Branching-Time Logics with Path Relativisation

Markus Latte[1]

Department of Computer Science, Ludwig-Maximilians-University Munich, Germany

Martin Lange[2]

School of Electrical Engineering and Computer Science, University of Kassel, Germany

Abstract

We define extensions of the full branching-time temporal logic CTL* in which the path quantifiers are relativised by formal languages of infinite words, and consider its natural fragments obtained by extending the logics CTL and CTL$^+$ in the same way. This yields a small and two-dimensional hierarchy of temporal logics parametrised by the class of languages used for the path restriction on one hand, and the use of temporal operators on the other. We motivate the study of such logics through two application scenarios: in abstraction and refinement they offer more precise means for the exclusion of spurious traces; and they may be useful in software synthesis where decidable logics without the finite model property are required. We study the relative expressive power of these logics as well as the complexities of their satisfiability and model-checking problems.

Keywords: temporal logic, CTL*, formal languages, expressive power, model checking, satisfiability

1. Introduction

Branching-time temporal logics are some of the most well-known and used specification languages for reactive systems. The most prominent ones are the logics CTL [12, 15] and CTL* [16]. While CTL has nice algorithmic properties — model checking is P-complete and satisfiability checking is EXPTIME-complete — its expressive power is very weak. CTL* amends this by unifying CTL with the linear-time temporal logic LTL [34]. This way, it can express important properties like fairness which is not possible in CTL. This comes at a certain price, though.

Email addresses: markus.latte@ifi.lmu.de (Markus Latte), martin.lange@uni-kassel.de (Martin Lange)

[1]Supported by the Doctorate Program "Program and Model Analysis" of the German Research Foundation (GRK 1480)

[2]The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

Model checking CTL* is naturally at least as expensive as it is for LTL. In fact it is no more expensive either, thus, it is PSPACE-complete [35]. The additional complexity introduced by merging branching-time with linear-time formalisms shows through in satisfiability checking which is 2-EXPTIME-complete [39, 17].

Temporal logics in general, and with it such branching-time logics, have become prominent because of the success that model checking — an automatic program verification method for correctness properties specified in temporal logics — has had over the past decades [22]. In particular, model checking was very successful in hardware verification because hardware can be modeled by finite-state systems. Verifying infinite-state systems has become more and more important in the domain of program verification since, and this is mainly due to the growing importance of software in reactive systems. Note that software usually leads to infinite-state systems because of the use of unbounded data structures, recursive functions, etc.

The model-checking complexities mentioned above hold with respect to finite models. Clearly, model checking infinite-state programs is undecidable in general but it remains decidable for certain classes of infinite-state programs, e.g. pushdown processes, and weak temporal logics like CTL and LTL. It is still just PSPACE-complete for LTL but EXPTIME-complete for CTL [9, 41].

Several extensions and variations of such branching-time logics have been considered since, usually with special purposes in mind: Timed CTL has been introduced in order to verify properties in which real-time effects play a crucial role [2]; action-based CTL considers models with more than one accessibility relation [33]; Graded CTL adds some possibility of counting [7]; etc.

Here we consider extensions of branching-time logics in which the path quantifiers can be relativised to traces that belong to a formal language of ω-words. This defines a hierarchy of extensions parametrised by the class of formal languages which can be used for the relativisation. In Section 2 we introduce these logics — based on CTL, CTL* and the lesser known CTL$^+$ which is known to be only as expressive as CTL [15] but exponentially more succinct [42, 1, 26].

Section 3 motivates the use of these logics through two scenarios: abstraction and software synthesis. Still, the main focus of this paper are the logics themselves. Here we study the hierarchy of expressive power we obtain from these logics (Section 5) with respect to different classes of formal languages, namely the ω-regular ones, the ω-context-free ones and the ω-visibly pushdown ones. We study their the computational complexity and decidability of their satisfiability problems (Section 6) and of their model-checking problems (Section 7).

2. CTL* with Path Relativisation

2.1. Transition Systems

Models of CTL* with path relativisation are transition systems which—as opposed to ordinary CTL* models and like models of action-based CTL—also have labelled edges and need not be total. Let Σ be a finite alphabet and $\mathcal{P}$ be a countably infinite set of atomic propositions. A *transition system* is a tuple

$\mathcal{T} = (\mathcal{S}, \to, \lambda)$ where $\mathcal{S}$ is a set of *states*, $\to \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the *transition relation*, and $\lambda : \mathcal{S} \to 2^{\mathcal{P}}$ labels each state with a finite set of propositions that are true in this state. We write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \to$.

In order to simplify technical details, we assume that Σ always contains a special character d and that each transition system has a distinct state end with $s \xrightarrow{\mathsf{d}} \mathsf{end}$ for every s including end itself. Furthermore, end has no other incoming or outgoing transitions than these. This means that transition systems are total in the sense that in any state at least a d-action is possible. However, afterwards nothing else is possible any more. Thus, taking a d-transition somehow indicates being in a deadlock state.

A *path* in $\mathcal{T}$ is an infinite sequence $\pi = s_0, a_0, s_1, a_1, \ldots$, alternating between states and edge labels such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \in \mathbb{N}$. The assumption above ensures that no maximal paths other than infinite ones exist. We write $\Pi_{\mathcal{T}}(s)$ for the set of all paths through $\mathcal{T}$ that start in s. For $i \in \mathbb{N}$, the denote by $\pi{\uparrow}i$ the ith suffix of π, i.e. $s_i, a_i, s_{i+1}, \ldots$.

A path $\pi = s_0, a_0, s_1, a_1, \ldots$ determines in a unique way its trace — the ω-word $a_0 a_1 a_2 \ldots$ over Σ. Abusing notation we will identify a path with its trace of edge labels and sometimes simply write $\pi \in L$ for a path π and a language L.

2.2. Formal Languages and Automata

As usual, let Σ be a finite alphabet. Then Σ^ω denotes the set of all infinite words over Σ; ϵ denotes the empty word. A *formal ω-language*, or just language from now on, is a subset of Σ^ω. We are particularly interested in three classes of languages: the ω-regular ones ωREG, the ω-context-free ones ωCFL, and the ω-visibly pushdown ones ωVPL.

In order to be able to use languages in formulas they need to be represented syntactically. Here we choose automata for this purpose: nondeterministic Büchi automata for ωREG [11], nondeterministic Büchi pushdown automata for ωCFL [36], and nondeterministic Büchi visibly pushdown automata for ωVPL [3].

A *nondeterministic Büchi automaton* is a tuple $\mathcal{A} = (Q, q_I, \delta, F)$ where Q is a finite set of *states*, $q_I \in Q$ is a designated *starting state*, $F \subseteq Q$ is a designated set of *acceptance states*, and $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*. A *run* of $\mathcal{A}$ on a word $w = a_0 a_1 \ldots \in \Sigma^\omega$ is a sequence of states $q_0, q_1, \ldots$ such that $q_0 = q_I$ and $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$.

A *nondeterministic Büchi pushdown automaton* (ω-PDA) is a tuple $\mathcal{A} = (Q, \Gamma, \bot, q_I, \delta, F)$ where Q, q_I and F are as above, Γ is a finite *stack alphabet*, $\bot \in \Gamma$ is a designated *bottom-of-stack* symbol, and δ is a finite subset of $Q \times \Gamma \times \Sigma \times Q \times \Gamma^*$. A *run* of $\mathcal{A}$ on a word $w = a_0 a_1 \ldots$ is a sequence of pairs of states and finite stacks over Γ of the form $(q_0, \gamma_0), (q_1, \gamma_1), \ldots$ such that $q_0 = q_I$, $\gamma_0 = \bot$, and for all $i \in \mathbb{N}$: $\gamma_i = \gamma B$, $\gamma_{i+1} = \gamma\gamma'$ and $(q_i, B, a_i, q_{i+1}, \gamma') \in \delta$ for some $B \in \Gamma$ and some $\gamma, \gamma' \in \Gamma^*$.

For the last kind of automaton we need a fixed partition of Σ into three disjoint parts Σ_{push}, Σ_{pop}, and Σ_{int}. A *nondeterministic Büchi visibly pushdown automaton* is a tuple $\mathcal{A} = (Q, \Gamma, \bot, q_I, \delta, F)$ as above with the exception of

$$\delta \subseteq (Q \times \Gamma \times \Sigma_{\mathsf{push}} \times Q \times \Gamma) \cup (Q \times \Gamma \times \Sigma_{\mathsf{pop}} \times Q) \cup (Q \times \Gamma \times \Sigma_{\mathsf{int}} \times Q) .$$

A *run* is a sequence of state-stack pairs as above with the provision that for all $i \in \mathbb{N}$ one of the following four cases holds.

- $a_i \in \Sigma_{\mathsf{push}}$, $\gamma_i = \gamma B$, $\gamma_{i+1} = \gamma BC$ for some $\gamma \in \Gamma^*$, $B, C \in \Gamma$, and $(q_i, B, a_i, q_{i+1}, C) \in \delta$.
- $a_i \in \Sigma_{\mathsf{pop}}$, $\gamma_i = \gamma B$ for some $\gamma \in \Gamma^*$, $B \in \Gamma$, $\gamma_{i+1} = \gamma$, and $(q_i, B, a_i, q_{i+1}) \in \delta$.
- $a_i \in \Sigma_{\mathsf{pop}}$, $\gamma_i = \gamma_{i+1} = \bot$, and $(q_i, \bot, a_i, q_{i+1}) \in \delta$.
- $a_i \in \Sigma_{\mathsf{int}}$, $\gamma_i = \gamma B$ for some $\gamma \in \Gamma^*$, $B \in \Gamma$, $\gamma_{i+1} = \gamma_i$ and $(q_i, B, a_i, q_{i+1}) \in \delta$.

Thus, a visibly pushdown automaton acts very much like a pushdown automaton, but the currently read input symbol determines whether something or nothing gets pushed onto the stack, or popped off the stack.

A run of any of these automata on a word w is *accepting* if it contains infinitely many states in F. The *language* $L(\mathcal{A})$ of the automaton $\mathcal{A}$ is the set of all words w for which there is an accepting run of $\mathcal{A}$ on w.

