Temporal Logic with Recursion

Florian Bruse
School of Electrical Engineering and Computer Science, University of Kassel, Germany
florian.bruse@uni-kassel.de

Martin Lange
School of Electrical Engineering and Computer Science, University of Kassel, Germany
martin.lange@uni-kassel.de

Abstract

We introduce extensions of the standard temporal logics CTL and LTL with a recursion operator that takes propositional arguments. Unlike other proposals for modal fixpoint logics of high expressive power, we obtain logics that retain some of the appealing pragmatic advantages of CTL and LTL, yet have expressive power beyond that of the modal μ-calculus or MSO. We advocate these logics by showing how the recursion operator can be used to express interesting non-regular properties. We also study decidability and complexity issues of the standard decision problems.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Program specifications

Keywords and phrases formal specification, temporal logic, expressive power

Digital Object Identifier 10.4230/LIPIcs.TIME.2020.6

1 Introduction

Temporal logic is a well-established formalism for the specification of the behaviour of dynamic systems, typically separated into two classes: linear-time vs. branching-time, reflecting the philosophical question of whether the future is determined of not [31]. There is a direct correspondence in computer science: the linear-time view regards programs as being standalone with input given at the beginning and a computation running without further interaction with the program's environment; in the branching-time view programs may be reactive, i.e. able to react to input as it occurs during a computation. The most prominent member of the linear-time family is LTL [29], the most prominent members of the branching-time family are CTL [8] and CTL* [10].

The classification into linear-time and branching-time typically has consequences with regards to expressiveness and computational complexity of the two major decision problems: satisfiability and model checking. For genuine linear-time logics, these two are closely related as model checking is a generalisation of validity checking, and validity checking can express model checking of finite-state systems. For LTL, these problems are PSPACE-complete [30]. The picture for branching-time logics is different: here, model checking is typically easier than satisfiability checking, for instance P- vs. EXPTIME-complete for CTL [9] and PSPACE- vs. 2EXPTIME-complete for CTL* [10, 11, 33].

Various extensions of these logics have been investigated for purposes of higher expressiveness: there are "semantic" extensions like action-based [21], dynamic logic [12], real-time [1], metric [18] and probabilistic temporal logics [13] for instance, as well as combinations thereof. Then there are "syntactic" extensions like the modal μ-calculus ($\mathcal{L}_\mu$) [19] with its explicit least and greatest fixpoint operators. It extends the expressive power to full regularity, i.e. that of Monadic Second-Order Logic [16] (up to bisimilarity).

Extensions in expressive power beyond that are possible, and sometimes even necessary for particular purposes. For instance, $\mathcal{L}_\mu$– and therefore temporal logics embeddable into it – have the finite model property [20]. Hence, they cannot be used to reason about inherent properties of infinite-state systems. Such an observation has led to the investigation of Propositional Dynamic Logic of Non-Regular Programs (PDL[CFL]) [15] for instance. It uses context-free instead of regular languages as in ordinary PDL and, thus, can express some non-regular properties but not all regular ones. When restricted to visibly pushdown languages (PDL[VPL]), it even becomes decidable [26]. Other extensions include Fixpoint Logic with Chop (FLC) [28], integrating process-algebraic operations into the modal μ-calculus, or Higher-Order Fixpoint Logic (HFL) [34] which incorporates a simply-typed λ-calculus into $\mathcal{L}_\mu$.

High expressive power also comes at a high price in two regards. First, it is tightly linked to high computational complexity including undecidability. In fact, the satisfiability checking problems for all the logics mentioned above here, capable of expressing non-regular properties, are (highly) undecidable. The second downside concerns pragmatics. Already $\mathcal{L}_\mu$ is commonly seen as unsuitable for a non-expert as writing temporal properties using least and greatest fixpoints is cumbersome and error-prone. This holds even more so for extensions like FLC and HFL.

We introduce extensions of LTL and CTL in order to make the formal specification of non-regular properties more widely available through temporal logics with a more intuitive syntax. We introduce a recursion operator, resulting the logics Recursive LTL, resp. CTL (RecLTL, RecCTL). The standard temporal operators from LTL and CTL are preserved even though the recursion operator is expressive enough to mimic these. Note that temporal operators like Until can be seen as abbreviations of infinite Boolean combinations of basic propositional and modal formulas. The recursion operator can be used to construct more complex infinite Boolean connections and, thus, express interesting properties, including non-regular ones. Semantically, it is defined via least fixpoints of monotone functions of order 1 over the powerset lattice of the underlying labelled transition systems. The model-theoretic design of RecLTL and RecCTL is inspired by the machinery underlying a complex logic like HFL, yet their pragmatics aims at more understandable and usable temporal specification languages.

