Automaton-Based Criteria for Membership in CTL

Udi Boker
Interdisciplinary Center (IDC) Herzliya, Israel

Yariv Shaulian
Interdisciplinary Center (IDC) Herzliya, Israel

Abstract

Computation Tree Logic (CTL) is widely used in formal verification, however, 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.

We provide sufficient conditions and necessary conditions for LTL formulas and ω-regular languages to be expressible in CTL. The conditions are automaton-based; We first tighten the automaton characterization of CTL to the class of Hesitant Alternating Linear Tree Automata (HLT), and then conduct the conditions by relating between the cycles of a word automaton for a given ω-regular language and the cycles of a potentially equivalent HLT.

The new conditions allow to simplify proofs of known results on languages that are definable, or not, in CTL, as well as to prove new results. Among which, they allow us to refute a conjecture by Clarke and Draghicescu from 1988, regarding a condition for a CTL* formula to be expressible in CTL.

Keywords LTL, CTL, automaton characterization

1 Introduction

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) [26] and in the latter is *Computation Tree Logic* (CTL) [5]. Roughly (and arguably) speaking, LTL is a more natural specification language, whereas CTL allows for more efficient verification algorithms.

LTL and CTL are known to be incomparable [17], and the quest for deciding their common fragment goes back to the 1980s. (An LTL formula φ, defining a word language L, is equivalent to a CTL formula ψ, defining a tree language L', if L' is the *derived language* of L, namely consisting of trees all of whose paths are in L.)

In 1988, Clarke and Draghicescu (now, Browne) presented an algorithm to determine, given a CTL formula, whether it has an equivalent LTL formula, while leaving the other direction open [4]. They did provide a necessary condition for an LTL formula to have an equivalent CTL formula, however using a non-standard equivalence relation; Instead of considering equivalence with respect to standard Kripke structures, as is usually done, they defined the equivalence with respect to Kripke structures with fairness constraints. They conjectured that this necessary condition is also sufficient, leaving it as another open question. (We refute the conjecture in Section 6.)

A major progress was made by Maidl in 2000, when she 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 its negation has an equivalent nondeterministic linear word automaton [19]. Yet, there was no algorithm to decide whether a given LTL formula satisfies the condition. Moreover, it was not clear whether LTL∩ACTL is equivalent to LTL∩CTL.

In 2008, Bojańczyk provided an algorithm to decide Maidl's condition, namely to decide whether a given ω-regular language has an equivalent nondeterministic linear word automaton. However, he also showed that Maidl's characterization does not capture LTL∩CTL, meaning that LTL∩ACTL is a strict fragment of LTL∩CTL [2]. This result is somewhat surprising, as LTL formulas apply to all paths, while a formula in CTL\ACTL must also quantify existentially over paths. In contrast, for the case of LTL∩AFMC, universality does not limit the expressive power [15].

Since 2008, there was no significant progress toward deciding the LTL∩CTL fragment. Moreover, there are currently very limited sufficient conditions and necessary conditions for deciding whether an LTL formula is expressible in CTL, even if considering conditions that do not add up to a complete decision procedure.

Beyond the theoretical interest in better understanding the connection between linear-time and branching-time temporal logics, there is also a potential for a practical benefit. 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. Although an exponential lower bound is known for such a translation [33], it might be useful in practice. Moreover, as claimed by Eisner and Fisman [8]: "the vast majority of properties used in practice belong to the overlap between CTL and LTL". Another area in which a characterization of their common fragment can be useful is synthesis, generalyzing the approach taken by Ehlers in [7], who improved synthesis procedures by using the automaton characterization of LTL∩ACTL.

We use an automaton 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 sufficient conditions.

LICS '18, July 9–12, 2018, Oxford, United Kingdom

ACM ISBN 978-1-4503-5583-4/18/07…$15.00
https://doi.org/10.1145/3209108.3209143

This work was supported by the Israel Science Foundation grant 1373/16.

Automaton characterization of CTL As opposed to LTL, which enjoys various automaton characterizations, such as deterministic counter-free Muller automata [21], nondeterministic counter-free Büchi automata [6], and alternating linear (also called "very-weak" or "1-weak") automata [6], the automaton characterization of CTL is not completely clear. It is stated in [31, Theorem 5.11] that CTL is equivalent to alternating linear tree automata. Yet, a close look on the definition of alternating tree automata in [31] reveals that it is different from the standard definition, as originally given in [24].

In [31], 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 [15, 16, 18, 23, 24, 28–30], alternating automata with ϵ-transitions have a few, slightly different, definitions; Sometimes the domain of the transition function is the set of states [31, 34] and sometimes the set of states together with the alphabet [33]; In addition, sometimes the Boolean connectives and the path quantifiers (or path directions) can be combined freely in the transition condition [11, 33] and sometimes only in a limited way [11, 31, 34].

In general, classic alternating tree automata and the various versions of alternating tree automata with ϵ-transitions are considered to be equivalent ([33, Proposition 1] and [11, Remark 9.4]). 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 [31] are strictly less expressive than standard alternating linear tree automata (ALT)—We prove that the class of hesitant-ALT (HLT) is equivalent to CTL, and thus also to the automata of [31], while being strictly less expressive than ALT.

The translation of CTL to HLT is given in [16], and we provide the other direction. Our translation of an HLT $\mathcal{A}$ to a CTL formula φ generalizes the technique used in [18] for translating an alternating linear word automaton to an LTL formula, by handling the subtle branching possibilities of tree automata. For showing that HLT 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 HLT $\mathcal{H}$ 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 $\mathcal{A}$.

Our conditions for LTL∩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∩CTL is expressible by deterministic Büchi automata (DBW) [13], one can concentrate on LTL formulas that are recognized by DBWs. Notice that this preliminary check is indeed decidable; One can translate the LTL formula to an equivalent deterministic Rabin automaton [27, 32], which has an equivalent DBW iff it has one on its own structure [12].

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 HLT. If the DBW is linear, namely has only cycles of size one, then by Maidl's condition [19], it obviously has an equivalent CTL, and even ACTL, formula. Intuitively, the core limitation of CTL in expressing a tree language derived 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, and satisfy 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.

We prove the necessary condition by assuming toward contradiction that a DBW $\mathcal{D}$ that does not satisfy the necessary condition has an equivalent HLT $\mathcal{H}$. 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 viewed as a deterministic tree automaton) stays in the relevant cycle for some paths and leaves it 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. 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 the uniqueness of each finite word u on which $\mathcal{D}$ can complete and leave a cycle. Moreover, There should be 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 the second requirement, can be decided by examining all of $\mathcal{D}$'s simple cycles. The second requirement 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 the given DBW $\mathcal{D}$, and whose length is up to exponentially the number of states in $\mathcal{D}$. The constructed formula may be syntactically, as well as semantically, in

CTL\ACTL; A simple DBW for Bojańczyk's language (see Figure 7), 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 $F(p \wedge Xp)$. 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.

Due to lack of space, some of the proofs appear in the appendix.

2 Preliminaries

2.1 Words and Trees

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.

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, $E \subseteq N \times N$ is a set of node transitions and ϵ is the root node. Given a tree T, we denote its set of nodes by $N(T)$, and for a node n, we write $Succ(n)$ 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 starts at the root of the tree. The notation π^i, for an integer i, stands for the suffix of π starting at index i.

Given an alphabet Σ, a Σ-*labeled tree* is a pair $\langle T, V \rangle$, where T is a tree and $V : N(T) \to \Sigma$ maps each node of T to a letter in Σ.

2.2 Temporal Logic

Let AP be a set of atomic propositions. The language of well-formed CTL* formulas is generated by the following context-free grammar: $\psi = \mathtt{true} \mid p \mid (\neg\psi) \mid (\psi \wedge \psi) \mid E\varphi$, and $\varphi = \psi \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid X\varphi \mid (\varphi U \varphi)$ where $p \in AP$. We use the following syntactic sugar: $F\varphi = trueU\varphi'$, $G\varphi = \neg F \neg\varphi$, $\varphi R \varphi' = \neg(\neg\varphi U \neg\varphi')$, $\varphi W \varphi' = \varphi U \varphi' \vee G\varphi$ and $A\psi = \neg E \neg\psi$. Proper CTL*-formulas are built using the nonterminal ψ. These formulas are called state formulas, while those created by the symbol φ are called path formulas.