A typical visibly pushdown ω-language is for instance $\{a^n b^n c^\omega \mid n \in \mathbb{N}\}$ if a is a push-symbol, b is a pop-symbol, and c is an int-symbol. Equally, $\{a^n b^n a^\omega \mid n \in \mathbb{N}\}$ is a visibly pushdown ω-language over the same alphabet, but $\{a^n b a^n c^\omega \mid n \in \mathbb{N}\}$ for instance is not, because one needs a push- as well as a pop-phase whilst reading the a-part of an input word in order to compare their lengths. However, it is not too hard to construct a Büchi pushdown automaton recognising this language.

The size of an automaton is $|\mathcal{A}| = |Q|$, i.e. the number of its states.

2.3. Syntax and Semantics of Path Relativised CTL*

Let $\mathfrak{A} \subseteq 2^{\Sigma^\omega}$ be a class of ω-languages. Formulas of CTL* *with path relativisation*, CTL$^*[\mathfrak{A}]$, are built like CTL* formulas with the difference that path quantifiers are syntactically indexed by languages of ω-words.

$$\varphi \ ::= \ q \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{X}\varphi \mid \varphi \,\mathtt{U}\, \varphi \mid \mathtt{E}_L \varphi$$

where $q \in \mathcal{P}$ and $L \in \mathfrak{A}$.

We take the liberty of identifying a language L with the smallest automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$. This will make the presentation of formulas in examples much more readable and yields a finite syntax for these logics with a well-defined notion of formula size.

Also, we restrict — unless stated otherwise — our attention to the classes ωREG, ωCFL and ωVPL as defined above, thus obtaining the three logics CTL$^*[\omega\text{REG}]$, CTL$^*[\omega\text{VPL}]$, and CTL$^*[\omega\text{CFL}]$.

The semantics of CTL$^*[\mathfrak{A}]$ is given as follows. Let $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ be a transition system as above. In particular, all paths in it are infinite. For any path $\pi = s_0, a_0, s_1, a_1, \ldots$ in $\mathcal{T}$ we have:

$$\begin{aligned}
\mathcal{T}, \pi \models q \quad & \text{iff} \quad q \in \lambda(s_0) \\
\mathcal{T}, \pi \models \neg\varphi \quad & \text{iff} \quad \mathcal{T}, \pi \not\models \varphi \\
\mathcal{T}, \pi \models \varphi \vee \psi \quad & \text{iff} \quad \mathcal{T}, \pi \models \varphi \text{ or } \mathcal{T}, \pi \models \psi
\end{aligned}$$

$$\begin{array}{rcl}
\mathcal{T},\pi \models \mathtt{X}\varphi & \text{iff} & \mathcal{T},\pi{\uparrow}1 \models \varphi \\
\mathcal{T},\pi \models \varphi\,\mathtt{U}\,\psi & \text{iff} & \exists i \in \mathbb{N} \text{ with } \mathcal{T},\pi{\uparrow}i \models \psi \text{ and } \forall j < i : \mathcal{T},\pi{\uparrow}j \models \varphi \\
\mathcal{T},\pi \models \mathtt{E}_L\varphi & \text{iff} & \exists \pi' \in \Pi_{\mathcal{T}}(s_0) \text{ s.t. } \pi' \in L \text{ and } \mathcal{T},\pi' \models \varphi
\end{array}$$

We introduce the abbreviation $\mathtt{E}\varphi := \mathtt{E}_{\Sigma^\omega}\varphi$, i.e. if the path relativiser does not restrict the choice of the path at all then it is not mentioned explicitly.

Two formulas are equivalent, written $\varphi \equiv \psi$, if for all transition systems $\mathcal{T}$, all its states s and all paths $\pi, \pi' \in \Pi_{\mathcal{T}}(s)$ we have: $\mathcal{T},\pi \models \varphi$ iff $\mathcal{T},\pi \models \psi$.

A state formula is a formula φ such that $\varphi \equiv \mathtt{E}\varphi$. Thus, the value of a state formula only depends on the first state of the path it is interpreted in. If φ is a state formula we will simply write $\mathcal{T}, s \models \varphi$. In order to enable a comparison to other formalisms we restrict our attention to state formulas.

Subformulas are defined as usual. The size of a formula φ, written as $|\varphi|$, is the number of its subformulas plus the sizes of the automata occurring in it.

2.4. Abbreviations and Fragments

In addition to the unrestricted path quantification we introduce more abbreviations like

- the Boolean constants $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathcal{P}$ and $\mathtt{ff} := \neg\mathtt{tt}$,
- other Boolean operators like $\wedge$ and $\rightarrow$ in the usual way,
- the standard temporal operators $\mathtt{F}\varphi := \mathtt{tt}\,\mathtt{U}\,\varphi$, $\mathtt{G}\varphi := \neg\,\mathtt{F}\,\neg\varphi$, and $\varphi\,\mathtt{R}\,\psi := \neg(\neg\varphi\,\mathtt{U}\,\neg\psi)$,
- universal path quantification via $\mathtt{A}_L\varphi := \neg\mathtt{E}_L\neg\varphi$ and $\mathtt{A}\varphi := \neg\mathtt{E}\neg\varphi$,
- operators from action-based CTL like $\mathtt{E}\,\mathtt{X}_a\,\varphi := \mathtt{E}_{a\Sigma^\omega}\,\mathtt{X}\,\varphi$, etc.

Clearly, these syntactic extensions do not extend the expressive power of the language. On the other hand, we will also consider syntactic restrictions guided by the standard fragments of CTL* and study them with respect to expressiveness and computational complexity of their decision problems.

Let $\mathfrak{A} \subseteq 2^{\Sigma^\omega}$ as above. Formulas of CTL *with Path Relativisation* are given by the following grammar.

$$\varphi \ ::= \ q \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{E}\,\mathtt{X}_a\,\varphi \mid \mathtt{E}_L(\varphi\,\mathtt{U}\,\varphi) \mid \mathtt{A}_L(\varphi\,\mathtt{U}\,\varphi)$$

where $q \in \mathcal{P}$, $a \in \Sigma$ and $L \in \mathfrak{A}$.

Similarly, formulas of CTL$^+$ *with Path Relativisation* are derived from φ in the following grammar.

$$\begin{array}{rcl}
\varphi & ::= & q \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{E}_L\psi \\
\psi & ::= & \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathtt{X}\varphi \mid \varphi\,\mathtt{U}\,\varphi
\end{array}$$

where, again, $q \in \mathcal{P}$ and $L \in \mathfrak{A}$.

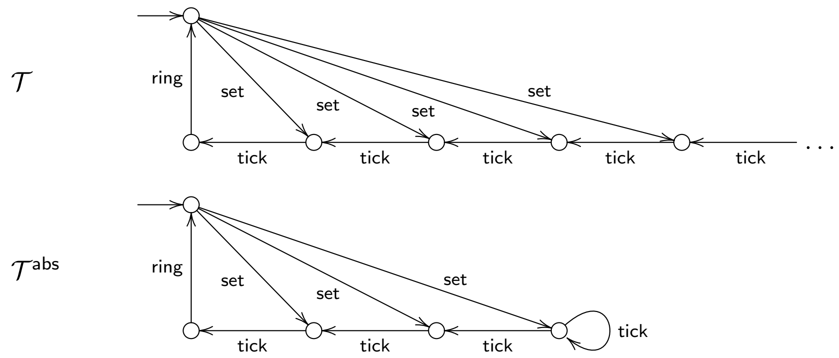


Figure 1: Transition system of an alarm clock and an abstraction.

It is not hard to see, that CTL[$\mathfrak{A}$] is a syntactic fragment of CTL$^+$[$\mathfrak{A}$] which in turn is a syntactic fragment of CTL*[$\mathfrak{A}$] for any fixed $\mathfrak{A}$ containing Σ^ω. Furthermore, all formulas of the two smaller logics are state formulas in the above sense.

3. Motivation

We motivate the study of path relativised branching-time logics through two scenarios: *abstraction and refinement*, and *software synthesis*.

3.1. Abstraction and Refinement

Verification of infinite-state (or just very large) systems is often computationally infeasible or undecidable, and therefore done by considering finite and smaller systems instead. The step from a larger to a smaller system can incur loss of information about the behaviour of the larger system, hence, the smaller system is an abstraction of the larger one. This is often done in a way such that the abstract system approximates the behaviour of the concrete one, for instance by having at least all the traces of the concrete one but possibly more [13]. If a property without existential path quantification is verified on the abstract system it then also holds on the concrete one. The other direction does not hold in general.

Consider an alarm clock $\mathcal{T}$ which can be set to count down an arbitrary number of steps and then ring. Its transition system is depicted in the top of Figure 1. Clearly, an alarm clock should ring eventually once it is set to a certain time, therefore, the alarm clock should not have a state from which an infinite tick-path exists. This property is specifiable in action-based CTL as $\mathtt{A}\,\mathtt{G}\,\neg\mathtt{E}\,\mathtt{G}_{\mathsf{tick}}\,\mathtt{tt}$.

Now consider an abstraction of this system which collapses all counter values that are greater than 2. This introduces a tick-loop in the rightmost state of the abstracted system $\mathcal{T}^{\mathsf{abs}}$ depicted at the bottom of Figure 1.

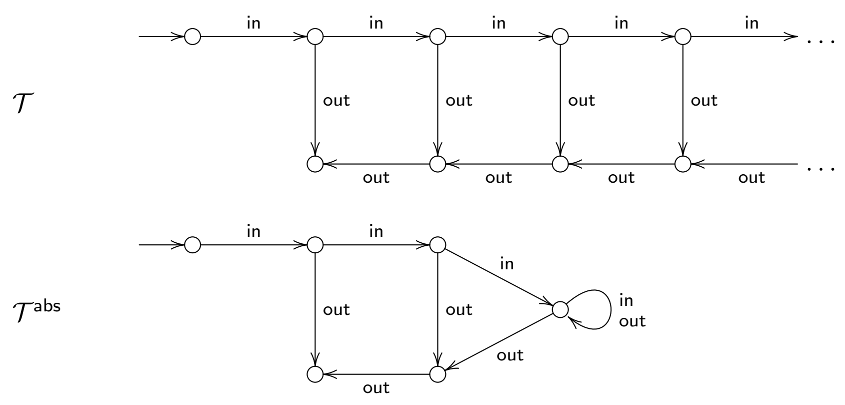


Figure 2: Transition system of a buffer system and an abstraction.