In Sect. 2 we introduce RecCTL and RecLTL formally but also try to build intuition about what they can be used for and how to use them. In Sect. 3 we study the expressive power of RecCTL and RecLTL formally by placing them into the hierarchy of the those logics mentioned above. In Sect. 4 we show that model checking RecCTL is EXPTIME-complete whereas model checking RecLTL and satisfiability checking for both is undecidable. We also present a decidable fragment of RecCTL. The paper concludes with remarks on further work in Sec. 5.

2 Designing Temporal Logics of Higher Expressiveness

We assume familiarity with the standard temporal logics CTL and LTL.

Labelled Transition Systems. Let $\mathcal{P}$ be a finite set of atomic propositions. A labeled transition system (LTS) is a tuple $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$ where $\mathcal{S}$ is a (potentially infinite) set of *states*, $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is the transition relation that is assumed to be total in the sense that for every $s \in \mathcal{S}$ there is a $t \in \mathcal{S}$ s.t. $s \rightarrow t$. Finally, $\ell : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ labels each state with the set of propositions that hold at this state. A *path* is an infinite sequence $\pi = s_0, s_1, \ldots$ s.t. $s_i \rightarrow s_{i+1}$ for all $i \geq 0$. We write $\pi(i)$ to denote the ith state on this path.

Infinitary modal logic. Infinitary modal logic adds infinitary junctors $\bigwedge_{i \in I}$, resp. $\bigvee_{i \in I}$ for arbitrary sets I to modal logic with the operators $\Diamond$ and $\Box$ or, $\mathtt{EX}$ and $\mathtt{AX}$ as they are usually written in the temporal setting. Hence, formulas can become infinitely wide, but every path in the syntax tree is still of finite length.

Over any set of LTS, any standard propositional temporal logic can be translated into infinitary modal logic. For instance, the CTL formula $\mathtt{EF}q$ expressing reachability of a q-state is equivalent to $\varphi_1 := \bigvee_{i \geq 0} \mathtt{EX}^i q$, even uniformly over the class of all LTS.

Not every infinitary modal formula corresponds to a (finite) temporal formula, though. For instance, $\varphi_2 := \bigvee_{i \geq 0} \mathtt{AX}^i q$ is not expressible in CTL, or even the modal μ-calculus [7].

Patterns of infinitary formulas expressible in temporal logics. The reason for the fact that φ_1 is expressible in CTL but φ_2 is not, is given by the interplay of two principles:

First, the operator $\mathtt{EX}$ commutes with disjunctions – $\mathtt{EX}\varphi \vee \mathtt{EX}\psi \equiv \mathtt{EX}(\varphi \vee \psi)$ – but $\mathtt{AX}$ does not. Hence, we have

$$\varphi_1 = \bigvee_{i \geq 0} \mathtt{EX}^i q = q \vee \bigvee_{i \geq 1} \mathtt{EX}^i q \equiv q \vee \mathtt{EX} \bigvee_{i \geq 0} \mathtt{EX}^i q = q \vee \mathtt{EX}\varphi_1 \ . \tag{1}$$

Second, the fixpoint operators in CTL, LTL and $\mathcal{L}_\mu$ are propositional, i.e. (monadic) second-order. In other words, they can only be used to define a least or greatest fixpoint recursion over an operator that is a modal formula itself (disregarding nested fixpoints for the moment). Eq. 1 shows that $\varphi_1 \equiv \mu Z.q \vee \mathtt{EX}\, Z$ in $\mathcal{L}_\mu$ terms with $q \vee \mathtt{EX}\, Z$ for a propositional variable Z describing the evaluation in each iteration of the fixpoint recursion.

Now consider φ_2 again. It is simply not possible to rewrite it in a way like φ_1, obtaining an $\mathcal{L}_\mu$ formula because $\mathtt{AX}\varphi \vee \mathtt{AX}\psi \not\equiv \mathtt{AX}(\varphi \vee \psi)$. Most importantly, $\varphi_2 \not\equiv \mu Z.q \vee \mathtt{AX}\, Z$.

This is not to say that the infinitary modal formula representing φ_2 could not be built up in a recursive way using fixpoints, as it is equivalent to the FLC formula $(\mu Z.\tau \vee (Z;\mathtt{AX}));q$. The syntax is not easily understood – see the literature on FLC for a detailed introduction [28, 23] – especially with $\mathtt{AX}$ becoming a 0-ary operator. The main point to observe here, though, is the structure of the formula $\tau \vee (Z;\mathtt{AX})$ defining the fixpoint iteration using a recursion anchor Z. It is, in a sense, *left*-linear as opposed to the *right*-linear recursion schemes definable in $\mathcal{L}_\mu$.

