an HALT $\mathcal{A}$ recognizing it. We describe a tree T that belongs to L, and via which we will show that $\mathcal{A}$ also accepts some tree T' not in L, reaching a contradiction.

Recall the construction of the tree T in the original proof. It used a set B of $\mathcal{A}$'s states and a set Y of trees. We denoted by m the number states in $\mathcal{A}$. Further, we hanged under the nodes n_i ($i \in [0..m-1]$) trees denoted by T_s, where s is a state in the set B, and $T_s \in Y \setminus L(\mathcal{A}^s)$. Under the node n_m we hanged a tree in Y. The tree T' resulted by replacing the subtree hanged under n_m with the singled-path tree labeled w.

We keep this construction, but redefine Y to be the set of trees in which all paths satisfy the following two conditions: i) the first $|y|$ labels form the word y, and ii) the labels of the suffix from the ($|y|+1$)'s position form a word in $L(\mathcal{D}^{q'})$.

Note that T is in L. Indeed, each path has the form of $v(xyz)^i yy'$ for some $y' \in L(\mathcal{D}^{q'})$ and $i \in \mathbb{N}$, which is in L (see Figure 3, and recall that though $L(D^{q'})$ is no longer Σ^ω, $y' \in L(D^{q'})$). Moreover, note that T' is not in L.

Analyzing accepting runs of $\mathcal{A}$ on T: Consider a run r of $\mathcal{A}$ that accepts T. As mentioned in the original proof, we can assume that in r, all states that are assigned to n_0^l are assigned by an A statement.

By the same argument that was used in the original proof, we get that a state s that is assigned to n_0^l by an A statement must not be in B. Thus, $\mathcal{A}^s$ accepts every tree in Y. Therefore, conceptually, a run of $\mathcal{A}^s$ can accept every tree in $L(D^{q'})\Delta$ after reading y. If, technically, it is not the case, we can change $\mathcal{A}$ to an equivalent HALT $\mathcal{A}'$, as described below, such that the above conceptual claim holds also technically.

Recall that $D^{q'}$ has an equivalent CTL formula, therefore by Theorem 7 there is an HALT $\mathcal{A}_{q'}$ equivalent to $\mathcal{D}^{q'}$. We denote its initial state by s'. We change $\mathcal{A}$ to an HALT $\mathcal{A}'$ that extends $\mathcal{A}$ with $|y|-1$ new states, namely $\{s'_1, \ldots, s'_{|y|-1}\}$, having the transitions $s \xrightarrow[A]{y_1} s'_1 \xrightarrow[A]{y_2} \ldots \xrightarrow[A]{y_{|y|-1}} s'_{|y|-1} \xrightarrow[A]{y_{|y|}} s'$. Notice that $\mathcal{A}$ and $\mathcal{A}'$ recognize the same language.

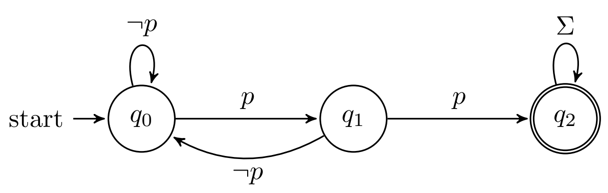


Figure 5: A DBW accepting the language of $F(p \wedge Xp)$, not expressible in CTL.

Deducing an accepting run of $\mathcal{A}'$ on T': Analogously to the arguments in the original proof, the node n^l_{m-1} and therefore also the node n_m, is not assigned any state of $\mathcal{A}$. Note that for $i \in [0..m-1]$ we prove that the node n^l_i is not assigned to any of the states $\{s_0, s_1, \ldots, s_i\}$ by changing the run of $\mathcal{A}$ on T in a way that it goes to the state s' when reading y, before reaching n^l_i. We should explain how the run continues from there and why it is still accepting. To this end, note that in each iteration s' is assigned to a subtree all of whose paths are of the form $z(xyz)^j yy'$ for some $y' \in L(\mathcal{D}^{q'})$ or of the form $z(xyz)^j w$, for $j \in \mathbb{N}$. Either way, we assumed it to be in $L(\mathcal{D}^{q'}) = L(\mathcal{A}^{s'})$. In particular, there is an accepting run of $\mathcal{A}^{s'}$ on that subtree. Therefore, we can have an accepting run of $\mathcal{A}$ on T', which does not belong to L. ☐

4.3 Examples

We present below some examples of using the basic and stronger necessary conditions, for showing that ω-regular languages cannot be expressed in CTL.

The first example deals with the language of the LTL formula $F(p \wedge Xp)$. It was already proven to be out of CTL in [10], but with much effort. We get it as a simple corollary of Theorem 13.

Corollary 15 ([10])**.** *The LTL formula $F(p \wedge Xp)$ is not expressible in CTL.*

Proof. Consider the DBW $\mathcal{D}$, presented in Figure 5. Note that it satisfies the necessary condition set by Theorem 13, since:

- $\mathcal{D}^{q_1}$ rejects some word $(\neg p^\omega)$.
- $\mathcal{D}^{q_1}$ accepts every tree that starts with p. Furthermore, p also occurs on a cycle from q_1 back to itself.

Therefore, it is not expressible in CTL. ☐

Note that in Theorem 13, it does not matter whether the states in the cycle are accepting or not. In the following corollary, for example, all of the states are accepting. Moreover, this corollary will be used in Section 6 for refuting an open conjuncture presented in [5].

Corollary 16. *The LTL formula $(p \wedge Xp)Rq$ is not expressible in CTL.*

Proof. Proven similarly to Corollary 15. Consider the DBW, $\mathcal{D}$, presented in Figure 6. The word $\{\neg p, \neg q\}$ is rejected by $\mathcal{D}$. In addition, $\mathcal{D}^{q_1}$ accepts every word that starts with $\{p, q\}$, while $\{p, q\}$ can also be found on the cycle $q_1 \to q_0 \to q_1$. ☐

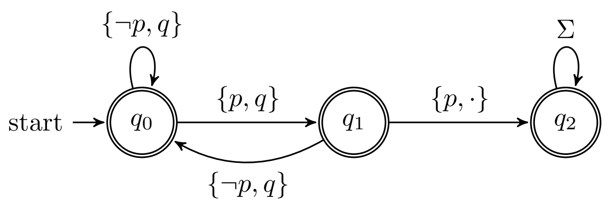


Figure 6: A DBW accepting the language of $(p \wedge Xp)Rq$, not expressible in CTL.

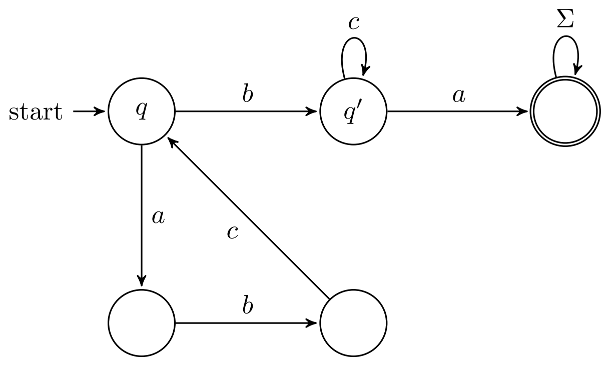


Figure 7: A DBW accepting the language $(abc)^*bc^*a\Sigma^\omega$, not expressible in CTL.

Observe that in Theorem 13, the required cycle from q back to itself need not be simple. An example for such a case is demonstrated in the following corollary.