It should be clear that $\mathcal{T} \models \mathtt{A}\,\mathtt{G}\,\neg\mathtt{E}\,\mathtt{G}_{\mathsf{tick}}\,\mathtt{tt}$ but that $\mathcal{T}^{\mathsf{abs}} \not\models \mathtt{A}\,\mathtt{G}\,\neg\mathtt{E}\,\mathtt{G}_{\mathsf{tick}}\,\mathtt{tt}$. The reason for this is the spurious trace that stays in the rightmost state forever. It is called spurious because it has only been created through the abstraction process since it does not exist in the concrete system.

It has been suggested not to check the original property on the abstract system but to amend not only the system through abstraction but the property as well, namely through the introduction of fairness predicates [8]. Take, for instance the fairness predicate $\Phi := \mathtt{G}\,\mathtt{F}\,\mathsf{tick} \Rightarrow \mathtt{G}\,\mathtt{F}\,\mathsf{ring}$, i.e. if infinitely many ticks are being done then infinitely many rings are being done, too. Now it is the case that $\mathcal{T}^{\mathsf{abs}} \models_{fair} \mathtt{A}\,\mathtt{G}\,\neg\mathtt{E}\,\mathtt{G}_{\mathsf{tick}}\,\mathtt{tt}$ under this fairness predicate, meaning that the CTL path quantifiers in this formula now only range over fair paths, i.e. those that satisfy the fairness predicate Φ.

While this does work in this particular case, the introduction of a fairness predicate seems rather arbitrary as well as its choice. Furthermore, the chosen fairness predicate almost contradicts the correctness property at hand. Hence, this is almost like only considering that part of the abstracted system which does satisfy the correctness property and then showing that it does indeed. In other words, finding the right fairness predicate may be as hard as showing correctness of the original system.

Branching-time temporal logics with path relativisation offer a more fine-tuned and more systematic way of amending the correctness properties. We will consider another example in which the introduction of fairness is not able to exclude spurious traces. Consider a system containing a buffer into which items can be placed and from which items can be taken. It works such that once something is taken out, it can only be emptied and nothing more can be put into it. The transition system $\mathcal{T}$ is depicted on top in Figure 2. An abstraction $\mathcal{T}^{\mathsf{abs}}$ which collapses all states containing more than 2 buffer items is depicted below that.

Now consider the correctness property stating that at no point is it possible to execute an out-action followed by an in-action. In action-based CTL it can be written as $\mathtt{A}\,\mathtt{G}\,\neg\mathtt{E}\,\mathtt{X}_{\mathsf{out}}\,\mathtt{E}\,\mathtt{X}_{\mathsf{in}}\,\mathtt{tt}$. Clearly, it is satisfied by the original system $\mathcal{T}$ and not satisfied by the abstraction $\mathcal{T}^{\mathsf{abs}}$ because of the spurious trace through the self-loop in the state representing all large buffer contents. The important observation about this is, though, that no fairness predicate can exclude all the spurious traces which cause the violation of the correctness property. This is simply because fairness is concerned with the infinite occurrence of states / actions, etc. or the absence thereof. The characteristics of the spurious traces in this case, however, is the single occurrence of an in-action after a single out-action. It is therefore sensible to restrict the path quantification to traces of the form $\mathsf{in}^\omega \cup \mathsf{in}^*\mathsf{out}^*\mathsf{d}^\omega$ where action d indicates, as introduced above, a transition into an imaginary deadlock state.

The issue about the right choice of path relativisation still persists, though. As in the first example, the trace predicate $\mathsf{in}^\omega \cup \mathsf{in}^*\mathsf{out}^*\mathsf{d}^\omega$ is somehow found miraculously. However, CTL[$\mathfrak{A}$] allows for a more automatic approach depending on $\mathfrak{A}$. Note that $\mathcal{T}$ is indeed a visibly pushdown system with push-action in and pop-action out. The language of its traces is a visibly pushdown language (ωVPL), characterised by the property that no in-action occurs after an out-action and on any prefix, the number of out-actions is at most as high as the number of in-actions. Let L be that language. Using CTL[ωVPL] it is then possible to replace the correctness property above by $\mathtt{A}\,\mathtt{G}\,\neg\mathtt{E}_{L\cap\Sigma^*\,\mathsf{out}\,\mathsf{in}\,\Sigma^\omega}\,\mathtt{F}\,\mathtt{tt}$ for instance and test that on the abstracted system. Note how this restricts path quantification to traces which are present in the original system only. This is of course the essence of excluding spurious traces.

It should be clear that the formalism of well-known branching-time logics with the additional path relativisation offers a systematic way of amending properties to be checked on the original system. The path relativiser can be used to restrict quantification to those paths that exist in the original system if the language of traces of that system is expressible in the class of formal languages $\mathfrak{A}$ used as a parameter to the logic. It is also a major advantage to base such expressive logics on well-known ones like CTL* and CTL. This could enable an automatic abstraction process in which the user provides the correctness properties in an intuitive language known to him like these branching-time logics, and the transformation into their path relativised variants is done in the background.

It should also be clear that the use of path relativised logics does not miraculously solve all the problems arising with sound and incomplete abstractions. In particular, there is no general recipe for the choice of the language used for the relativisation, just as there is none for the exact fairness predicate in the example above. It remains to be seen how heuristics can help to guide such choices [27]. The advantage of path relativisation based on formal languages over specific path relativisations using fairness constraints for instance lies in the much greater power that the former provides for the exclusion of spurious traces.

3.2. Software Synthesis

Synthesis is the problem of automatically generating, given a specification φ, a model $\mathcal{T}$ of φ. Note that this is more general than the satisfiability problem since it implicitly answers the question of whether or not φ is satisfiable.

The synthesis problem has been considered before, in particular in the context of automatically generating controllers, i.e. components of systems that guide the system's behaviour such that a given specification is satisfied [4]. Such work has considered temporal logics which possess the finite model property, i.e. for which every satisfiable formula has a finite model. Synthesis for specification languages with the finite model property therefore results in the automatic generation of finite programs, for instance hardware. Software gives rise to infinite-state models, though, through the use of unbounded variable values, data structures, recursion, etc.

Thus, in order to be able to synthesise software (skeletons) one needs specification logics that are decidable but do not have the finite model property. Examples of such logics are rare, in particular in the area of program logics which are often required to be bisimulation-invariant. Hence, the lack of finite model property must not be based on the lack of bisimulation-invariance. An example of such a logic is PDL with Intersection whose satisfiability problem is decidable [14] and which can make non bisimulation-invariant assertions like "there is an infinite path and no loop". Clearly, such an assertion can only be satisfied in an infinite model.

The branching-time logics using path relativisation with ω-visibly pushdown language satisfy these two requirements: lack of finite model property and decidability. Another example of such a logic is PDL with Recursive Programs [31] which is used here in order to obtain decidability and complexity results for the branching-time logics. Compared to that, the logics proposed here have the advantage of a more intuitive syntax which is useful for pragmatic aspects in software synthesis.

4. Related Formalisms

We briefly mention some logics that are related to the ones presented here. The branching-time logics CTL and CTL* have been extended in various ways to make up for all sorts of deficiencies. For instance, there are several versions of action-based CTL which are interpreted — as the logics in this paper and as opposed to ordinary CTL — over transition systems with edge labels [6, 10, 32]. They refine the temporal operators in CTL using regular languages (of finite words) in one way or the other. Clearly, their expressive power does not extend further than ω-regular tree languages and this is why we do not consider them any further. In particular, we leave the question of determining the exact relationship between CTL[ωREG] and these action-based variants of CTL to the reader since the focus of this paper are the non-regular extensions of CTL.

Extensions of branching-time logics with specifications of infinite traces have also been considered, in particular ECTL and ECTL$^+$ which add to CTL and CTL$^+$, resp., temporal operators of the form `FG` and `GF` [16]. This is done in order to remedy their weakness of not being able to express fairness properties. Still, it should be clear that this extension does not capture the whole world of ω-regularity, hence, CTL[ωREG] and CTL$^+$[ωREG] are certainly not embeddable into those extensions.

There have also been investigations into extensions of CTL (but not CTL* or CTL$^+$) beyond regularity [5]. Extended CTL — the name does not seem to identify the logic uniquely — enhances the temporal operators in CTL with formal languages of finite words. These are used to constrain the moments in which an *until* property should be fulfilled for instance, or to relax a *generally* formula. These investigations consider classes of languages beyond the regular ones as it is done here. However, the two formalisms are quite different since here we use the formal languages to constrain paths rather than moments along a path.

Finally, we mention Propositional Dynamic Logic over a class $\mathfrak{A}$ of formal languages of finite words [19], PDL[$\mathfrak{A}$], which is one of the very rare logics which has also been considered with non-regular features in mind. This simply is standard modal logic with an infinite set of accessibility relations. In case of $\mathfrak{A}$ being the class of regular languages they form a Kleene algebra, i.e. are describable by regular expressions. On the other hand, the modal operator $\langle L\rangle$, where L is such a description, can also be read as “there is a finite path whose trace is in the language L”. It is then easy to consider PDL over other classes of languages, for instance the context-free ones [23] or the visibly pushdown ones [31]. Furthermore, we consider its extension by tests and the delta operator [37, 31] which can be used to postulate the existence of an *infinite* path whose trace belongs to some language. This will be one of the main tools through which we obtain upper bounds on the complexity of the branching-time logics considered here, as well as measure their relative expressive power.

Remembering the setting of abstract interpretation as described in the previous section, it should be clear that only the non-regular extensions of the logics mentioned here could be suitable for the task described above. $\Delta\text{PDL}^?[\omega\text{VPL}]$ is a very powerful logic with suitable algorithmic and model-theoretic properties. However, it does not provide an elegant syntax, in particular the temporal operators that one has in CTL[ωVPL] need to be encoded in the formal language part in $\Delta\text{PDL}^?[\omega\text{VPL}]$. This makes it less suitable for a generic amendment of correctness properties in abstract interpretation. Extended CTL, as mentioned above, does provide such a nice syntax but it is not clear how the use of formal languages of finite words could be used for the systematic exclusion of infinite spurious traces.

