Extended Computation Tree Logic

Roland Axelsson[1], Matthew Hague[2], Stephan Kreutzer[2], Martin Lange[3], and Markus Latte[1]

[1] Department of Computer Science, Ludwig-Maximilians-Universität Munich,
Email: {roland.axelsson,markus.latte}@ifi.lmu.de
[2] Oxford University Computing Laboratory,
Email: {Matthew.Hague,stephan.kreutzer}@comlab.ox.ac.uk
[3] Department of Elect. Engineering and Computer Science, University of Kassel, Germany,
Email: martin.lange@uni-kassel.de

Abstract. We introduce a generic extension of the popular branching-time logic CTL which refines the temporal until and release operators with formal languages. For instance, a language may determine the moments along a path that an until property may be fulfilled. We consider several classes of languages leading to logics with different expressive power and complexity, whose importance is motivated by their use in model checking, synthesis, abstract interpretation, etc. We show that even with context-free languages on the until operator the logic still allows for polynomial time model-checking despite the significant increase in expressive power. This makes the logic a promising candidate for applications in verification. In addition, we analyse the complexity of satisfiability and compare the expressive power of these logics to CTL* and extensions of PDL.

1 Introduction

Computation Tree Logic (CTL) is one of the main logical formalisms for program specification and verification. It appeals because of its intuitive syntax and its very reasonable complexities: model checking is PTIME-complete [9] and satisfiability checking is EXPTIME-complete [12]. However, its expressive power is low.

CTL can be embedded into richer formalisms like CTL* [13] or the modal μ-calculus $\mathcal{L}_\mu$ [23]. This transition comes at a price. For CTL* the model checking problem increases to PSPACE-complete [32] and satisfiability to 2EXPTIME-complete [14, 35]. Furthermore, CTL* cannot express regular properties like “something holds after an even number of steps”. The modal μ-calculus is capable of doing so, and its complexities compare reasonably to CTL: satisfiability is also EXPTIME-complete, and model checking sits between PTIME and NP$\cap$coNP. However, it is much worse from a pragmatic perspective since its syntax is notoriously unintuitive.

Common to all these (and many other) formalisms is a restriction of their expressive power to at most regular properties. This follows since they can be embedded into (the bisimulation-invariant) fragment of monadic second-order logic on graphs. This restriction yields some nice properties — like the finite model property and decidability — but implies that these logics cannot be used for certain specification purposes.

For example, specifying the correctness of a communication protocol that uses a buffer requires a non-underflow property: an item cannot be removed when the buffer is empty. The specification language must therefore be able to track the buffer's size. If the buffer is unbounded, as is usual in software, this property is non-regular and a regular logic is unsuitable. If the buffer is bounded, the property is regular but depends on the actual buffer capacity, requiring a different formula for each size. This is unnatural for verification purposes. The formulas are also likely to be complex as they essentially have to hard-code numbers up to the buffer length. To express such properties naturally one has to step beyond regularity and consider logics of corresponding expressive power.

Also, consider program synthesis where, instead of verifying a program, one wants to automatically generate a correct program (skeleton) from the specification. This problem is very much linked to satisfiability checking, except, if a model exists, one is created and transformed into a program. This is known as controller synthesis and has been done mainly based on satisfiability checking for the modal μ-calculus [4]. The finite model property restricts the synthesization to finite state programs, i.e. hardware and controllers, etc. In order to automatically synthesize software (e.g. recursive functions) one has to consider non-regular logics.

Finally, consider the problem of verifying programs with infinite or very large state spaces. A standard technique is to abstract the large state space into a smaller one [10]. This usually results in spurious traces which then have to be excluded in universal path quantification on the small system. If the original system was infinite then the language of spurious traces is typically non-regular and, again, a logic of suitable expressive power is needed to increase precision [26].

In this paper we introduce a generic extension of CTL which provides a specification formalism for such purposes. We refine the usual until operator (and its dual, the release operator) with a formal language defining the moments at which the until property can be fulfilled. This leads to a family of logics parametrised by a class of formal languages. CTL is an ideal base logic because of its wide-spread use in actual verification applications. Since automata easily allow for an unambiguous measure of input size, we present the precise definition of our logics in terms of classes of automata instead of formal languages. However, we do not promote the use of automata in temporal formulas. For pragmatic considerations it may be sensible to allow more intuitive descriptions of formal languages such as Backus-Naur-Form or regular expressions.

As a main result we extend CTL using context-free languages, significantly increasing expressive power, while retaining polynomial time model-checking. Hence, we obtain a good balance between expressiveness — as non-regular properties become expressible — and low model-checking complexity, which makes this logic very promising for applications in verification. We also study model-checking for the new logics against infinite state systems represented by (visibly) pushdown automata, as they arise in software model-checking, and obtain tractability results for these. For satisfiability testing, equipping the path quantifiers with visibly pushdown languages retains decidability. However, the complexity increases from EXPTIME for CTL to 3EXPTIME for this new logic.

The paper is organised as follows. We formally introduce the logics and give an example demonstrating their expressive power in Section 2. Section 3 discusses related formalisms. Section 4 presents results on the expressive power of these logics, and

Section 5 and 6 contain results on the complexities of satisfiability and model checking. Finally, Section 7 concludes with remarks on further work. Due to space restrictions this paper contains no detailed proofs in its main part. A full version with all proof details is available online at `http://arxiv.org/abs/1006.3709`.

2 Extended Computation Tree Logic

Let $\mathcal{P} = \{p, q, \ldots\}$ be a countably infinite set of *propositions* and Σ be a finite set of *action names*. A *labeled transition system* (LTS) is a $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$, where $\mathcal{S}$ is a set of states, $\rightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ and $\ell : \mathcal{S} \rightarrow 2^{\mathcal{P}}$. We usually write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \rightarrow$. A *path* is a maximal sequence of alternating states and actions $\pi = s_0, a_1, s_1, a_2, s_2, \ldots$, s.t. $s_i \xrightarrow{a_{i+1}} s_{i+1}$ for all $i \in \mathbb{N}$. We also write a path as $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$. Maximality means that the path is either infinite or it ends in a state s_n s.t. there are no $a \in \Sigma$ and $t \in \mathcal{S}$ with $s_n \xrightarrow{a} t$. In the latter case, the domain $dom(\pi)$ of π is $\{0, \ldots, n\}$. And otherwise $dom(\pi)$:=$\mathbb{N}$.

We focus on automata classes between deterministic finite automata (DFA) and nondeterministic pushdown automata (PDA), with the classes of nondeterministic finite automata (NFA), (non-)deterministic visibly pushdown automata (DVPA/VPA) [2] and deterministic pushdown automata (DPDA) in between. Beyond PDA one is often faced with undecidability. Note that some of these automata classes define the same class of languages. However, translations from nondeterministic to deterministic automata usually involve an exponential blow-up. For complexity estimations it is therefore advisable to consider such classes separately.

We call a class $\mathfrak{A}$ of automata *reasonable* if it contains automata recognising Σ and Σ^* and is closed under equivalences, i.e. if $\mathcal{A} \in \mathfrak{A}$ and $L(\mathcal{A}) = L(\mathcal{B})$ and $\mathcal{B}$ is of the same type then $\mathcal{B} \in \mathfrak{A}$. $L(\mathcal{A})$ denotes the language accepted by $\mathcal{A}$.

Let $\mathfrak{A}, \mathfrak{B}$ be two reasonable classes of finite-word automata over the alphabet Σ. Formulas of *Extended Computation Tree Logic over* $\mathfrak{A}$ *and* $\mathfrak{B}$ (CTL[$\mathfrak{A}$,$\mathfrak{B}$]) are given by the following grammar, where $\mathcal{A} \in \mathfrak{A}$, $\mathcal{B} \in \mathfrak{B}$ and $q \in \mathcal{P}$.

$$\varphi \quad ::= \quad q \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathtt{E}(\varphi \mathtt{U}^{\mathcal{A}} \varphi) \mid \mathtt{E}(\varphi \mathtt{R}^{\mathcal{B}} \varphi)$$

Formulas are interpreted over states of a transition system $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$ in the following way.

- $\mathcal{T}, s \models q$ iff $q \in \ell(s)$
- $\mathcal{T}, s \models \varphi \vee \psi$ iff $\mathcal{T}, s \models \varphi$ or $\mathcal{T}, s \models \psi$
- $\mathcal{T}, s \models \neg\varphi$ iff $\mathcal{T}, s \not\models \varphi$
- $\mathcal{T}, s \models \mathtt{E}(\varphi \mathtt{U}^{\mathcal{A}} \psi)$ iff there exists a path $\pi = s_0, a_1, s_1, \ldots$ with $s_0 = s$ and $\exists n \in dom(\pi)$ s.t. $a_1 \ldots a_n \in L(\mathcal{A})$ and $\mathcal{T}, s_n \models \psi$ and $\forall i < n : \mathcal{T}, s_i \models \varphi$.
- $\mathcal{T}, s \models \mathtt{E}(\varphi \mathtt{R}^{\mathcal{A}} \psi)$ iff there exists a path $\pi = s_0, a_1, s_1, \ldots$ with $s_0 = s$ and for all $n \in dom(\pi)$: $a_1 \ldots a_n \notin L(\mathcal{A})$ or $\mathcal{T}, s_n \models \psi$ or $\exists i < n$ s.th. $\mathcal{T}, s_i \models \varphi$.

As usual, further syntactical constructs, like other boolean operators, are introduced as abbreviations. We define $\mathtt{A}(\varphi \mathtt{U}^{\mathcal{A}} \psi) := \neg\mathtt{E}(\neg\varphi \mathtt{R}^{\mathcal{A}} \neg\psi)$, $\mathtt{A}(\varphi \mathtt{R}^{\mathcal{A}} \psi) := \neg\mathtt{E}(\neg\varphi \mathtt{U}^{\mathcal{A}} \neg\psi)$, as well as $Q\mathtt{F}^{\mathcal{A}}\varphi := Q(\mathtt{tt}\mathtt{U}^{\mathcal{A}}\varphi)$, $Q\mathtt{G}^{\mathcal{A}}\varphi := Q(\mathtt{ff}\mathtt{R}^{\mathcal{A}}\varphi)$ for $Q \in \{\mathtt{E}, \mathtt{A}\}$. For presentation, we also use languages L instead of automata in the temporal operators. For instance, $\mathtt{EG}^L\varphi$ is $\mathtt{EG}^{\mathcal{A}}\varphi$ for some $\mathcal{A}$ with $L(\mathcal{A}) = L$. This also allows us to easily define the original CTL operators: $Q\mathtt{X}\varphi := Q\mathtt{F}^{\Sigma}\varphi$, $Q(\varphi\mathtt{U}\psi) := Q(\varphi\mathtt{U}^{\Sigma^*}\psi)$, $Q(\varphi\mathtt{R}\psi) := Q(\varphi\mathtt{R}^{\Sigma^*}\psi)$, etc. The size of a formula φ is the number of its unique subformulas plus the sum of the sizes of all automata in φ, with the usual measure of size of an automaton.

The distinction between $\mathfrak{A}$ and $\mathfrak{B}$ is motivated by the complexity analysis. For instance, when model checking $\mathtt{E}(\varphi\mathtt{U}^{\mathcal{A}}\psi)$ the existential quantifications over system paths and runs of $\mathcal{A}$ commute and we can guess a path and an accepting run in a step-wise fashion. On the other hand, when checking $\mathtt{E}(\varphi\mathtt{R}^{\mathcal{A}}\psi)$ the existential quantification on paths and universal quantification on runs (by $\mathtt{R}$ — "on all prefixes …") does not commute unless we determinise $\mathcal{A}$, which is not always possible or may lead to exponential costs.

However, $\mathfrak{A}$ and $\mathfrak{B}$ can also be the same and in this case we denote the logic by CTL[$\mathfrak{A}$]. Equally, by EF[$\mathfrak{A}$], resp. EG[$\mathfrak{B}$] we denote the fragments of CTL[$\mathfrak{A}$,$\mathfrak{B}$] built from atomic propositions, boolean operators and the temporal operators $\mathtt{EF}^{\mathcal{A}}\varphi$, resp. $\mathtt{EG}^{\mathcal{B}}\varphi$ only. Since the expressive power of the logic only depends on its class of *languages* rather than *automata*, we will write CTL[REG], CTL[VPL], CTL[CFL], etc. to denote the logic over regular, visibly pushdown, and context-free languages, represented by any type of automaton. We close this section with a CTL[VPL] example which demonstrates the buffer-underflow property discussed in the introduction.

Example. Consider a concurrent producer/consumer scenario over a shared buffer. If the buffer is empty, the consumer process requests a new resource and halts until the producer delivers a new one. Any parallel execution of these processes should obey a non-underflow property (NBU): at any moment, the number of produce actions is sufficient for the number of consumes.

If the buffer is realised in software it is reasonable to assume that it is unbounded, and thus, the NBU property becomes non-regular. Let $\Sigma = \{p, c, r\}$, where p stands for *production* of a buffer object, c for *consume* and r for *request*. Consider the VPL $L = \{w \in \Sigma^* \mid \ |w|_c = |w|_p$ and $|v|_c \leq |v|_p$ for all $v \preceq w\}$, where $\preceq$ denotes the prefix relation. We express the requirements in CTL[VPL].