Corollary 17. *The language $L =$ "all paths belong to $(abc)^*bc^*a\Sigma^\omega$" is not definable in CTL.*

Proof. Consider the DBW $\mathcal{D}$ in Figure 7. The word bca appears on a cycle from q back to itself (the cycle $abcabc$). In addition, $\mathcal{D}^q = \mathcal{D}$ accepts every word that starts with bca, and rejects the word a^ω. □

The added value of the stronger condition (Theorem 14) is demonstrated by the following example, not covered by the basic condition. It also demonstrates how one can inductively use the stronger necessary condition together with sufficient conditions—The language of $\mathcal{D}^{q'}$ is in CTL by the sufficient condition presented in Section 5.4, and therefore, due to Theorem 14, the following language L is not in CTL.

Corollary 18. *The language $L =$ "all paths belong to $(abc)^*b((b+c)^*a)^\omega$" is not definable in CTL.*

Proof. Consider the DBW $\mathcal{D}$ in Figure 8 and the states q and q' of $\mathcal{D}$. Denote $x = a$, $y = b$ and $z = c$. Then y appears on a cycle from q back to itself and also is the path from q to q'.

Note that by the sufficient condition of Section 5.4, $\mathcal{D}^{q'}$ is almost linear and thus has an equivalent CTL formula. The rest of the stronger conditions hold as well, therefore by Theorem 14, there is no equivalent CTL formula for $\mathcal{D}$. □

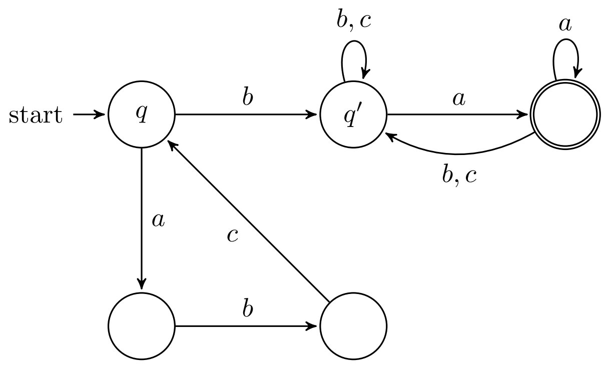


Figure 8: A DBW accepting the language $(abc)^*b((b+c)^*a)^\omega$, not expressible in CTL.

Analogously, it can be shown that the formula $F(p \wedge Xp) \wedge GFp$ is not expressible in CTL.

5 Sufficient Conditions for LTL $\cap$ CTL

The main sufficient condition narrows down the necessary condition by requiring, among other things, that the DBW leaves cycles with unique words. Its correctness proof is constructive, defining an equivalent CTL formula. The resulting formula contains both universal and existential path quantification, and indeed, the sufficient condition is shown to capture languages in LTL$\cap$(CTL$\setminus$ACTL).

In Section 5.4, we provide another, simpler, sufficient condition that can be combined with the main condition, for allowing the inductive construction of more involved CTL-expressible DBWs. See, for example, Corollary 28. Moreover, by combining the sufficient conditions and the necessary condition, one can define more involved DBWs that cannot be expressed in CTL. See, for example, Corollary 18.

5.1 The Main Condition

DBWs that satisfy the condition are required to have some special segment, which we dub the "decisive part", containing the initial state and no accepting states. A run of the DBW can leave the decisive part only upon reading a special delimiting letter e, going to a state that has an equivalent CTL formula. In addition, in the decisive part, the way out of every cycle should be unique.

We start by defining formally what we mean by a "way out of a cycle". Notice that we distinguish between two variants of going out of a cycle; Before and after completing a full cycle.

Definition 19 (Escaping Words)**.** Consider a DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, a simple cycle C of $\mathcal{D}$, and a finite word $w = w_1 \dots w_l$.

1. We say that $\mathcal{D}$ *leaves* C from a state q of C via the word w if the following hold:
   - The outdegree of q is greater than one;

   - The run of $\mathcal{D}^q$ on $w_1 \dots w_{l-1}$ visits states only in C, and does not visit any state more than once; and
   - The run of $\mathcal{D}^q$ on w reaches a state not in C.

2. We say that $\mathcal{D}$ *leaves C early* from a state q of C via the word w if in addition to the requirements presented in (1), for every $j \in [1..l{-}1]$, the outdegree of the state $\delta(q, w_1 \dots w_j)$ is one. In this case, we call the word w an *early escaping word.*

3. We say that $\mathcal{D}$ *leaves C cyclically* from a state q_i of C via the word w if in addition to the requirements presented in (1), the word $w_1 \dots w_{l-1}$ completes C, namely $\delta(q, w_1 \dots w_{l-1}) = q$. In this case, we call the word w a *cyclic escaping word.*

Note that an escaping word can be both early and cyclic, in the case that the cycle contains a single state with outdegree above 1.

Given a DBW $\mathcal{D}$, a cycle C of $\mathcal{D}$, and a state q of C, we define the following sets of escaping words.

$$EarlyEscape(C, q) = \{w \in \Sigma^* \mid \mathcal{D} \text{ leaves } C \text{ early from } q \text{ via } w\},$$

$$EarlyEscape(C) = \bigcup_{q \in C} EarlyEscape(C, q),$$

$$EarlyEscape(q) = \bigcup_{\{C \ \mid \ C \in Cycles(q)\}} EarlyEscape(C, q),$$

Similarly, we define the sets $CyclicEscape(C, q)$, $CyclicEscape(C)$, and $CyclicEscape(q)$.

We continue with the formal definition of the sufficient condition. We define the constraints that a DBW should satisfy in order to be "decisive", and then show that every decisive DBW can be translated to CTL.

Definition 20. A DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ is *decisive* if there is a subset $Q' \subseteq Q$ that contains the initial state of $\mathcal{D}$, such that $\mathcal{D}|_{Q'}$ satisfies the following:

1. It is counter-free.

2. There is a letter $e \in \Sigma$ s.t. for every state $q \in Q'$, the automaton $\mathcal{D}^{\delta(q,e)}$ has an equivalent CTL formula.

3. For every letter $\sigma \neq e$ and a state $q \in Q'$ it holds that $\delta(q, \sigma) \in Q'$.

4. Q' contains no accepting states.

5. If $\mathcal{D}$ leaves a simple cycle $C \subseteq Q'$ early from a state q via a finite word w then for every state $q' \neq q$ of Q', we have $\delta(q', w) = \emptyset$.

The subset Q' is dubbed the *decisive part of $\mathcal{D}$*.

Observe that except for the second constraint, it is decidable to check whether a given DBW satisfies the constraints. Regarding the second constraint, we obviously do not know to decide it, as such a decision procedure will solve the question of whether a DBW is CTL-expressible. The idea behind it is to allow the inductive construction of involved CTL-expressible DBWs—Starting with obvious languages that are known to be in CTL, such as `true`, one can inductively apply the above condition, as well as other sufficient conditions, such as the one of Section 5.4, for getting a CTL-expressible DBW. (See, for example, Corollary 28.)

We briefly explain the intuitive reason for requiring each of the other constraints. The counter-free constraint follows the known equivalence of LTL and counter-free NBWs [7] and non-counting languages [23]. The uniqueness of the escaping words allows the equivalent CTL formula to "synchronize" whenever some path of the input tree leaves a simple cycle. The delimiting letter e allows the formula to constantly wait for an escaping word w until e occurs; Without it, the escaping word would have been awaited even after going out of the decisive part. Regarding the limitation of not having accepting states in the decisive part, we believe that it can be relaxed, and we partially address it in the additional condition, provided in Section 5.4.