Adding recursion to temporal logics. The aim of this paper is to design expressive extensions of CTL and LTL that retain their nice pragmatic features, in particular an intuitive and readable syntax. Likewise, the semantics needs to be – at the same time – mathematically sound. We aim for a moderate increase in expressive power in the sense of the example above: on top of standard CTL and LTL, it should be possible to express certain additional patterns of infinitary modal formulas, for instance infinitary disjunctions over uniform families of $\mathtt{AX}$-formulas.

As an example, recall that two paths $\pi = s_0, s_1, \ldots$ and $\pi' = s'_0, s'_1, \ldots$ of an LTS $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$ are called *trace-equivalent* iff $\ell(s_i) = \ell(s'_i)$ for all $i \geq 0$. The existence of two non-trace-equivalent paths can be expressed in infinitary modal logic as

$$\varphi_{\mathsf{nonlin}} := \bigvee_{p \in \mathcal{P}} \bigvee_{i \geq 0} \mathtt{EX}^i p \wedge \mathtt{EX}^i \neg p$$

In other words, $\neg\varphi_{\mathsf{nonlin}}$ states that all paths beginning in the state of evaluation, are trace-equivalent or, equivalently, that the LTS (from that state) is bisimilar to a word.

Note that in $\varphi_{\mathsf{nonlin}}$, the $\mathtt{EX}$-operators are "guarded" by a conjunction. It is therefore not possible to rewrite this formula into a least fixpoint recursion of $\mathcal{L}_\mu$ in the style of Eq. 1. In order to capture such patterns of infinitary modal logic, a temporal logic would have to provide operators which allow the build-up of modal formulas "behind" the recursion anchor, similar to the FLC formula equivalent to φ_2 above.

An appropriate mechanism for creating such effects can be borrowed from HFL: it lifts the recursion anchor from the propositional level to a higher one. In HFL, this can be a function of arbitrary order; here we restrict ourselves to order 1 to keep the resulting logic reasonably simple. The recursion anchor then becomes a function whose arguments are temporal formulas. In order to manipulate the arguments, we use propositional variables as symbolic names for the argument values in each recursive call.

Thus, our recursive temporal logic should have a recursion operator which defines a recursion anchor in the form of a variable, say $\mathcal{F}$. At the same time, it needs to provide symbolic names for some arguments to $\mathcal{F}$, say $x_1, \dots, x_n$, and then the definition of the recursion can use these as standard temporal formulas, as well as $\mathcal{F}$ applied to n formulas. Likewise, the entire recursion formula would have to be applied to n propositional formulas, which are just the initial arguments for the recursive iteration. Such a formula could look like

$$\varphi_p := \big(\underbrace{\mathtt{rec}\,\mathcal{F}(y,z).(y \land z) \lor \mathcal{F}(\mathtt{EX}y, \mathtt{EX}z)}_{\Psi}\big)(p, \neg p) \ .$$

From a pragmatic point of view, recursion can be read in a natural way using unfolding and replacement of symbolic argument variables, i.e. using β-reduction. Here, we would get

$$\begin{aligned}\varphi_p \equiv & (p \land \neg p) \lor \Psi(\mathtt{EX}p, \mathtt{EX}\neg p) \equiv (p \land \neg p) \lor (\mathtt{EX}p \land \mathtt{EX}\neg p) \lor \Psi(\mathtt{EXEX}p, \mathtt{EXEX}\neg p) \\ \equiv & \ldots \equiv \bigvee_{i \geq 0} \mathtt{EX}^i p \land \mathtt{EX}^i \neg p \ .\end{aligned}$$

In other words, such an extension of CTL (with appropriately defined semantics) would be able to express $\varphi_{\mathsf{nonlin}}$ via $\bigvee_{p \in \mathcal{P}} \varphi_p$.