1. $\mathtt{AGEX}^p\mathtt{tt}$: "at any time it is possible to produce an object"
2. $\mathtt{AG}^L(\mathtt{AX}^c\mathtt{ff} \wedge \mathtt{EX}^r\mathtt{tt})$: "whenever the buffer is empty, it is impossible to consume and possible to request"
3. $\mathtt{AG}^{\overline{L}}(\mathtt{EX}^c\mathtt{tt} \wedge \mathtt{AX}^r\mathtt{ff})$: "whenever the buffer is non-empty it is possible to consume and impossible to request"
4. $\mathtt{EFEG}^{\overline{c^*}}\mathtt{ff}$: "at some point there is a consume-only path"

Combining the first three properties yields a specification of the scenario described above and states that a *request* can only be made if the buffer is empty. For the third properly, recall that VPL are closed under complement [2]. Every satisfying model gives a raw implementation of the main characteristics of the system. Note that if it is always possible to *produce* and possible to *consume* iff the buffer is not empty, then a straight-forward model with self-loops p, c and r does not satisfy the specification. Instead, we require a model with infinitely many different p transitions. If we strengthen the specification by adding the fourth formula, it becomes unsatisfiable.

3 Related Formalisms

Several suggestions to integrate formal languages into temporal logics have been made so far. The goal is usually to extend the expressive power of a logic whilst retaining its intuitive syntax. The most classic example is Propositional Dynamic Logic (PDL) [17] which extends Modal Logic with regular expressions.

Similar extensions — sometimes using finite automata instead of regular expressions — of Temporal Logics have been investigated a long time ago. The main purpose has usually been the aim to increase the expressive power of seemingly weak specification formalisms in order to obtain at least ω-regular expressivity, but no efforts have been made at that point in order to go beyond that. This also explains why such extensions were mainly based on LTL [39, 36, 24, 20], i.e. not leaving the world of linear-time formalisms.

The need for extensions beyond the use of pure temporal operators is also witnessed by the industry-standard *Property Specification Language* (PSL) [1] and its predecessor ForSpec [3]. However, ForSpec is a linear-time formalism and here we are concerned with branching-time. PSL does contain branching-time operators but they have been introduced for backwards-compatibility only.

On the other hand, some effort has been made with regards to extensions of branching-time logics like CTL [5, 7, 29]. These all refine the temporal operators of this logic with regular languages in some form.

Thus, while much effort has been put into regular extensions of standard temporal logics, little is known about extensions using richer classes of formal languages. We are only aware of extensions of PDL by context-free languages [19] or visibly pushdown languages [27]. The main yardstick for measuring the expressive power of CTL[$\mathfrak{A}$,$\mathfrak{B}$] will be therefore be PDL and one of its variants, namely PDL with the Δ-construct and tests, ΔPDL$^?[\mathfrak{A}]$, [17, 33]. Note: for a class $\mathfrak{A}$ of automata, CTL[$\mathfrak{A}$] is a logic using such automata on finite words only, whereas ΔPDL$^?[\mathfrak{A}]$ uses those and their Büchi-variants on infinite words. In the following we will use some of the known results about ΔPDL$^?[\mathfrak{A}]$. For a detailed technical definition of its syntax and semantics, we refer to the literature on this logic [18].

There are also temporal logics which obtain higher expressive power through other means. These are usually extensions of $\mathcal{L}_\mu$ like the Modal Iteration Calculus [11] which uses inflationary fixpoint constructs or Higher-Order Fixpoint Logic [37] which uses higher-order predicate transformers. While most regular extensions of standard temporal logics like CTL and LTL can easily be embedded into $\mathcal{L}_\mu$, little is known about the relationship between richer extensions of these logics.

4 Expressivity and Model Theory

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. We use $\mathcal{L} \lneq_f \mathcal{L}'$ to denote that such a translation exists, but there are formulas of $\mathcal{L}'$ which are not equivalent to any formula in $\mathcal{L}$. Also, we write $\mathcal{L} \equiv_f \mathcal{L}'$ if $\mathcal{L} \leq_f \mathcal{L}'$ and $\mathcal{L}' \leq_f \mathcal{L}$. We will drop the index if a potential blow-up is of no concern.

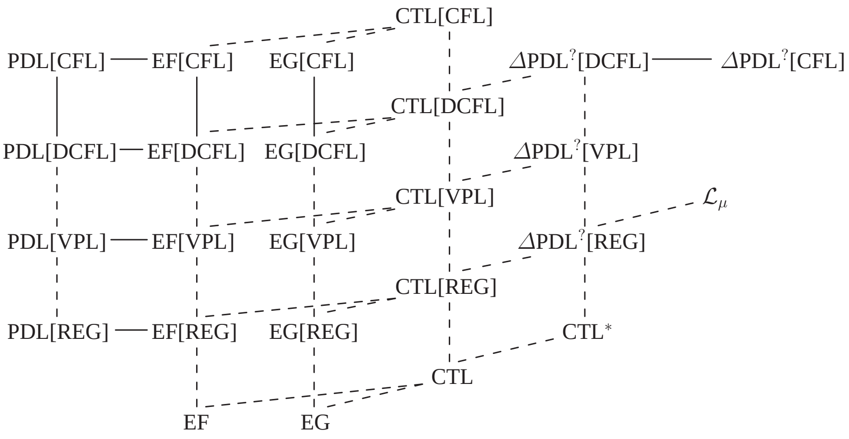


Fig. 1. The expressive power of Extended Computation Tree Logic.

A detailed picture of the expressivity results regarding the most important CTL[$\mathfrak{A}$] logics is given in Fig. 1. A (dashed) line moving upwards indicates (strict) inclusion w.r.t. expressive power. A horizontal continuous line states expressive equivalence. The following proposition collects some simple observations.

Proposition 4.1. *1. For all* $\mathfrak{A}, \mathfrak{B}$*:* CTL $\lneq_{\mathsf{lin}}$ CTL[$\mathfrak{A}$,$\mathfrak{B}$].
2. *For all* $\mathfrak{A}, \mathfrak{A}', \mathfrak{B}, \mathfrak{B}'$*: if* $\mathfrak{A} \leq \mathfrak{A}'$ *and* $\mathfrak{B} \leq \mathfrak{B}'$ *then* CTL[$\mathfrak{A}$,$\mathfrak{B}$] $\leq$ CTL[$\mathfrak{A}$',$\mathfrak{B}$'].

CTL[$\mathfrak{A}$] extends PDL[$\mathfrak{A}$] since the latter is just a syntactic variation of the EF[$\mathfrak{A}$] fragment. On the other hand, CTL[$\mathfrak{A}$] can — in certain cases — be embedded into PDL[$\mathfrak{A}$]'s extension ΔPDL$^?$[$\mathfrak{A}$]. This, however, requires a transformation from automata on finite words to automata on infinite words which shows that these two formalisms are conceptually different.

Theorem 4.2. *1. For all* $\mathfrak{A}$*:* PDL[$\mathfrak{A}$] $\equiv_{\mathsf{lin}}$ EF[$\mathfrak{A}$].
2. *For all* $\mathfrak{A}, \mathfrak{B}$*:* EF[$\mathfrak{A}$] $\lneq_{\mathsf{lin}}$ CTL[$\mathfrak{A}$,$\mathfrak{B}$].
3. *For all* $\mathfrak{A}, \mathfrak{B}$*:* CTL[$\mathfrak{A}$,$\mathfrak{B}$] $\leq_{\mathsf{lin}}$ ΔPDL$^?$[$\mathfrak{A} \cup \mathfrak{B}$], *if* $\mathfrak{B}$ *is a class of deterministic automata.*
4. ΔPDL$^?$[PDA] $\equiv_{\mathsf{lin}}$ ΔPDL$^?$[DPDA].

Note that CFL does not admit deterministic automata. Hence, part 3 is not applicable in that case. If for some classes $\mathfrak{A}$,$\mathfrak{B}$ the inclusion in part 3 holds, then it must be strict. This is because fairness is not expressible in CTL[$\mathfrak{A}$] regardless of what $\mathfrak{A}$ is, as demonstrated by the following.

Theorem 4.3. *The* CTL**-formula* $\texttt{EGF}q$ *expressing fairness is not equivalent to any* CTL[$\mathfrak{A}$, $\mathfrak{B}$] *formula, for any* $\mathfrak{A}$, $\mathfrak{B}$.

Fairness can be expressed by $\Delta\mathcal{A}_{\mathsf{fair}}$, where $\mathcal{A}_{\mathsf{fair}}$ is the standard Büchi automaton over some alphabet containing a test predicate $q?$ that recognises the language of all infinite paths on which infinitely many states satisfy q.

Corollary 4.4. *1. For all* $\mathfrak{A}, \mathfrak{B}$*:* CTL* $\not\leq$ CTL[$\mathfrak{A}$,$\mathfrak{B}$].

2. *There are no* $\mathfrak{A}$,$\mathfrak{B}$ *such that any* CTL[$\mathfrak{A}$,$\mathfrak{B}$] *is equivalent to the* ΔPDL$^?$[REG] *formula* $\Delta\mathcal{A}_{\mathsf{fair}}$.

At least in the case of CFLs, the premise to part 3 of Thm. 4.2 cannot be dropped. Indeed, the formula $\mathtt{EG}^L p$ is not expressible as a ΔPDL$^?$[CFL]-formula where L is the language of palindromes.

Theorem 4.5. CTL[CFL] $\not\leq$ ΔPDL$^?$[CFL].

Finally, we provide some model-theoretic results which will also allow us to separate some of the logics with respect to expressive power. Not surprisingly, CTL[REG] has the finite model property which is a consequence of its embedding into the logic ΔPDL$^?$[REG]. It is not hard to bound the size of such a model given that ΔPDL$^?$[REG] has the small model property of exponential size.

Proposition 4.6. *Every satisfiable* CTL[REG] *formula has a finite model. In fact, every satisfiable* CTL[NFA,DFA]*, resp.* CTL[NFA,NFA] *formula has a model of at most exponential, resp. double exponential size.*

We show now that the bound for CTL[NFA] cannot be improved.

Theorem 4.7. *There is a sequence of satisfiable* CTL[NFA]*-formulas* $(\psi_n)_{n\in\mathbb{N}}$ *such that the size of any model of* ψ_n *is at least doubly exponential in* $|\psi_n|$.

The next theorem provides information about the type of models we can expect. This is useful for synthesis purposes.

Theorem 4.8. *1. There is a satisfiable* CTL[VPL] *formula which does not have a finite model.*

2. *There is a satisfiable* CTL[DCFL] *formula which has no pushdown system as a model.*
3. *Every satisfiable* CTL[VPL] *formula has a visibly pushdown system as a model.*

Proof (Sketch of Part 3). The satisfiability problem for CTL[VPL] can be translated into that of a non-deterministic Büchi visibly pushdown tree automaton (VPTA). An unrolling of this automaton does not necessarily lead to the claimed visibly pushdown system. First, such a system might admit paths which violate the Büchi condition. And secondly, the lack of determinism combines successors of different transitions undesirably. However, Thm. 4.2 Part 3 states that CTL[VPL] can be translated into ΔPDL$^?$[VPL] whose satisfiability problem reduces to the emptiness problem for stair-parity VPTA [27]. There exists an exponential reduction from stair-parity VPTA to parity tree automata (PTA) which preserves satisfiability. The emptiness test is constructive in the sense that for every PTA accepting a non-empty language there exists a finite transition system which satisfies this PTA. This system can be translated back into a visibly pushdown system satisfying the given CTL[VPL]- or ΔPDL$^?$[VPL]-formula. Implementing this idea, however, requires some care and is technically involved. □

Putting Thm. 4.5, Prop. 4.6 and Thm. 4.8 together we obtain the following separations. Note that the first three inequalities of the corollary can also be obtained from language theoretical observations.

Corollary 4.9. CTL[REG] $\lneq$ CTL[VPL] $\lneq$ CTL[DCFL] $\lneq$ CTL[CFL].

5 Satisfiability

In this section we study the complexity of the satisfiability problem for a variety of CTL[$\mathfrak{A}$,$\mathfrak{B}$] logics. The presented lower and upper bounds, as shown in Fig. 2, also yield sharp bounds for EF[_] and CTL[_].

Theorem 5.1. *The satisfiability problems for* CTL[DPDA, _] *and for* CTL[_, DPDA] *are undecidable.*

Proof. Harel et al. [19] show that PDL over regular programs with the one additional language $L{:=}\{a^n b a^n \mid n \in \mathbb{N}\}$ is undecidable. Since $L \in$ DCFL $\supseteq$ REG, the logic EF[DPDA] is undecidable and hence so is CTL[DPDA, _]. As for the second claim, the undecidable intersection problem of two DPDA, say $\mathcal{A}$ and $\mathcal{B}$, can be reduced to the satisfiability problem of the CTL[_, DPDA]-formula $\mathtt{AF}^{\mathcal{A}}\mathtt{AXff} \ \wedge \ \mathtt{AF}^{\mathcal{B}}\mathtt{AXff}$. Note that a single state with no outgoing transitions still has outgoing paths labeled with ϵ. This formula is therefore only satisfiable if $L(\mathcal{A}) \cap L(\mathcal{B}) \neq \emptyset$. □

Theorem 5.2. *The upper bounds for the satisfiability problem are as in Fig. 2.*

Proof. By Thm. 4.2(3), CTL[$\mathfrak{A}$, $\mathfrak{B}$] can be translated into $\triangle$PDL$^?$[$\mathfrak{A} \cup \mathfrak{B}$] with a blow-up that is determined by the worst-case complexity of transforming an arbitrary $\mathfrak{A}$-automaton into a deterministic one. The claim follows using that REG $\subseteq$ VPL and that the satisfiability problem for $\triangle$PDL$^?$[REG] is in EXPTIME [15] and for $\triangle$PDL$^?$[VPL] is in 2EXPTIME [27]. □

The hardness results are more technically involved.