The formal claim on the correctness of the main sufficient condition is the following.

Theorem 21. *Every decisive DBW has an equivalent CTL formula.*

We start the proof of Theorem 21 with defining the CTL formula that corresponds to a given decisive DBW, and afterwards prove that it is indeed equivalent to the DBW. For better readability, we use the typographical convention of `word` when using a word that refers to a CTL formula.

5.2 The Equivalent CTL Formula

Consider a decisive DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ over an alphabet Σ with a decisive part $Q' \subseteq Q$. We define below a corresponding CTL formula ψ, which we will show to be equivalent to $\mathcal{D}$.

For every state p in Q', we have a formula $\texttt{State}(p)$ that "describes" it, based on the simple cycles to which p belongs. We also define such formulas for the states in $\delta(Q', e)$. In addition, we have the formula `Orientation` that occasionally "synchronizes" a node of the input tree with the corresponding state of Q'.

A formula for each state in $Q' \cup \delta(Q', e)$: We define below the formula of a state p, using some subformulas that will be defined afterwards.

We begin with the states in $\delta(Q', e)$. Let p be a state in Q'. We know that every state transitioned by e from p has an equivalent CTL formula. In other words, $\mathcal{D}^{\delta(p,e)}$ is equivalent to φ_p, for some CTL formula φ_p. We simply define $\texttt{State}(\delta(p, e)) := \varphi_p$.

We continue with a state $p \in Q'$. It consists of three different parts explained below. In general, the first part ensures local validity, the second deals with letters that do not appear on cycles of p, and the last handles cycles.

$$\begin{aligned}
\texttt{State}(p) \quad := \quad & \texttt{LookAhead}(p) \\
\wedge \quad & \bigvee_{\{\sigma \ \mid \ Cycles(p,\sigma)=\emptyset\}} \sigma \wedge AX \ \texttt{State}(\delta(p,\sigma)) \\
\wedge \quad & \bigvee_{\{\sigma \ \mid \ Cycles(p,\sigma)\neq\emptyset\}} \sigma \wedge \Big(\bigvee_{C \in Cycles(p,\sigma)} \texttt{EarlyLeave}(C,p) \vee \texttt{Cycle}(C)\Big)
\end{aligned}$$

`LookAhead`(p): This formula guarantees that the labels of the next nodes of each path in the input tree match $\mathcal{D}$, by "looking ahead" a fixed number of steps. This fixed number is defined to be the length of the longest cycle in Q' plus one, and is denoted by l.

For example, consider a state p that has the following direct paths: $p \xrightarrow{a} \cdot \xrightarrow{a,c} \cdot \xrightarrow{b} \cdot$; $p \xrightarrow{a} \cdot \xrightarrow{b} \cdot \xrightarrow{a} \cdot$; and $p \xrightarrow{b} \cdot \xrightarrow{a} \cdot \xrightarrow{a,b} \cdot$. Then $\texttt{LookAhead}(p) = (a \vee b) \wedge (a \rightarrow AX((a \vee b \vee c) \wedge AX((a \vee c) \rightarrow AX(b) \wedge b \rightarrow AX(a)) \wedge (b \rightarrow AX(a \rightarrow AX(a \vee b)))$.

`EarlyLeave`(p, C): Consider a state p that belongs to a cycle $C \subseteq Q'$. We define a formula that checks if there is a path of the tree on which $\mathcal{D}$ leaves C early from p. For ease of notations, for a word $w \in \Sigma^*$, we define

$$\texttt{PathExists}(w) = EX\,(w_1 \wedge EX\,(w_2 \cdots \wedge EX\,w_{|w|})),$$

and use it to define

$$\texttt{EarlyLeave}(C, p) = \bigvee_{w \in EarlyEscape(C,p)} \texttt{PathExists}(w).$$

`Cycle`(C): This formula deals with trees all of whose paths start with a complete cycle of C. Basically, the formula validates that every path stays in the cycle, until encountering some cyclic-escaping path. Consider a cycle $C = \Sigma_0\Sigma_1 \cdots \Sigma_{k-1}$ that starts and ends in a state p, where $\Sigma_i \subseteq \Sigma$, for every $i \in [0..k-1]$. For defining the formula $\texttt{Cycle}(C)$, we first define the following formulas.

- For every $i \in [0..k-1]$, a formula that corresponds to Σ_i, namely

$$\varphi_{\Sigma_i} = \bigvee_{\sigma \in \Sigma_i} \sigma.$$

- For every $x \in [0..k-1]$ and CTL formula ξ, a formula that promises for every path that if it completes the cycle C starting from the x-th position of C, then its k-step descendants, in all paths, satisfy the formula ξ. Formally,

$$\texttt{IfThen}(C, \xi)_x = \varphi_{\Sigma_x} \rightarrow AX(\varphi_{\Sigma_{x+1 \bmod k}} \rightarrow \cdots AX(\varphi_{\Sigma_{x+k-1 \bmod k}} \rightarrow AX\xi)).$$

- We use the `IfThen` formula for defining a formula stating that if all paths complete a cycle along C, starting from the x-th position of C, then they also take one more step on C.

$$\texttt{Stay}(C)_x = \texttt{IfThen}(C, \Sigma_x)_x,$$

- Next, we gather the instances of the above formula for all positions of the cycle C, such that the resulting formula is indifferent to the starting position.

$$\texttt{Stay}(C) = \bigwedge_{x=0}^{k-1} \texttt{Stay}(C)_x,$$

- A formula stating that some path takes its way out of C after completing a cycle.

$$\texttt{CyclicLeave}(C) = \bigvee_{w \in CyclicEscape(C)} \texttt{PathExists}(w),$$

Then,

$$\texttt{Cycle}(C) = A\ \texttt{Stay}(C)\ U\ \texttt{CyclicLeave}(C).$$

`Orientation`: The formula allows to synchronize the current node of the read tree with the corresponding state of $\mathcal{D}$. Due to the uniqueness of the escaping words, it can trigger the synchronization whenever an escaping word occurs at *some* path of the read tree.

Once detecting an escaping word, `Orientation` triggers the formula of the state on which the paths diverge. It is done as long as the letter e is not read, meaning that the run of the DBW on the tree is still in a state in Q'.

If an early escaping word $w\sigma$ is detected, we can be sure that the $|w|$ upcoming positions in the tree share the same future, reaching together the same state. Therefore, we trigger the formula of that state. On the other hand, that property cannot be guaranteed in case of a cyclic escaping word, as $\mathcal{D}$ may leave earlier on some paths without completing a cycle. Therefore we enforce the formula only on paths that complete the cycle, using `IfThen`.

Note that for every cyclic escaping word $w\sigma$ of a cycle C, there is a unique state of C from which the word is escaping. Therefore, we can use the notation $\texttt{IfThen}(C,\cdot)_{w\sigma}$ to describe $\texttt{IfThen}(C,\cdot)_x$.

$$\texttt{Orientation} = A(\texttt{GoodEarlyEscapes} \wedge \texttt{GoodCyclicEscapes})Ue,$$

where

$$\texttt{GoodEarlyEscapes} = \bigwedge_{q \in Q'} \bigwedge_{w\sigma \in EarlyEscape(q)} PathExists(w\sigma) \to (AX)^{|w|}\texttt{State}(\delta(q,w)),$$

and

$$\texttt{GoodCyclicEscapes} = \bigwedge_{q \in Q'} \bigwedge_{C \in Cycles(q)} \bigwedge_{w\sigma \in CyclicEscape(C,q)} PathExists(w\sigma) \to \texttt{IfThen}(C, \texttt{State}(\delta(q,w)))_{w\sigma}.$$

The overall formula: for a decisive DBW $\mathcal{D}$ with an initial state q_0, the corresponding CTL formula is

$$\psi = \texttt{State}(q_0) \wedge \texttt{Orientation}$$

5.3 Correctness

In this subsection we prove the correctness of our construction. We start with some propositions on the structure of decisive DBWs.