5. Expressivity

We begin by investigating the (relative) expressive power of the logics introduced above. An overview of the partial order that these logics form w.r.t. expressivity is given in Figure 3. An arrow pointing from $\mathcal{L}$ to $\mathcal{L}'$ indicates that $\mathcal{L}'$ is at least as expressive as $\mathcal{L}$. Formally, we write $\mathcal{L} \leq_f \mathcal{L}'$ with $f \in \{\mathsf{lin}, \mathsf{exp}\}$ to state that for every formula $\varphi \in \mathcal{L}$ there is an equivalent $\psi \in \mathcal{L}'$ with at most a linear or exponential (respectively) blow up in size. Implicitly, such a statement involving a language L for path relativisation requires that L is given in terms of an automaton as named in Section 2.3.

$$\begin{tikzcd}
 & \Delta\mathrm{PDL}^?[\omega\mathrm{REG}] \arrow[r, dashed] & \Delta\mathrm{PDL}^?[\omega\mathrm{VPL}] \arrow[r, dashed] & \Delta\mathrm{PDL}^?[\omega\mathrm{CFL}] \\
\mathrm{CTL}^* \arrow[r, dashed] & \mathrm{CTL}^*[\omega\mathrm{REG}] \arrow[r, dashed] \arrow[u] & \mathrm{CTL}^*[\omega\mathrm{VPL}] \arrow[r, dashed] \arrow[u] & \mathrm{CTL}^*[\omega\mathrm{CFL}] \arrow[u] \\
\mathrm{CTL}^+ \arrow[r, dashed] \arrow[u, dashed] & \mathrm{CTL}^+[\omega\mathrm{REG}] \arrow[r, dashed] \arrow[u, dashed] & \mathrm{CTL}^+[\omega\mathrm{VPL}] \arrow[r, dashed] \arrow[u, dashed] & \mathrm{CTL}^+[\omega\mathrm{CFL}] \arrow[u, dashed] \\
\mathrm{CTL} \arrow[r, dashed] \arrow[u, leftrightarrow] & \mathrm{CTL}[\omega\mathrm{REG}] \arrow[r, dashed] \arrow[u] & \mathrm{CTL}[\omega\mathrm{VPL}] \arrow[r, dashed] \arrow[u] & \mathrm{CTL}[\omega\mathrm{CFL}] \arrow[u]
\end{tikzcd}$$

Figure 3: Hierarchy of expressive power.

We write $\mathcal{L} \lneq_f \mathcal{L}'$ to denote that such a translation exists, but there are formulas of $\mathcal{L}'$ which are not equivalent to any formula in $\mathcal{L}$. Such a relation is depicted using a dashed arrow in Figure 3. In any case, we will drop the index if a potential blow-up is of no concern.

To shorten statements about the variety of logics, $\mathrm{CTL}^\blacklozenge$— including their path relativisations — is used to denote either of CTL, CTL^+ or CTL^*.

First we prove that using path relativisation at all increases the expressive power in any case.

Theorem 1. $\mathrm{CTL}^\blacklozenge \lneq_{\mathsf{lin}} \mathrm{CTL}^\blacklozenge[\mathfrak{A}]$ *for any* $\mathfrak{A} \supsetneq \{\Sigma^\omega, \emptyset\}$.

Proof. The embedding of CTL, CTL^+ and CTL^*, respectively, is trivial using Σ^ω as a quantifier restriction. The strictness of this embedding is a very simple consequence of the fact that any logic $\mathrm{CTL}^\blacklozenge$ cannot distinguish path labels. Suppose $\mathfrak{A} \supsetneq \{\Sigma^\omega, \emptyset\}$. Then there is an $L \in \mathfrak{A}$ such that there are words $w, v \in \Sigma^\omega$ with $w \in L$ and $v \notin L$. Now consider the formula $\mathtt{E}_L\ \mathtt{G}\ \mathtt{tt}$. It is satisfied by the model which has a single path labelled w only such that all states are labelled with $\emptyset$ for instance. Suppose there was an equivalent $\mathrm{CTL}^\blacklozenge$ formula φ. Now consider the model that arises from the above by changing the path label into v. Then this still satisfies φ but it does not satisfy $\mathtt{E}_L\ \mathtt{G}\ \mathtt{tt}$ anymore. □

It should be clear that using a richer syntax or a bigger class of formal languages cannot result in less expressive power which is expressed by the next two statements.

Proposition 2. $\mathrm{CTL}^\blacklozenge[\mathfrak{A}] \leq \mathrm{CTL}^\blacklozenge[\mathfrak{B}]$ *for* $\mathfrak{A} \subseteq \mathfrak{B}$.

Proposition 3. $\mathrm{CTL}[\mathfrak{A}] \leq \mathrm{CTL}^+[\mathfrak{A}] \leq \mathrm{CTL}^*[\mathfrak{A}]$ *for all* $\mathfrak{A}$.

We show that $\Delta\mathrm{PDL}^?[\mathfrak{A}]$ does indeed bound the expressive power of the branching-time logics from above. We consider this separately for $\mathrm{CTL}[\mathfrak{A}]$ on one hand and $\mathrm{CTL}^*[\mathfrak{A}]$ (and with it also $\mathrm{CTL}^+[\mathfrak{A}]$) on the other because of the involved blow-ups in the respective translations.

Theorem 4. $\mathrm{CTL}^*[\mathfrak{A}] \leq_{\mathsf{exp}} \Delta\mathrm{PDL}^?[\mathfrak{A}]$ *for any class* $\mathfrak{A}$ *which is closed under intersection with* ω*-regular languages.*

Proof. The translation, say $\widetilde{\cdot}$, is defined inductively. We detail some cases only. For a formula $\mathtt{E}_L\varphi$, its translation is $\Delta\mathcal{A}$ where $\mathcal{A}$ is the intersection of the automaton represented by L with an automaton $\mathcal{B}$. For the latter, the LTL-part of φ can be rephrased [40] as a Büchi automaton of exponential size in $|\varphi|$. That is, any outermost path-quantified subformula ψ is replaced by a test-operator $\widetilde{\psi}?$ in $\mathcal{B}$. The case of $\mathtt{A}_L\varphi$ can be handled by using negation and complementation. Finally, the size of the constructed formula is exponential in the size of the given one, as duplications of subformulas do not affect the size. $\square$

All considered language classes are closed under intersection with ω-regular languages. Since CTL[$\mathfrak{A}$] is a fragment of CTL*[$\mathfrak{A}$], we obtain — for free — a translation of CTL[$\mathfrak{A}$] into ΔPDL$^?$[$\mathfrak{A}$] from Theorem 4. In that case, the exponential blow-up is unnecessary though. In fact, there is a linear translation.

Theorem 5. CTL[$\mathfrak{A}$] $\leq_{\mathsf{lin}}$ ΔPDL$^?$[$\mathfrak{A}$] *for* $\mathfrak{A} \in \{\omega\text{REG}, \omega\text{VPL}, \omega\text{CFL}\}$.

Proof. The embedding is proved by induction on the structure of formulas in CTL[$\mathfrak{A}$]. We detail only the case of $\theta := \mathtt{E}_{\mathcal{A}}(\varphi\,\mathtt{U}\,\psi)$ for an automaton $\mathcal{A}$ with states Q, initial state q_0, and final states F. Let φ' and ψ' be the translations of φ and ψ, respectively. The translation of θ is $\langle\mathcal{B}\rangle\mathtt{tt}$ where $\mathcal{B}$ is an automaton of the same kind as $\mathcal{A}$ with states containing $Q \times \{0,1\}$, initial state $(q_0, 0)$ and final states $F \times \{1\}$. The second component shall denote whether the $\mathtt{U}$ has been fulfilled. Let $p \xrightarrow[act]{a} q$ denote a transition in $\mathcal{A}$ leading from state p by reading $a \in \Sigma \cup \{\epsilon\}$ to state q while performing operation act on the stack — if applicable. Then $\mathcal{B}$ contains the following three transitions for $p \xrightarrow[act]{a} q$.

$$(q,0) \xrightarrow[nop]{?\varphi'} \xrightarrow[act]{a} (q',0) \qquad (q,0) \xrightarrow[nop]{?\psi'} \xrightarrow[act]{a} (q',1) \qquad (q,1) \xrightarrow[act]{a} (q',1)$$

$\square$

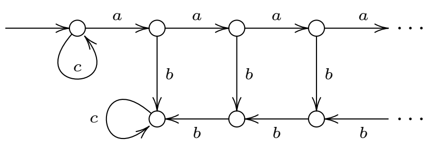
Next we consider some model-theoretic properties and use them to separate logics along the horizontal axis in Figure 3. A logic is said to have the *finite model property* if every satisfiable formula of that logic has a finite transition system as a model.

Theorem 6. CTL[ωVPL] *does not have the finite model property.*

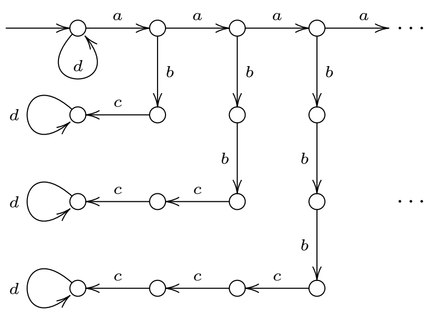
Proof. Consider the visibly pushdown alphabet Σ with a push-action a, a pop-action b and an internal action c. Let $L := \{a^n b^m c^\omega \mid n \neq m\}$ and take the CTL[ωVPL]-formula

$$\varphi \;=\; \mathtt{E}_{a^\omega}\,\mathtt{G}\,\mathtt{E}_{b^* c^\omega}\,\mathtt{G}\,\mathtt{tt} \quad\wedge\quad \mathtt{A}_L\,\mathtt{G}\,\mathtt{ff}\ .$$

Assume it has a finite model of size n. As the formula enforces a path $a^{n+1}b^{n+1}c^\omega$, the model also contains a path $a^m b^{n+1} c^\omega$ with $m < n+1$. However, this path violates $\mathtt{A}_L\,\mathtt{G}\,\mathtt{ff}$. On the other hand, φ is satisfiable: a model is shown in Figure 4(a). $\square$



(a) For Theorem 6.



(b) For Theorem 8.

Figure 4: Models used in the proofs of Theorem 6 and 8.

An immediate consequence is the separation of the logics over ω-regular and the ones over ω-visibly pushdown languages.

Corollary 7. $\mathrm{CTL}^{\blacklozenge}[\omega\mathrm{REG}] \lneq \mathrm{CTL}^{\blacklozenge}[\omega\mathrm{VPL}]$.