For defining the semantics of CTL*, let $\langle T, V \rangle$ be a 2^{AP}-labeled tree and π a path of it. We say that π *satisfies* φ ($\pi \models \varphi$) when the following hold, where $p \in AP$ and φ, φ_1, and φ_2 are path formulas.

- $\pi \models \mathtt{true}$. • $\pi \models p$ iff $p \in V(\pi_0)$. • $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$.
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$. • $\pi \models X\varphi$ iff $\pi^1 \models \varphi$.
- $\pi \models \varphi_1 U \varphi_2$ iff $\exists i \in \mathbb{N}$ s.t. $\pi^i \models \varphi_2$ and $\forall j \in [0..i-1]$, $\pi^j \models \varphi_1$.

Similar definitions are for state formulas and trees with the addition of: $\langle T, V \rangle \models E\varphi$ iff there is a path π of T 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.

An LTL formula φ is a CTL* path-formula that does not contain A or E, e.g. $F(p \wedge Xp)$. In the context of CTL*, we treat φ as the state formula $A\varphi$. A CTL formula is a CTL* state-formula s.t. each path-quantifier is followed immediately by one of the temporal operators $\{X, U, R, G, F, W\}$, e.g. $EFAGp$. The formula $EXp \wedge AFGp$ is neither an LTL nor a CTL formula.

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. We assume that all of Q's states are reachable from some initial state. When $|Q_0| = 1$ and It is *deterministic* if Q_0 is a singleton and for every $q \in Q$ and $\sigma \in \Sigma$, $|\delta(q, \sigma)| \le 1$. Then we refer to $\delta(q, \sigma)$ as a state and not as a 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 $i \ge 0$, we have $r_{i+1} \in \delta(r_i, w_i)$. A run r is accepting if it visits the accepting states infinitely often. Formally, $inf(r) = \{q \in Q \mid$ for infinitely many $i \in \mathbb{N}$, we have $r_i = q\}$, and r is accepting iff $inf(r) \cap \alpha \neq \emptyset$.

The language that $\mathcal{A}$ *recognizes* (*accepts*), denoted by $L(\mathcal{A})$, is the set of words on which $\mathcal{A}$ has an accepting run. 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 to the domain $Q' \times \Sigma$.

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.

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. An automaton $\mathcal{A}$ is called *counter-free*, if $w^m \in L_{p,p}$ implies $w \in L_{p,p}$ for every state p of $\mathcal{A}$, word $w \in L_{p,p}$, and $m \ge 1$.

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.

Formally, an *alternating Büchi tree automaton* (ABT) is a tuple $\langle \Sigma, Q, q_0, \delta, \alpha \rangle$ where 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.

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 rooted at 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 $m \models \theta$. This definition is by induction on the structure of θ, where the boolean connectives, `true`, and `false` are dealt in the usual way. Further:

- $m \models (E, q)$ if the corresponding node $r(m)_n$ of m in T has a successor n' and there exists some successor m' of m, such that $r(m') = (n', q)$.
- $m \models (A, q)$ 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 $m \models \delta(q, V(n))$.

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 $\delta(q, V(n)) = \texttt{true}$ 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 that are labeled by a state from α. An automaton accepts a tree 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 $m \in Succ(m')$, $r(m) = (n, q)$, and $m' \models (A, q)$.

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.

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 alternation-free μ-calculus (AFMC) [25].

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]. An ALT is *hesitant*, denoted by HLT, 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 Connecting Automata and Temporal Logic

In temporal logic, formulas are interpreted over a set AP of atomic propositions. On the other hand, ABTs operate on Σ-labeled trees. When correlating between them, we take the alphabet Σ to be the power set of the atomic propositions, namely $\Sigma = 2^{AP}$.

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 $\langle T, V\rangle \models \varphi$.

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].

2.4.1 ω-regular∩CTL = LTL∩CTL ⊆ DBW

It was shown in [10] that the language of a CTL* formula is the derived language of some ω-regular language 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∩CTL = LTL∩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 AFMC formula. As CTL is subsumed by AFMC [20], we have:

Corollary 2.1 ([15])**.** *Let φ be an LTL formula of a language L, equivalent to some CTL formula. Then, there is a DBW recognizing L.*

In other words, we know that LTL∩CTL ⊆ DBW. Notice that the sets are not equal. Moreover, we have LTL∩CTL ⊊ LTL∩DBW. For example, the LTL formula $F(p \wedge Xp)$ is not expressible in CTL [9], while expressible by a DBW.

3 CTL is Equivalent to HLT

It is stated in [31, p. 710, Theorem 5.11] that CTL is equivalent to alternating linear tree automata. Yet, as elaborated on in the introduction, a close look on the definition of alternating tree automata in [31] reveals that it is different from the standard definition, as originally given in [24]. We show that alternating linear tree automata as defined in [31], are strictly less expressive than standard alternating linear tree automata (ALT)—We prove that HLT is equivalent to CTL, and thus also to the corresponding automata of [31], and that they are strictly less expressive than ALT.

We start by presenting the equivalence and later, in Section 3.1, show that HLT indeed tightly characterizes CTL; Relaxing either

[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 HLT.

the linearity or the hesitant obligations of the automata results in strictly more expressive automata.

Theorem 3.1. *CTL formulas and HLTs have the same expressiveness.*

The proofs of both directions of Theorem 3.1 are constructive, and generate CTL formulas and HLTs whose size is linear in each other.

For the translation of CTL to HLT, a construction was presented [16]. It considers automata on ordered trees, but it is also suitable, almost as is, for HLTs, which run on unordered trees.

For the other direction, we show that every HLT can be translated to an equivalent CTL formula, adapting the technique used in [18] for translating a linear alternating word automaton into an LTL formula. The challenge in the adaptation is how to properly generalize the technique from words to trees. Indeed, as explained in the Introduction and will be demonstrated in Section 3.1, small variations in the definition of alternating linear tree automata determine whether or not they are equivalent to CTL.

Construction. Consider an HLT $\mathcal{A} = \langle \Sigma, Q = \{q_0, q_1, \ldots, q_n\}, q_0, \delta, \alpha\rangle$. 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 $i \leq j$, and that $q_n = q_{\mathtt{true}}$. 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.

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

The ToCTL function, translating states and transition conditions of $\mathcal{A}$ into CTL formulas, is defined in the expected way by induction on the structure of the transition condition. The non-trivial translation is of the atomic subformula q_i, for $i \in [0..n]$. Recall that the Weak-until temporal operator, defined by $\varphi W \psi := \varphi U \psi \vee G\varphi$, and Let ρ_1 and ρ_2 be subformulas of θ.

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

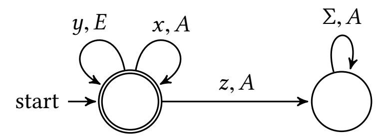

Figure 1. An ALT that has no equivalent HLT.

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 $\mathtt{true}$.

Lemma 3.2. *For every $i \in [0..n]$, the CTL formula $\mathtt{ToCTL}(q_i)$ is equivalent to $\mathcal{A}^{q_i}$.*

3.1 Tightness

We show that both the linearity and the hesitant properties of an HLT are indeed essential for the equivalence with CTL. That is, we prove that non-linear hesitant AWT (HWT) 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 analogous.)

The inequality HLT<HWT is straightforward. It is easy to present an HWT that recognizes the language of $AF(p \wedge XP)$, but no CTL formula captures that language [9]. Thus by Theorem 3.1, the described HWT has no equivalent HLT.

For showing that HLT<ALT, we provide an ALT $\mathcal{A}$, as depicted in Figure 1, and prove that it does not have an equivalent HLT. Intuitively, an HLT cannot follow the unboundedly many alternations between A- and E-transitions that $\mathcal{A}$ allows. The technique used in the proof shares ideas with the proof of Theorem 4.1, and is detailed in the appendix.