The formal syntax. Let $\mathcal{P} = \{p, q, \dots\}$ be a finite set of atomic propositions, $\mathcal{V}_1 = \{x, y, \dots\}$ and $\mathcal{V}_2 = \{\mathcal{F}, \mathcal{G}, \dots\}$ be sets of *propositional*, resp. *recursion* variables. Formulas of *Recursive CTL* (RecCTL) are given by the grammar

$$\begin{aligned}\varphi, \psi \ &::= \ p \mid x \mid \varphi \land \psi \mid \neg\varphi \mid \mathtt{EX}\varphi \mid \mathtt{E}(\varphi\,\mathtt{U}\,\psi) \mid \Phi(\varphi, \dots, \varphi) \\ \Phi \ &::= \ \mathcal{F} \mid \mathtt{rec}\,\mathcal{F}(x_1, \dots, x_k).\varphi\end{aligned}$$

where $p \in \mathcal{P}$, $k \geq 0$, $x, x_1, \dots, x_k \in \mathcal{V}_1$ and $\mathcal{F} \in \mathcal{V}_2$. The formulas derived from φ, ψ are called *propositional*, those derived from Φ are called *first-order*.

Other Boolean and temporal operators are defined in the usual way, for instance $\mathtt{AX}\varphi := \neg\mathtt{EX}\neg\varphi$, $\mathtt{EF}\varphi := \mathtt{E}(\mathtt{tt}\,\mathtt{U}\,\varphi)$, $\mathtt{AG}\varphi := \neg\mathtt{EF}\neg\varphi$, and will be used freely henceforth. The notions of a subformula, a free or bound occurrence of a variable are the usual ones.

The semantics of the recursion operator will be explained later on using least fixpoints in complete function lattices. This makes the Bekič Lemma [5] available which allows formulas with mutual dependencies between recursion variables to be written down in a more readable form. A formula in *vectorial form*, cf. [3] for its use in $\mathcal{L}_\mu$, is a

$$\mathtt{rec}_{\,i} \begin{pmatrix} \mathcal{F}_1(x_1, \dots, x_{k_1}) & . & \varphi_1 \\ & \vdots & \\ \mathcal{F}_n(x_1, \dots, x_{k_n}) & . & \varphi_n \end{pmatrix} (\psi_1, \dots, \psi_k)$$

s.t. $1 \le i \le n$ and $k = k_i$. Informally, this defines not just one but several functions $\mathcal{F}_1, \dots, \mathcal{F}_n$ which may all depend on each other in a mutually recursive way formalised in the φ_j's. In the end, the function named by $\mathcal{F}_i$ is applied to the initial arguments $\psi_1, \dots, \psi_k$.

We will also write $\mathtt{fun}(x_1, \dots, x_k).\varphi$ instead of $\mathtt{rec}\,\mathcal{F}(x_1, \dots, x_k).\varphi$ when $\mathcal{F}$ does not occur in φ.

Well-formed formulas. Clearly, not every formula generated by the formal grammar above is meaningful. For instance, in $(\mathtt{rec}\,\mathcal{F}(x).x \wedge \mathcal{F}(\mathtt{EX}x))(p, \neg p)$ the number of formal parameters of $\mathcal{F}$ does not match the number of given parameters. Such mistakes are easy to spot; henceforth, we assume that all formulas are well-formed in this respect.

However, further restrictions need to be imposed before a formal semantics can be given, since the addition of the recursion operator requires careful handling of negations.

Consider φ_p from above. Its subformula Ψ can be seen as a function mapping a pair of propositions to a proposition. When interpreted over an LTS with state set $\mathcal{S}$, such a function is an object of type $2^{\mathcal{S}} \times 2^{\mathcal{S}} \to 2^{\mathcal{S}} =: M_{1,2}$ (functions of order 1 with 2 arguments). Note that $M_{1,2}$ is a complete lattice when equipped with the point-wise order where $f \le g$ iff $f(x, y) \le g(x, y)$ for all $x, y \in 2^{\mathcal{S}}$. This also is true when restricted to just monotonic functions from $M_{1,2}$. Since recursion should be explained using least fixpoints, we are interested in the function $f\colon M_{1,2} \to M_{1,2}$ that maps a (monotonic) function $\mathcal{F}\colon M_{1,2}$ to the function $(y, z) \mapsto (y \cap z) \cup \mathcal{F}(\mathtt{EX}y, \mathtt{EX}z)\colon M_{1,2}$. All operators used here are monotonic in the usual powerset lattice $2^{\mathcal{S}}$ and, hence, if $\mathcal{F}$ is monotonic, so is f. Thus, the Knaster-Tarski Theorem [32, 17] yields that f has a least fixpoint F, which is a natural candidate for the semantics of Ψ.