We first observe that due to the counter-free property, cyclic words along the same cycle are unique.

Proposition 22. *For every cycle C of a counter-free DBW $\mathcal{D}$ and states q and q' in C, let u and u' be finite words on which $\mathcal{D}$ goes from q and q' back to themselves along C, respectively. Then $u \neq u'$.*

Next, we use the second constraint in the definition of decisive automata, for assuming without loss of generality that the delimiting letter e does not appear along a cycle of the decisive part.

Proposition 23. *For every decisive DBW $\mathcal{D}$, there is an equivalent decisive DBW with a decisive part Q'', such that e does not appear on any cycle of Q''.*

Proof. Let Q' be a decisive part of $\mathcal{D}$. If e does not appear in any cycle of Q' we are done; If it does, we may change $\mathcal{D}$ to achieve such a form without altering its language. First, we create a copy $\bar{\mathcal{D}}$ of $\mathcal{D}$, which will not be in the decisive part. If $\mathcal{D}$ moves from a state $q \in Q'$ to a state $s \in Q'$ when reading e, we refer q instead to its corresponding state in $\bar{\mathcal{D}}$. Note that we haven't changed the language of $\mathcal{D}$ by doing so. Moreover, there is still an equivalent CTL formula to the state transitioned by e, as a copy of a state that already has an equivalent CTL. We do so for each transition that includes e, getting the requested form. □

In addition, we assume the uniqueness of cyclic escaping words, as it can be deduced from the uniqueness of early escaping words:

Proposition 24. *If $\mathcal{D}$ leaves a cycle $C \subseteq Q'$ cyclically from a state q via a finite word w then for every state $q' \neq q$ of Q', we have $\delta(q', w) = \emptyset$.*

Proof. Assume that $\mathcal{D}$ leaves a cycle $C \subseteq Q'$ cyclically from a state q via a finite word w, and there is a state q' of Q' s.t. $\delta(q', w) \neq \emptyset$. We will prove that $q = q'$ by "zipping" the paths from those states on the word w.

First, note that w must not be an early escaping word, as otherwise Definition 20.5 would not hold. Therefore, there is an inner state q_1 on the cycle with outdegree greater than 1. Thus, w can be rewritten as $w = w_1 w_1' \sigma$ for two finite words $w_1 \in \Sigma^*$, $w_1' \in \Sigma^+$ and a letter $\sigma \in \Sigma$, s.t. $\mathcal{D}$ leaves C early from the state $q_1 = \delta(q, w_1)$ via $w_1'\sigma$. We use again Definition 20.5, this time on the early escaping word $w_1'\sigma$, and learn that this word is legal only from q_1. Therefore, $\delta(q', w_1) = \delta(q, w_1) = q_1$.

We apply similar argument on the word w_1. As mentioned above, q_1's outdegree is greater than 1, therefore it leaves C with some letter $\sigma_1 \in \Sigma$. It implies w_1 can be rewritten as $w_2 w_2'$ such that $\mathcal{D}$ leaves C early from $q_2 := \delta(q, w_2)$ via $w_2'\sigma_1$. We infer that $\delta(q', w_2) = \delta(q, w_2) = q_2$.

We continue iteratively on $i \in \mathbb{N}$, as long as there is a state between q and q_i on C with outdegree greater than 1, getting the state q_{i+1}. Finally, we get a state q_k, for some $k \in \mathbb{N}$, such that every state between q and q_k has outdegree of 1. Moreover, $\mathcal{D}$ leaves C early from q with $w_k \sigma_k$. It must imply that $q = q'$. □

We now turn to show the equivalence of the given decisive DBW and the corresponding CTL formula.

Proof of Theorem 21. Consider a decisive DBW $\mathcal{D}$, and let ψ be the corresponding CTL formula, as per Section 5.5.2. We need to show that $L(\psi) = L(D)\Delta$. Namely, we should prove that for every Σ-labeled tree $\langle T, V\rangle$, it holds that $\langle T, V\rangle \models \psi$ iff for every path π of T, the word $V(\pi)$ is accepted by $\mathcal{D}$.

Left To Right ($L(\psi) \subseteq L(D)\Delta$). We show that $\mathcal{D}$ accepts all paths of a tree T that satisfies ψ, by proving the following "local" lemma, which roughly states that once ψ holds in a node of the tree T, then $\mathcal{D}$ has a corresponding run prefix on every path prefix of some length k, and that all nodes that are k-levels ahead also satisfy ψ.

The "local" lemma guarantees also the "global" requirement, since its iterative repetition is finite—Due to the satisfaction of ψ, e must eventually occur in every path, and therefore $\mathcal{D}$ accepts the path.

Lemma 25. *Let $q \in Q'$ be a state of $\mathcal{D}$ and $\langle T, V\rangle$ a Σ-labeled tree, such that $\langle T, V\rangle \vDash \mathtt{State}(q) \wedge \mathtt{Orientation}$. Then:*

1. *Either T's root is labeled with e (satisfying* `Orientation`*), and every subtree in depth 1 satisfies $\mathtt{State}(\delta(q, e))$ (due to $\mathtt{State}(q)$); Or*

2. *For every path π of T, there exists an integer $k \geq 1$ and a state $q' \in Q'$ s.t. the run of $\mathcal{D}$ on the first k positions of π is legal and leads to a state q', namely $\delta(q, V(\pi_0 \ldots \pi_{k-1})) = q'$ and the subtree $\langle T|_{\pi_k}, V\rangle$ satisfies both $\mathtt{State}(q')$ and* `Orientation`*, namely $\langle T|_{\pi_k}, V\rangle \vDash \mathtt{State}(q') \wedge \mathtt{Orientation}$.*

Proof. Suppose $\langle T, V\rangle \vDash \mathtt{State}(q) \wedge \mathtt{Orientation}$. By definition of the formula `State`, the transition from q upon reading the label of the root of T, denoted by σ, is either on some cycles containing q or leads q to a state q' that has no way back to q. If the latter holds then the required containment is easily shown: by the definition of `State`, we have $\langle T, V\rangle \vDash \sigma \wedge AX\, \mathtt{State}(q')$. For every path π it holds that $\langle T|_{\pi_1}, V\rangle \vDash \mathtt{State}(q')$. If $\sigma = e$ then case (1) is satisfied. Otherwise, $\langle T|_{\pi_1}, V\rangle \vDash \mathtt{Orientation}$ holds since $\langle T, V\rangle$ satisfies `Orientation` and the Until condition of `Orientation` has not been satisfied by the root of T.

We turn to the interesting case, where the transition from q upon reading σ is on some cycles containing q. Recall that we assumed e does not appear in cycles of Q', therefore $\sigma \neq e$. In addition, the definition of `State` dictates that there exists a cycle C that contains q, such that one of the following cases:

I. $\langle T, V\rangle \vDash \mathtt{EarlyLeave}(C, q)$, or

II. $\langle T, V\rangle \vDash \mathtt{Cycle}(C)$

Case I: We learn about the existence of a path in T such that its labels form a word $w \in \Sigma^*$ on which $\mathcal{D}$ leaves C early from q. By Definition 19.2, the outdegrees of $\mathcal{D}$'s states in the path defined by w from q are all 1.