Theorem 3.3. *Hesitant alternating linear tree automata (HLT) are strictly less expressive than alternating linear tree automata (ALT).*

4 Necessary Conditions for LTL∩CTL

There are currently very limited techniques for showing that an LTL formula cannot be expressed in CTL. Emerson and Halpern showed in [9] that the LTL formula $F(p \wedge Xp)$ is not expressible in CTL. They used, as stated in [3], a long and complicated inductive argument that required about 2 journal pages to present, while not allowing for easy generalizations to other examples.

Later on in [4], Clarke and Draghicescu 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 6.1). 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 HLT characterization of CTL. The techniques can be used for easily showing that $F(p \wedge Xp)$ is not expressible in CTL, as well as many other formulas, among which is $(p \wedge Xp)Rq$, which will serve us in refuting the aforementioned conjecture of Clarke and Draghicescu.

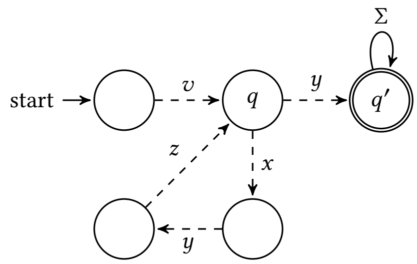

Figure 2. A schematic DBW that cannot be expressed in CTL.

We start with a basic necessary condition, which we will strengthen in Section 4.2. Recall that LTL∩CTL ⊆ DBW [14]. 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 4.1. *Let L be the derived language of a DBW $\mathcal{D}$. 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 2).*
- *$\mathcal{D}^q$ 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 3.1, there is an HLT $\mathcal{A}$ that recognizes L.

We shall describe a tree T that belongs to L, as depicted in Figure 3, 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 HLT 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 HLT.

Let $v \in \Sigma^*$ be a finite word on which $\mathcal{D}$ reaches q. 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 $w \in \Sigma^\omega$ be a word rejected by $\mathcal{D}^q$ (See Figure 2 for a sketch of $\mathcal{D}$).

The tree** T **(see Figure 3): We define Y to be the set of trees 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, in this case, of $\mathcal{A}$ on $vxyzw$ can be accomplished by changing an accepting run of $\mathcal{A}$ on a singled-path tree that starts with vx; the prefix of such a run can be 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_0^l 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, $\mathcal{A}^s$ can accept every tree. (If it is not the case, we change $\mathcal{A}$ to an HLT $\mathcal{A}'$ that extends $\mathcal{A}$ with $|y| - 1$ new states, namely $\{s'_1, \ldots, s'_{|y|-1}\}$, having the transitions $s \xrightarrow[A]{y_1} s'_1 \xrightarrow[A]{y_2} \ldots \xrightarrow[A]{y_{|y|-1}} s'_{|y|-1} \xrightarrow[A]{y_{|y|}} \mathtt{true}$. 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_0^l. Indeed, if r does not assign s_0 to n_0^l then r' is simply r. If r assigns s_0 to n_0^l only by an E statement, r' assigns s_0 to n_0^r instead. Otherwise, by the above argument, s_0 does not belong to B and has a transition to $\mathtt{true}$ after reading the word y. Hence, the run r' can lead s_0 to $\mathtt{true}$ when reading y, before reaching n_0, implying that no state that is assigned to n_0 can assign s_0 to n_0^l. (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_i^l is not assigned any of the states in $\{s_0, s_1, \ldots, s_i\}$. In particular, the node n_{m-1}^l, 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 4.1 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

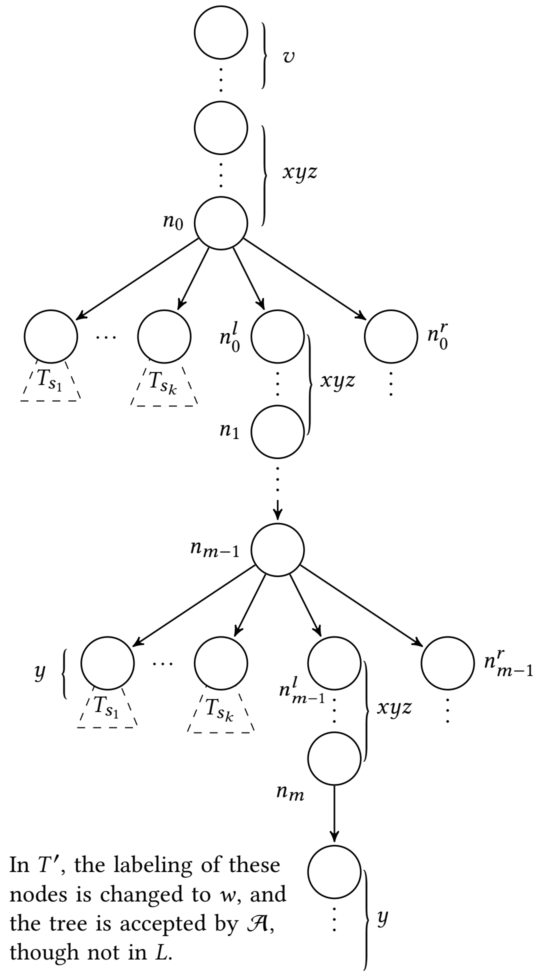

Figure 3. The tree T used in the proof of Theorem 4.1

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 4.1, considered a state q', s.t $\mathcal{D}^{q'}$ accepts every word.

We shall allow $\mathcal{D}^{q'}$ to recognize a richer variety of languages, in particular, CTL-recognizable languages that satisfy 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 4.4.

Theorem 4.2. *[Theorem 4.1 Extended] Let L be the derived language of a DBW $\mathcal{D}$. If there is a state q of $\mathcal{D}$ s.t. the following hold, then L is not expressible in CTL.*

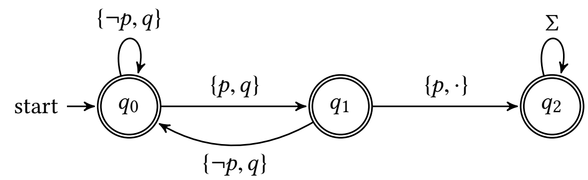

Figure 4. A DBW for $(p \wedge Xp)Rq$, not expressible in CTL.

- *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 $w \notin L(\mathcal{D}^q)$, s.t. $\forall i \in \mathbb{N}$, $z(xyz)^i w \in L(\mathcal{D}^{q'})$.*
- *For every word $y' \in L(\mathcal{D}^{q'})$ and $\forall i \in \mathbb{N}$, $z(xyz)^i yy' \in L(\mathcal{D}^{q'})$.*

Notice that Theorem 4.1 is a special case of Theorem 4.2, 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 sketch. We extent the proof of Theorem 4.1 by updating the set 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 $L(\mathcal{D}^{q'})$.

Since the tree we build depends on Y, we get an updated version of T. The tree T' is obtained from T by replacing the subtree hanged under n_m with the singled-path tree labeled w. We have, as before, that T is in L and T' is not.

As $\mathcal{D}^{q'}$ is expressible in CTL, there exists, by Theorem 3.1, an HLT $\mathcal{A}_{q'}$ equivalent to $\mathcal{D}^{q'}$. We may thus assume that $\mathcal{A}$, or an HLT equivalent to $\mathcal{A}$, has a nondeterministic path from its initial state to the initial state of $\mathcal{A}_{q'}$ upon reading y.

We then continue along the lines of the proof of Theorem 4.1, and show that the only guards (states) on the end of the path that we examine, namely the states that are assigned to the node n_m, are states of $\mathcal{A}_{q'}$ and not of the original automaton $\mathcal{A}$. Hence, the states assigned to n_m cannot "catch" the bad path we hanged, implying that $\mathcal{A}$ accepts T', which leads to a contradiction. ◻

4.3 Examples of Using the Necessary Conditions

The first example concerns the LTL formula $(p \wedge Xp)Rq$, recognized by the DBW in Figure 4. Note that in Theorem 4.1, it does not matter whether the states in the cycle are accepting or not, and in this example all of the states are accepting. This formula will also serve us in Section 6 to refute a conjuncture presented in [4].

Corollary 4.3. *The LTL formula $(p \wedge Xp)Rq$ is not expressible in CTL.*

Similarly, we are able to easily show that the LTL formula $F(p \wedge Xp)$ is not expressible in CTL. This was already proved in [9], but with much effort.