Having negation in a specification logic is desirable, yet negation is clearly not a monotonic operator. This is not problematic in CTL and LTL, where negation can only occur in places where the implicit recursion in e.g. the operator $\mathtt{U}$ is not affected by negation in one of its arguments. Already $\mathcal{L}_\mu$ does not allow unrestricted use of negation, since $\mu X.\neg X$ for instance cannot be given a proper semantics. The solution is to restrict the syntax to only allow negation in front of non-variable atoms, or to require that recursion variables only occur under an even number of negations in the defining fixpoint formula.

The first way is also viable mathematically here, but it would restrict the pragmatics of this logic strongly since one may often want to specify undesired properties, i.e. use negation on top level for instance. Hence, we aim for a syntactic criterion that allows negation to be used as freely as possible. Unfortunately, the comparatively simple rule used in $\mathcal{L}_\mu$ does not generalise so easily.

Consider the toy example $(\mathtt{rec}\,\mathcal{F}(x).\,(\mathtt{fun}\,y.\neg y)(\mathcal{F}(x)))\,p$. It appears to be harmless, since negation occurs only in front of the propositional variable y but not any recursion variable. However, unfolding and β-reducing this to $(\mathtt{rec}\,\mathcal{F}(x).\,\neg\mathcal{F}(x))\,p$ reveals that the negation in front of $\mathcal{F}$ was simply hidden away in the anonymous function $\mathtt{fun}\,y.\neg y$ which is clearly not monotonic.

While the use of this antitonic function violates the monotonicity requirement of the function defined by $\mathcal{F}$, functions that are antitonic in one of their arguments are not necessarily problematic in general. In fact, much of the expressive power of RecCTL and RecLTL would be lost if antitonic functions were forbidden in general. Consider

$$\varphi_{\mathsf{unbound}} := \neg\big(\mathtt{rec}\,\mathcal{F}(x, y).(x \wedge \neg y) \vee \mathcal{F}(\mathtt{EX}x, \mathtt{EX}y)\big)(p, \mathtt{EX}p)$$

stating that there is no n such that p can be reached in n steps but not in $n+1$. Clearly, the function $\mathcal{F}$ is antitonic in its second argument. However, the function $\mathcal{F} \mapsto ((x, y) \mapsto (x \cap \overline{y}) \cup \mathcal{F}(\mathtt{EX}x, \mathtt{EX}y)$ is indeed monotonic in $\mathcal{F}$ and therefore has a least fixpoint which is why the recursion in $\varphi_{\mathsf{unbound}}$ is well-defined in this case. In this instance, we make use of the fact that $M_{1,2}$ stays a complete lattice when restricted to functions that are monotonic in their first argument, and antitonic in the second argument, whence any monotonic function that maps a function from this sublattice back into the sublattice has a least fixpoint. In fact, this holds for all $M_{1,j}$ and all partitions of the arguments into monotonic and antitonic.

In the following we will devise a syntactic criterion that allows antitonic functions to be used in a harmless way, i.e. such that a formal semantics can still be given via least fixpoints. The criterion is necessarily more complex than the $\mathcal{L}_\mu$ one about occurrence under an even number of negations as seen above; on the other hand we also do not require an entire type system as it is the case in HFL.

We will call a formula well-formed if, in addition to the constraint on matching numbers of arguments, it is possible to separate each list of formal and given parameters into two parts of monotonically and antitonically used arguments, such that based on this separation we can establish that the former ones are only used positively and the latter ones are only used negatively. In the following we will make the meaning of this precise. For the moment, suppose that recursive definitions and calls are written as

$$\big(\mathtt{rec}\,\mathcal{F}(x_1,\ldots,x_k \mid y_1,\ldots,y_{k'}).\,\varphi\big) \quad \text{resp.} \quad \mathcal{F}(\varphi_1,\ldots,\varphi_k \mid \psi_1,\ldots,\psi_{k'})\ .$$

Either part of such a list can also be empty. For example, we woudl write $\varphi_{\mathsf{unbound}}$ as

$$\neg\big(\mathtt{rec}\,\mathcal{F}(x \mid y).(x \wedge \neg y) \vee \mathcal{F}(\mathtt{EX}x \mid \mathtt{EX}y)\big)(p \mid \mathtt{EX}p)$$

declaring, in particular, x to be used monotonically and y to be used antitonically. The other possibilities for separating the arguments would not pass the following check about positive, resp. negative use. We say that $x \in \mathcal{V}_1$ is used positively in x and $\mathcal{F} \in \mathcal{V}_2$ is used positively in $\mathcal{F}(x_1,\ldots,x_k \mid y_1,\ldots,y_{k'})$. Moreover, x, or $\mathcal{F}$ is used