Recall that $\langle T, V\rangle \vDash \mathtt{LookAhead}(q)$ and therefore the next l levels of the tree indeed respect $\mathcal{D}$. Note that w has no more than l letters, since as every other word in $EarlyEscape(q)$, it follows some cycle, up to its last letter, and cannot use the same edge twice. We conclude that all of the $k := |w| - 1$ first levels of the tree follow the same path in $\mathcal{D}$.

Furthermore, since $\langle T, V\rangle \vDash \mathtt{Orientation}$, the fact that there is a path labeled w that is an early escaping word "triggers" `Orientation` to guarantee that $\langle T, V\rangle \vDash (AX)^k \mathtt{State}(q')$ where $q' = \delta(q, w_1 \ldots w_k)$. In other words, every subtree in the k-th level of T satisfies $\mathtt{State}(q')$.

Note that e does not appear in these k levels, since these levels correspond to a cycle of $\mathcal{D}$. Therefore, `Orientation`'s Until condition has not been fulfilled yet so `Orientation` still holds at the k-th level, and we are done.

Notice that we presented a single integer k with respect to which the lemma holds for all paths. In the second case below, the situation will be different, having a possibly different k for each path.

Case II: $\langle T, V\rangle \models \texttt{Cycle}(C) = A\ \texttt{Stay}(C)\ U\ \texttt{CyclicLeave}(C)$. Notice first that since $\texttt{LookAhead}(q)$ holds in the root of T, the first l levels of the input tree indeed correspond to a legal run of $\mathcal{D}$. In addition, we can assume that no path of T starts with an escaping word on which $\mathcal{D}$ leaves C early from q, as we dealt with this case before. Therefore, $\mathcal{D}$ completes the cycle C along every path prefix of T.

Recall that when $\mathcal{D}$ completes the cycle C along every path prefix of T, $\texttt{Stay}(C)$ promises (if holds) another step on C. That is, as long as $\texttt{Stay}(C)$ holds in π, we know that the labels on the next $|C|$ levels correspond to a proper continuation of $\mathcal{D}$'s run on C.

Eventually on every path π of T it holds that $\langle T|_{\pi_j}, V\rangle \models \texttt{CyclicLeave}(C)$, for some $j \geq 0$. In addition, for every path starting from π_j we know that the next $|C| - 1$ levels correspond to C; This is the case, since either $j = 0$ and $\texttt{LookAhead}(q)$ guarantees it or $j \geq 1$ and $\texttt{Stay}(C)$ guarantees it.

We claim that the requested k is $j + |C| - 1$. We have already shown that the labels of the nodes of π respect $\mathcal{D}$ up to π_k (including). It is left to show that $\langle T|_{\pi_k}, V\rangle \models \texttt{State}(q') \wedge \texttt{Orientation}$ for $q' = \delta(q, V(\pi_0 \dots \pi_k))$.

Since $\texttt{CyclicLeave}(C)$ holds at the subtree of π_j, we know that there is a path starting at π_j that starts with a word $w\sigma'$ for $w \in \Sigma^*$ and a letter σ', on which $\mathcal{D}$ leaves C cyclically from some state q''. It must follow that $q'' = \begin{cases} q & j = 0 \\ \delta(q, V(\pi_0 \dots \pi_{j-1})) & j \geq 1 \end{cases}$ because otherwise w is a valid word from two different states on C, contradicting Proposition 22.

The formula $\texttt{Orientation}$ still holds at $\langle T|_{\pi_k}, V\rangle$, for the same reason presented in the first case.

Therefore, similarly to the first case, we infer that the cyclic escaping word $w\sigma'$ "triggers" $\texttt{Orientation}$ to guarantee that the subtree $\langle T|_{\pi_k}, V\rangle$ satisfies $\texttt{State}(\delta(q'', w))$. Now, we claim that $\delta(q'', w) = q'$, implying that $\langle T|_{\pi_k}, V\rangle \models \texttt{State}(q')$, as required. Indeed, $\delta(q'', w) = \delta(q'', V(\pi_j \dots \pi_k))$, since we saw that all $k - j + 1 = |C|$ steps from π_j (including) properly correspond to $\mathcal{D}$'s run on C, and $\delta(q'', V(\pi_j \dots \pi_k)) = \delta(q, V(\pi_0 \dots \pi_k)) = q'$. $\square$

Right to Left ($L(\psi) \supseteq L(D)\Delta$). Let $\langle T, V\rangle$ be a Σ-labeled tree all of whose paths are accepted by $\mathcal{D}$. Without loss of generality, we assume that for every alphabet letter σ, $\mathcal{D}$ remains in the same strongly connected component upon reading σ in q_0; Otherwise, the proof proceeds by induction on the strongly connected components of $\mathcal{D}$. Note that the base case of the induction is covered since states transitioned by e have an equivalent CTL formula (Definition 20.2).

The outline of the proof consists of two main claims. The first concerns the formula $\texttt{State}$ and is more local, saying that if a labeled tree belongs to the language of D^q, for some state q of $\mathcal{D}$, then it also satisfies $\texttt{State}(q)$. The second is more global, claiming that $\langle T, V\rangle$ satisfies $\texttt{Orientation}$.

The first claim is captured by the following lemma.

Lemma 26. *Let T' be a subtree of T. If $\langle T', V\rangle \in L(D^q)$ for some state $q \in Q'$ of $\mathcal{D}$, then $\langle T', V\rangle \models \texttt{State}(q)$.*

Proof. Let T' be a subtree of T such that $\langle T', V\rangle \in L(\mathcal{D}^q)$, for some state $q \in Q'$ of $\mathcal{D}$, and let σ be the label of T''s root.

First, by the construction of LookAhead, it is easy to see that $\langle T', V\rangle \vDash$ LookAhead(q).

We assumed w.l.o.g. that $Cycles(q, \sigma) \neq \emptyset$, thus we should prove that there is a cycle $C \in Cycles(q, \sigma)$, such that $\langle T', V\rangle$ satisfies either EarlyLeave(C, q) or Cycle(C). In fact, we will prove it for every cycle $C \in Cycles(q, \sigma)$.

Indeed, if there is a path of T' labeled with an early-escaping word from $EarlyEscape(C, q)$, then EarlyLeave(C, q) is satisfied. Otherwise, on every path prefix of T', the run of $\mathcal{D}$ remains in the cycle C, meaning all paths complete C. Consider a path π of T'. If CyclicLeave(C, p) doesn't hold on the subtrees $\langle T|_{\pi_0}, V\rangle$, we can infer that the run of $\mathcal{D}$ remain on C for one more step along each path, therefore Stay(C) holds. We can repeat this process concluding that as long as CyclicLeave(C, p) doesn't hold on the subtree $\langle T|_{\pi_i}, V\rangle$ it implies that Stay(C) does, meaning $\langle T', V\rangle \vDash$ Cycle(C). ☐

For showing that $\langle T, V\rangle \vDash$ Orientation, let π be a path of T. We should show that every escaping-word "trigger" is handled correctly until encountering the letter e. Recall that by definition Q' does not contain accepting states and can be leaved only with the letter e, therefore e occurs at some position j in $V(\pi)$.