The added value of the stronger condition (Theorem 4.2) is demonstrated by the DBW $\mathcal{D}$ of Figure 5, not covered by the basic condition. It also demonstrates how one can inductively use a necessary condition together with a sufficient condition—The language of $\mathcal{D}^{q'}$ is in CTL by the sufficient condition presented in Section 5.2, and therefore, due to Theorem 4.2, the language of $\mathcal{D}$ is not in CTL.

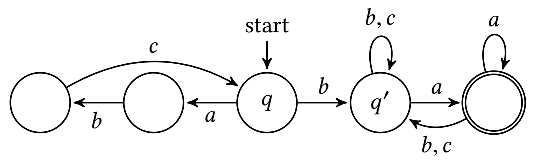

Figure 5. A DBW for $(abc)^*b((b+c)^*a)^\omega$, not expressible in CTL.

Corollary 4.4. *The language L = "all paths belong to $(abc)^*b((b+c)^*a)^\omega$" is not definable in CTL.*

Analogously, $F(p \wedge Xp) \wedge GFp$ is not expressible in CTL.

5 Sufficient Conditions for LTL∩CTL

The main sufficient condition narrows down the necessary condition by requiring, among other things, that the DBW $\mathcal{D}$ leaves cycles with unique words. (More precisely, the requirement is that if $\mathcal{D}$ leaves a cycle with some word w then the valid runs of $\mathcal{D}$ on w from all states coincide in their last two states.) 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∩(CTL\ACTL).

In Section 5.2, 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 5.5. Moreover, by combining the sufficient conditions and the necessary conditions, one can define more involved DBWs that cannot be expressed in CTL. See, for example, Corollary 4.4.

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 5.1 (Escaping Words)**.** Consider a DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, a simple cycle $C = \langle q{=}q_1, q_2, \ldots, q_m, q_{m+1}{=}q \rangle$ of $\mathcal{D}$, and a finite word $w = w_1 \ldots w_l$, where $l \in [2..m{+}1]$.

We say that $\mathcal{D}$ *leaves* C from q via the word w if the following hold:

- The outdegree of q is greater than one;
- The run of $\mathcal{D}^q$ on w follows C up to the last letter (excluding), namely $\delta(q_j, w_j) = q_{j+1}$ for every $j \in [1..l{-}1]$ and $\delta(q_l, w_l) \neq \begin{cases} q_2 & q_l = q_1 \\ q_{l+1} & \textit{otherwise} \end{cases}$.

When the run of $\mathcal{D}^q$ on the word $w_1 \ldots w_{l-1}$ completes C, namely when $l{-}1 = m$, we say that $\mathcal{D}$ *leaves* C *cyclically* from q and otherwise we say that $\mathcal{D}$ *leaves* C *early* from q.

We call the word w a *cyclic* (resp. *early*) *escaping word.*

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

Definition 5.2. 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 $C \subseteq Q'$ from a state q via a finite word $w\sigma$ (the last letter of the escaping word is σ) then for every state $q' \in Q'$, we have $\delta(q', w\sigma) = \emptyset$ or $\delta(q', w) = \delta(q, w)$. That is, all paths over $w\sigma$ in Q' coincide in their last two states.

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 5.2, for getting a CTL-expressible DBW. (See, for example, Corollary 5.5.)

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 [6] and non-counting languages [22]. 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 5.2.

Theorem 5.3. *Every decisive DBW has an equivalent CTL formula.*

proof sketch. We first define the CTL formula that corresponds to a given decisive DBW, and afterwards prove that it is indeed equivalent to the DBW.

Consider a decisive DBW $\mathcal{D} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ over an alphabet Σ with a decisive part $Q' \subseteq Q$. For every state p in Q', we have a formula `State`(p) that "describes" it, based on the simple cycles to which p belongs. We also define such formulas for the states in $\delta(Q', e)$.

We have in addition the formula `Orientation` that occasionally "synchronizes" a node of the input tree with the corresponding state of Q'. 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'. Note that we use here the existential power of CTL.

The overall CTL formula corresponding to D is

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

The correctness proof details the intuitive explenation above, using "local" and "global" claims. The local claim states that a tree satisfies the formula $\mathtt{State}(p)$, for some state p, iff the first fixed amount of levels of the tree are legal prefixes from the state p. The global claim states that $\mathtt{Orientation}$ holds until the letter e appears. By combining the two, we are able to prove the equivalence of our formula and the DBW. ◻

5.2 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 condition toward such an extension is the following. 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 $\delta(q', e) = \emptyset$; II) q_e is an accepting state ($q_e \in \alpha$); and III) If removing all the transitions on the letter e, the automaton becomes linear. See, for example, Figure 6.

We show that an almost linear DBW $\mathcal{D}$ has an equivalent CTL formula by translating it to an equivalent HLT $\mathcal{H} = \langle \Sigma, Q_{\mathcal{H}} = Q \cup \{h_0, q'_e\}, \delta_{\mathcal{H}}, h_0, \alpha \cup \{q'_e\} \rangle$. 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 $\mathtt{true}$. That is, for every $q' \in Q$ such that $\delta(q', e) = q_e$, we have $\delta_{\mathcal{H}}(q', e) = \mathtt{true}$.
- Adding a new universal state q'_e that is also an accepting state. It goes to itself on every letter, and on e 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 other 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.

GFp is a simple example of a language the has an almost linear DBW. Another, more involved example, is given in Figure 6. As described, the HLT in the figure simulates the DBW, by initializing a new copy of the "linearized" DBW on every occurrence of e.

5.3 Examples

In [2], Bojańczyk proved that there is a language L expressible in both LTL and CTL but not in ACTL. The following corollary of Theorem 5.3 is an alternative proof to the expressibility of L in CTL, based on the DBW of Figure 7, in which the early (and in this case also cyclic) escaping words are *bac* (from q_1) and *abc* (from q_3).

Corollary 5.4 ([2]). *The language L = "all paths belong to $(ab)^* a(ab)^* c^\omega$" is definable in CTL.*

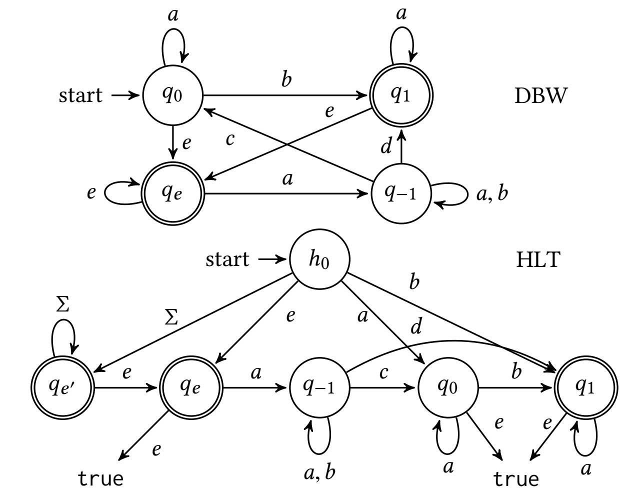

Figure 6. An almost-linear DBW and its equivalent HLT. In the HLT, all transitions are universal A-transitions.

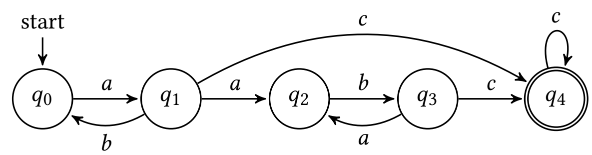

Figure 7. A DBW for $(ab)^* a(ab)^* c^\omega$, expressible in CTL.

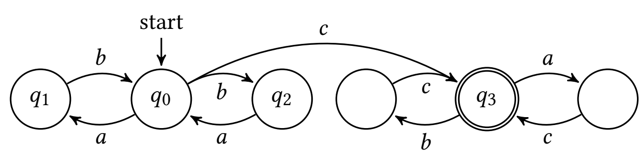

Figure 8. A DBW for $(ab + ba)^* c(ac + bc)^\omega$, expressible in CTL.