Theorem 5.3. *1.* CTL[DFA, NFA] *and* CTL[_, DVPA] *are 2EXPTIME-hard.*
2. CTL[DVPA, NFA] *and* CTL[_, DVPA $\cup$ NFA] *are 3EXPTIME-hard.*

Corollary 5.4. *The lower bounds for the satisfiability problem are as in Fig. 2.*

Proof. As CTL is EXPTIME-hard [12], so is CTL[_, _]. The 2EXPTIME lower bound for PDL[DVPA] [27] is also a lower bound for CTL[DVPA, _] due to Thm. 4.2. Finally, Thm. 5.3 and Prop. 4.1(2) complete the picture. □

In the remaining part of this section we sketch the proof of Thm. 5.3. For each of the four lower bounds, we reduce from the word problem of an alternating Turing machine T with an exponentially or doubly exponentially, resp., space bound. These problems are 2EXPTIME-hard and 3EXPTIME-hard [8], respectively.

A run of such a machine can be depicted as a tree. Every node stands for a configuration — that is, for simplicity, a bounded sequence of cells. An universal choice corresponds to a binary branching node, and an existential choice to an unary node. We aim to construct a CTL[_,_]-formula φ such that each of its tree-like models resembles a tree expressing a successful run of T on a given input. Thereto, the configurations are linearized — an edge becomes a chain of edges, in the intended model, and a node represents a single cell. The content of each cell is encoded as a proposition. However, the linearization separates neighboring cells of consecutive configurations. Between these

	DFA	NFA	DVPA	VPA	DPDA, PDA
DFA, NFA	EXPTIME	2EXPTIME	2EXPTIME	3EXPTIME	undec.
DVPA, VPA	2EXPTIME	3EXPTIME	2EXPTIME	3EXPTIME	undec.
DPDA, PDA	undec.	undec.	undec.	undec.	undec.

Fig. 2. The time complexities of checking satisfiability for a CTL[$\mathfrak{A}$,$\mathfrak{B}$] formula. Entries denote completeness results. The rows contain different values for $\mathfrak{A}$ as the results are independent of whether or not the automata from this class are deterministic.

cells, certain constraints have to hold. So, the actual challenge for the reduction is that φ must bridge this exponential or doubly exponential, resp., gap while be of a polynomial size in n, i.e. in the input size to T.

We sketch the construction for CTL[DFA, NFA]. The exponential space bound can be controlled by a binary counter. Hence, the constraint applies only to consecutive positions with the same counter value. To bridge between two such positions, we use a proof obligation of the form $\mathtt{AU}^{\mathcal{A}}$ for a NFA $\mathcal{A}$. In a tree model, we say that a node has a *proof obligation* for an $\mathtt{AU}$-formula iff that formula is forced to hold at an ancestor but is not yet satisfied along the path to the said node. The key idea is that we can replace $\mathcal{A}$ by an equivalent automaton $\mathcal{D}$ without changing the models of φ. In our setting, $\mathcal{D}$ is the deterministic automaton resulting from the powerset-construction [30]. In other words, we simulate an exponentially sized automaton. Here, the mentioned obligation reflects the value of the counter and the expected content of a cell.

One of the building blocks of φ programs the obligation with the current value of the counter. Thereto, we encode the counter as a chain of labels in the model, say $(\mathtt{bit}_i^{b_i})_{1\leq i\leq n}$ where $b_i \in \mathbb{B}$ is the value of the ith bit. The automaton $\mathcal{A}$ contains states q_i^b for all $1 \leq i \leq n$ and $b \in \mathbb{B}$. Initially, it is ensured that $\mathcal{D}$ is in the state $\{q_i^b \mid 1 \leq i \leq n, b \in \mathbb{B}\}$. Informally, this set holds all possibilities for the values of each bit. In $\mathcal{A}$, any q_i^b has self-loops for any label except for $\mathtt{bit}_i^{\neg b}$. Hence, a traversal of a chain eliminates invalid bit assignments from the subset and brings $\mathcal{D}$ into the state $\{q_i^{b_i} \mid 1 \leq i \leq n\}$ which characterizes the counter for which the chain stands. Finally for matching, a similar construction separates proof obligations depending on whether or not they match the counter: unmatched obligations will be satisfied trivially, and matching ones are ensured to be satisfied only if the expected cell is the current one.

For the other parts involving DVPA, again, the constructed formula φ shall imitate a successful tree of T on the input. The space bound can be controlled by a counter with appropriate domain. The constraints between cells of consecutive configurations, however, are implemented differently. We use a deterministic VPA to push all cells along the whole branch of the run on the stack — configuration by configuration. At the end, we successively take the cells from the stack and branch. Along each branch, we use the counter to remove exponential or doubly exponential, resp., many elements from stack to access the cell at the same position in the previous configuration. So, as a main component of φ we use either $\mathtt{AU}^{\mathcal{A}}\mathtt{AXff}$ or $\mathtt{AG}^{\mathcal{A}}\mathtt{ff}$ for some VPA $\mathcal{A}$. In the case of a doubly exponential counter, the technique explained for CTL[DFA, NFA] can be applied. But this time, a proof obligation expresses a bit number and its value.

6 Model Checking

In this section we consider model-checking of CTL[$\mathfrak{A}$, $\mathfrak{B}$] against finite and infinite transition systems, obtained as the transition graphs of (visibly) pushdown automata. Note that undecidability is quickly obtained beyond that. For instance model checking the genuine CTL fragment EF is undecidable over the class of Petri nets, and for EG model checking becomes undecidable of the class of Very Basic Parallel Processes [16].

6.1 Finite State Systems

The following table summarises the complexities of model checking CTL[$\mathfrak{A}$,$\mathfrak{B}$] in finite transition systems in terms of completeness. Surprisingly, despite its greatly increased expressive power compared to CTL, CTL[PDA,DPDA] remains in PTIME. In general, it is the class $\mathfrak{B}$ which determines the complexity. The table therefore only contains one row ($\mathfrak{A}$) and several columns ($\mathfrak{B}$). Note that PDA covers everything down to DFA while DPDA covers DVPA and DFA.

	DPDA	NFA	VPA	PDA
PDA	PTIME	PSPACE	EXPTIME	undec.

Theorem 6.1. *Model checking of finite state systems against* CTL[PDA,DPDA] *is in* PTIME*,* CTL[PDA,VPA] *is in* EXPTIME*, and* CTL[PDA,NFA] *is in* PSPACE*.*

Proof (Sketch). To obtain a PTIME algorithm for CTL[PDA,DPDA] we observe that — as for plain CTL — we can model check a CTL[$\mathfrak{A}$,$\mathfrak{B}$] formula bottom-up for any $\mathfrak{A}$ and $\mathfrak{B}$. Starting with the atomic propositions one computes for all subformulas the set of satisfying states, then regards the subformula as a proposition. Hence, it suffices to give algorithms for $\mathtt{E}(x\mathtt{U}^{\mathcal{A}}y)$ and $\mathtt{E}(x\mathtt{R}^{\mathcal{B}}y)$ for propositions x and y.

We prove the case for $\mathtt{E}(x\mathtt{U}^{\mathcal{A}}y)$ by reduction to non-emptiness of PDA which is well-known to be solvable in PTIME. Let $\mathcal{T}{=}(\mathcal{S}, \rightarrow, \ell)$ be an LTS and $\mathcal{A}{=}(Q, \Sigma, \Gamma, \delta, q_0, F)$. We construct for every $s \in \mathcal{S}$ a PDA $\mathcal{A}_{\mathcal{T}}{=}(Q \times \mathcal{S}, \Sigma, \Gamma, \delta', (q_0, s), F')$, where

$$F'{:=}\{(q, s) \mid q \in F \text{ and } y \in \ell(s)\} \text{ and}$$
$$\delta'((q, s), a, \gamma){:=}\{(q', s') \mid q' \in \delta(q, a, \gamma) \text{ and } s \xrightarrow{a} s' \text{ and } x \in \ell(s)\}.$$

Clearly, if $\mathcal{L}(\mathcal{A}_{\mathcal{T}}) \neq \emptyset$ then there exist simultaneously a word $w \in \mathcal{L}(\mathcal{A})$ and a path π in $\mathcal{T}$ starting at s and labeled with w, s.t. x holds everywhere along π except for the last state in which y holds. Note that this takes time $\mathcal{O}(|\mathcal{S}| \cdot |\mathcal{A}| \cdot |\mathcal{T}|)$.

The same upper bound can be achieved for ER-formulas. However, they require the automaton to be deterministic. This is due to the quantifier alternation in the release operator, as discussed in Sect. 2.

We show containment in PTIME by a reduction to the problem of model checking a fixed LTL formula on a PDS. Let $\mathcal{T}$ and $\mathcal{A}$ be defined as above except that $\mathcal{A}$ is deterministic. We construct a PDS $\mathcal{T}_{\mathcal{A}} = (Q \times \mathcal{S} \cup \{g, b\}, \Gamma, \Delta, \ell')$, where ℓ' extends ℓ by $\ell'(b) = \text{dead}$ for a fresh proposition dead. Intuitively, g represents “good” and

b “bad” states, i.e. dead-end states, in which $\mathtt{E}(x\mathtt{R}^{\mathcal{A}}y)$ has been fulfilled or violated, respectively. Furthermore, Δ contains the following transition rules:

$$((q,s),\gamma) \hookrightarrow \begin{cases} (g,\epsilon) & \text{if } x \in \ell'(s) \text{ and } (q \in F \text{ implies } y \in \ell'(s)) \\ (b,\epsilon) & \text{if } q \in F \text{ and } y \notin \ell'(s) \\ ((q',s'),w) & \text{if none of the above match and there ex. } a \in \Sigma\text{, s.t.} \\ & s \xrightarrow{a} s' \text{ and } (q',w) \in \delta(q,a,\gamma) \text{ for some } \gamma \in \Gamma, w \in \Gamma^* \end{cases}$$

Note that $|\mathcal{T}_{\mathcal{A}}| = \mathcal{O}(|\mathcal{T}| \cdot |\mathcal{A}|)$. Now consider the LTL formula $\mathtt{F}\mathtt{dead}$. It is not hard to show that $s \not\models_{\mathcal{T}} \mathtt{E}(x\mathtt{R}^{\mathcal{A}}y)$ iff $((q_0,s),\epsilon) \models_{\mathcal{T}_{\mathcal{A}}} \mathtt{F}\mathtt{dead}$. The fact that model checking a fixed LTL formula over a PDS is in PTIME [6] completes the proof.

To show that CTL[PDA,NFA] is in PSPACE we reduce $\mathtt{E}(x\mathtt{R}^{\mathcal{B}}y)$ to the problem of checking a fixed LTL formula against a determinisation of the NFA $\mathcal{B}$. This is a repeated reachability problem over the product of a Büchi automaton and a determinisation of the NFA. Since we can determinise by a subset construction, we can use Savitch’s algorithm [31] and an on-the-fly computation of the edge relation. Because Savitch’s algorithm requires logarithmic space over an exponential graph, the complete algorithm runs in PSPACE.

Using the fact that every VPA can be determinised at a possibly exponentially cost [2], we obtain an algorithm for CTL[PDA,VPA]. □

We now consider the lower bounds.

Theorem 6.2. *For fixed finite state transition systems of size 1, model checking for* EF[VPA] *is PTIME-hard,* EG[NFA] *is PSPACE-hard,* EG[VPA] *is EXPTIME-hard, and* EG[PDA] *is undecidable.*

Proof (Sketch). It is known that model checking CTL is PTIME-complete. Thus, the model checking problems for all logics between CTL and CTL[CFL] are PTIME-hard. However, for EF[VPL] it is already possible to strengthen the result and prove PTIME-hardness of the expression complexity, i.e. the complexity of model checking on a fixed transition system. The key ingredient is the fact that the emptiness problem for VPA is PTIME-hard.[1]

Model checking the fragment EG[$\mathfrak{A}$] is harder, namely PSPACE-hard for the class REG already. The proof is by a reduction from the n-tiling problem [34] resembling the halting problem of a nondeterministic linear-space bounded Turing Machine. Two aspects are worth noting. First, this result — as opposed to the one for the fragment EF[$\mathfrak{A}$] — heavily depends on the fact that $\mathfrak{A}$ is a class of nondeterministic automata. For $\mathfrak{A}$ = DFA for instance, there is no such lower bound unless PSPACE = PTIME. The other aspect is that the formulas constructed in this reduction are of the form $\mathtt{EG}^{\mathcal{A}}\mathtt{ff}$, no boolean operators, no multiple temporal operators, and no atomic propositions are needed.

The principle is that tilings can be represented by infinite words over the alphabet of all tiles. Unsuccessful tilings must have a finite prefix that cannot be extended to become successful. We construct an automaton $\mathcal{A}$ which recognises unsuccessful prefixes.

[1] This can be proved in just the same way as PTIME-hardness of the emptiness problem for PDA.

Every possible tiling is represented by a path in a one-state transition system with universal transition relation. This state satisfies the formula $\mathtt{EG}^{\mathcal{A}}\mathtt{ff}$ iff a successful tiling is possible.