- positively, resp. negatively in $\varphi_1 \wedge \varphi_2$, $\mathtt{EX}\varphi_1$, $\mathtt{E}(\varphi_1 \mathtt{U} \varphi_2)$ or $\mathtt{rec}\,\mathcal{F}(x_1,\ldots,x_k \mid y_1,\ldots,y_{k'}).\,\varphi_1$, if it is used positively, resp. negatively in φ_1 or φ_2;
- positively, resp. negatively in $\neg\varphi$ if it is used negatively, resp. positively in φ;
- positively in $\mathcal{G}(\varphi_1,\ldots,\varphi_k \mid \psi_1,\ldots,\psi_{k'})$ or $\big(\mathtt{rec}\,\mathcal{G}(\ldots).\,\varphi\big)(\varphi_1,\ldots,\varphi_k \mid \psi_1,\ldots,\psi_{k'})$ if it is used positively in one of the φ_i, or negatively in one of the ψ_i;
- negatively in $\mathcal{G}(\varphi_1,\ldots,\varphi_k \mid \psi_1,\ldots,\psi_{k'})$ or $\big(\mathtt{rec}\,\mathcal{G}(\ldots).\,\varphi\big)(\varphi_1,\ldots,\varphi_k \mid \psi_1,\ldots,\psi_{k'})$ if it is used negatively in one of the φ_i, or positively in one of the ψ_i.

Intuitively, the polarity of use switches at an actual negation, and when the subformula in question is an argument right of the separator in a recursive call. Having defined a notion of positive and negative occurrences, we can restrict the syntax accordingly to ensure that the following semantics will be well-defined. Note that x or $\mathcal{F}$ can be used both positively and negatively in a formula.

▶ **Definition 1.** *A formula of* RecCTL *is called* well-formed *if it is possible to separate the formal and given arguments of recursive definitions and calls into two lists each such that the following conditions hold.*

- *If $\big(\mathtt{rec}\,\mathcal{F}(x_1,\ldots,x_k \mid y_1,\ldots,y_{k'}).\,\varphi\big)(\varphi_1,\ldots,\varphi_l \mid \psi_1,\ldots,\psi_{l'})$ is a recursive definition, then $k = l$ and $k' = l'$, and if $\mathcal{F}(\varphi_1,\ldots,\varphi_m \mid \psi_1,\ldots,\psi_{m'})$ is a recursive call of the same variable, then $k = m$ and $k' = m'$,*
- *If $\mathtt{rec}\,\mathcal{F}(x_1,\ldots,x_k \mid y_1,\ldots,y_{k'}).\,\varphi$ is a recursive definition then none of the x_i is used negatively in φ, and none of the y_i is used positively in φ, and*
- *If $\mathtt{rec}\,\mathcal{F}(x_1,\ldots,x_k \mid y_1,\ldots,y_{k'}).\,\varphi$ is a recursive definition then $\mathcal{F}$ is not used negatively in φ.*

The formal semantics. Let $\mathcal{T} = (\mathcal{S}, \to, \ell)$ be an LTS, let φ_0 be a well-formed formula of RecCTL. An environment is a function from $\eta \colon \mathcal{V}_1 \cup \mathcal{V}_2 \to 2^{\mathcal{S}} \cup \bigcup_{j\geq 0} M_{1,j}$ where the value of a variable matches its type in φ_0 and the declared monotonicity, resp. antitonicity of its arguments. Here $M_{1,j}$ is the space of order-1 functions with j arguments.

The formal semantics assigns a proposition, i.e. set of states to each propositional subformula φ of φ_0, and a first-order function to each first-order subformula Φ of φ of as follows.