The next corollary demonstrates how both the main sufficient condition, concerning decisive DBWs, and the additional sufficient condition, concerning almost-linear DBWs, can be combined. Consider the DBW $\mathcal{D}$, presented in Figure 8. First, note that $\mathcal{D}^{q_3}$ is an almost linear DBW and therefore it has an equivalent CTL. In addition, note that $\mathcal{D}$ is decisive (c servers as a delimiting letter). Therefore, by Theorem 5.3, we get the expressibility in CTL.

Corollary 5.5. *The language L = "all paths belong to $(ab+ba)^* c(ac+bc)^\omega$" is definable in CTL.*

6 On CTL* Formulas Expressible in CTL

Clarke and Draghicescu give in [4] a necessary condition for a CTL* formula to be expressible in CTL over Kripke structures with fairness constraints. We first formally define the latter, as used for example in [1, 4].

A *Kripke structure with fairness constraints* over an alphabet 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. (One may also assume that each set of $\mathcal{F}$ defines a strongly connected component.)

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 in 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 $F \in \mathcal{F}$.

The semantics of CTL* with respect to a Kripke structure with fairness constraints is defined using only the fair paths of the structure. That is, one should take the following change in the satisfiability definition: $\langle M, s \rangle \models E\varphi$ iff there is a fair path π' starting from s s.t. $\pi' \models \varphi$.

We provide next the necessary condition of [4].

Theorem 6.1 ([4]). *Let $M = \langle S, R, L, \mathcal{F} \rangle$ and $M' = \langle S, R, L, \mathcal{F}' \rangle$ be Kripke Structure with Fairness Constraints, where the set of constraints $\mathcal{F}'$ extends $\mathcal{F}$. Then for all CTL formulas φ and all states $s \in S$, $\langle M, s \rangle \models \varphi$ if and only if $\langle M', s \rangle \models \varphi$*

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

Conjecture 6.2 ([4]). *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 $s \in S$, $\langle M, s \rangle \models \varphi$ and $\langle M', s \rangle \not\models \varphi$ or $\langle M, s \rangle \not\models \varphi$ and $\langle M', s \rangle \models \varphi$.*

We refute Conjuncture 6.2 by showing that the CTL* formula $E(p \vee Xp)Uq$ is not expressible in CTL (already w.r.t. Kripke structures without fairness constrains), while no two Kripke structures with fairness constraints satisfy the conjecture's requirements.

Corollary 6.3. *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 4.3. □

For showing that the condition of Conjecture 6.2 does not hold for the formula $E(p \vee Xp)Uq$, we will use the following lemma from [4], 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 6.4 ([4]). *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 6.2.

Theorem 6.5. *Conjecture 6.2 of [4] is false.*

Proof. We claim that the formula $\varphi = E(p \vee Xp)Uq$ is a counter example for Conjecture 6.2. By Corollary 6.3, φ is not expressible in CTL. We will show that the condition of Conjecture 6.2 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, we will show that $\langle M, s \rangle \models \varphi$ iff $\langle M', s \rangle \models \varphi$.

Let φ^d stand for the LTL formula $(p \vee Xp)Uq$. 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, $\langle M, s \rangle \models \varphi$ iff there is a finite prefix of a fair computation that satisfies φ^d. Hence, we get the following by Lemma 6.4: $\langle M, s \rangle \models \varphi$ iff there is a prefix $\pi \in Prefix(M, s)$ s.t. $\pi \models \varphi^d$ iff there is a prefix $\pi' \in Prefix(M', s)$ s.t. $\pi' \models \varphi^d$ iff $\langle M', s \rangle \models \varphi$. □

7 Conclusions

We tightened the automaton characterization of CTL to the class of hesitant alternating linear tree automata (HLT), and used it to provide some necessary conditions and some sufficient conditions for an LTL formula to be expressible in CTL. The new conditions allow to simplify proofs of known results on languages that are definable, or not, in CTL, as well as to prove many new results.

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 deciding 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 HLT characterization of CTL is useful also for conditions on the membership of tree languages in CTL. We used it for showing that CTL < ALT, and for refuting a conjecture in [4] regarding a sufficient condition for a CTL* formula to be 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. 1994. Equivalences for fair Kripke Structures. In *Proc. ICALP*. 364–375.
[2] M. Bojańczyk. 2008. The common fragment of ACTL and LTL. In *Proc. of FoSSaCS*. Springer, 172–185.
[3] E.M. Clarke. 2008. The birth of model checking. In *25 Years of Model Checking*. Springer, 1–26.
[4] E.M. Clarke and I.A. Draghicescu. 1988. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency (LNCS)*, Vol. 354. 428–437.
[5] E.M. Clarke and E.A. Emerson. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs (LNCS)*, Vol. 131. Springer, 52–71.
[6] V. Diekert and P. Gastin. 2008. First-order definable languages. *Logic and automata* 2 (2008), 261–306.
[7] Rüdiger Ehlers. 2012. ACTL ∩ LTL Synthesis. In *Proc. of CAV*. 39–54.
[8] C. Eisner. 2007. PSL for runtime verification: Theory and practice. *LNCS* 4839 (2007), 1–8.
[9] E.A. Emerson and J.Y. Halpern. 1986. Sometimes and Not Never Revisited: On Branching Versus Linear Time. *J. ACM* 33, 1 (1986), 151–178.
[10] O. Grumberg and R.P. Kurshan. 1994. How linear can branching-time be. In *Proc. 1st Int. Conf. on Temporal Logic*, Vol. 827. Springer, 180–194.
[11] D. Kirsten. 2002. *Alternating Tree Automata and Parity Games.* 153–167.
[12] S.C. Krishnan, A. Puri, and R.K. Brayton. 1994. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations (LNCS)*, Vol. 834. Springer, 378–386.
[13] O. Kupferman, S. Safra, and M.Y. Vardi. 1996. Relating word and tree automata. In *Proc. 11th IEEE Symp. on Logic in Computer Science.* 322–333.
[14] O. Kupferman, S. Safra, and M.Y. Vardi. 2006. Relating word and tree automata. *Ann. Pure Appl. Logic* 138, 1-3 (2006), 126–146.
[15] O. Kupferman and M.Y. Vardi. 2005. From linear time to branching time. *ACM Transactions on Computational Logic* 6, 2 (2005), 273–294.
[16] O. Kupferman, M.Y. Vardi, and P. Wolper. 2000. An Automata-Theoretic Approach to Branching-Time Model Checking. *J. ACM* 47, 2 (2000), 312–360.
[17] L. Lamport. 1980. "Sometimes" is sometimes "Not never" - on the temporal logic of programs. In *Proc. of PoPL*. 174–185.
[18] C. Löding and W. Thomas. 2000. Alternating automata and logics over infinite words. In *Theoretical Computer Science (LNCS)*, Vol. 1872. Springer, 521–535.
[19] M. Maidl. 2000. The common fragment of CTL and LTL. In *Proc. FoCS*. 643–652.
[20] K.L. McMillan. 1993. *Symbolic Model Checking.* Kluwer Academic Publishers.

[21] R. McNaughton. 1966. Testing and Generating Infinite Sequences by a Finite Automaton. *Information and Control* 9 (1966), 521–530.
[22] R. McNaughton and S. Papert. 1971. *Counter-Free Automata.* MIT Pres.
[23] D.E. Muller, A. Saoudi, and P.E. Schupp. 1992. Alternating automata, the weak monadic theory of trees and its complexity. *Theoretical Computer Science* 97, 2 (1992), 233–244.
[24] D.E. Muller and P.E. Schupp. 1987. Alternating automata on infinite trees. *Theoretical Computer Science* 54 (1987), 267–276.
[25] D.E. Muller and P.E. Schupp. 1995. Simulating Alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science* 141 (1995), 69–107.
[26] A. Pnueli. 1977. The temporal logic of programs. In *Proc. FoCS.* 46–57.
[27] S. Safra. 1988. On the Complexity of ω-Automata. In *Proc. FoCS.* 319–327.
[28] M.Y. Vardi. 1995. Alternating automata and program verification. In *Computer Science Today –Recent Trends and Developments (LNCS)*, Vol. 1000. 471–485.
[29] M.Y. Vardi. 1997. Alternating automata – unifying truth and validity checking for temporal logics. In *Proc. of the 14th Int. Conf. on Automated Deduction (Lecture Notes in Artificial Intelligence)*, W. McCune (Ed.), Vol. 1249. Springer, 191–206.
[30] M.Y. Vardi. 1998. Sometimes and Not Never Re-revisited: On Branching Versus Linear Time. In *International Conference on Concurrency Theory.* Springer, 1–17.
[31] M.Y. Vardi and T. Wilke. 2008. Automata: from logics to algorithms. *Logic and automata* 2 (2008), 629–736.
[32] M.Y. Vardi and P. Wolper. 1986. An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. on Logic in Computer Science.* 332–344.
[33] T. Wilke. 1999. CTL$^+$ is exponentially more succinct than CTL. In *Proc. of FoSSaCS (LNCS)*, Vol. 1738. Springer, 110–121.
[34] T. Wilke. 2001. Alternating Tree Automata, Parity Games, and Modal μ-Calculus. *Bulletin of the Belgian Mathematical Society Simon Stevin* 8, 2 (2001), 359.