However, if we increase the language class to CFL we are able to encode an undecidable tiling problem. The octant tiling problem asks for a successful tiling of the plane which has successively longer rows [34]. Since the length of the rows is unbounded, we need non-determinism and the unbounded memory of a PDA to recognise unsuccessful prefixes.

The situation is better for VPA. When used in EF-operators, visibly pushdown languages are not worse than regular languages, even for nondeterministic automata. This even extends to the whole of all context-free languages.

In EG-operators VPA increase the complexity of the model checking problem even further in comparison to NFA to EXPTIME. We reduce from the halting problem for alternating linear-space bounded Turing machines. An accepting computation of the machine can be considered a *finite* tree. We encode a depth-first search of the tree as a word and construct a VPA $\mathcal{A}$ accepting all the words that do not represent an accepting computation. As in previous proofs, one then takes a one-state transition system with universal transition relation and formula $\mathtt{EG}^{\mathcal{A}}\mathtt{ff}$. □

6.2 Visibly Pushdown Systems

We consider model checking over an infinite transition system represented by a visibly pushdown automaton. The following summarises the complexity results in terms of completeness.

	DFA,DVPA	NFA,VPA	DPDA
DFA . . . VPA	EXPTIME	2EXPTIME	undec.

Theorem 6.3. *Model checking visibly pushdown systems against* CTL[VPA,DVPA] *is in* EXPTIME*, whereas against* CTL[VPA,VPA] *it is in 2*EXPTIME*.*

Proof (sketch). To obtain the first result, we follow the game approach hinted at in Section 2 (hence the restriction to DVPA). We reduce the model checking problem to a Büchi game played over a PDS, which is essentially the product of the formula (including its automata) and the model. That is, for example, from a state $(s, \varphi_1 \wedge \varphi_2)$ the opponent can move to (s, φ_1) or (s, φ_2) — the strategy is to pick the subformula that is not satisfied. The stack alphabet is also a product of the model stack and the formula VPA stack. For a temporal operator augmented with a VPA, the formula VPA component is set to $\perp$ to mark its bottom of stack. Then the automaton is simulated step-wise with the model. At each step the appropriate player can decide whether to attempt to satisfy a subformula, or continue simulating a path and run. Since deciding these games is EXPTIME [38], we get the required result. The second result follows by determinisation of the VPA. □

Theorem 6.4. *Model checking visibly pushdown systems against* CTL[DFA] *is hard for* EXPTIME*,* EG[NFA] *is hard for 2*EXPTIME*, and* EF[DPDA] *and* EG[DPDA] *are undecidable.*

Proof (sketch). EXPTIME-hardness follows immediately from the EXPTIME-hardness of CTL over pushdown systems [21] and that CTL is insensitive to the transition labels.

2EXPTIME-hardness is similar to Bozzelli's 2EXPTIME-hardness for CTL* [25]. This is an intricate encoding of the runs of an alternating EXPSPACE Turing machine. The difficulty lies in checking the consistency of a guessed work tape of exponential length. We are able to replace the required CTL* subformula with a formula of the form $\mathtt{EG}^A$, giving us the result.

The undecidability results are via encodings of a two counter machine. Intuitively, the visibly pushdown system simulates the machine, keeping one counter in its stack. It outputs the operations on the second counter (appropriately marked to meet the visibly condition) and the DPDA checks for consistency. In this way we can simulate two counters. □

6.3 Pushdown Systems

For pushdown systems we have the following complexity-theoretic completeness results.

	DFA	NFA	DVPA
DFA/ NFA	EXPTIME	2EXPTIME	undec.

Theorem 6.5. *Model checking pushdown systems against* CTL[NFA,DFA] *is in EXPTIME, against* CTL[NFA,NFA] *it is in 2EXPTIME, against* EF[DVPA] *and* EG[DVPA] *it is undecidable.*

Proof (sketch). The decidability results are similar to the case of visibly pushdown systems; we simply drop the visibly restriction. The lower bounds which do not follow from the results on VPA can be obtained by a reduction from two counter machines. □

7 Conclusion and Further Work

To the best of our knowledge, this is the first work considering a parametric extension of CTL by arbitrary classes of formal languages characterising the complexities of satisfiability and model checking as well as the expressive power and model-theoretic properties of the resulting logics in accordance to the classes of languages. The results show that some of the logics, in particular CTL[VPL] may be useful in program verification because of the combination of an intuitive syntax with reasonably low complexities of the corresponding decision problems.

Some questions still remain to be answered. First, it is open whether the relationships are strict between logics which are connected by solid vertical lines in Fig. 1. Moreover, the presented separations are rather coarse. Hence, it is desirable to have a generic approach to separate logics, e.g. CTL[$\mathfrak{A}$] $\lneq$ CTL[$\mathfrak{B}$] whenever $\mathfrak{A}$ is a "reasonable" subset of $\mathfrak{B}$.

It is an obvious task for further work to consider CTL* or CTL$^+$ as the base for similar extensions, and to characterise the expressive power and the complexities of the resulting logics.

References

1. Inc. Accellera Organization. Formal semantics of Accellera property specification language, 2004. In Appendix B of `http://www.eda.org/vfv/docs/PSL-v1.1.pdf`.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Ann. ACM Symp. on Theory of Computing, STOC'04*, pages 202–211, 2004.
3. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property specification language. In *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'02*, volume 2280 of *LNCS*, pages 296–311, Grenoble, France, 2002. Springer.
4. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 303(1):7–34, 2003.
5. 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.
6. 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.
7. T. Brázdil and I. Cerná. Model checking of regCTL. *Computers and Artificial Intelligence*, 25(1), 2006.
8. Ashok K. Chandra, Dexter C. Kozen, and Larry J.Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
9. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logics of Programs: Workshop*, volume 131 of *LNCS*, pages 52–71, Yorktown Heights, New York, 1981. Springer.
10. 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.
11. A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logics. *ACM Transactions on Computational Logic*, 5(2):282–315, 2004.
12. 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.
13. 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.
14. 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.
15. 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.
16. J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.
17. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
18. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
19. D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.
20. J. G. Henriksen and P. S. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied Logic*, 96(1–3):187–207, 1999.
21. I. Walukiewicz. Model checking ctl properties of pushdown systems. In *FSTTCS*, pages 127–138, 2000.

22. D. Kirsten. *Automata Logics, and Infinite Games – A Guide to Current Research*, chapter 9 – Alternating Tree Automata and Parity Games, pages 405–411. Number 2500 in LNCS. Springer, 2002.
23. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.
24. O. Kupferman, N. Piterman, and M. Y. Vardi. Extended temporal logic revisited. In *Proc. 12th Int. Conf. on Concurrency Theory, CONCUR'01*, volume 2154 of *LNCS*, pages 519–535. Springer, 2001.
25. L. Bozzelli. Complexity results on branching-time pushdown model checking. *Theor. Comput. Sci.*, 379(1-2):286–297, 2007.
26. 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 *LNAI*, pages 19–33. Springer, 2010.
27. C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *J. Log. Algebr. Program.*, 73(1-2):51–69, 2007.
28. Ch. 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.
29. 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.
30. M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal*, 2(3):115–125, 1959.
31. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.
32. 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.
33. R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982.
34. 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.
35. 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.
36. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
37. M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004.
38. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.
39. P. Wolper. Temporal logic can be more expressive. In *SFCS '81: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, pages 340–348, Washington, DC, USA, 1981. IEEE Computer Society.

A Proof Details

We provide the details for a number of the proofs omitted from the main paper. However, due to the length of the combined proofs, we only present the most interesting results here. For a complete set of results, please refer to the full version available from http://web.comlab.ox.ac.uk/people/Stephan.Kreutzer/csl10.pdf.

Semantics of PDL with the Δ-operator and tests. Formulas Form and programs Prog of $\Delta\mathrm{PDL}^?[\mathfrak{A}]$ for some $\mathfrak{A}$ over an alphabet Σ are the least sets satisfying the following.

1. $\mathcal{P} \subseteq \mathsf{Form}$.
2. If $\varphi, \psi \in \mathsf{Form}$ then $\varphi \vee \psi \in \mathsf{Form}$, $\neg\varphi \in \mathsf{Form}$.
3. If $\varphi \in \mathsf{Form}$, $\mathcal{A} \in \mathsf{Prog}$ then $\langle\mathcal{A}\rangle\varphi \in \mathsf{Form}$.
4. $\mathfrak{A} \subseteq \mathsf{Prog}$.
5. For every $\mathfrak{A}$-automaton $\mathcal{A}$ over $\Sigma \cup \{\varphi? \mid \varphi \in \mathsf{Form}\}$ we have $\mathcal{A} \in \mathsf{Prog}$.
6. If $\mathcal{A} \in \mathsf{Prog}$ and $\mathcal{A}'$ results from $\mathcal{A}$ by equipping it with a Büchi condition on states, then $\Delta\mathcal{A}' \in \mathsf{Form}$.

$\Delta\mathrm{PDL}^?[\mathfrak{A}]$ consists of all elements of Form which are constructed in this way. The fragment PDL[$\mathfrak{A}$] is obtained by removing clauses (5) and (6). The semantics is again defined over states of transition systems. The clauses for atomic propositions and the boolean operators are as usual. For the other constructs, we use the fact that programs and formulas are defined inductively. For a $\mathcal{T} = (\mathcal{S}, \to, \ell)$ with edge labels in Σ' and a finite subset $\Phi \subset \mathsf{Form}$ of formulas let $\mathcal{T}^{\Phi}$ result from $\mathcal{T}$ by adding, for every $s \in \mathcal{S}$ and every $\varphi \in \Phi$, a transition $s \xrightarrow{\varphi?} s$ if $\mathcal{T}, s \models \varphi$. For a formula φ let $?(\varphi)$ be the set of all tests $\psi?$ occurring in φ syntactically.

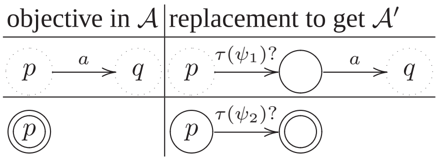
$$\begin{aligned}
\mathcal{T}, s \models \langle\mathcal{A}\rangle\varphi \ \text{ iff } & \exists\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n \text{ in} \\
& \mathcal{T}^{?(\langle\mathcal{A}\rangle\varphi)} \text{ with } s_0 = s \text{ s.t.} \\
& \text{(i) } a_1 \ldots a_n \in L(\mathcal{A}), \text{ and} \\
& \text{(ii) } \mathcal{T}, s_n \models \varphi. \\
\mathcal{T}, s \models \Delta\mathcal{A} \ \text{ iff } & \exists\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \text{ in } \mathcal{T}^{?(\langle\mathcal{A}\rangle\varphi)} \\
& \text{with } s_0 = s \text{ and } a_1 a_2 \ldots \in L(\mathcal{A}).
\end{aligned}$$

A.1 Proof of Thm. 4.2

Theorem

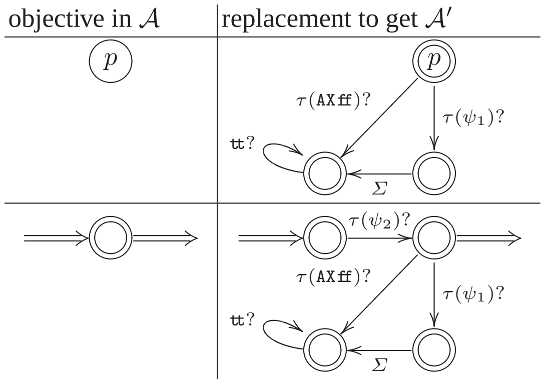
1. For all $\mathfrak{A}$: PDL[$\mathfrak{A}$] $\equiv_{\mathsf{lin}}$ EF[$\mathfrak{A}$].
2. For all $\mathfrak{A}, \mathfrak{B}$: EF[$\mathfrak{A}$] $\lneq_{\mathsf{lin}}$ CTL[$\mathfrak{A}$,$\mathfrak{B}$].
3. For all $\mathfrak{A}, \mathfrak{B}$: if $\mathfrak{B}$ is a class of deterministic automata then CTL[$\mathfrak{A}$,$\mathfrak{B}$] $\leq_{\mathsf{lin}}$ $\Delta\mathrm{PDL}^?[\mathfrak{A}\cup\mathfrak{B}]$.

Proof. The first two cases are left to the full version. Here we concentrate on the third case. We focus on finite state automaton only. However, the proof can be extended to the two kinds of pushdown automata considered in the report—every subsequent replacement can be extended to push, pop and internal operations. Tests are internal operations, anyway. The proposed translation of the ER-formulas relies on an translation of the possibly larger formula $\texttt{AXff} \equiv \neg\texttt{E}(\texttt{tt}\texttt{U}^{\Sigma}\texttt{tt})$. As latter does not involve any ER-formula we may assume an appropriate induction principle. The translations of proposition and boolean operation are straight forward. Given a CTL-formula $\texttt{E}(\psi_1\texttt{U}^{\mathcal{A}}\psi_2)$, we construct an automaton $\mathcal{A}'$ by modifying $\mathcal{A}$ as follows.



Each dotted circle matches either a final or non-final state. The function τ refers to the translation for those formulas for which the induction hypothesis is applicable. Obviously $\mathcal{T}, s \models \texttt{E}(\psi_1\texttt{U}^{\mathcal{A}}\psi_2)$ iff $\mathcal{T}, s \models \langle\mathcal{A}'\rangle\texttt{tt}$.