Proof. A $\mathrm{CTL}^{\blacklozenge}[\omega\mathrm{REG}]$-formula can be translated into a $\Delta\mathrm{PDL}^{?}[\omega\mathrm{REG}]$-formula by Proposition 3 and Theorem 4. However, the latter has the finite model property [19, Thm. 3.2]. $\square$

We continue along the same lines in order to obtain a separation between the logics over ω-visibly pushdown languages and the ones over ω-context-free languages. A logic is said to have the *visibly pushdown model property* if every satisfiable formula of this logic has a model which can be represented as the transition graph of a visibly pushdown automaton. Furthermore, it has the *pushdown model property* if the same holds for models which can be represented as transition graphs of pushdown automata.

Theorem 8. $\mathrm{CTL}[\omega\mathrm{CFL}]$ *does not have the pushdown model property.*

Proof. Let $L_1 := \{a^i b^j c^* d^\omega \mid i \neq j\}$ and let $L_2 := \{a^* b^i c^j d^\omega \mid i \neq j\}$ be two ωCFLs over the alphabet $\{a, b, c, d\}$. Consider the $\mathrm{CTL}[\omega\mathrm{CFL}]$-formula

$$\varphi \quad = \quad \mathtt{E}_{a^\omega}\ \mathtt{G}\ \mathtt{E}_{b^* c^* d^\omega}\ \mathtt{G}\ \mathtt{tt} \quad \wedge \quad \mathtt{A}_{L_1}\ \mathtt{G}\ \mathtt{ff} \quad \wedge \quad \mathtt{A}_{L_2}\ \mathtt{G}\ \mathtt{ff} \quad \wedge \quad \mathtt{A}_{\overline{a^* b^* c^* d^\omega}}\ \mathtt{G}\ \mathtt{ff}\ .$$

In any model $\mathcal{T}$ the set of all paths — starting from the initial state — is $\{a^n b^n c^n d^\omega \mid n \in \mathbb{N}\}$. For the sake of contradiction, assume that such a model was representable by a pushdown system. Now, we take any state of the system as a final state which can handle the input d. The obtained pushdown automaton (on finite words) would accept the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ which is of course impossible. On the other hand, the formula φ is indeed satisfiable: a model is depicted in Figure 4(b). $\square$

In order to conclude a gap in expressive power we also need an upper bound on the logics over ω-visibly pushdown languages similar to the finite model property of the regular ones used above. Clearly, the finite model property cannot be used because its lack has already been established in Theorem 6. However, the visibly pushdown model property works in this case. We state it here for the logic $\mathrm{CTL}^*[\omega\mathrm{VPL}]$. The proof uses a close inspection of the decision procedure for the stronger logic $\Delta\mathrm{PDL}^?[\omega\mathrm{VPL}]$, hence that logic also has this property.

Theorem 9. $\mathrm{CTL}^\blacklozenge[\omega\mathrm{VPL}]$ *has the visibly pushdown model property.*

Proof. Any formula $\varphi \in \mathrm{CTL}^\blacklozenge[\omega\mathrm{VPL}]$ can be translated into an equivalent formula $\varphi' \in \Delta\mathrm{PDL}^?[\omega\mathrm{VPL}]$ according to Proposition 3 and Theorem 4. Such a formula is satisfiable iff the language of a certain stair-parity visibly-pushdown tree automaton is not empty [29]. Every tree in the language represents a model. The emptiness test can be rephrased as whether or not the $\exists$-player has a winning strategy in the corresponding visibly pushdown game with a stair-parity acceptance condition. This game can be translated [30] into a parity game. Following this chain, any winning strategy for the $\exists$-player in the latter game can be lifted to a visibly pushdown system which models φ. $\square$

The expressivity gap is then a consequence of this upper bound and the lower bound stated in Theorem 8 with the trivial observation that every visibly pushdown system is also a pushdown system.

Corollary 10. $\mathrm{CTL}^\blacklozenge[\omega\mathrm{VPL}] \lneq \mathrm{CTL}^\blacklozenge[\omega\mathrm{CFL}]$.

Next we separate logics along the vertical axis. The main tool for this is the (in-)expressibility of fairness, as it is for the separation between CTL^* on one hand and CTL^+ and CTL on the other. Fairness describes the linear-time property "infinitely often p" for some atomic proposition p. A branching-time logic is said to be able to express fairness (w.r.t. p) if it can formalise the existence of a path on which p holds infinitely often.

We remark that it is known that CTL^+ is only as expressive as CTL despite its richer syntax [15]. This does not carry over if fairness is introduced, though: The logic ECTL^+ which enhances CTL^+ with temporal operators $\mathtt{GF}$ and $\mathtt{FG}$ for expressing fairness and its complement, is strictly more expressive as ECTL, the corresponding counterpart on top of CTL [16]. We suspect that $\mathrm{CTL}^+[\mathfrak{A}]$ is also strictly more expressive than $\mathrm{CTL}[\mathfrak{A}]$ for any reasonable class $\mathfrak{A}$.

Lemma 11. $\mathrm{CTL}^+[\mathfrak{A}]$ *cannot express fairness for any class* $\mathfrak{A}$ *of languages.*

Proof. For the sake of contradiction, assume that there is $\mathrm{CTL}^+[\mathfrak{A}]$-formula φ expressing this property. This would also characterise fairness over transition systems using a single accessibility relation only. For such systems the quantifiers are relativised either to $\emptyset$ or to Σ^ω. Hence, φ can be understood as a CTL^+-formula. But fairness is not expressible in CTL^+ [16, 15]. $\square$

On the other hand, $\mathtt{E}_{\Sigma^\omega}\ \mathtt{G}\,\mathtt{F}p$ expresses fairness in $\mathrm{CTL}^*[\mathfrak{A}]$ for as long as $\mathfrak{A}$ contains the universal language Σ^ω.

Corollary 12. $\mathrm{CTL}[\mathfrak{A}] \lneq \mathrm{CTL}^*[\mathfrak{A}]$ *and* $\mathrm{CTL}^+[\mathfrak{A}] \lneq \mathrm{CTL}^*[\mathfrak{A}]$ *for any class* $\mathfrak{A}$.

6. Satisfiability

We present results on the decidability and computational complexity of the satisfiability problem for CTL$^*[\mathfrak{A}]$ and its fragments over the main classes of formal languages of infinite words considered here. We start with a simple and not too surprising undecidability result.

Theorem 13. *The satisfiability problem for* CTL$[\mathfrak{A}]$ *is undecidable for every class* $\mathfrak{A} \supseteq \omega$CFL.

Proof. Remember that the inclusion problem for two context-free languages is undecidable [24, Thm. 8.12]. Let L_1 and L_2 be two context-free languages over an alphabet Σ and let $\# \notin \Sigma$. The CTL[ωCFL]-formula $\mathtt{E}_{L_1\#^\omega}\, \mathtt{G}\, \mathtt{tt} \wedge \mathtt{A}_{L_2\#^\omega}\, \mathtt{G}\, \mathtt{ff}$ is unsatisfiable iff $L_1 \subseteq L_2$. $\square$

Decidability can be achieved when considering smaller classes of languages that possess nice algorithmic and algebraic properties, e.g. ωREG and ωVPL for instance. There is a difference in using CTL on one hand and CTL$^+$ or CTL* on the other. Thus, we examine the extensions of CTL first. The complexity of CTL[ωREG] is easily being characterised.

Theorem 14. *The satisfiability problem for* CTL[ωREG] *is* EXPTIME*-complete.*

Proof. The upper bound is a consequence of the linear translation into the logic ΔPDL$^?$[ωREG] in Theorem 5. It is known that the satisfiability problem of the latter is in EXPTIME [18]. The lower bound trivially follows from the fact that the satisfiability problem for plain CTL is EXPTIME-hard already which is easily shown by adapting the corresponding proof for PDL[REG] [19]. $\square$

Using ωVPL instead of ωREG makes the satisfiability problem exponentially more difficult.

Theorem 15. *The satisfiability problem for* CTL[ωVPL] *is* 2*-*EXPTIME*-complete.*

Proof. The upper bound follows again from the linear translation of CTL[ωVPL] into ΔPDL$^?$[ωVPL] — c.f. Theorem 5 — whose satisfiability problem is in 2-EXPTIME [31].

The lower bound can be proved by a reduction from the tiling game problem for the $2^n \times \mathbb{N}$-corridor which is 2-EXPTIME-hard [38] using the same ideas that led to the 2-EXPTIME-hardness proof of the satisfiability problem for PDL[VPL] [29].

Given a finite set T of tiles with two relations $H, V \subseteq T^2$ that denote horizontal and vertical matchings of adjacent tiles and a designated tile t_0, players 1 and 2 place tiles from T on the aforementioned corridor as follows. Tile t_0 is being placed in the first cell of the first row. Player 2 always has to complete a whole row of 2^n many tiles. Player 1 then plays a tile in the first cell of the next row, etc. A player wins when the opponent cannot place a tile that matches the preceding one in the row regarding H and the tile in the same column of the previous row regarding V. Player 2 also wins every infinite play.

The decision problem at hand is then to decide, given such a tiling system $\mathcal{G}$, whether or not player 2 has a winning strategy. Such a strategy can easily be represented as a tree of nodes labelled with tiles from T such that every node at a height which is a multiple of 2^n has successors for every possible choice of player 1, and every other node has exactly one successor, namely the choice of player 2 in that position. It is then possible to construct a formula $\varphi^n_{\mathcal{G}}$ of size polynomial in $\mathcal{G} = (T, H, V, t_0)$ and n, that is satisfiable iff player 2 has a winning strategy for the game on $2^n \times \mathbb{N}$ as described above.