A Appendix

A.1 Additional Preliminaries

Given a (labeled) tree T and a node $n \in N(T)$, we denote by $T|_n$ the (labeled) subtree of T rooted at n.

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 $\sigma \in \Sigma$, $\delta(q_0, \sigma) = \rho$.

A.2 Proofs of Section 3

Theorem 3.1. *CTL formulas and HLTs have the same expressiveness.*

We show that the described construction indeed produces for every HLT $\mathcal{A}$ 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 the structure of the transition condition θ of the state q_i. We start with a lemma on the correctness of this structural induction.

Lemma A.1. *Let $i \in [0..n{-}1]$, and assume that for every $k > i$, the HLT $\mathcal{A}_k$ 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 A_i^ρ accepts T iff $T \models$* $\texttt{ToCTL}(\rho)$.

Proof. By induction on the structure of ρ.

Base cases:

- $\rho = \texttt{true}$: By definition, T satisfies $\texttt{true}$, and $A_i^{\texttt{true}}$ accepts every tree.
- $\rho = \texttt{false}$: By definition, T does not satisfy $\texttt{false}$, and $A_i^{\texttt{false}}$ 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 corresponding node in T and 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 semantics of CTL, 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, there is an accepting run R_l of $\mathcal{A}_k$ on $T|_l$. Recall that for each such node l, there is a corresponding node 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 A.2. *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 $\mathtt{ToCTL}(q_i) = A\varphi_{i,stay} W \varphi_{i,leave}$.

I. $L(\mathcal{A}_i) \subseteq L(\mathtt{ToCTL}(q_i))$: 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 A.1, $T \models \mathtt{ToCTL}(\theta'_{i,\sigma})$, 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. $L(\mathtt{ToCTL}(q_i)) \subseteq L(\mathcal{A}_i)$: Let T be a Σ-labeled tree s.t. $T \models \mathtt{ToCTL}(q_i)$. We will present an accepting run of $\mathcal{A}_i$ on T. As $T \models \mathtt{ToCTL}(q_i)$, in particular $T \models \varphi_{i,stay} \vee \varphi_{i,leave}$. That is, there is $\sigma \in \Sigma$ s.t. $T \models \psi_\sigma$, and either $T \models \mathtt{ToCTL}(\theta_{i,\sigma})$ or $T \models \mathtt{ToCTL}(\theta'_{i,\sigma})$.
If $T \models \mathtt{ToCTL}(\theta'_{i,\sigma})$ then by Lemma A.1, $\mathcal{A}_i^{\theta'_{i,\sigma}}$ 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, $T \models \mathtt{ToCTL}(\theta_{i,\sigma})$, and by Lemma A.1, $\mathcal{A}_i^{\theta_{i,\sigma}}$ 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, $T|_l \models \varphi_{i,stay} \vee \varphi_{i,leave}$. 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 $\mathtt{ToCTL}(\theta'_{i,\sigma})$ 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 $\mathtt{ToCTL}(\theta'_{i,\sigma})$, 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 A.2, considering the initial state q_0, we get the correction of our construction, concluding the proof of Theorem 3.1.

Theorem 3.3. *Hesitant alternating linear tree automata (HLT) are strictly less expressive than alternating linear tree automata (ALT).*

Proof. Consider the ALT $\mathcal{A}$ described in Figure 1, and let $L = L(\mathcal{A})$. Assume toward contradiction that there exists an HLT $\mathcal{H}$ 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 $T_s^x \in \overline{L(\mathcal{H}^s)} \cap L$, and for every state $s \in Y$, there exists a tree $T_s^y \in \overline{L(\mathcal{H}^s)} \cap \overline{L}$.

The tree** T **(see Figure 9): 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_s^x. 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 is labeled y and has for every $s \in Y$, the subtree T_s^y. 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_s^x, and two identical subtrees, denoted by n_1^l and n_1^r. n_1^l 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 $T|_{n_i^l}$ is also in L, since it is labeled y and has a subtree in L. Recall that by definition, $T|_{n_i^r}$ is identical to $T|_{n_i^l}$, therefore also in L. It is left to show that the rest of the children of n_i are in L. Namely, that T_s^x is in L for every $s \in X$, which indeed holds by the definition of T_s^x. 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_s^y for $s \in Y$, which by definition are not in L. Therefore n_i^l has no child in L, and thus $T|_{n_i^l} \notin L$. 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. □

A.3 Proofs of Section 4

Theorem 4.2. *[Theorem 4.1 Extended] Let L be the derived language of a DBW $\mathcal{D}$. If there is a state q of $\mathcal{D}$ s.t. the following hold, then L is not expressible in CTL.*

- *There is a 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 $w \notin L(\mathcal{D}^q)$, s.t. $\forall i \in \mathbb{N}$, $z(xyz)^i w \in L(\mathcal{D}^{q'})$.*
- *For every word $y' \in L(\mathcal{D}^{q'})$ and $\forall i \in \mathbb{N}$, $z(xyz)^i yy' \in L(\mathcal{D}^{q'})$.*

Proof. We explain below how we change the proof of Theorem 4.1 in order to suit Theorem 4.2.

Similarly to the proof of Theorem 4.1, we assume toward contradiction that L is expressible in CTL, and therefore there is an HLT $\mathcal{A}$ 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 of 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 2, and recall that though $L(\mathcal{D}^{q'})$ is no longer Σ^ω, $y' \in L(\mathcal{D}^{q'})$). Moreover, note that T' is not in L.

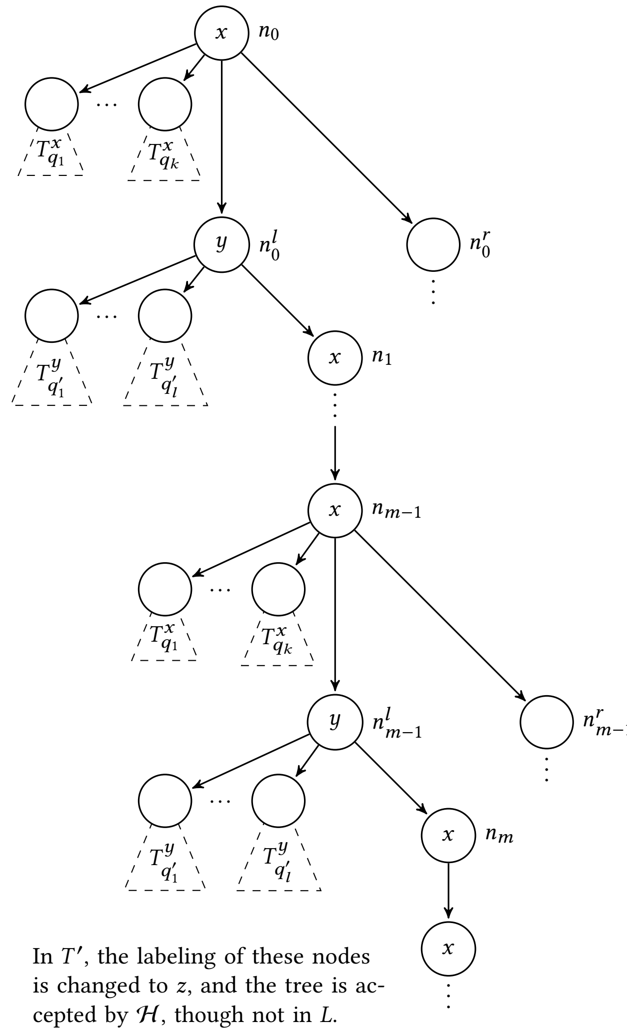

Figure 9. The tree T used in the proof of Theorem 3.3

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, $\mathcal{A}^s$ can accept every tree in $L(\mathcal{D}^{q'})\Delta$ after reading y. If, technically, it is not the case, we can change $\mathcal{A}$ to an equivalent HLT $\mathcal{A}'$, as described below, such that the above conceptual claim holds also technically.