And as for a formula $\varphi{:=}\texttt{E}(\psi_1\texttt{R}^{\mathcal{A}}\psi_2)$, the automaton $\mathcal{A}$ is assumed to be complete, and is turned into a safety ω-automaton $\mathcal{A}'$ as follows. The translation of φ is $\Delta\mathcal{A}'$.



The double arrows indicate either in- or outgoing edges. For the later discussions we wish $\mathcal{A}'$ to be deterministic. Therefore, the edge $\tau(\psi_1)?$ is omitted iff $\tau(\texttt{AXff}) = \tau(\psi_1)$. Note that in this case, the Σ-transition is not eligible anyway as $\tau(\psi_1)?$ reports a dead-end state in the LTS.

Let $\pi = s_0, a_1, s_1, \ldots$ be a path witnessing $\mathcal{T}, s_0 \models \texttt{E}(\psi_1\texttt{R}^{\mathcal{A}}\psi_2)$. As $\mathcal{A}$ is deterministic, the run of $\mathcal{A}$ on a prefix π' of π is a prefix of the run on π. Thus the witnessing path can be turned into a run in $\mathcal{A}'$: As long as ψ_1 does not hold we follow the trace of $\mathcal{A}$. In particular, if a final state in $\mathcal{A}$ is reached then the respective edge $\tau(\psi_2)?$ can be passed. Now, if the current state has no children then $\mathcal{A}'$ can follow the edges $\tau(\texttt{AXff})?$, and for ever $\texttt{tt}?$. And if ψ_1 holds in the current state of the LTS the proof obligation vanishes for the next state onwards. Hence $\mathcal{A}'$ can take the way $\tau(\psi_1)?$.

Conversely, let $\pi = s_0, a_1, s_1, \ldots$ be a path witnessing $\mathcal{T}, s_0 \models \Delta\mathcal{A}'$. The run on this path has a prefix—maybe the whole run—which corresponds to a run of $\mathcal{A}$ on π where ψ_2 is ensured every time $\mathcal{A}$ recognizing the present word, that is, the states in the lower line of replacement are not taken. If the prefix is infinite the run is an infinite witness for φ. Otherwise, the suffix has the shape $(\tau(\mathtt{AX}\mathtt{tt})?)\ (\mathtt{tt}?)^*$ or $(\tau(\psi_1)?)\ \Sigma\ (\mathtt{tt}?)^*$. Both alternatives presents a finite witness of φ. In particular, the last state of the first witness has no successors.

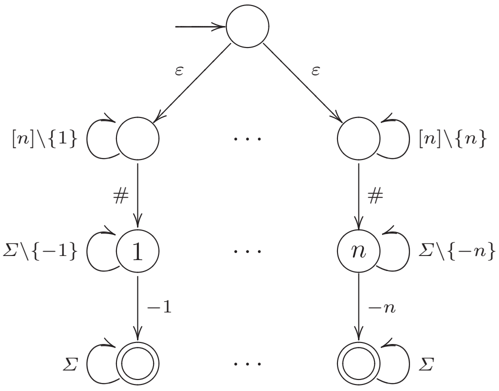
Finally, the transition is only linearly increasing.

□

A.2 Proof of Thm. 4.7

Theorem. There is a sequence of satisfiable CTL[NFA]-formulas $(\psi_n)_{n\in\mathbb{N}}$ such that the size of any model of ψ_n is at least doubly exponential in $|\psi_n|$.

Proof. Fix an even number $n > 0$. Let $[n]{:=}\{1,\ldots,n\}$. Let $\mathcal{A}$ be the following NFA over the alphabet $\Sigma{:=}\{-n,\ldots,-1,\#,1,\ldots,n\}$.



Let $Q \supset [n]$ be set of its states. The ϵ-transition can be eliminated with a linear overhead. However, the ϵ-transitions are more convenient for presentation purposes. In any case, the size of $\mathcal{A}$ is linear in n. Let $\mathcal{D}$ be a deterministic automaton for $\mathcal{A}$ obtained from the standard powerset construction [30]. Although we do not use $\mathcal{D}$ explicitly, it allows us to say that at a node of a model there is a proof obligation for $\mathtt{AF}^{\mathcal{A}}\_$ in a state $S \subseteq Q$, for instance.

Let $S \subseteq [n]$. Consider a Hintikka model of some formula φ and let $\mathtt{AF}^{\mathcal{A}}p$ occur in some node. Suppose that we have the control over the formulas φ, or over the Hintikka model, respectively. Now, we can set up $\mathcal{A}$ with the set S as follows. Let $[n] \setminus S = \{s_1,\ldots,s_\ell\}$. Consider a path π passing the labels $s_1,\ldots,s_\ell,\#$ such that along the path p does not hold. At the end of this path, there is a proof obligation for $\mathtt{AF}^{\mathcal{A}}p$ in the state S (w.r.t to $\mathcal{D}$). Iterating this construction with different sets S yields to many proof obligations for the $\mathtt{AF}^{\mathcal{A}}$ along the iteration.

As for the lower bound, we construct a formula φ polynomially sized in n such that any of its tree model consists of two phases. The first one creates exponential many proof obligations for some instances $\mathtt{AF}^{\mathcal{A}}p$ along the path. There are doubly exponential many such paths. In the second phase the model satisfies these obligations but it also materializes all the obligations. The set of proof obligations will be so that the materialization is characteristic for this set. This property prevents any model from sharing the different materializations. To be more precise, the first phase is built from smaller blocks, called S-blocks. For each set $S \subseteq [n]$ of size $n/2$ there is a leaf such that the block imposes an additional proof obligation for $\mathtt{AF}^{\mathcal{A}}p$ in the state S. The first phase consists of $b := \binom{n}{n/2}/2$ many[2] layers of S-blocks. For each list $\boldsymbol{S} := S_1, \ldots, S_{n/2}$ with each element in $\binom{[n]}{n/2}$, there is a path (starting from the root) which reaches the second phase and which has collected proof obligations for $\mathtt{AF}^{\mathcal{A}}p$ in the state S_i for any $i \in [n/2]$. In the last phase, the model can pick out $b = \binom{n}{n/2} - b$ sets in $\binom{[n]}{n/2}$. Only for these sets the model has a path. For a set $\{a_1, \ldots, a_{n/2}\} \in \binom{[n]}{n/2}$, the path touches the labels $-a_1$, …, $-a_{n/2}$ in some order. The node after the last label has no successor, and it is the only state on the path at which p holds. The passed labels transform the proof obligations. The only node which can fulfill the the proof obligations is a dead-end node. The combination of both properties implement the said materialization. This final phase is implemented by a so-called T-block.

The encoding of this paradigm uses two kinds of counters: one to iterate the S-blocks, and the others to control the branching in any T-block. We write $\boldsymbol{C}$ for a list of n (distinct) propositions which are intended to be used as n-bit counter.

Let $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$—possibly indexed—be counters, $\ell \in \mathbb{N}$, $v \in \{0, \ldots, 2^n - 1\}$, and $\varDelta \subseteq \varSigma$. There are CTL-formulas of polynomial size (in n) which encode the following properties.

Formula	Property
$\ulcorner \boldsymbol{C} = v \urcorner$, $\ulcorner \boldsymbol{C} \neq v \urcorner$	The counter $\boldsymbol{C}$ has (not) the value v.
$\ulcorner \mathtt{AX}^{\varDelta} \boldsymbol{A} = \boldsymbol{B} \urcorner$	The value of $\boldsymbol{A}$ in any $\varDelta$-successor is the value of $\boldsymbol{B}$ of the current state.
$\ulcorner \mathtt{AX}^{\varDelta} \boldsymbol{A} = \boldsymbol{B} + 1 \urcorner$	The value of $\boldsymbol{A}$ in any $\varDelta$-successor is the successor value of $\boldsymbol{B}$ of the current state. If $\boldsymbol{B}$ represents 2^n the behavior is undefined.
$\ulcorner \boldsymbol{A} = \sum_{i=0}^{\ell} \boldsymbol{B}_i \urcorner$	The value of $\boldsymbol{A}$ is the sum of the values of $\boldsymbol{B}_i$ for all $i = 0 \ldots \ell$. Here, we allow (polynomial many) additional counters, respectively variables, to compute the sum successively.

The final formula φ uses the propositions p, and the counters $\boldsymbol{C}$ and $\boldsymbol{C}_i$ for $i = 0, \ldots, n$.

Encoding of S-blocks. For $\varDelta \subseteq \varSigma$ and ψ a CTL-formula, the formula

$$!\mathtt{X}^{\varDelta}\psi := \mathtt{AX}^{\varSigma \setminus \varDelta}\mathtt{ff} \ \wedge \ \mathtt{EX}^{\varDelta}\mathtt{tt} \ \wedge \ \mathtt{AX}^{\varDelta}\psi$$

forces that for any of its models there are only $\varDelta$-successors and at each of them ψ holds. Note that for an $a \in \varDelta$ there might be more than one a-successors.

The enumeration of all $S \in \binom{n}{n/2}$ is constructed level by level. An element S is enumerated increasingly. Thereto, the auxiliary formulas $\varphi_{m,\ell}$ are introduced for ℓ the number of levels remaining and m the maximal number seen along an enumeration so far.

$$\varphi_{m,0} := !\mathtt{X}^{\{\#\}}\mathtt{tt}$$

[2] Indeed, $\binom{n}{n/2} = \frac{(n-1)! \cdot n}{(n/2)!(n/2-1)! \cdot n/2} = 2\binom{n-1}{n/2}$ is even.

$$\varphi_{m,\ell} := !\mathtt{X}^{\{m+1,\dots,n+1-\ell\}}\neg p \qquad\qquad \text{if } \ell > 0$$

Finally, an S-block is forced by

$$\sigma := \mathtt{AF}^{\mathcal{A}} p \ \wedge\ \neg p \ \wedge\ \varphi_{0,n/2} \ \wedge \bigwedge_{\substack{m\in[n]\\ k\in[n/2]}} \mathtt{AX}^{\Sigma^{k-1}\{m\}}\varphi_{m,n/2-k}.$$

Any (tree) model of σ enumerates all subsets of $[n]$ of size $n/2$, and ensures that along the enumeration p does not hold while the proof obligation $\mathtt{AF}^{\mathcal{A}} p$ is imposed on the root. That is, for any sequence $a_1, \dots, a_{n/2+1}$ in Σ the following properties are equivalent.

- $a_1, \dots, a_{n/2}$ is a strictly increasing sequence in $[n]$, and $a_{n/2+1} = \#$.
- there exists a path $s_0, a_1, s_1, a_2, s_2, \dots$ starting at s such that $s_i \models \neg p$ for all $i \in \{0, \dots, n/2\}$, and $s_0 \models \mathtt{AF}^{\mathcal{A}} p$.

Encoding of T-blocks. An T-block is a tree with b leaves. The encoding is similar to that of an S-block. Additionally, at each node v we use a counter C_0 and counters C_i for each outgoing label $-i$. The counter C_0 contains the number of leaves of the tree[3] at v. Similarly, C_i stands for the number of leaves at the respective subtree. The counters C_i must sum up to C_0. In analogy to $\varphi_{m,\ell}$, each formula $\psi_{m,\ell}$ is responsible for a certain level. However, the expression $!\mathtt{X}^{\Delta}$ is replaced by a variation additionally depending on the counter $\boldsymbol{C}_i$.

$$\begin{aligned}
\psi_{m,0} &:= \ulcorner \boldsymbol{C}_0 = 1 \urcorner \ \wedge\ p \ \wedge\ \mathtt{AX}\,\mathtt{ff} \\
\psi_{m,\ell} &:= \neg p \ \wedge \bigwedge_{a\in\Sigma\setminus\{-n,\dots,-1\}} \mathtt{AX}^{\{a\}}\mathtt{ff} \\
&\qquad \wedge \bigwedge_{i=m+1}^{n+1-\ell} \Big\{ \Big(\ulcorner \boldsymbol{C}_i \neq 0 \urcorner \leftrightarrow \mathtt{EX}^{\{-i\}}\mathtt{tt} \Big) \\
&\qquad\qquad\qquad \wedge \ulcorner \mathtt{AX}^{\{-i\}} \boldsymbol{C}_0 = \boldsymbol{C}_i \urcorner \Big\}
\end{aligned}$$

A T-block is represented by the formula τ defined as

$$\ulcorner \boldsymbol{C}_0 = b \urcorner \ \wedge\ \psi_{0,n/2} \ \wedge \bigwedge_{\substack{m\in[n]\\ k\in[n/2]}} \mathtt{AX}^{\Sigma^{k-1}\{-m\}}\psi_{m,n/2-k}$$

Encoding. Now, the S-blocks can be iterated b-times.

$$\begin{aligned}
\varphi := \ulcorner \boldsymbol{C} = 0 \urcorner \ \wedge\ \mathtt{AG}^{\Sigma^*} \Big(& \ulcorner \mathtt{AX}^{\Sigma\setminus\{\#\}} \boldsymbol{C} = \boldsymbol{C} \urcorner \wedge \\
& \ulcorner \mathtt{AX}^{\{\#\}} \boldsymbol{C} = \boldsymbol{C} + 1 \urcorner \Big)
\end{aligned}$$

[3] Because CTL is bisimilar, there might be more than one out-going edge with a given label $a \in \Sigma$. In this case, we pick out one such edge. So, the term “tree” refers to the tree thinned out.

$$\wedge\ \mathtt{AG}^{\Sigma^*\{\#\}}\,(\ulcorner \boldsymbol{C} \neq b \urcorner \to \sigma)$$
$$\wedge\ \mathtt{AG}^{\Sigma^*\{\#\}}\,(\ulcorner \boldsymbol{C} = b \urcorner \to \tau)$$

φ is satisfiable. We construct a tree model of φ. Obviously, the existence of the first phase—as mentioned in the introductive text—is guaranteed because τ and φ without its last conjunct have bisimilar models only. Given a path π from the root to the last element of the first phase, it remains to show how to continue with a T-blocks. By the construction of σ and φ, there are sets $S_1, \ldots, S_b \in \binom{[n]}{n/2}$ such that the path has collected only proof obligation of $\mathtt{AF}^{\mathcal{A}}p$ for the states S_1 to S_b. Let $\mathcal{S} := \{S_i \mid i \in [b]\}$. Now, set

$$\mathcal{T} := \left\{ T \in \binom{[n]}{n/2} \mid [n] \setminus T \notin \mathcal{S} \right\}.$$

Note that $|\mathcal{T}| = \binom{n}{n/2} - |\mathcal{S}| \geq \binom{n}{n/2} - b = b$. Choose a subset $\mathcal{T}' \subseteq \mathcal{T}$ of size $\binom{n}{n/2} - b$. The formula τ forces b branches. Therefore, for each $T \in \mathcal{T}'$ we construct a branch which passes the labels $-t_1, \ldots, -t_b$, where $t_1, \ldots, t_b$ is an increasing enumeration of T. For any $S \in \mathcal{S}$, the sets S and T are not disjoint. Indeed, if they are disjoint then $[n] \setminus T = S$ as both have the same size $n/2$. But this is contradiction to $T \in \mathcal{T}$. The non-disjointness ensures that any proof obligation in S is turned into an obligation for a set of states containing a final state, after passing the labels $-t_1, \ldots, -t_b$. However, this state models p, and hence all proof obligations disappear.