Consider some node π_i of T for $i < j$. Note that $\langle T|_{\pi_i}, V\rangle \vDash D^q$ for $q = \delta(q_0, V(\pi_0 \dots \pi_{i-1}))$. If there is no cyclic- or early-escaping word that starts at π_i, no trigger is raised, and the Orientation formula is vacuously satisfied. Otherwise, there is a cyclic- or early-escaping word $w\sigma$ that starts at π_i. We consider blow the two cases.

Early-escaping word: We need to show that all nodes on the $|w|$-th level of $T|_{\pi_i}$ satisfy State$(\delta(q, w))$. Consider such a node and the subtree T' that it induces. By definition of early-escaping words, the path that $\mathcal{D}$ makes when reading w from the state q only visits states with outdegree one, until the last visited state. Therefore, since $\langle T|_{\pi_i}, V\rangle \vDash D^q$, when $\mathcal{D}$ runs on the paths of $T|_{\pi_i}$, it necessarily goes along the states on the path from q to $\delta(q, w)$. Thus, $\langle T', V\rangle \vDash D^{\delta(q,w)}$ and by Lemma 26, we get that $\langle T', V\rangle \vDash$ State$(\delta(q, w))$, as required.

Cyclic-escaping word: We should show that $\langle T|_{\pi_i}, V\rangle \vDash IfThen(C, \texttt{State}(\delta(q, w)))_{w\sigma}$. Namely, to show that if a path completes C from the state in which $w\sigma$ is legal, then the subtree on that path in the $|w|$-th level satisfies State$(\delta(q, w))$.

Indeed, consider such a path π', and let T' be the labeled subtree in its $|w|$-th level. By Proposition 24, the only state from which $w\sigma$ is legal is q. Therefore, $T' \in L(D^{\delta(q,w)})$. Thus, by Lemma 26, we get that T' satisfies State$(\delta(q, w))$, as required. ☐

Observe that the length of the constructed formula might be exponentially longer than the number of states in the translated DBW. The reason for the blowup comes from the fact that we examine each simple cycle of the automaton, and there might be exponentially many simple cycles even in decisive DBWs.

5.4 An Additional Sufficient Condition

A future direction of extending the main condition is to handle DBWs in which the decisive part can contain accepting states. A simple case where a DBW with a cycle that contains accepting states can be translated to CTL is as follows. We say that a DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ is *almost linear* if I) There is a letter e that after reading it, $\mathcal{D}$ always moves to a specific state. That is, there is a letter $e \in \Sigma$ and a state $q_e \in Q$, s.t. for every state $q' \in Q$, either $\delta(q', e) = q_e$ or $\delta(q', e) = \emptyset$; and II) q_e is an accepting state ($q_e \in \alpha$) III) If removing all the transitions on the letter e, the automaton becomes linear. See, for example, Figure 10.

We show that an almost linear DBW $\mathcal{D}$ has an equivalent CTL formula by translating it to an equivalent HALT $\mathcal{H} = \langle \Sigma, Q_{\mathcal{H}} = Q \cup \{h_0, q'_e\}, \delta_{\mathcal{H}}, h_0, \alpha \cup \{q'_e\} \rangle$. The translation transforms $\mathcal{D}$ into $\mathcal{H}$ through the following steps:

- All the transitions of $\mathcal{D}$ become A-transitions of $\mathcal{H}$. That is, for every $\sigma \in \Sigma$ and $q \in Q$, we have $\delta_{\mathcal{H}}(q, \sigma) = (A, \delta(q, \sigma))$.
- Changing all the transitions that enter q_e to lead to `true`. That is, for every $q' \in Q$ such that $\delta(q', e) = q_e$, we have $\delta_{\mathcal{H}}(q', e) = \texttt{true}$.
- Adding a new universal state q'_e that is also an accepting state. It goes to itself on every letter, except for e, on which it also goes to q_e. That is, $\delta_{\mathcal{H}}(q'_e, e) = (A, q_e) \wedge (A, q'_e)$, and for every $\sigma \neq e$, $\delta_{\mathcal{H}}(q'_e, \sigma) = (A, q'_e)$.
- Adding a new transient state h_0 that is also the new initial state. It imitates q_0 on the first read letter and universally also goes to q'_e. That is, for every $\sigma \in \Sigma$, $\delta_{\mathcal{H}}(h_0, \sigma) = (A, \delta(q_0, \sigma)) \wedge (A, q'_e)$.

Notice that the second change makes $\mathcal{H}$ linear, and the next two changes keep it linear, having h_0 and q'_e the first and second states of $\mathcal{H}$, respectively. Observe also that $\mathcal{H}$ only uses universality, having no nondeterminism.

Correctness. We show that the languages of $\mathcal{D}$ and $\mathcal{H}$ are equal by proving mutual containment. For showing that $L(\mathcal{D}) \subseteq L(\mathcal{H})$, consider an accepting run of $\mathcal{D}$ on a given tree. We claim that every path in the run-tree is accepting; It is clear for the path labeled with q'_e since it is an accepting state. It is true also for paths with e label, as they were "released" by $\mathcal{H}$ by sending each one of them to `true`. The rest of the paths can be found within the run-tree of $\mathcal{D}$ on that tree, and therefore are also accepting.

For the other direction, consider a tree that is rejected by $\mathcal{D}$. It must contain a path that is not in $L(\mathcal{D})$. Observe that a finite word is rejected by $\mathcal{D}$ iff it is rejected by $\mathcal{H}$. Hence, it is left to consider the case where the run of $\mathcal{D}$ on that path reaches only finitely many times an accepting state. In particular to q_e. It implies that at some point $\mathcal{H}$ is forced to imitate $\mathcal{D}$, therefore to reject the tree.

A known simple example of an almost linear DBW is given in Figure 9. Its language is captured by the LTL formula GFp. Indeed, the derived language is expressible in CTL by the formula $AG\,AFp$. Another, more involved example, is given in Figure 10. As described above, the HALT in the figure simulates the DBW, by initializing a new copy of the "linearized" DBW on every occurrence of e.

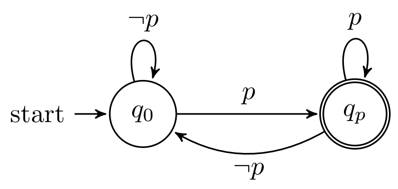


Figure 9: A DBW accepting the language of GFp, expressible in CTL.

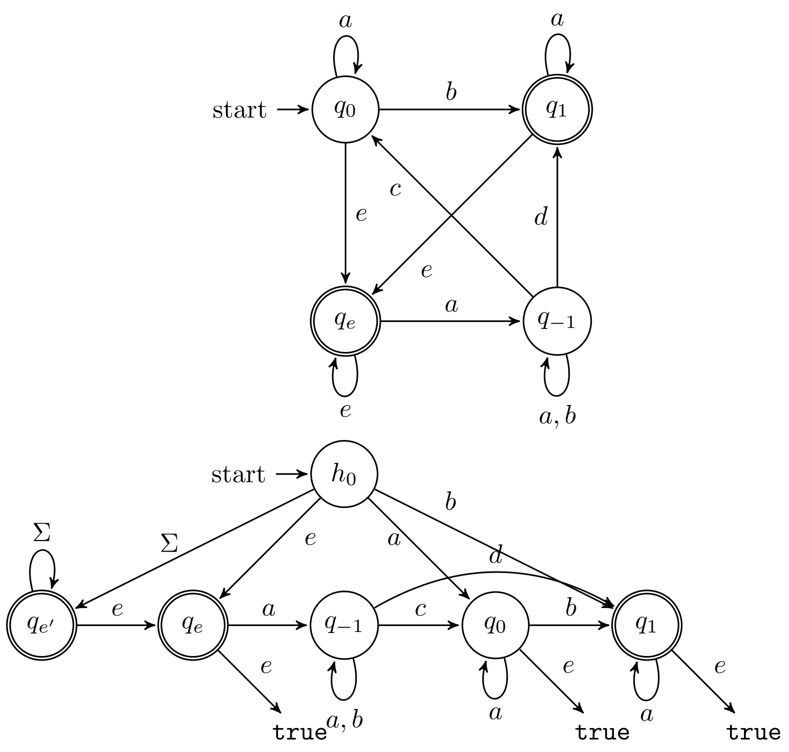


Figure 10: An almost-linear DBW and its equivalent HALT. In the HALT, all transitions are A-transitions, and whenever going on some letter to more than one state, it is done universally.