Recall that $\mathcal{D}^{q'}$ has an equivalent CTL formula, therefore by Theorem 3.1 there is an HLT $\mathcal{A}_{q'}$ equivalent to $\mathcal{D}^{q'}$. We denote its initial state by s'. We change $\mathcal{A}$ to an HLT $\mathcal{A}'$ that extends $\mathcal{A}$ with $|y|-1$ new states, namely $\{s'_1, \ldots, s'_{|y|-1}\}$, having the transitions $s \xrightarrow[A]{y_1} s'_1 \xrightarrow[A]{y_2} \cdots \xrightarrow[A]{y_{|y|-1}} s'_{|y|-1} \xrightarrow[A]{y_{|y|}} s'$. Notice that $\mathcal{A}$ and $\mathcal{A}'$ recognize the same language.

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. □

Corollary 4.4. *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 5 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 5.2, $\mathcal{D}^{q'}$ is almost linear and thus has an equivalent CTL formula. The rest of the stronger conditions hold as well, therefore by Theorem 4.2, there is no equivalent CTL formula for $\mathcal{D}$. □

A.4 Proofs of Section 5

We start with 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 define $Cycles(q)$ to be the set of simple cycles that include q, and $Cycles(q, \sigma)$ to be the set of simple cycles that include both the state q and $\delta(q, \sigma)$, such that the two states are adjacent along the cycle.

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 $CyclicEscape(q)$. Finally, we define

$$Escape(q) = EarlyEscape(q) \cup CyclicEscape(q)$$

We continue with the theorem proof.

Theorem 5.3. *Every decisive DBW has an equivalent CTL formula.*

A.4.1 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 will have a formula $\mathtt{State}(p)$ that "describes" it, based on the simple cycles to which p belongs. We will also define such formulas for the states in $\delta(Q', e)$. In addition, we will 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 some CTL formula φ^p. We simply define $\mathtt{State}(\delta(p, e)) := \varphi^p$.

We continue with a state $p \in Q'$. Consider first the case that p has outdegree 1. Since there is only one possible next-state to p, we may demand that all paths that continue from p in the tree satisfy the next-state's formula. Namely,

$$\mathtt{State}(p) = \bigvee_{\sigma \in \Sigma} \varphi_\sigma \wedge AX\mathtt{State}(\delta(p, \sigma))$$

Where

$$\varphi_\sigma = (\bigwedge_{a \in \sigma} a) \wedge (\bigwedge_{a \notin \sigma} \neg a)$$

Upfront, it might seem that the above definition of $\mathtt{State}(p)$ is improper, as it might cause a cyclic definition of the overall CTL formula, defining the formula of each state according to the formula of the next state along a cycle. Yet, observe that we only allow this definition in the case that the outdegree of p is 1. Recall that all states in the decisive part of the automaton are rejecting. Thus, a cyclic definition might only occur in case that there is a cycle in the automaton of rejecting states, all of which having outdegree 1. Every state in this cycle is thus equivalent to `false`. Hence, we can remove such cycles in a pre-processing phase, and assume at this phase that there are no such cycles.

We continue with the case where p has outdegree of 2 or above. The matched formula 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.

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

$\mathtt{LookAhead}(p)$***:*** 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 $\mathtt{LookAhead}(p) = (\varphi_a \vee \varphi_b) \wedge (\varphi_a \to AX((\varphi_a \vee \varphi_b \vee \varphi_c) \wedge AX((\varphi_a \vee \varphi_c) \to AX(\varphi_b) \wedge \varphi_b \to AX(\varphi_a)) \wedge (\varphi_b \to AX(\varphi_a \to AX(\varphi_a \vee \varphi_b)))$.

$\mathtt{EarlyLeave}(p, C)$***:*** 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

$$\mathtt{PathExists}(w) = \varphi_{w_1} \wedge EX\,(\varphi_{w_2} \wedge EX\,(\varphi_{w_3} \cdots \wedge EX\,\varphi_{w_{|w|}})),$$

and use it to define

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

Before defining $\texttt{Cycle}(C)$, we define the formula $\texttt{IfThen}$ that it will use. The formula takes two arguments: a finite sequence of states $q_1, \ldots, q_m$ for some $m \in \mathbb{N}$ and a CTL formula ξ. Intuitively, the formula ensures that if a path has a prefix u of length m, such that the run of the automaton from q_1 over u is exactly the sequence of states $[q_1..q_m]$, then this path, after the prefix u, satisfies ξ. Formally, for every $i \in [1..m-1]$, let Σ_i be the set of letters that are legit as a transition from q_i to q_{i+1}. We define

$$\texttt{IfThen}([q_1..q_m], \xi) = \varphi_{\Sigma_1} \rightarrow AX(\varphi_{\Sigma_2} \rightarrow \cdots AX(\varphi_{\Sigma_{m-1}} \rightarrow AX\xi)).$$

where

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

$\texttt{Cycle}(C)$*:* 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 $C = \langle q_1, \ldots, q_k, q_1 \rangle$. For defining the formula $\texttt{Cycle}(C)$, we first define the following formulas.

- We use the $\texttt{IfThen}$ formula for defining a formula stating that if a path of the tree completes a cycle along C, starting from the x-th position of C, then it also takes one more step on C.

$$\texttt{Stay}(C)_x = \texttt{IfThen}(\langle q_x, .., q_k, q_1, .., q_{x-1} \rangle, \Sigma_x),$$

- 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,

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

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

Once detecting an escaping word, $\texttt{Orientation}$ 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 escaping word $w\sigma$ is detected, we can ensure that paths in the tree that are related to w share the same future, reaching together the same state. Therefore, we trigger the formula of that state.

$$\texttt{Orientation} = A\, \texttt{GoodEscapes}\ Ue,$$

where $\texttt{GoodEscapes}$ =

$$\bigwedge_{q \in Q'} \bigwedge_{w\sigma \in Escape(q)} PathExists(w\sigma) \rightarrow \texttt{IfThen}(sequence_{q,w}, \texttt{State}(\delta(q, w))).$$

and $sequence_{q,w}$ is the list of states that the automaton visits while running on w from q. It starts with q and ends with $\delta(q, w)$.

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

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

A.4.2 Correctness

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 A.3. *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 A.4. *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. □

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

Proof of Theorem 5.3. Consider a decisive DBW $\mathcal{D}$, and let ψ be the corresponding CTL formula, as per Section A.4.1. 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 A.5. *Let $q \in Q'$ be a state of $\mathcal{D}$ and $\langle T, V \rangle$ a Σ-labeled tree, such that $\langle T, V \rangle \models \texttt{State}(q) \land \texttt{Orientation}$. 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 $\delta(q, V(\pi_0 \dots \pi_{k-1})) = q'$ and the subtree $\langle T|_{\pi_k}, V\rangle$ satisfies both* $\mathtt{State}(q')$ *and* Orientation*, namely $\langle T|_{\pi_k}, V\rangle \models \mathtt{State}(q') \wedge \mathtt{Orientation}$.*

Proof. Suppose $\langle T, V\rangle \models \mathtt{State}(q) \wedge \mathtt{Orientation}$. 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 the formula State, we have $\langle T, V\rangle \models \varphi_\sigma \wedge AX\, \mathtt{State}(q')$. For every path π it holds that $\langle T|_{\pi_1}, V\rangle \models \mathtt{State}(q')$. If $\sigma = e$ then case (1) is satisfied. Otherwise, $\langle T|_{\pi_1}, V\rangle \models \mathtt{Orientation}$ 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 for every cycle $C \in Cycles(q, \sigma)$, one of the following holds:

1. $\langle T, V\rangle \models \mathtt{EarlyLeave}(C, q)$, or
2. $\langle T, V\rangle \models \mathtt{Cycle}(C)$

We split into two cases:

I. $\exists C \in Cycles(q, \sigma)$ s.t. $\langle T, V\rangle \models \mathtt{EarlyLeave}(C, q)$, or
II. $\forall C \in Cycles(q, \sigma)$ it holds that $\langle T, V\rangle \not\models \mathtt{EarlyLeave}(C, q)$

Notice first that since $\mathtt{LookAhead}(q)$ holds in the root of T, the first l levels of the input tree indeed correspond to a legal run of $\mathcal{D}$. (Recall that l stands for the length of the longest cycle in Q' plus one.)