Lower bound. Consider a model $\mathcal{T}$ of φ. Because $\binom{2k}{k} \geq 2^k$ for any $k \in \mathbb{N}$, the set $\binom{\binom{[n]}{n/2}}{b}$ has at least doubly exponential size in n. For any set $\mathcal{S} \in \binom{\binom{[n]}{n/2}}{b}$ there is a rooted path $\pi_{\mathcal{S}}$ through the S-blocks of $\mathcal{T}$ which got proof obligations for $\mathtt{AF}^{\mathcal{A}}p$ for every $S \in \mathcal{S}$ and ends at the first node of a T-block. Let $\mathcal{S}$ and $\mathcal{S}'$ two different sets in $\binom{\binom{[n]}{n/2}}{b}$. As for the lower bound, it suffices to show that the last nodes of $\pi_{\mathcal{S}}$ and $\pi_{\mathcal{S}'}$ are different. Assume that they are identical. The T-block starting at the last node shows b branches, each naming (the negative of each element of) a set $T \subseteq [n]$ of size $n/2$. As in the case of satisfiability, the proof obligations got transformed by each branch. Since a T-block is a dead end, a transformed proof obligation must refer to a set which contains a final state of $\mathcal{A}$. Therefore, T must intersect with any element of $\mathcal{S} \cup \mathcal{S}'$. That is, $([n] \setminus T) \notin \mathcal{S} \cup \mathcal{S}'$. In total, each of the $b = \binom{n}{n/2} - b$ branches names a different set which is not in $\mathcal{S} \cup \mathcal{S}'$. So, $|\mathcal{S} \cup \mathcal{S}'| = b$. Being of size b, both $\mathcal{S}$ and $\mathcal{S}'$ are identical. Contradiction. □

A.3 Proof of Thm. 4.8

Theorem.

1. There is a satisfiable CTL[VPL] formula which does not have a finite model.
2. There is a satisfiable $\varphi \in$ CTL[DCFL] s.t. no pushdown system is a model of φ.
3. Every satisfiable CTL[VPL] formula has a model which is a visibly pushdown system.

We commit the first two cases to the full version. Here we prove part three, beginning with the following lemma.

Lemma A.1. *Every satisfiable* CTL[VPL] *formula has a model which is a visibly pushdown system.*

Proof. Beforehand, we harmonize the definitions of two kinds of automata, and of a push down system.

Let $\Sigma = (\Sigma_{\mathtt{c}}, \Sigma_{\mathtt{r}}, \Sigma_{\mathtt{i}})$ be a pushdown alphabet [2]. For the following three definitions, Q refers to a set of states, $q_0 \in Q$ to an initial state, Γ to a stack alphabet containing the bottom-of-stack symbol $\bot$, and $col : Q \to \mathbb{N}$ to a function coloring the states Q. Moreover, we implicitly use the standard [2, 22] notations of a configuration, and of a run on ω-words over Σ and on infinite trees over Σ, respectively. For simplicity, let T be the set $(Q \times \Sigma_{\mathtt{c}} \times (\Gamma \setminus \{\bot\} \times Q)^*) \cup (Q \times \Sigma_{\mathtt{i}} \times Q^*) \cup (Q \times \Sigma_{\mathtt{r}} \times \Gamma \times Q^*)$. We write $\langle (q_1, B_1), \ldots (q_n, B_n) \rangle$ for an element in $(\Gamma \setminus \{\bot\} \times Q)^*$. A *ordered visibly pushdown system* (oVPS) over Σ is a tuple $P = (Q, \Gamma, \delta, q_0)$ such that $\delta \subseteq T$ and δ is deterministic. An oVPS P induces an Σ-labeled and ordered tree by unrolling δ. A *parity tree automaton* over Σ is a tuple $\mathcal{A} = (Q, \delta, q_0, col)$ such that $\delta \subseteq Q \times \Sigma \times Q^*$. A *stair parity visibly pushdown tree automaton* [28] over Σ is a tuple $\mathcal{A} = (Q, \Gamma, \delta, q_0, col)$ such that $\delta \subseteq T$. Any such automaton is said to be *satisfiable* if there exists a tree which it accepts.

Given a stair parity VPTA $\mathcal{A}$, we construct an oVPS such that its induced tree is accepted by $\mathcal{A}$. As for the claim of Thm. 3, for any ΔPDL$^?$[VPA]- and any CTL[VPA]-formula φ there is a stair parity VPTA which accepts exactly the unique diamond path and unique Δ-path Hintikka tree models of φ [27, Lem. 24]. By the announced implication there exists a oVPS which admits such a Hintikka model for φ. From this, one obtains a VPS [28] satisfying φ, as just as one gets a tree model from a Hintikka model [27, Prop. 23].

Let $\mathcal{A} = (Q^{\mathcal{A}}, \Gamma, \delta^{\mathcal{A}}, q_0^{\mathcal{A}}, col^{\mathcal{A}})$ be a stair parity visibly pushdown tree automaton over a a pushdown alphabet $\Sigma = (\Sigma_{\mathtt{c}}, \Sigma_{\mathtt{r}}, \Sigma_{\mathtt{i}})$.

Definition A.2. *Wlog.* $col^{\mathcal{A}} \colon Q^{\mathcal{A}} \to \mathbb{N} \setminus \{0\}$*, and* $Q^{\mathcal{A}} = \{1, \ldots, |Q^{\mathcal{A}}|\}$*. The parity tree automaton* $\mathcal{B} := (Q^{\mathcal{B}}, \delta^{\mathcal{B}}, q_0^{\mathcal{B}}, col^{\mathcal{B}})$ *is defined as follows.*

- $Q^{\mathcal{B}} := \big(Q^{\mathcal{A}} \times \Gamma \times 2^{Q^{\mathcal{A}}}\big) \mathbin{\dot\cup} \{\checkmark\}$.
- $q_0^{\mathcal{B}} := (q_0^{\mathcal{A}}, \bot, \emptyset)$.
- $col^{\mathcal{B}}\big((q, \_, \_)\big) := col^{\mathcal{A}}(q)$ *for all* $q \in Q^{\mathcal{A}}$*, and* $col^{\mathcal{B}}(\checkmark) := 0$.

The relation $\delta^{\mathcal{B}}$ *is given by case distinction on* Σ*.*

Always: $(\checkmark, a, \langle \checkmark \rangle) \in \delta^{\mathcal{B}}$ *for all* $a \in \Sigma$*.*
For all $a \in \Sigma_{\mathtt{i}}$ **and** $(q, a, \langle q_1, \ldots, q_k \rangle) \in \delta^{\mathcal{A}}$**:** *Then* $((q, \gamma, R), a, \langle (q_1, \gamma, R), \ldots, (q_k, \gamma, R) \rangle) \in \delta^{\mathcal{B}}$ *for all* $\gamma \in \Gamma$*.*
For all $a \in \Sigma_{\mathtt{r}}$ **and** $(q, \gamma, a, \langle q_1, \ldots, q_k \rangle) \in \delta^{\mathcal{A}}$**:** *Then* $((q, \bot, R), a, \langle (q_1, \bot, R), \ldots, (q_k, \bot, R) \rangle) \in \delta^{\mathcal{B}}$*. And* $((q, \gamma, R), a, \langle \checkmark \rangle) \in \delta^{\mathcal{B}}$ *if* $q_i \in R$ *for all* $i = 1, \ldots, k$*.*
For all $a \in \Sigma_{\mathtt{c}}$ **and** $(q, a, \langle (\gamma_1, q_1), \ldots, (\gamma_k, q_k) \rangle) \in \delta^{\mathcal{A}}$**:** *Let* $R_1, \ldots, R_k \subseteq Q^{\mathcal{A}}$ *be arbitrary. Then* $((q, \gamma', R'), a, \langle \boldsymbol{w}_1 \ldots \boldsymbol{w}_k \rangle) \in \delta^{\mathcal{B}}$ *where* $\boldsymbol{w}_i$ *for* $i = 1, \ldots, k$ *is a vector over* $Q^{\mathcal{B}}$ *of length* $1 + |Q^{\mathcal{A}}|$*. Its first component is* (q_i, γ_i, R_i)*, followed by* (r, γ', R') *if* $r \in R_i$*, or by* $\checkmark$ *otherwise, for all* $r \in Q^{\mathcal{A}}$ *increasingly.*

Note that from any transition in $\mathcal{B}$ its generating transition in $\mathcal{A}$ can be reconstructed.

Lemma A.3. *If $\mathcal{A}$ is satisfiable then so $\mathcal{B}$ is.*

Proof. Suppose that $\mathcal{A}$ accepts a tree $t_{\mathcal{A}}$. Let $t'_{\mathcal{A}}$ be the tree $t_{\mathcal{A}}$ but additionally annotated with configurations of $\mathcal{A}$ witnessing that $t_{\mathcal{A}}$ is accepted by $\mathcal{A}$. Starting from the root, the tree $t'_{\mathcal{A}}$ is successively rearranged to a tree $t_{\mathcal{B}}$ accepted by $\mathcal{B}$. Let a node v be given. If at v the automaton $\mathcal{A}$ does an internal operation or a pop operation then this nodes remains. Now, assume that $\mathcal{A}$ does a push operation along v to a child w. Consider the occurrences of all pop operations corresponding to the push operation from v to w on all branches arising from w. Let R be the states reached by $\mathcal{A}$ as a result of the exhibited pop operation. Hence, for any $r \in R$ there is a subtree t_r below w annotated with the state r. For all $r \in Q^{\mathcal{A}}$, increasingly, the node v got the following subtree as a sibling. If $r \in R$ then we take t_r and otherwise some (infinite) tree. The new sibling are inserted right after v and a head of its siblings in the first place.

The construction ensures that the resulting tree is accepted by $\mathcal{B}$. Indeed, let π be a path starting in $q_0^{\mathcal{B}}$. If π touches $\checkmark$, it keeps doing so. Hence, the path is accepted. Otherwise, the path corresponds to a branch in $\mathcal{A}$ where the immediate run corresponding to a maximally matching word [2] are omitted. For each such word, a branch is forked, cf. the first component of the w_is in Def. A.2. Hence, π corresponds to a branch in $\mathcal{A}$. However, the positions of the maximally matching words are not taken into account for the acceptance condition. But, this restriction is just the stair parity condition. Hence, π is accepted.

Definition A.4. *Let $\mathcal{C}$ be a parity tree automaton over Σ with states Q and transitions δ. A triple (V, E, r, ℓ) is a* finite interpretation *for $\mathcal{C}$ iff V is a finite set of nodes, $E : V \to V^+$ is a successor function with ordered children, $r \in V$ is its root, and $\ell \colon V \to (Q \times \Sigma)$ is a labeling function which in conform with $\mathcal{C}$. That is, $E(v_0) = (v_1, \ldots, v_n)$ and $\ell(v_i) = (q_i, a_i)$ for all $i \in \{0, \ldots, n\}$ imply $(q_0, a_0, (a_1, \ldots, q_n)) \in \delta$, for any $v_0, \ldots, v_n \in V$, $q_0, \ldots, q_n \in Q$, and $a_0, \ldots, a_n \in \Sigma$. Such a finite interpretation is a* finite model *of $\mathcal{C}$ iff $\mathcal{C}$ accepts the tree resulting from unrolling (V, E, r, ℓ) at its root. The labels of this tree follow the Σ-part of ℓ.*

Theorem A.5. *Any satisfiable parity tree automaton has a finite model.*

Proof. The emptiness problem can be reduced to the question whether or not the automaton player has a winning strategy for a finite parity game [22]. The set of winning position is computable. Hence, fixing one outgoing edge of a position of the automaton player leads directly to the claimed graph.