This formula is obtained as the conjunction of several parts. The first one says that every node is labelled with exactly one tile, and the root of the tree is labelled with t_0. We use T as a set of atomic propositions and the transition label a to mark the part of the tree that represents the strategy. There also are labels b and c such that the a-part forms a complete subtree, and every node in this subtree has additional paths of the form b^*c^ω. However, all nodes on a path of the form b^*c^ω must have a unique tile label. Then it is possible to refer to the label of the first node on such a path by asserting this about the entire subtree on such paths.

$$sane \;:=\; t_0 \wedge \mathtt{A}_{\overline{a^\omega \cup a^* b^* c^\omega}}\, \mathtt{G}\, \mathtt{ff} \wedge \mathtt{A}\,\mathtt{G}\,\Big(\bigvee_{t \in T} t \wedge \bigwedge_{t' \neq t} \neg t'\Big) \wedge \mathtt{A}\,\mathtt{G}\,\Big(\bigwedge_{t \in T} t \rightarrow \mathtt{A}_{b^* c^\omega}\,\mathtt{G}\, t\Big)$$

We furthermore use n propositions $x_0, \ldots, x_{n-1}$ to model a counter which is being used in order to detect every node that marks the start of a new row. Let $null_x := \bigwedge_{i=0}^{n-1} \neg x_i$ and $full_x := \bigwedge_{i=0}^{n-1} x_i$ say that the counter value is 0, respectively, that its maximal value $2^n - 1$.

$$\begin{aligned} count_x \;:=\; null_x \;\wedge\; \mathtt{A}_{a^\omega}\,\mathtt{G}\,\Big((full_x \;\wedge\; \mathtt{A}\,\mathtt{X}_a\, null_x) \vee{} \\ \bigvee_{i=0}^{n-1} \neg x_i \wedge \mathtt{A}\,\mathtt{X}_a\, x_i \wedge \big(\bigwedge_{j<i} x_j \wedge \mathtt{A}\,\mathtt{X}_a\, \neg x_j\big) \;\wedge{} \\ \bigwedge_{j>i} (x_j \rightarrow \mathtt{A}\,\mathtt{X}_a\, x_j) \wedge (\neg x_j \rightarrow \mathtt{A}\,\mathtt{X}_a\, \neg x_j)\Big) \end{aligned}$$

Then we need to say that the horizontal matching relation is satisfied.

$$horiz \;:=\; \mathtt{A}_{a^\omega}\,\mathtt{G}\,\Big(\neg full_x \rightarrow \bigwedge_{t \in T} \big(t \rightarrow \bigwedge_{(t,t') \notin H} \mathtt{A}\,\mathtt{X}_a\, \neg t'\big)\Big)$$

Now all the remains to be said is that every position that is not at the start of a row has a successor in distance 2^n with a vertically matching tile, and for every position at the start of a row there is such a successor at that distance for every matching tile. We require that every node that is reachable along a's only has a b-path of length 2^n, followed by c's only. In order to achieve that length we use propositions $y_0, \ldots, y_{n-1}$ to model a second counter, and formulas $null_y$ and $full_y$ just like their x-counterparts above.

$$
\begin{aligned}
\mathit{count}_y \;:=\;\; & \mathtt{A}_{a^\omega}\,\mathtt{G}\Big(\mathtt{E}\,\mathtt{X}_b\,\mathtt{tt} \;\wedge\; \mathtt{A}\,\mathtt{X}_b\,\mathit{null}_y \;\wedge\; \mathtt{A}_{b^+c^\omega}\,\mathtt{G}\,\big((\mathit{full}_y \wedge \mathtt{A}\,\mathtt{X}_b\,\mathtt{ff} \wedge \mathtt{E}\,\mathtt{X}_c\,\mathtt{tt}) \vee \\
& \qquad \big(\mathtt{A}\,\mathtt{X}_c\,\mathtt{ff} \wedge \mathtt{E}\,\mathtt{X}_b\,\mathtt{tt} \wedge \bigvee_{i=0}^{n-1} \neg y_i \wedge \mathtt{A}\,\mathtt{X}_b\,y_i \wedge (\bigwedge_{j<i} y_j \wedge \mathtt{A}\,\mathtt{X}_b\,\neg y_j) \\
& \qquad \wedge\; \bigwedge_{j>i} (y_j \rightarrow \mathtt{A}\,\mathtt{X}_b\,y_j) \wedge (\neg y_j \rightarrow \mathtt{A}\,\mathtt{X}_b\,\neg y_j))\big)\big)\Big)
\end{aligned}
$$

We can then use the ωVPL $L := \{a^n b^n c^\omega \mid n \geq 1\}$ in order to relate a node in this tree to all nodes at distance 2^n from it. Remember that the b-paths starting in any node reachable via a's have length exactly 2^n.

$$
\begin{aligned}
\mathit{vert} \;:=\;\; & \mathtt{A}_{a^\omega}\,\mathtt{G}\,\Big(\big(\neg \mathit{null}_x \;\wedge\; \bigwedge_{t\in T} t \rightarrow \bigvee_{(t,t')\in V} \mathtt{E}_L\,\mathtt{F}\,(\mathtt{E}\,\mathtt{X}_c\,\mathtt{tt} \wedge t')\big) \vee \\
& \qquad \big(\mathit{null}_x \;\wedge\; \bigwedge_{t\in T} t \rightarrow \bigwedge_{(t,t')\in V} \mathtt{E}_L\,\mathtt{F}\,(\mathtt{E}\,\mathtt{X}_c\,\mathtt{tt} \wedge t')\big)\Big)
\end{aligned}
$$

Finally, let $\varphi^n_{\mathcal{G}} := \mathit{sane} \wedge \mathit{count}_x \wedge \mathit{count}_y \wedge \mathit{horiz} \wedge \mathit{vert}$. We leave it to the reader to verify that a winning strategy for the tiling game $\mathcal{G}$ can be extracted from a model for this formula if player 2 has such a strategy, and that the formula is unsatisfiable otherwise. $\square$

Now we turn to the stronger languages CTL* and CTL$^+$. Upper bounds can be obtained using the linear translation of CTL[$\mathfrak{A}$] and the exponential translation of CTL*[$\mathfrak{A}$] into ΔPDL$^?$[$\mathfrak{A}$], as presented in the previous section.

Theorem 16. *The satisfiability problem for* CTL*[ωREG] *is in* 2-EXPTIME, *and that for* CTL*[ωVPL] *is in* 3-EXPTIME.

Proof. This follows from Theorem 4 since satisfiability of ΔPDL$^?$[ωREG] is in EXPTIME [17] and for ΔPDL$^?$[ωVPL] it is in 2-EXPTIME [29]. $\square$

We remark that it is also possible to obtain the same upper bounds by refining the tableau- and automata-based decision procedure for CTL* which reduces the satisfiability problem to the problem of solving a doubly exponentially large parity game [21] with a single exponential number of priorities. These can be solved in doubly exponential time. The path relativisation can be included as follows. Every E-paths get equipped with an automaton for the respective language. Doing so imposes an additional requirement on the acceptance condition. Similarly, automata are attached to A-paths. Since an A-path may be duplicated among the children in a tableau, the respective automaton must be deterministic. In case of CTL*[ωREG] the refinement stays within the doubly exponential size and single exponential bound on number of priorities, and so does it for CTL*[ωVPL]. However, in this case the resulting game is not a simple parity game anymore but a stair-parity game. Solving these games is exponential in the size of the

	Satisfiability				Model Checking			
$\mathfrak{A}$	—	ωREG	ωVPL	ωCFL	—	ωREG	ωVPL	ωCFL
CTL$^*[\mathfrak{A}]$	2-EXPTIME	2-EXPTIME	3-EXPTIME	undec.	PSPACE	PSPACE	EXPTIME	EXPTIME
CTL$^+[\mathfrak{A}]$	2-EXPTIME	2-EXPTIME	3-EXPTIME	undec.	Δ_2^P	Δ_2^P	Δ_2^P	Δ_2^P
CTL$[\mathfrak{A}]$	EXPTIME	EXPTIME	2-EXPTIME	undec.	P	P	P	P

Figure 5: Complexity results for the satisfiability and the model-checking problem.

game [30] resulting in a 3-EXPTIME upper bound altogether. The advantage of this approach simply is that there is an implementation of the aforementioned CTL* decision procedure [20]. This could in principle be extended to a decision procedure for CTL$^*[\omega$REG$]$ and CTL$^*[\omega$VPL$]$.

Matching lower bounds are not too difficult to achieve, in particular the one for CTL$^*[\omega$REG$]$ which holds for CTL$^+[\omega$REG$]$ already. Regarding the logics over ωVPL, the 2-EXPTIME-hardness proof can easily be amended to a 3-EXPTIME-hardness proof. Instead of requiring b-paths of length $2^{p(n)}$, one requires them to be of length $2^{2^{p(n)}}$. Additionally, one has to mark the beginning of each segment of length $2^{2^{p(n)}}$ instead of just $2^{p(n)}$ in order to prescribe the correct branching structure. While logics like PDL and CTL can only count up to some exponential number, it is known that one can formalise the existence of a path of length $2^{2^{p(n)}}$ in CTL$^+$ already [26].

Theorem 17. *The satisfiability problem for* CTL$^+[\omega$REG$]$ *is* 2-*EXPTIME-hard, and that for* CTL$^+[\omega$VPL$]$ *is* 3-*EXPTIME-hard.*

Proof. The 2-EXPTIME lower bound for CTL$^+$ [25] is trivially inherited by CTL$^+[\omega$REG$]$. The 3-EXPTIME lower bound for CTL$^+[\omega$VPL$]$ can be obtained by a simple adaption of the 2-EXPTIME-hardness result for PDL[VPL] using a reduction from the word problem for alternating, doubly exponential space bounded Turing Machines and the known trick to count up to $2^{2^{p(n)}}$ for some polynomial p in CTL$^+$ [26]. □

Figure 5 summarises the decidability and complexity results presented in this section and compares them to the corresponding results for the pure branching-time logics — i.e. those without path relativisation — in the last row. The stated complexity classes denote completeness under polynomial time reductions.

7. Model Checking

In this section, we investigate the model-checking problem for the path-relativised logics. A summary is given in Figure 5. Although the satisfiability problem for CTL$^\blacklozenge[\omega$CFL$]$ is undecidable, its model-checking problem is solvable efficiently.

Theorem 18. *The model-checking problem for* CTL$[\mathfrak{A}]$ *over finite transition systems is in* P *if* $\mathfrak{A} \subseteq \omega$CFL, *and hard for* P *if* $\Sigma^\omega \in \mathfrak{A}$.