$$\begin{aligned}
[\![p]\!]^{\mathcal{T}}_{\eta} &= \{s \in \mathcal{S} \mid p \in \ell(s)\} \\
[\![x]\!]^{\mathcal{T}}_{\eta} &= \eta(x) \\
[\![\varphi \wedge \psi]\!]^{\mathcal{T}}_{\eta} &= [\![\varphi]\!]^{\mathcal{T}}_{\eta} \cap [\![\psi]\!]^{\mathcal{T}}_{\eta} \\
[\![\neg\varphi]\!]^{\mathcal{T}}_{\eta} &= \mathcal{S} \setminus [\![\varphi]\!]^{\mathcal{T}}_{\eta} \\
[\![\mathtt{EX}\varphi]\!]^{\mathcal{T}}_{\eta} &= \{s \in \mathcal{S} \mid \text{ there exists } t \in [\![\varphi]\!]^{\mathcal{T}}_{\eta} \text{ such that } s \to t\} \\
[\![\mathtt{E}(\varphi \,\mathtt{U}\, \psi)]\!]^{\mathcal{T}}_{\eta} &= \{s \in \mathcal{S} \mid \text{ there exists path } \pi, \text{ integer } i \text{ such that} \\
&\qquad s = \pi(0), \pi(i) \in [\![\psi]\!]^{\mathcal{T}}_{\eta} \text{ and } \pi(j) \in [\![\varphi]\!]^{\mathcal{T}}_{\eta} \text{ for all } 0 \leq j < i\} \\
[\![\mathcal{F}]\!]^{\mathcal{T}}_{\eta} &= \eta(\mathcal{F}) \\
[\![\mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_k).\, \varphi]\!]^{\mathcal{T}}_{\eta} &= \mathrm{LFP}\, f \mapsto \big((S_1, \ldots, S_k) \mapsto [\![\varphi]\!]^{\mathcal{T}}_{\eta[\mathcal{F} \mapsto f, x_1 \mapsto S_1, \ldots, x_k \mapsto S_k]}\big) \\
[\![\Phi(\varphi_1, \ldots, \varphi_k)]\!]^{\mathcal{T}}_{\eta} &= [\![\Phi]\!]^{\mathcal{T}}_{\eta}([\![\varphi_1]\!]^{\mathcal{T}}_{\eta}, \ldots, [\![\varphi_k]\!]^{\mathcal{T}}_{\eta})
\end{aligned}$$

Notions like satisfaction ($\mathcal{T}, s \models \varphi$), satisfiability and equivalence ($\varphi \equiv \psi$) are defined as usual.

The following lemma states that this semantics is well-defined, in particular, that least fixpoints as used in the second to last clause do indeed exist. This is guaranteed by well-formedness of φ_0 which in turn guarantees monotonicity of the function whose least fixpoint is used to give meaning to the recursion operator in the penultimate clause.

▶ **Lemma 2.** *Let φ be a well-formed* RecCTL *sentence, let η be an environment and let $\mathcal{T}$ be an LTS. Then $[\![\psi]\!]^{\mathcal{T}}_{\eta}$ is well-defined.*

Since $[\![\varphi]\!]^{\mathcal{T}}_{\eta}$ does not depend on η, we simply write $[\![\varphi]\!]^{\mathcal{T}}$ and drop the environment. The lemma's proof is purely technical but standard by induction on the structure of φ.

We also state a fundamental equivalence which is very helpful for understanding formulas. The proof is simply by combining the well-known equivalence-preserving principles of fixpoint unfolding and β-reduction, and is therefore omitted. As usual, $\varphi[\psi_1/x_1, \ldots \psi_k/x_k]$ denotes the simultaneous replacement of every free occurrence of x_i by ψ_i.

▶ **Lemma 3.** *For any $\varphi, \psi_1, \ldots, \psi_k$ we have*

$$(\mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_k).\varphi)(\psi_1, \ldots, \psi_k) \equiv \varphi[\psi_1/x_1, \ldots, \psi_k/x_k, \mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_k).\varphi/\mathcal{F}]$$

The linear-time case. RecLTL is obtained from RecCTL syntactically by removing the path quantifiers E and A just like the syntax of LTL can be obtained from CTL in this way. A RecLTL formula is interpreted over a linear-time structure π, i.e. a transition system with a single path only. The semantics is defined in the same way as for RecCTL. Given a RecLTL formula φ, an LTS $\mathcal{T}$ with state s, and a path π, we write $\pi \models \varphi$ to denote that the path π satisfies φ. We write $\mathcal{T}, s \models \varphi$ iff all paths starting in s satisfy φ. In other words, the usual for-all-paths semantics for linear-time formulas can also be applied to the richer language RecLTL.

It is not hard to see that Lemmas 2 and 3 as well as previously worked out concepts like well-formedness etc. hold for RecLTL as well.