Finally, the translation in Def. A.2 and the reduction in Lem. A.3 can be reversed.

Definition A.6. *Let $G = (V, E, r, \ell)$ a finite model of $\mathcal{B}$. Then G induces an oVPS $P := (V, \Gamma^P, \delta^P, r)$, where the stack alphabet Γ^P is $(Q \to V) \,\dot\cup\, \{\bot\}$. The transition relation δ^P is given as follows. Let $v \in V$ be labeled with $(\_, a) \in Q \times \Sigma$. For any $a \in \Sigma_{\mathtt{i}}$, δ^P contains $(v, a, E(v))$. And for any $a \in \Sigma_{\mathtt{r}}$, δ^P contains $(v, a, \bot, E(v))$ and $(v, a, \rho, \rho(v))$ for any function $\rho : Q \to V$. As for the push operations, let $E(v) = \boldsymbol{v}_1 \ldots \boldsymbol{v}_k$ and let $\boldsymbol{v}_i = v_{i,0}, \ldots, v_{i,|Q|}$ for each i, due to the conformity of G with $\mathcal{B}$. Then δ^p contains $(v, a, ((\rho_1, v_{1,0}), \ldots, (\rho_k, v_{k,0})))$ where $\rho_i : Q \to V$ is some (fixed) function such that $\rho_i(q) = v_{i,q}$ if the Σ-part of $\ell(v_{i,q})$ is not $\checkmark$.*

Because, in the tree resulting from unrolling G, no rooted branch reaches the state ✓, transitions leaving this state need not be translated.

Theorem A.7. *Let G be a finite model of $\mathcal{B}$. Then the oVPS P is a model of $\mathcal{A}$.*

Proof. In the unrolled tree of P, any maximal path π which starts at the root is infinite, following the labeling function. Analogously to the proof of Lem. A.3, such a path meets the stair parity condition. Indeed, it suffices to consider the interrupted path which skips the minimally matching words in the factorization of (the word labeling) π. Such an interrupted path corresponds to a path in G meeting the parity condition of $\mathcal{B}$. Hence, π fulfills the stair parity condition for $\mathcal{A}$.

As for the underdetermination of the functions ρ_i in the case $\Sigma_{\mathtt{c}}$: if the Σ-part of $\ell(v_{i,q})$ is ✓, the value of $\rho_i(q)$ is irrelevant as the function will be never evaluated at q—as long as only rooted paths are considered. This is ensured by the condition "$q_i \in R$" in the case $\Sigma_{\mathtt{r}}$ of Def. A.2 and by the conformity of G with $\mathcal{B}$.

This completes the proof of Lemma A.1 and therefore Thm. 4.8 Part 3. □

B Proofs omitted in Section 5

B.1 Proof of Thm. 5.3

Theorem. The following items hold.

1. CTL[DFA, NFA] satisfiability is hard for 2EXPTIME.
2. CTL[DVPA, NFA] satisfiability is hard for 3EXPTIME.

The reduction uses the alternating tiling problem.

Definition B.1. *The* alternating tiling problem *is the following. Given a set T of tiles, $H, V \subseteq T^2$, $s \in T$, $f \colon \mathbb{N} \to \mathbb{N}$, and $\alpha \colon T \to \{0,1,2\}$ such that $H \subseteq \{(t,t') \mid \alpha(t) = \alpha(t')\}$ decide whether there is a* tiling tree*. That is, a finite tree such that*

- *any node is labeled with $t_1, \ldots, t_m$ for $m{:=}f(|T|)$,*
- *$t_1 = s$ for the root,*
- *$t_i H t_{i+1}$ for all $1 \leq i < m$,*
- *the node has $\alpha(t_m)$ successors, and*
- *for each successor labeled with $t'_1, \ldots, t'_m$ holds $t_i V t'_i$ for all $1 \leq i \leq m$.*

The function α realizes alternation. Note that, if the range of α is $\{0,1\}$ the definition corresponds the usual one version for one player [34]. Therefore, we refer to a node in a tiling tree as a *row* and to its components as *columns*. So, H represent the horizontal and V the vertical matching relation.

To describe the complexity of alternating tiling we assume a reasonable encoding of T, H et cetera. In particular, the function f is given as a term. As we want to characterize complexity classes far beyond EXPTIME the usual corridor tiling [34] does not suffice because an explicit naming of the width would require to much space.

Combining the technique of tiling and alternation [8], we obtain the following characterization.

Lemma B.2. *The class of alternating tiling problems where their functions f is exponential is 2EXPTIME-complete. Similar, the restriction to doubly exponential functions is complete for 3EXPTIME.*

In Def. B.1, the restriction on H with respect to α is not necessary for the completeness for the respective completity class. However, it simplifies that subsequent hardness proof for CTL[DFA,NFA].

Proof (of Thm. 5.3(1)). Given an alternating tiling problem consisting of T, H, V, s, f and α as in Def. B.1 such that f is exponential. Set $n{:=}|T|$, $m{:=}f(n)$ and let m' be the number of bits to count from 0 to $m-1$, that is $n'{:=}\lfloor \log_2(m-1)\rfloor + 1$. Note that n' is polynomially bounded in n. W.l.o.g. $T = \{1, \dots, n\}$.

It is pretty easy to find a CTL-formula φ such that any of its models looks like an tiling tree (up to bisimulation). Thereto, the tiles are encoded by propositions, say $t_1, \dots, t_n$. Any sequent of tiles in a node of the tree is represented by a chain of nodes in the model of the respective length. The length is ensured by a binary counter with n' bits. In (pure) CTL all properties can specified except for the constraint on V. Therefore, the formula would need to look about m steps into the future while have a size polynomial in n.

The V-constraint refers only to any those two immediately consecutive positions on which the counter has the same value. To bridge between those two positions, a proof obligation is created by an $\mathtt{AU}^{\mathcal{A}}$-subformula. The key idea is that for the correctness we can replace $\mathcal{A}$ by the deterministic automaton obtained from the standard powerset-construction [30]. In other words, we are allowed to construct an exponentially sized automaton but which has a small description. The mentioned obligation reflects the value of the counter and the expected tile at the second position. However, its creating requires that the outgoing edge is replaced by a chain of edges. Each edge copies another bit from the counter to the proof obligation. As long as the nodes of the model represent the same row, the programmed proof obligation are not armed, that is, they can not reach any final state. The change to the next row arms the obligations. Along the path to the second position, at every tile position an appendix in the model checks every proof obligation. If the current value of the counter does not match the stored value in the obligation the model ensures that the obligation is satisfied trivially. Otherwise, the (only remaining) obligation matches the chosen tile with the expected tile. Finally at every second change of the row, the model disposes of the proof obligations.

Formally, we will construct a formula φ over the alphabet

$$\Sigma{:=}\{\mathtt{nextCol}, \mathtt{nextRow}, \mathtt{ifNeq}, \mathtt{then}, \mathtt{else}\} \cup \Gamma$$

where $\Gamma{:=}\{\mathtt{bit}_i^b \mid i \in [n], b \in \mathbb{B}\}$. As boolean values we use 0 and 1. The label `nextCol` separates two columns in the same row, and `nextRow` indicates a new node in the tiling tree. The set Γ is used to program the proof obligations, which are verified with help of `ifNeq`, `then` and `else`. Besides the already mentioned propositions $t_1, \dots, t_n$ for tiles, we use $c_1, \dots, c_{n'} {\equiv:} \boldsymbol{c}$ as an n' bit counter ranging from 0 to $m-1$. Arithmetical operations involving this counter are described informally in quotes because these only plays a minor role. However, these operations have short encodings as CTL-formulas, that is, their size is polynomially bounded in n'. Additionally, the proposition $\mathtt{dir}$ is used to force two sons whenever α gets two.

Define $p^0{:=}\neg p$ and $p^1{:=}p$ for any proposition p. For a label $a \in \Sigma$ and a CTL-formula ψ, $!\mathtt{X}^a\psi{:=}\mathtt{EX}^a\mathtt{tt} \wedge \mathtt{AX}^a\psi$ denotes that there is at least one a-successor and ψ hold at these successors. Moreover, instead of automata we also use regular expressions as annotations to CTL-formulas.

The tiling problem is translated into the formula

$$\varphi \ := \ \text{“}\boldsymbol{c} = 0\text{”} \wedge \mathtt{AG}^{\{\varepsilon\} \cup \Sigma^* \{\mathtt{nextCol},\mathtt{nextRow}\}}\psi$$

where ψ is the conjunction of the following lines and the automaton $\mathcal{A}$ is depicted in Fig. 3.

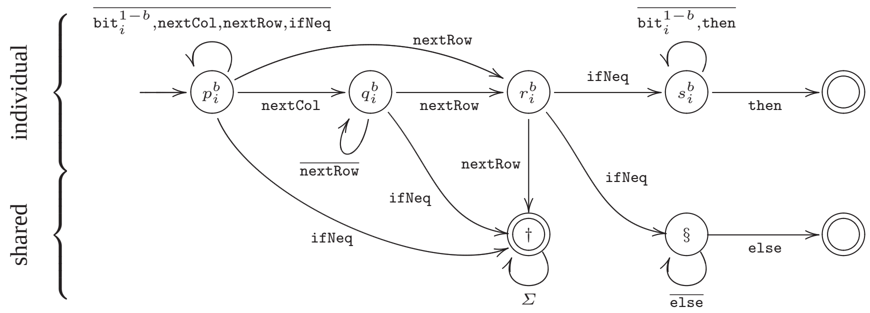

individual
shared
$\overline{\mathtt{bit}_i^{1-b},\mathtt{nextCol},\mathtt{nextRow},\mathtt{ifNeq}}$
nextRow
$\overline{\mathtt{bit}_i^{1-b},\mathtt{then}}$
p_i^b
nextCol
q_i^b
nextRow
r_i^b
ifNeq
s_i^b
then
$\overline{\mathtt{nextRow}}$
nextRow
ifNeq
ifNeq
ifNeq
†
§
else
Σ
$\overline{\mathtt{else}}$

Fig. 3. Automaton $\mathcal{A}$. Overlined labels mean their complement with respect to Σ. The individual part is present for any $i \in [n]$ and for any $b \in \mathbb{B}$. So, it has $10n + 3$ states where $2n$ are initial ones.

$$\bigvee_{i\in[n]} t_i \wedge \bigwedge_{j\in[n]\setminus\{i\}} \neg t_j \tag{1}$$

$$\bigwedge_{i\in[n']} \bigwedge_{b\in\mathbb{B}} c_i^b \to \mathtt{AX}^{\Gamma^{i-1}}\mathtt{EX}^{\mathtt{bit}_i^b}\mathtt{tt} \tag{2}$$

$$\mathtt{EX}^{\mathtt{ifNeq}}\mathtt{tt} \tag{3}$$

$$\bigwedge_{i\in[n']} \bigwedge_{b\in\mathbb{B}} c_i^b \to \mathtt{AX}^{\mathtt{ifNeq}\ \Gamma^{i-1}}\mathtt{EX}^{\mathtt{bit}_i^b}\mathtt{tt} \tag{4}$$

$$\mathtt{AX}^{\mathtt{ifNeq}\ \Gamma^{n'}}!\mathtt{X}^{\mathtt{then}}\mathtt{dispose} \tag{5}$$

$$\mathtt{AX}^{\mathtt{ifNeq}\ \Gamma^{n'}\mathtt{then}}!\mathtt{X}^{\mathtt{else}}(\neg\mathtt{dispose} \wedge \mathtt{AX}\mathtt{ff}) \tag{6}$$

$$\bigwedge_{i\in[n]} (\mathtt{AX}^{\mathtt{ifNeq}\ \Gamma^{n'}\mathtt{then}\ \mathtt{else}}t_i) \leftrightarrow t_i \tag{7}$$

$$\bigwedge_{i\in[n],\alpha(i)\neq 0} t_i \to \mathtt{AF}^{\mathcal{A}}\left(\bigvee_{j\in[n],iVj} t_j \vee \mathtt{dispose}\right) \tag{8}$$

$$\text{“}\boldsymbol{c} < m-1\text{”} \to \bigvee_{i,j\in[n],iHj} t_i \wedge \mathtt{AX}^{\Gamma^{n'}} \mathtt{!X}^{\mathtt{nextCol}} t_j \tag{9}$$

$$\text{“}\boldsymbol{c} < m-1\text{”} \to \text{“}\mathtt{AX}^{\Gamma^{n'}\,\mathtt{nextCol}} \boldsymbol{c} = \boldsymbol{c}+1\text{”} \tag{10}$$

$$\left(\text{“}\boldsymbol{c} = m-1\text{”} \wedge \bigvee_{i\in[n],\alpha(i)>0} t_i\right) \to \mathtt{AX}^{\Gamma^{n'}} \mathtt{!X}^{\mathtt{nextRow}}(\mathtt{dispose} \wedge \text{“}\boldsymbol{c} = 0\text{”}) \tag{11}$$

$$\left(\text{“}\boldsymbol{c} = m-1\text{”} \wedge \bigvee_{i\in[n],\alpha(i)=2} t_i\right) \to \bigwedge_{b\in\mathbb{B}} \mathtt{AX}^{\Gamma^{n'}} \mathtt{EX}^{\mathtt{nextRow}} \mathtt{dir}^{b} \tag{12}$$

$$\left(\text{“}\boldsymbol{c} = m-1\text{”} \wedge \bigvee_{i\in[n],\alpha(i)=0} t_i\right) \to \mathtt{EX}^{\mathtt{ifNeq}\ \mathtt{else}} \mathtt{dispose} \tag{13}$$

The formula φ is obviously a CTL[DFA,NFA]-formula and its size is polynomially bounded in n.