Case I: Let wa be an early escaping word of minimal length that can be found from the root of T and let C be the cycle on which $\mathcal{D}$ leaves C early from q via wa. By definition of wa we know that each (legal) path in T follows C for the next $k := |w|$ levels of the tree, as otherwise there would have been a shorter escaping word.

Note that wa 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. Since the following l levels are legal, we conclude that all of the $k \leq l$ first levels of the tree follow the same path in $\mathcal{D}$.

Furthermore, since $\langle T, V\rangle \models \mathtt{Orientation}$, the fact that there is a path labeled wa that is an early escaping word "triggers" Orientation to guarantee that $\mathtt{IfThen}(sequence_{q,w}, \mathtt{State}(\delta(q, w))$ holds. Hence, since every path follows C for the next k steps, we have by the definition of the IfThen formula that every subtree in the k-th level of T satisfies $\mathtt{State}(\delta(q, w))$.

Note that e does not appear in these k levels, since these levels correspond to a cycle of Q'. 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: The assumption in this case implies that for each cycle $C \in Cycles(q, \sigma)$ it holds that $\langle T, V\rangle \models \mathtt{Cycle}(C)$. In particular, there exists such a cycle and we denote it by C. So $\langle T, V\rangle \models A\, \mathtt{Stay}(C)\; U\; \mathtt{CyclicLeave}(C)$. In addition, we know that $\langle T, V\rangle \not\models \mathtt{EarlyLeave}(C, q)$, therefore no path of T starts with an escaping word on which $\mathcal{D}$ leaves C early from q. Thus, $\mathcal{D}$ 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, $\mathtt{Stay}(C)$ promises (if holds) another step on C. That is, as long as $\mathtt{Stay}(C)$ 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, there is some $j \geq 0$, such that $\langle T|_{\pi_j}, V\rangle \models \mathtt{CyclicLeave}(C)$. 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 $\mathtt{LookAhead}(q)$ guarantees it or $j \geq 1$ and $\mathtt{Stay}(C)$ 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 $\langle T|_{\pi_k}, V\rangle \models \mathtt{State}(q') \wedge \mathtt{Orientation}$ for $q' = \delta(q, V(\pi_0 \dots \pi_k))$.

Since $\mathtt{CyclicLeave}(C)$ 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 A.3.

The formula 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" Orientation to guarantee that the subtree $\langle T|_{\pi_k}, V\rangle$ satisfies $\mathtt{IfThen}(sequence_{q'',w}, \mathtt{State}(\delta(q'', w)))$. Since the path labeled w satisfies the if-clause, we deduce that $\langle T|_{\pi_k}, V\rangle \models \mathtt{State}(\delta(q'', w))$

Now, we claim that $\delta(q'', w) = q'$, implying that $\langle T|_{\pi_k}, V\rangle \models \mathtt{State}(q')$, 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 $\delta(q'', V(\pi_j \dots \pi_k)) = \delta(q, V(\pi_0 \dots \pi_k)) = q'$. ◻

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 5.2.2).

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

The first claim is captured by the following lemma.

Lemma A.6. *Let T' be a subtree of T. If $\langle T', V\rangle \in L(\mathcal{D}^q)$ for some state $q \in Q'$ of $\mathcal{D}$, then $\langle T', V\rangle \models \mathtt{State}(q)$.*

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. We will further assume that q's outdegree is greater than 1 and handle the other case later.

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

We assumed w.l.o.g. that $Cycles(q, \sigma) \neq \emptyset$, thus we should prove that for each cycle $C \in Cycles(q, \sigma)$ the labeled tree $\langle T', V\rangle$ satisfies either $\mathtt{EarlyLeave}(C, q)$ or $\mathtt{Cycle}(C)$.

Indeed, let C be a cycle in $Cycles(C, q)$. If there is a path of T' labeled with an early-escaping word from $EarlyEscape(C, q)$, then $\mathtt{EarlyLeave}(C, q)$ 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 $\mathtt{CyclicLeave}(C, p)$ 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 $\mathtt{Stay}(C)$ holds. We can repeat this process concluding that as long as $\mathtt{CyclicLeave}(C, p)$ doesn't hold on the subtree $\langle T|_{\pi_i}, V\rangle$ it implies that $\mathtt{Stay}(C)$ does, meaning $\langle T', V\rangle \vDash \mathtt{Cycle}(C)$.

For the case where q's outdegree is 1, recall that the $\mathtt{State}(q)$ formula is $AX\mathtt{State}(\delta(p, \sigma))$ for the letter σ with which the tree node is labeled. Since all children of the tree node follow the run of $\mathcal{D}$ from q, which necessarily leads to the single next-state of q in $\mathcal{D}$, every path of the tree satisfies the formula $\mathtt{State}$ until $\mathcal{D}$ reaches a state with outdegree bigger than 1, and from there the correctness follows from the previous case. □

For showing that $\langle T, V\rangle \vDash \mathtt{Orientation}$, 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 $\langle T|_{\pi_i}, V\rangle \in L(\mathcal{D}^q)$ for $q = \delta(q_0, V(\pi_0 \ldots \pi_{i-1}))$. If there is no cyclic- or early-escaping word that starts at π_i, no trigger is raised, and the $\mathtt{Orientation}$ formula is vacuously satisfied. Otherwise, there is a cyclic- or early-escaping word $w\sigma$ that starts at π_i. Let q' be a state and C a cycle, such that $\mathcal{D}$ leaves C from q' via $w\sigma$.

We should show that $\mathtt{IfThen}(sequence_{q',w}, \mathtt{State}(\delta(q', w))$ holds. That is, we need to prove that if a prefix of π_i forms a word u, such that $\mathcal{D}^{q'}$ has the same run on w and on u, then the subtree T' that starts after this prefix of π_i satisfies $\mathtt{State}(\delta(q', w))$.

Indeed, observe first that $u\sigma$ is also an escaping word, since $w\sigma$ is an escaping word and w and u share the same route from q', hence leave C together. By Definition 5.2.5 escaping words coincide in their last two states, therefore $\delta(q', u) = \delta(q, u)$ and $\delta(q', w) = \delta(q, w)$. Since we also have that $\delta(q', u) = \delta(q', w)$, we get that $\delta(q, u) = \delta(q, w)$.

As $\langle T|_{\pi_i}, V\rangle \in L(\mathcal{D}^q)$, we have that $\langle T', V\rangle \in L(\mathcal{D}^{\delta(q,u)})$. By Lemma A.6 it follows that $\langle T', V\rangle \vDash \mathtt{State}(\delta(q, u))) = \mathtt{State}(\delta(q, w))$, 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.

A.4.3 Almost Linear DBWs

Lemma A.7. *An almost linear DBW $\mathcal{D}$ is equivalent to its constructed HLT $\mathcal{H}$, described in Section 5.2.*

Proof. 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 tree T. We claim that every path in the run-tree of $\mathcal{H}$ on T 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 some e label, as they were "released" by $\mathcal{H}$ by sending each one of them to $\mathtt{true}$. The rest of the paths are contained in the run-tree of $\mathcal{D}$ on T, and therefore are also accepting.

For the other direction, consider a tree T 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, and in particular q_e. It implies that at some point $\mathcal{H}$ is forced to imitate $\mathcal{D}$, and therefore to reject T. □