Proof. Given a formula $\varphi \in \mathrm{CTL}[\mathfrak{A}]$ and a transition system $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$, we compute inductively the set of states in $\mathcal{T}$ which satisfy a subformula of φ. Thereto, we extend λ with those formulas. The cases are similar to that of pure CTL. We detail the case of a formula $\mathtt{E}_L(\varphi \,\mathtt{U}\, \psi)$ for $L \in \mathfrak{A}$. For the purpose of a simple presentation assume that L is given as a finite-state Büchi automaton $\mathcal{A} = (Q, q_0, \delta, F)$ where Q is the set of states, $q_0 \in Q$, the transition relation $\delta \in Q \times \Sigma \times Q$, and $F \subseteq Q$ are the final states. We construct for every state $s \in \mathcal{S}$ an automaton $\mathcal{B}_s := (Q \times \mathcal{S} \times \{0,1\}, (q_0, s, 0), \delta', F')$ which recognises witnessing paths for $\mathtt{E}_L(\varphi \,\mathtt{U}\, \psi)$ starting at s. The last component of the state is 1 iff the eventuality is satisfied. So, δ' consists of

$$\begin{array}{lr} ((q,s,0), a, (q', s', 0)) & \text{if } \varphi \in \lambda(s) \\ ((q,s,i), a, (q', s', 1)) & \text{if } \psi \in \lambda(s) \text{ or } i = 1 \end{array}$$

where each line requires $q' \in \delta(q, a)$ and $s \xrightarrow{a} s'$ for some $a \in \Sigma$. Finally, $F' := F \times \mathcal{S} \times \{1\}$. A similar construction is available for ω-PDAs. The emptiness check for this ω-PDA can be done in P [9, Thm. 3.2] — note that the Büchi condition is expressible in LTL. Finally, CTL is hard for P. Hence, so is $\mathrm{CTL}[\mathfrak{A}]$. $\square$

Next we classify the model-checking problem for $\mathrm{CTL}^+[\mathfrak{A}]$. It turns out to have the same complexity as the model-checking problem for ordinary CTL^+, namely it is complete for the class Δ_2^P in the polynomial hierarchy. This is defined as P^{NP} — the class of all problems solvable in polynomial time by a deterministic machine with access to an NP-oracle.

Theorem 19. *The model-checking problem for* $\mathrm{CTL}^+[\mathfrak{A}]$ *over finite transition systems is* Δ_2^P*-complete for* $\mathfrak{A} \in \{\omega\mathrm{REG}, \omega\mathrm{VPL}, \omega\mathrm{CFL}\}$.

Proof. The lower bound is trivially inherited from CTL^+ which is known to have a Δ_2^P-hard model-checking problem [28]. For the corresponding upper bound it suffices to show that $\mathrm{CTL}^+[\omega\mathrm{CFL}\ ]$ belongs to Δ_2^P. For this purpose, the proposed decision procedure recursively computes and memorises for each state subformula its set of satisfying states. For each such subformula φ, the procedure operates in non-deterministic polynomial time and may use the result for the subformulas of φ atomically.

Let $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ be the considered transition system, $s \in \mathcal{S}$ be a state, and $\mathtt{E}_L\psi$ be the quested formula where L is given by a pushdown automaton $\mathcal{A}$. The procedure firstly rewrites the outermost $\mathtt{R}$-connectives in $\mathtt{U}$- and $\mathtt{G}$-connectives. Secondly, it guesses the satisfied side of all those disjunctions which do not occur under a temporal operator. Hence, we may assume that ψ is a conjunction of literals, of $\mathtt{X}$-formulas, of $\mathtt{G}$-formulas, and of $\mathtt{U}$-formulas. Thirdly, the procedure guesses the order in which the $\mathtt{U}$-formulas get fulfilled. Based on this order, the procedure constructs an ω-pushdown automaton $\mathcal{B}$ similar to that in the proof of Theorem 18. The automaton is the product of $\mathcal{A}$ and $\mathcal{T}$ and, in addition, verifies the $\mathtt{X}$- and $\mathtt{G}$-formulas and the fulfilment of the $\mathtt{U}$-formulas in the guessed order. The verification takes advantage of the memorization as the subformulas of the said formulas are state formulas. A run of this automaton represents a witness for the formula $\mathtt{E}_L\psi$. The size of the automaton is polynomial in the size of $\mathcal{A}$, $\mathcal{T}$ and of ψ. Again, the non-emptiness can be checked in polynomial time. Note that if we shifted the non-determinism from the procedure to the automaton $\mathcal{B}$, the size of $\mathcal{B}$ would become exponential. ☐

In contrast to the previous logics, the complexity for model checking CTL$^*[\mathfrak{A}]$ depends on the concrete $\mathfrak{A}$.

Theorem 20. *The model-checking problem for* CTL$^*[\omega$REG$]$ *over finite transition systems is* PSPACE*-complete.*

Proof. The hardness statement follows from Theorem 1 using that the model-checking problem for CTL* is PSPACE-hard [35].

To prove that the model-checking problem is in PSPACE, we use the memorisation technique for state formulas as we did in the proofs of Theorem 18 and 19. Let $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ be the considered transition system, let s be a state in $\mathcal{S}$, $\mathtt{E}_L\varphi$ the considered state formula and let L be given by a finite-state Büchi automaton $\mathcal{A}$. As the state subformulas of φ are considered as atoms, we may assume that λ also lists such subformulas and that φ is a LTL-formula essentially. For such a formula there is [40] a finite-state Büchi automaton $\mathcal{B}$ which recognises exactly all paths fulfilling φ. The size of $\mathcal{B}$ is exponential in $|\varphi|$. To fit the space requirement, the model-checking algorithm does not construct $\mathcal{B}$ explicitly but a short description [40, Thm. 3.6] in space polynomial in $|\varphi|$. That is, every state of $\mathcal{B}$ is representable in space polynomial in $|\varphi|$, and the description contains a polynomial-space decision procedure for every relation in $\mathcal{B}$.

Based on the product of $\mathcal{T}$, $\mathcal{A}$ and $\mathcal{B}$, let $\mathcal{C}$ be a finite-state Büchi automaton which guesses a path in $\mathcal{T}$ successively starting at the state s and which runs $\mathcal{A}$ and $\mathcal{B}$ on this path. Hence, $\mathcal{T}, s \models \mathtt{E}_L\varphi$ iff the language of $\mathcal{C}$ is not empty. The model-checking algorithm computes a short description of $\mathcal{C}$ and runs an emptiness-test on that description. Both parts are performable in space polynomial in $|\mathtt{E}_L\varphi|$ [40, Lem. 2.5]. ☐

Theorem 21. *The model-checking problems for* CTL$^*[\omega$VPL$]$ *and* CTL$^*[\omega$CFL$]$ *over finite transition systems are in* EXPTIME*.*

Proof. The proof follows the lines of the proof of Theorem 20. But this time, $\mathcal{A}$ is an ω-PDA and the algorithm can construct the automaton $\mathcal{B}$ explicitly as we have enough space. Therefore, $\mathcal{C}$ is also an ω-PDA of exponential size. The emptiness test can be performed in time polynomial in $|\mathcal{C}|$, cf. the proof of Theorem 18. All in all, the algorithm requires exponential time. ☐

Theorem 22. *The model-checking problems for* CTL$^*[\omega$VPL$]$ *and* CTL$^*[\omega$CFL$]$ *over finite transition systems are hard for* EXPTIME*.*

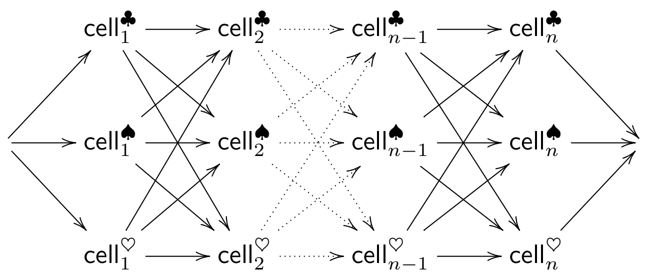
Proof. Similar to the proof of Theorem 15, a tiling game but for the $n \times \mathbb{N}$-corridor is used. The decision problem whether player 2 has a winning strategy is EXPTIME-hard [38].

The reduction to CTL*[ωVPL] produces for every tiling problem $\mathcal{G} = (T, H, V, t_0)$ and binary parameter $n \in \mathbb{N}$ a transition system $\mathcal{T}$, a state s and a formula $\mathtt{E}_{L(\mathcal{A})}\varphi$ — their components are sketched later — such that $\mathcal{T}, s \models \mathtt{E}_{L(\mathcal{A})}\varphi$ iff player 2 has a winning strategy. A witness of the formula represents a depth-first traversal of the tree representing the strategy. However, as this kind of serialisation cannot handle infinite branches we may assume that $\mathcal{G}$ does not admit any infinite run. (Technically, an encoding of an n-bit counter into the tiling ensures this requirement.) The H-constraint can be checked by $\mathcal{A}$ because two horizontally adjacent tiles occur as neighbouring labels in the witness of $\mathtt{E}_{L(\mathcal{A})}\varphi$. The stack of the ω-pushdown automaton $\mathcal{A}$ is used for bookkeeping the traversal.

The only property which $\mathcal{A}$ cannot handle is the V-constraint because the constraint affects positions far apart. However, this job can be assigned to φ. A row of tiles $(t_1, \ldots, t_n)$ is encoded by unfulfilled eventualities. These eventualities can be created by the formula $\bigwedge_{i=1}^{n} \gamma_i^{t_i}$ where

$$\gamma_i^t := \bigwedge_{s \in T, s \neq t} (\neg\mathsf{cell}_i^s)\, \mathtt{U}\, \mathsf{cell}_i^t$$

and where cell_i^t is an atomic proposition stating that the ith column is tiled with t. A basic building block for transition system $\mathcal{T}$ addresses the choice of a next row. For $T = \{\clubsuit, \spadesuit, \heartsuit\}$ this block looks as follows. The labels for the edges are omitted.


$\mathsf{cell}_1^{\clubsuit}$
$\mathsf{cell}_2^{\clubsuit}$
$\mathsf{cell}_{n-1}^{\clubsuit}$
$\mathsf{cell}_n^{\clubsuit}$
$\mathsf{cell}_1^{\spadesuit}$
$\mathsf{cell}_2^{\spadesuit}$
$\mathsf{cell}_{n-1}^{\spadesuit}$
$\mathsf{cell}_n^{\spadesuit}$
$\mathsf{cell}_1^{\heartsuit}$
$\mathsf{cell}_2^{\heartsuit}$
$\mathsf{cell}_{n-1}^{\heartsuit}$
$\mathsf{cell}_n^{\heartsuit}$

Assume that the witness for $\mathtt{E}_{L(\mathcal{A})}\varphi$ arrives from the left, and that φ has ensured open eventualities $\gamma_i^{t_i}$ for $i = 1, \ldots, n$. The witness must, thus, choose the trace which corresponds to the unfulfilled eventualities. If the formula

$$\mathit{nextRow} := \bigwedge_{i=1}^{n} \underbrace{\mathtt{X} \ldots \mathtt{X}}_{i \text{ times}} \left(\bigwedge_{t_1 \in T} \left(\mathsf{cell}_i^{t_1} \rightarrow \bigvee_{\substack{t_2 \in T \\ (t_1, t_2) \in V}} \mathtt{X}\gamma_i^{t_2} \right) \right)$$

is forced at the leftmost node, *nextRow* ensures the creation of a new row which matches the vertical constraint.