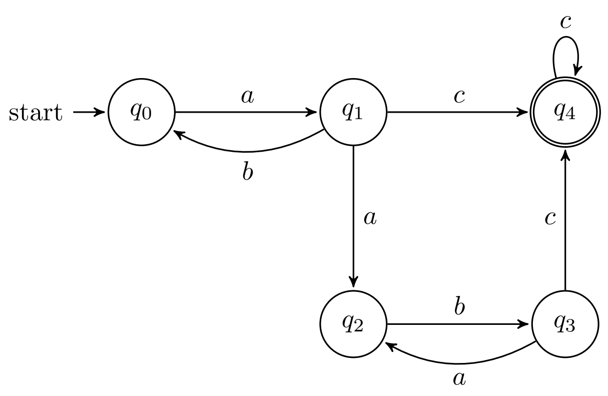


Figure 11: A DBW accepting the language of $(ab)^*a(ab)^*c^\omega$, expressible in CTL.

5.5 Examples

In [3], Bojańczyk proved that there is a language L expressible in both LTL and CTL, but not in ACTL. The following corollary of Theorem 14 provides an alternative proof to the expressibility of L in CTL.

Corollary 27 ([3])**.** *The language $L =$ "all paths belong to $(ab)^*a(ab)^*c^\omega$" is definable in CTL.*

Proof. The DBW presented in Figure 11 is a decisive DBW. The letter c plays to role of e (using the lemma's notation). Indeed, the derived language accepted by q_4 is equivalent to AGc. Note that the other requirements hold, allowing to apply Theorem 14. □

The following corollary demonstrates how both the main sufficient condition, concerning decisive DBWs, and the additional sufficient condition, concerning almost-linear DBWs, can be combined.

Corollary 28. *The language $L =$ "all paths belong to $(ab + ba)^*c(ac + bc)^\omega$" is definable in CTL.*

Proof. Consider the DBW $\mathcal{D}$, presented in Figure 12. First, note that $\mathcal{D}^{q_3}$ is an almost linear DBW; the only letter on which a run of D^{q_3} can move to q_3 is c, which does not appear anywhere else on $\mathcal{D}^{q_3}$. In addition, when removing the c-transitions out of $\mathcal{D}^{q_3}$, it becomes linear. Therefore it has an equivalent CTL.

Now, note that $\mathcal{D}$ is decisive; Its decisive part consists of q_0, q_1, and q_2. As a delimiting letter we have c, on which $\mathcal{D}$ moves to a state that has an equivalent CTL formula, as shown above. Therefore, by Theorem 21, $\mathcal{D}$ is expressible in CTL. □

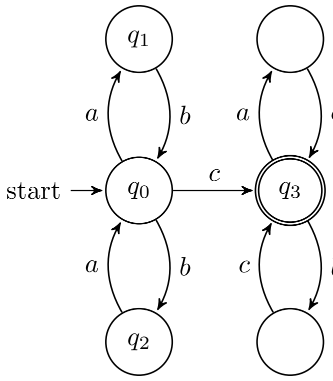


Figure 12: A DBW accepting the language $(ab+ba)^*c(ac+bc)^\omega$, expressible in CTL.

6 On CTL* Formulas Expressible in CTL

Clarke and Draghicescu (now Browne) give in [5] a necessary condition for a CTL* formula to be expressible in CTL over Kripke structures with fairness constraints.

Theorem 29 ([5]). *Let $M = \langle S, R, L, \mathcal{F}\rangle$ be a Kripke Structure with Fairness Constraints, and let $M' = \langle S, R, L, \mathcal{F}'\rangle$ where the set of constraints $\mathcal{F}'$ extends $\mathcal{F}$. Then for all CTL formulas φ and all states $s \in S$, $M, s \models \varphi$ if and only if $M', s \models \varphi$*

They were unable to prove that this condition is also sufficient, leaving it as a conjuncture.

Conjecture 30 ([5]). *Let φ be a CTL* formula. If φ is not expressible in CTL, then it is possible to find two Kripke structures with fairness constraints $M = \langle S, R, L, \mathcal{F}\rangle$ and $M' = \langle S, R, L, \mathcal{F}'\rangle$ with $\mathcal{F}'$ an extension of $\mathcal{F}$ such that for some state $s \in S$ either $M, s \models \varphi$ and $M', s \not\models \varphi$, or $M, s \not\models \varphi$ and $M', s \models \varphi$.*

We refute Conjuncture 30 by showing that the CTL* formula $E(p \vee Xp)Uq$ is not expressible in CTL (already with respect to Kripke structures without fairness constrains), while no two Kripke structures with fairness constraints satisfy the conjecture's condition.

Corollary 31. *The formula $E(p \vee Xp)Uq$ is not expressible in CTL.*

Proof. As a negation of the formula $A(\neg p \wedge X\neg p)R\neg q$, which is not expressible in CTL by Corollary 16. $\square$

For showing that the condition of Conjecture 30 does not hold for the formula $E(p \vee Xp)Uq$, we will use the following lemma from [5], regarding the prefixes of computations in Kripke structures with fairness constraints. Given a Kripke structure $M = \langle S, R, L, \mathcal{F}\rangle$ with fairness constraints and a state $s \in S$, we denote by $Prefix(M, s)$ the set of finite prefixes of fair computations of M that start at s.

Lemma 32 ([5])**.** *Let $M = \langle S, R, L, \mathcal{F} \rangle$ be a Kripke structure with fairness constraints, and let $M' = \langle S, R, L, \mathcal{F}' \rangle$ where the set of constraints $\mathcal{F}'$ extends $\mathcal{F}$. Let s be a state of M. Then, $Prefix(M, s) = Prefix(M', s)$.*

We are now in place to refute Conjecture 30.

Theorem 33. *Conjecture 30 of [5] is false.*

Proof. We claim that the formula $\varphi = E(p \vee Xp)Uq$ is a counter example for Conjecture 30. By Corollary 31, φ is not expressible in CTL. We will show that the condition of Conjecture 30 does not hold for φ, that is, for every Kripke structure with fairness constraints M, an extension of it M' and a state s of M, the following holds, $M, s \models \varphi$ iff $M', s \models \varphi$.