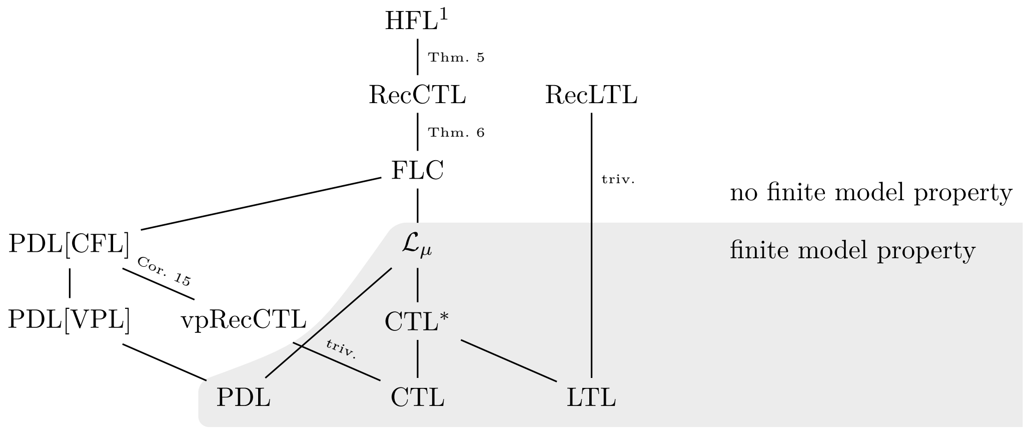

Figure 1 Placing RecCTL and RecLTL into the hierarchy of temporal logics w.r.t. expressive power.

3 The Power of Recursion in Temporal Logics

Recall the RecCTL example $\varphi_{\mathsf{nonlin}}$ above and remember that $\neg\varphi_{\mathsf{nonlin}}$ states that all paths emerging from the state under consideration are trace-equivalent, i.e. indistinguishable through the sequence of their propositional labels. This gives an easy satisfiability-preserving reduction from RecLTL into RecCTL.

▶ **Theorem 4.** *For every* $\varphi \in$ RecLTL *there is an equi-satisfiable* $\varphi' \in$ RecCTL *such that* $|\varphi'| = \mathcal{O}(|\varphi|)$.

Proof. Simply take $\varphi' := \widehat{\varphi} \wedge \neg\varphi_{\mathsf{nonlin}}$ where $\widehat{\varphi}$ results from φ by replacing every subformula of the form $\mathtt{X}\psi$ with $\mathtt{AX}\psi$. The second conjunct requires a model of φ' to only have trace-equivalent paths which are of infinite length by assumption. Thus, if φ has a model π then this is clearly a model of φ'. Moreover, any path of an LTS model for φ' is a model for φ. ◀

Next we place RecCTL and RecLTL into the expressiveness hierarchy of well-known (and some lesser known) temporal and modal fixpoint logics. Note that the models under consideration here are transition systems without edge labels, as they are usually used for temporal logics like CTL and LTL. The results of this section are presented for this class of structures, even though logics like $\mathcal{L}_\mu$ are typically interpreted over the richer class of transition system *with* edge labels. The results can easily be extended to this richer class, provided that the syntax of RecCTL and RecLTL is extended to speak about a-successors rather than just successors, for example by replacing $\mathtt{EX}$ with $\mathtt{EX}_a$ for any edge label.

This hierarchy is shown in Fig. 1. It contains the standard temporal logics CTL and LTL as fragments of CTL*, which in turn is known to be embeddable into $\mathcal{L}_\mu$. Above, there are the expressive logics mentioned in the introduction, namely

- *Fixpoint Logic with Chop* (FLC): it interprets every formula as a predicate transformer mapping a set of states to a set of states of an LTS. Predicates, the basic semantic objects of $\mathcal{L}_\mu$, can be seen as constant predicate transformers which explains why FLC extends $\mathcal{L}_\mu$ [28].
- *Higher-Order Fixpoint Logic* (HFL): it allows functions of arbitrary higher-order to be built from modal and Boolean operators as well as fixpoints. Its fragment HFL^1 is obtained by restricting all functions to first order. This includes predicate transformers as they can be seen as *unary* functions of order 1. Hence, FLC is embeddable into HFL^1 [34].

The graph also includes the dynamic logic PDL as it is closely related to CTL and its non-regular extension PDL[CFL], in order to complete the picture of expressiveness of temporal logics, in particular to give a better feeling for the power of RecCTL and RecLTL. The embedding of PDL[CFL] into FLC was shown in [25]. At last, it includes the fragment vpRecCTL of RecCTL which will be discussed in Sect. 4 in the context of decidability questions.

The picture also draws the distinguishing line of regularity vs. non-regularity in terms of possessing the finite model property (FMP). It is lost for all of these logics that are not embeddable into $\mathcal{L}_\mu$ [23].

RecCTL can be placed between FLC and HFL^1. We refrain from presenting the full (and sometimes cumbersome) syntax and semantics of these two logics here. Instead we refer to the existing literature for full details [28, 34] and only give the main ideas here.

▶ **Theorem 5.** *Every* RecCTL *formula can equivalently be expressed in* HFL^1*.*

Proof. (Sketch) