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 $\mathcal{L}_\mu$ [22]. This transition comes at a price. For CTL* the model checking problem increases to PSPACE-complete [30] and satisfiability to 2EXPTIME-complete [14, 33]. 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 [25].

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 [37, 34, 23, 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, 27]. 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 [26]. 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, 31]. 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 [35] 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.

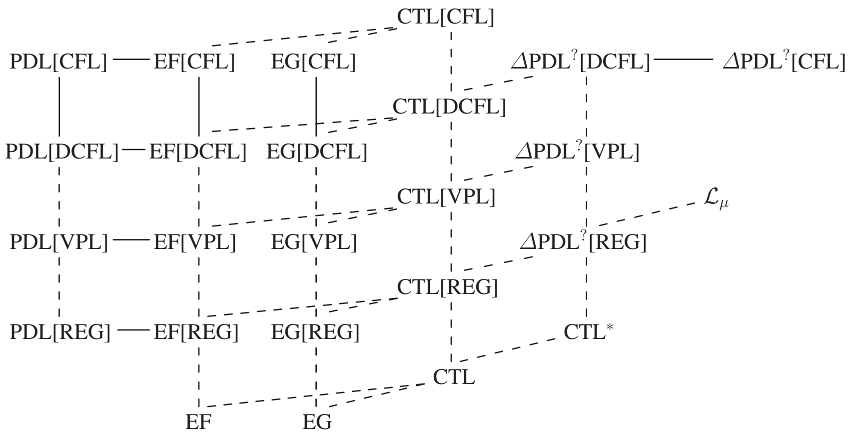

Fig. 1. The expressive power of Extended Computation Tree Logic.

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 [26]. 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 ΔPDL$^?$[$\mathfrak{A} \cup \mathfrak{B}$] 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 ΔPDL$^?$[REG] is in EXPTIME [15] and for ΔPDL$^?$[VPL] is in 2EXPTIME [26]. □

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] [26] 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 $\texttt{AU}^{\mathcal{A}}$ for a NFA $\mathcal{A}$. In a tree model, we say that a node has a *proof obligation* for an $\texttt{AU}$-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 [28]. 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 $(\texttt{bit}_i^{b_i})_{1\leq i\leq n}$ 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 $\texttt{bit}_i^{\neg b}$. 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 $\texttt{AU}^{\mathcal{A}}\texttt{AXff}$ or $\texttt{AG}^{\mathcal{A}}\texttt{ff}$ 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 $\ell'(b)=\mathsf{dead}$ 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 [29] 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 [32] 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 [32]. 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 2EXPTIME.*

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 [36], 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 2EXPTIME, 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* [24]. 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. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

23. 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.
24. L. Bozzelli. Complexity results on branching-time pushdown model checking. *Theor. Comput. Sci.*, 379(1-2):286–297, 2007.
25. 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.
26. C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *J. Log. Algebr. Program.*, 73(1-2):51–69, 2007.
27. 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.
28. M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal*, 2(3):115–125, 1959.
29. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.
30. 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.
31. R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982.
32. 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.
33. 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.
34. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
35. 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.
36. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.
37. 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.