The formula (1) ensures that exactly one tile is chosen, (2) programs the proof obligation (for the V-constraint) generated by (8). The verification is performed by (3)–(7). The formulas (9)–(12) ensure that the columns of a node in the tiling tree are enumerated, and that the tree is branching with respect to α. The formula (13) is the counterpart to (9) and just ensures that proof obligation at the leaves are satisfied. (Alternatively, (2)–(7) could be excluded for the very last column.)

If we neglect the V-constraint, the reduction is sound and complete. As for the V-constraint, we describe the life of a proof obligation on a tree model of φ. An excerpt is given in Fig. 4.

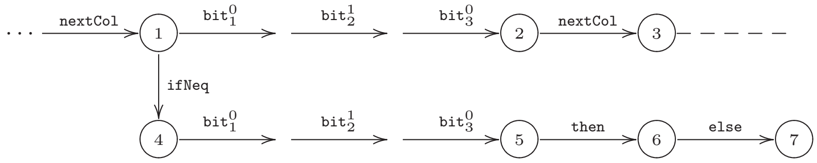


Fig. 4. Excerpt of a model for φ. This part depicts a single column which is neither the first nor the last one of a row. The second line shows the appendix which verifies the proof obligation for the V-constraint. At the node 1 the formulas t_7, $\neg c_3^1$, c_2 and $\neg c_1$ shall hold, at the node 6 the proposition $\mathtt{dispose}$, and at node 7 the proposition t_7.

Let Q be the set of states of $\mathcal{A}$. If we say that there is a proof obligation in a certain state $Q' \subseteq Q$, we refer to the deterministic substitute of $\mathcal{A}$ obtained from the powerset construction. Beginning at the node 1, the formula (8) admits a proof obligation for $t_j \vee \mathtt{dispose}$ (for some $j \in [n]$) in the state $\{p_i^b \mid i \in [n], b \in \mathbb{B}\}$. The intended trace is the first line in Fig. 4. After passing the label `nextRow` the automaton reaches the state $\{q_i^b \mid i \in [n], b \in \mathbb{B}, 1 \models c_i^b\}$, that is, the state reflect the content of the counter at node 1. As for the second line, the proof obligation vanishes because `dispose` holds at the node 6. Moreover, the obligation remains while passing another columns of the same row. Changing the row for the first time, the obligation changes to $\{r_i^b \mid i \in [n], b \in \mathbb{B}, 1 \models c_i^b\}$ where the node 1 refers to the node which admits the proof obligation. As long as we follow the first line, the state remains until we change the row for the second time. This brings the obligation in the state $\{\dagger\}$. The formulas 5 and 13 offers a node with models `dispose` and ensure that the proof obligation disappears. Note that after the first change of the row there is also a node modelling `dispose`. But the state of the obligation does not contain a final state of $\mathcal{A}$ at this time.

Now, we consider a proof obligation in the second line after passing `nextCol` for the first time. The label `ifNeq` switches the state to $\{\S, s_i^b \mid i \in [n], b \in \mathbb{B}, 1 \models c_i^b\}$. Again the node 1 refers to the node which admits the proof obligation. At node 5 the obligation either reaches the state $\{\S\}$ or some proper super set. The second case can only happen if the programmed counter and the counter of the current column differ. In this case, the formula (5) disposes the obligation. Otherwise, the state of the obligation does not contain a final state when reaching the node 6. By (6) and (7), the tile t_j—as represented by the obligation—must be the tile of the current column. □

C Proofs omitted in Section 6

C.1 Proof of Thm. 6.3

Theorem. Model checking visibly pushdown automata against CTL[VPA,DVPA] is in EXPTIME, and CTL[VPA,VPA] is in 2EXPTIME.

We split the proof into separate lemmas. For VPA rules we use the notation $(q, \gamma, a, push(b), q')$, $(q, \gamma, a, rew(b), q')$ and (q, γ, a, pop, q'), and omit the input character γ for PDS rules.

Lemma C.1. *Model checking* CTL[VPA,DVPA] *over visibly pushdown automata is in* EXPTIME*.*

Proof. We reduce the model checking problem for CTL[VPA, DVPA] over VPA to a Büchi game over a PDS. Since deciding the winner in such a game is EXPTIME [38], we obtain an EXPTIME algorithm for the model checking problem.

Without loss of generality, we assume all VPA have a bottom of stack symbol that is neither popped nor pushed and are complete. We also assume all formulas are in positive normal form.

The game has the following transitions. The state set and alphabet is defined implicitly. We begin with some standard formula to game translation. The alphabet becomes a set of pairs, (a, b). The first component corresponds to the model VPA, the second to the formula VPA being evaluated. All states annotated $begin$ are controlled by the existential player. The universal positions are $(s, \varphi_1 \wedge \varphi_2)$. The following rules are for all characters a and b.

- $(win, (a, b), rew((a, b)), win)$.

- $((s,p)^{begin},(a,b),rew((a,b)),win)$ if s satisfies the atomic proposition p.
- $((s,\neg p)^{begin},(a,b),rew((a,b)),win)$ if s does not satisfy the atomic proposition p.
- $((s,\varphi_1 \vee \varphi_2)^{begin},(a,b),rew((a,b)),(s,\varphi_i)^{begin})$ for $i \in \{1,2\}$.
- $((s,\varphi_1 \wedge \varphi_2)^{begin},(a,b),rew((a,b)),(s,\varphi_1 \wedge \varphi_2))$.
- $((s,\varphi_1 \wedge \varphi_2),(a,b),rew((a,b)),(s,\varphi_i)^{begin})$ for $i \in \{1,2\}$.

For path formulas, we form a product with the VPA labelling the formula. We begin by adding a bottom of stack symbol to the stack in the formula VPA's component. For $\mathtt{E}(\varphi_1 \mathtt{U}^A \varphi_2)$ we allow the existential player to decide whether to complete the until formula or postpone completion until later. When postponing, the opponent can check whether the until will eventually be completed, or whether the condition on the until holds. When progressing the game, the existential player is able to choose both the move of the formula VPA and the model VPA. The existential positions are $(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2))$ and $(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),move)$. The universal positions are $(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait)$.

- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^A \varphi_2))^{begin},(a,b),rew((a,\perp)),(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_{q_0^A}} \varphi_2)))$.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2)),(a,b),rew((a,b)),(s,\varphi_2)^{begin})$ for all a,b and q is accepting.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2)),(a,b),rew((a,b)),(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait))$ for all a,b.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait),(a,b),rew(a),(s,\varphi_1)^{begin})$ for all a,b.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait),(a,b),rew((a,b)),(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),move))$ for all a,b.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),move),(a,b),push((a',b')),(s',\mathtt{E}(\varphi_1 \mathtt{U}^{A_{q'}} \varphi_2)))$ whenever we have the rules $(s,\gamma,a,push(a'),s')$ and $(q,\gamma,b,push(b'),q')$.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),move),(a,b),rew((a',b')),(s',\mathtt{E}(\varphi_1 \mathtt{U}^{A_{q'}} \varphi_2)))$ whenever there is $(s,\gamma,a,rew(a'),s')$ and $(q,\gamma,b,rew(b'),q')$.
- $((s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),move),(a,b),pop,(s',\mathtt{E}(\varphi_1 \mathtt{U}^{A_{q'}} \varphi_2)))$ whenever (s,γ,a,pop,s') and (q,γ,b,pop,q').

The remaining path formulas are similar, but the roles of the players are altered accordingly. In the case $\mathtt{A}(\varphi_1 \mathtt{U}^A \varphi_2)$, when satisfaction is postponed, since the property must hold for all paths, first the opponent picks a transition of the model, then the existential player picks a move in A. The existential positions are $(s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2))$ and $(s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),t_s)$. The universal positions are $(s,\mathtt{E}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait)$. Note that $\mathtt{A}(\varphi_1 \mathtt{U}^A \varphi_2)$ is an abbreviation for a $\neg\mathtt{E}(\neg\varphi_1 \mathtt{R}^A \neg\varphi_2)$. Due to the discussion in Section 2, correctness of the reduction relies on A being deterministic.

- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^A \varphi_2))^{begin},(a,b),rew((a,\perp)),(s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_{q_0^A}} \varphi_2)))$.
- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2)),(a,b),rew((a,b)),(s,\varphi_2)^{begin})$ and q is accepting.
- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2)),(a,b),rew((a,b)),(s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait))$.
- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait),(a,b),rew((a,b)),(s,\varphi_1)^{begin})$.
- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),wait),(a,b),rew((a,b)),(s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),t_s))$ where t_s is a transition from s,a.
- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),t_s),(a,b),push((a',b')),(s',\mathtt{A}(\varphi_1 \mathtt{U}^{A_{q'}} \varphi_2)))$ whenever we have $t_s = (s,\gamma,a,push(a'),s')$ and $(q,\gamma,b,push(b'),q')$.
- $((s,\mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2),t_s),(a,b),rew((a',b')),(s',\mathtt{A}(\varphi_1 \mathtt{U}^{A_{q'}} \varphi_2)))$ whenever we have $t_s = (s,\gamma,a,rew(a'),s')$ and $(q,\gamma,b,rew(b'),q')$.

- $((s, \mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2), t_s), (a, b), pop, (s', \mathtt{A}(\varphi_1 \mathtt{U}^{A_{q'}} \varphi_2)))$ whenever $t_s = (s, \gamma, a, pop, s')$ and (q, γ, b, pop, q').

The release operators are defined analogously. We begin with $\mathtt{E}(\varphi_1 \mathtt{R}^{A} \varphi_2)$. The existential positions are $(s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2))$ and $(s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), move)$. The universal positions are $(s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait)$ and $(s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), t_s)$. Here we also rely on the fact that the VPA in the formulas are deterministic.

- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A} \varphi_2))^{begin}, (a, b), rew((a, \bot)), (s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_{q_0^A}} \varphi_2)))$.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2)), (a, b), rew((a, b)), (s, \varphi_1)^{begin})$.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2)), (a, b), rew((a, b)), (s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait))$.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait), (a, b), rew((a, b)), (s, \varphi_1)^{begin})$ where q is accepting.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait), (a, b), rew((a, b)), (s, \mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2), move))$.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), move), (a, b), rew((a, b)), (s, \mathtt{A}(\varphi_1 \mathtt{U}^{A_q} \varphi_2), t_s))$ where t_s is a transition from s, a.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), t_s), (a, b), push((a', b')), (s', \mathtt{E}(\varphi_1 \mathtt{R}^{A_{q'}} \varphi_2)))$ whenever we have $t_s = (s, \gamma, a, push(a'), s')$ and $(q, \gamma, b, push(b'), q')$.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), t_s), (a, b), rew((a', b')), (s', \mathtt{E}(\varphi_1 \mathtt{R}^{A_{q'}} \varphi_2)))$ whenever we have $t_s = (s, \gamma, a, rew(a'), s')$ and $(q, \gamma, b, rew(b'), q')$.
- $((s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), t_s), (a, b), pop, (s', \mathtt{E}(\varphi_1 \mathtt{R}^{A_{q'}} \varphi_2)))$ whenever $t_s = (s, \gamma, a, pop, s')$ (q, γ, b, pop, q').

And finally, $\mathtt{A}(\varphi_1 \mathtt{R}^{A} \varphi_2)$. The existential positions are $(s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2))$. The universal positions are $(s, \mathtt{E}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait)$.

- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A} \varphi_2))^{begin}, (a, b), rew((a, \bot)), (s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_{q_0^A}} \varphi_2)))$.
- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2)), (a, b), rew((a, b)), (s, \varphi_1)^{begin})$.
- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2)), (a, b), rew((a, b)), (s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait))$.
- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait), (a, b), rew(a), (s, \varphi_2)^{begin})$ where q is accepting.
- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait), (a, b), push((a', b')), (s', \mathtt{A}(\varphi_1 \mathtt{R}^{A_{q'}} \varphi_2)))$ whenever we have $(s, \gamma, a, push(a'), s')$ and $(q, \gamma, b, push(b'), q')$.
- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait), (a, b), rew((a', b')), (s', \mathtt{A}(\varphi_1 \mathtt{R}^{A_{q'}} \varphi_2)))$ whenever we have $(s, \gamma, a, rew(a'), s')$ and $(q, \gamma, b, rew(b'), q')$.
- $((s, \mathtt{A}(\varphi_1 \mathtt{R}^{A_q} \varphi_2), wait), (a, b), pop, (s', \mathtt{A}(\varphi_1 \mathtt{R}^{A_{q'}} \varphi_2)))$ whenever (s, γ, a, pop, s') and (q, γ, b, pop, q').

The game has a Büchi winning condition. All states are accepting except for states containing an U operator. Since these formulas must always eventually be satisfied, they are not accepting. Since we assume all VPA are complete, play will only get stuck when a literal is not satisfied, in which case the existential player will lose.

Given a CTL[VPA] formula φ and a VPA B, we can check whether B satisfies φ by asking whether the existential player wins the game described above from the control state (s_0, φ^{begin}) with the initial stack contents. Such games can be solved in EXPTIME [38]. □

Lemma C.2. *Model checking* CTL[VPA,VPA] *over visibly pushdown automata is in 2EXPTIME.*

Proof. The proof follows from the exponential cost of determinising the VPA, and Lemma C.1. □