Let φ^d stand for the LTL formula $(p \wedge Xp)Uq$. Note that φ^d defines a co-safety language: a word satisfies φ^d iff it has a finite prefix that "approves" the word. Thus, $M, s \models \varphi$ iff there is a finite prefix of a fair computation that satisfies φ^d. Hence, we get the following by Lemma 32:
$M, s \models \varphi$ iff
there is a finite computation prefix $\pi \in Prefix(M, s)$ such that $\pi \models \varphi^d$ iff
there is a finite computation prefix $\pi' \in Prefix(M', s)$ such that $\pi' \models \varphi^d$ iff
$M', s \models \varphi$. □

7 Conclusions and Future Work

We clarified the automaton characterization of CTL, showing that CTL is equivalent to hesitant alternating linear tree automata (HALT) and strictly less expressive than alternating linear tree automata. Using the HALT characterization of CTL, we provided some necessary conditions and some sufficient conditions for an LTL formula to be expressible in CTL. The conditions simplify the (non) membership proofs of some LTL formulas that are known (not) to be in CTL, and provide means for deducing the (non) membership of many other LTL formulas.

There is still a big gap between our necessary conditions and sufficient conditions. We believe that the automaton approach we have taken can be further pursued toward generalizing the conditions, and maybe even toward resolving the longstanding open problem of the common fragment of LTL and CTL. In particular, one can look into generalizing the sufficient condition, by allowing the decisive part of the considered DBW to have accepting states.

The HALT characterization of CTL is useful also for conditions on the membership of tree languages in CTL. We used it for showing that alternating linear tree automata are strictly more expressive than CTL, and for refuting a conjecture by Clarke and Draghicescu from 1988, regarding a sufficient condition for a CTL* formula to be expressible in CTL. The automaton approach may be further studied toward resolving the more general open problem of deciding whether a given CTL* formula is in CTL.

Lastly, the constructive technique that we used in the sufficient condition, for translating a certain kind of DBWs into CTL formulas, might be useful also for translating certain kinds of counter-free NBWs into LTL formulas. Counter-free NBWs are known to be equivalent to LTL, yet the current equivalence proofs are complicated and indirect.

References

[1] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A.L. Sangiovanni-Vincentelli. Equivalences for fair kripke structures. In *Proc. 21st Int. Colloq. on Automata, Languages, and Programming*, 1994.

[2] O. Bernholtz and O. Grumberg. Branching time temporal logic and $\mathcal{A}$morP$\mathcal{H}$Ous tree automata. In *Proc. 4th Int. Conf. on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 1993.

[3] M. Bojańczyk. The common fragment of ACTL and LTL. In *International Conference on Foundations of Software Science and Computational Structures*, pages 172–185. Springer, 2008.

[4] E.M. Clarke. The birth of model checking. In *25 Years of Model Checking*, pages 1–26. Springer, 2008.

[5] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 1988.

[6] 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*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[7] V. Diekert and P. Gastin. First-order definable languages. *Logic and automata*, 2:261–306, 2008.

[8] Rüdiger Ehlers. ACTL $\cap$ LTL synthesis. In *CAV*, pages 39–54. Springer, 2012.

[9] C. Eisner. PSL for runtime verification: Theory and practice. *Lecture Notes in Computer Science*, 4839:1–8, 2007.

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

[11] O. Grumberg and R.P. Kurshan. How linear can branching-time be. In *Proc. 1st Int. Conf. on Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 180–194. Springer, 1994.

[12] D. Kirsten. Alternating tree automata and parity games. *Automata logics, and infinite games*, pages 405–411, 2002.

[13] O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. In *Proc. 11th IEEE Symp. on Logic in Computer Science*, pages 322–333, 1996.

[14] O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006.

[15] O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

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

[17] L. Lamport. "Sometimes" is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symp. on Principles of Programming Languages*, pages 174–185, 1980.

[18] C. Löding. Automata on infinite trees. *Available online from author*, 2011.

[19] C. Löding and W. Thomas. Alternating automata and logics over infinite words. In *Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2000.

[20] M. Maidl. The common fragment of CTL and LTL. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, pages 643–652, 2000.

[21] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[22] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[23] R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Pres, 1971.

[24] 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 IEEE Symp. on Logic in Computer Science*, pages 422–427, 1988.

[25] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloq. on Automata, Languages, and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 275 – 283. Springer, 1986.

[26] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoretical Computer Science*, 97(2):233–244, 1992.

[27] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[28] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[29] D. Niwiński. Fixed point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189(1–2):1–69, 1997.

[30] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[31] M.Y. Vardi. Alternating automata and program verification. In *Computer Science Today –Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995.

[32] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.

[33] M.Y. Vardi. Alternating automata – unifying truth and validity checking for temporal logics. In W. McCune, editor, *Proc. of the 14th Int. Conf. on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 191–206. Springer, 1997.

[34] M.Y. Vardi. Sometimes and not never re-revisited: On branching versus linear time. In *International Conference on Concurrency Theory*, pages 1–17. Springer, 1998.

[35] M.Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[36] M.Y. Vardi and T. Wilke. Automata: from logics to algorithms. *Logic and automata*, 2:629–736, 2008.

[37] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 1999.

[38] T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 8(2):359, 2001.

תקציר

לוגיקת הזמן המתפצל CTL נמצאת בשימוש נרחב בתחום האימות הפורמלי. למרות זאת, בשונה מלוגיקת הזמן הקווי LTL, הקשר שלה לאוטומטים מעל מילים ועצים לא מובן עדיין במלואו. מעבר לכך, לא ידוע האם קיים אלגוריתם המכריע, בהנתן נוסחת LTL, האם היא ניתנת להבעה ב-CTL, והתנאים המספיקים או ההכרחיים הידועים לבעיה מאוד מוגבלים.

קופרמן, ורדי וולפר הראו בשנת 2000 שניתן לתרגם כל נוסחת CTL לאוטומט מתחלף הססן לינארי מעל עצים (hesitant alternating linear tree automaton – HALT). בהמשך לעבודתם, אנו מראים שאוטומטי HALT מאפיינים את CTL: אנחנו מוכיחים שניתן לתרגם HALT ל-CTL, וכן שללא דרישת ההססנות או דרישת הלינאריות, התרגום איננו אפשרי.

אנו עושים שימוש באיפיון HALT של CTL בכדי להציג תנאים מספיקים ותנאים הכרחיים לשייכות ל-CTL עבור נוסחאות LTL ושפות ω-רגולריות. התנאים שלנו מתבססים על העובדה שלכל שפה ω-רגולריות הניתנת להבעה ב-CTL, יש אוטומט בּוּכי דטרמיניטי (DBW) המקבל אותה. אנו מנתחים את המעגלים של ה-DBW ואת הקשר שלהם למעגלים באוטומט HALT העשוי להיות שקול לו.

התנאי ההכרחי הבסיסי שאנחנו מספקים עבור DBW $\mathcal{D}$ הניתן להבעה ב-CTL, הוא חוסר קיום של מעגל C, עליו ניתן למצוא מילה שלאחר קריאתה ממצב ב-C, $\mathcal{D}$ יכול לקבל כל מילה. התנאי המספיק מצמצם את התנאי ההכרחי ודורש בין היתר שאוטומט ה-DBW יעזוב מעגלים עם מילים ייחודיות. ההוכחה הניתנת הינה קונסטרוקטיבית, ומגדירה נוסחת CTL שקולה.

לבסוף, תוך שימוש בתנאי ההכרחי, אנחנו מפריכים השערה שניתנה ע"י קלארק ודרגיצ'סקו בשנת 1988 לגבי תנאי עבור נוסחאות CTL* להיות ניתנות להבעה גם על ידי CTL.

עבודה זו בוצעה בהדרכתו של דר' אודי בוקר מבי"ס אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה.

המרכז הבינתחומי בהרצליה
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

תנאים מבוססי אוטומטים לשייכות ל-CTL

מאת
יריב שאוליאן

עבודת תזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.
במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

ספטמבר 2017