In order to link the automaton to φ, every edge to a state in the final transition system will be labelled with the name of that state. In this manner, $\mathcal{A}$

```
 1  guess and push an initial tiling row $R$;
 2  push the minimal tile which player 1 can attach to $R$,
 3    and enter an accepting loop if impossible;
 4  while stack is not empty do
 5      choose either
 6          pop a tile $t$ and a tiling row $R$;
 7          let $t$' be the minimal tile $> t$
 8            which player 1 can attach to $R$
 9            and reject otherwise;
10          push tiling row $R$ and tile $t$';
11      or
12          pop a tile $t$ and a tiling row $R$;
13          reject if there is title $> t$
14            which player 1 can attach to $R$;
15      end-choose;
16      guess a tiling row $S$ starting with $t$ and push it;
17      push the minimal tile which player 1 can attach to $S$,
18        and enter an accepting loop if impossible;
19  end-while;
20  enter an accepting loop;
```

Figure 6: Algorithm for the ω-visibly pushdown automaton $\mathcal{A}$ as used in Theorem 22.

is notified about the witnessing path and can force decision by rejecting wrong ways. Thus we have established a bidirectional communication.

The ω-visibly pushdown automaton $\mathcal{A}$ implements the non-deterministic algorithm shown in Figure 6. Its stack contains a sequence of pairs each consisting of a tiling row R and a single tile t. The tile t is the next possible move of player 1 on the row which follows R. To enumerate all her possible moves, we assume that there is total order on the tiles. Every guess of a row shall fulfil the H-constraint. In line 1, $\mathcal{A}$ guesses an initial row — the choice of player 1 is fixed to t_0 in this case — and the first possible move of player 1 for the next row. For this purpose, $\mathcal{A}$ only need to keep the first column of R in memory. All push- and pop-operations of tiling rows are sent to φ by reading a piece of the input.

As long as the strategy tree is not entirely traversed, the algorithm considers the next possible move t of player 1 on a row R. In line 5 the algorithm guesses whether or not there is a further move t' of player 1 on R. The guess is verified in line 7 and 13, respectively. The row R is sent to φ to create the open eventualities. This anticipatory test is needed because these eventualities are used in different ways in each branch. In both cases, the eventualities are used to correctly guess the next row in line 16. But if player 1 has a further choice to play on R, the choice must to be stored on the stack. For this purpose, the transition system has a copy of the show transition system such that

$$\mathit{revealRow} := \bigwedge_{i=1}^{n} \underbrace{\mathtt{X}\ldots\mathtt{X}}_{i \text{ times}} \left(\bigwedge_{t_1 \in T} \left(\mathsf{cell}_i^{t_1} \rightarrow \mathtt{X}\gamma_i^{t_1} \right) \right)$$

can reveal the unfulfilled eventualities and send the tiles successively to $\mathcal{A}$ — which in turn pushes the row on the stack.

The guess of the next row happens as described at the beginning. If the whole tree is traversed the automaton forces the witness to go to a sink state and stay there. Hence, $\mathcal{A}$ requires a coBüchi acceptance condition only. The alphabet can be chosen as a pushdown alphabet.

All in all, the final transition system $\mathcal{T}$ is mainly the flow chart of the algorithm in Figure 6 but the nodes for the lines 6, 10, 12 and 16 are replaced by copies of the shown transition system. The final formula is

$$\mathtt{E}_{L(\mathcal{A})}\,\mathtt{G}\Big(\quad (\mathit{branch1} \to (\mathit{revealRow} \wedge \underbrace{\mathtt{X}\ldots\mathtt{X}}_{n \text{ times}}\,\mathit{nextRow})) \\ \wedge (\mathit{branch2} \to \mathit{nextRow})\Big)$$

where *branch1* and *branch2* are propositions indicating the branch chosen in line 5. □

Corollary 23. *The model-checking problems for* CTL*[ωVPL] *and* CTL*[ωCFL] *over finite transition systems are* EXPTIME*-complete.*

Proof. By Theorems 21 and 22. □

8. Conclusion and Further Work

We have considered extensions of the well-known full branching-time temporal logic CTL* using path relativisation with formal languages of ω-words, investigated expressivity and complexity issues and shown a possible area of application for such logics.

While it clearly still remains to be seen whether or not these logics can prove to be useful in the domain of abstract interpretation as well as elsewhere, there are also some open questions regarding the theory behind this family of logics. In particular, it is currently unknown whether or not CTL$^+$[$\mathfrak{A}$] equals CTL[$\mathfrak{A}$] in expressive power. We suspect that this is not the case for non-trivial classes $\mathfrak{A}$.

Another open problem is the suspected separation between CTL*[$\mathfrak{A}$] and ΔPDL$^?$[$\mathfrak{A}$]. It seems like the latter should have higher expressive power by being able to synchronise — via the test operator — between labels along a path and formulas that hold on this path. Note that in CTL*[$\mathfrak{A}$] these are two entirely separate issues: the path relativiser selects the path according to its edges, the temporal part of the formula is satisfied (or not) on this path because of its states.

It is of course also possible to compare path relativised branching-time logics more thoroughly with other existing formalisms, for instance the logics over ωREG with existing regular extensions of CTL or extensions with fairness predicates.

References

[1] M. Adler and N. Immerman. An $n!$ lower bound on formula size. In *Proc. 16th Symp. on Logic in Computer Science, LICS'01*, pages 197–208, Boston, MA, USA, 2001. IEEE.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[3] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Ann. ACM Symp. on Theory of Computing, STOC'04*, pages 202–211, 2004.

[4] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[5] R. Axelsson, M. Hague, S. Kreutzer, M. Lange, and M. Latte. Extended computation tree logic. In *Proc. 17th Int. Conf. on Logic for Programming and Artificial Reasoning, LPAR'07*, volume 6397 of *LNCS*, pages 67–81. Springer, 2010.

[6] I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In *Proc. 10th Int. Conf. on Computer Aided Verification, CAV'98*, volume 1427 of *LNCS*, pages 184–194. Springer, 1998.

[7] A. Bianco, F. Mogavero, and A. Murano. Graded computation tree logic. In *Proc. 24th Symp. on Logic in Computer Science, LICS'09*, pages 342–351. IEEE Computer Society, 2009.

[8] D. Bosnacki, N. Ioustinova, and N. Sidorova. Using fairness to make abstractions work. In *Proc. 11th Int. Workshop on Model Checking Software, SPIN'04*, volume 2989 of *LNCS*, pages 198–215. Springer, 2004.

[9] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. 8th Int. Conf. on Concurrency Theory, CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

[10] T. Brázdil and I. Cerná. Model checking of regCTL. *Computers and Artificial Intelligence*, 25(1), 2006.

[11] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, CA, USA, 1962. Stanford University Press.

[12] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.

[13] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[14] S. Danecki. Nondeterministic propositional dynamic logic with intersection is decidable. In A. Skowron, editor, *Proc. 5th Symp. on Computation Theory*, volume 208 of *LNCS*, pages 34–53, Zaborów, Poland, 1984. Springer.

[15] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

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

[17] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 2000.

[18] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *Foundations of Computer Science, Annual IEEE Symposium on*, pages 328–337, 1988.

[19] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[20] O. Friedmann and M. Lange. A solver for modal fixpoint logics. In *Proc. 6th Workshop on Methods for Modalities, M4M-6*, volume 262 of *Electronic Notes in Theoretical Computer Science*, pages 99–111, 2010.

[21] O. Friedmann, M. Lange, and M. Latte. A decision procedure for CTL* based on tableaux and automata. In *Proc. Int. Joint Conf. on Automated Reasoning, IJCAR'10*, volume 6173 of *LNCS*, pages 331–345. Springer, 2010.

[22] O. Grumberg and H. Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCS*. Springer, 2008.

[23] D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[24] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[25] J. Johannsen and M. Lange. CTL$^+$ is complete for double exponential time. In *Proc. 30th Int. Coll. on Automata, Logics and Programming, ICALP'03*, volume 2719 of *LNCS*, pages 767 – 775. Springer, 2003.

[26] M. Lange. A purely model-theoretic proof of the exponential succinctness gap between CTL$^+$ and CTL. *Information Processing Letters*, 108:308–312, 2008.

[27] M. Lange and M. Latte. A CTL-based logic for program abstractions. In *Proc. 17th Workshop on Logic, Language, Information and Computation, WoLLIC'10*, volume 6188 of *LNCS*, pages 19–33. Springer, 2010.

[28] F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking CTL$^+$ and FCTL is hard. In *Proc. 4th Conf. on Foundations of Software Science and Computation Structures, FOSSACS'01*, volume 2030 of *LNCS*, pages 318–331. Springer, 2001.

[29] C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *Journal of Logic and Algebraic Programming*, 73(1-2):51–69, 2007.

[30] C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. 24th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'04*, volume 3328 of *LNCS*, pages 408–420. Springer, 2004.

[31] C. Löding and O. Serre. Propositional dynamic logic with recursive programs. In *Proc. 9th Int. Conf. on Foundations of Software Science and Computation Structures, FOSSACS'06*, volume 3921 of *LNCS*, pages 292–306. Springer, 2006.

[32] R. Mateescu, P. T. Monteiro, E. Dumas, and H. de Jong. Computation tree regular logic for genetic regulatory networks. In *Proc. 6th Int. Conf. on Automated Technology for Verification and Analysis, ATVA'08*, volume 5311 of *LNCS*, pages 48–63. Springer, 2008.

[33] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes*, pages 407–419, Berlin, Heidelberg, New York, 1990. Springer.

[34] A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. on Foundations of Computer Science, FOCS'77*, pages 46–57, Providence, RI, USA, 1977. IEEE.

[35] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.

[36] L. Staiger. *Handbook of formal languages, vol. 3: beyond words*, chapter ω-Languages, pages 339–387. Springer, 1997.

[37] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982.

[38] P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker, Inc., 1997.

[39] M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing, STOC'85*, pages 240–251, Baltimore, USA, 1985. ACM.

[40] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[41] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

[42] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'99*, volume 1738 of *LNCS*, pages 110–121. Springer, 1999.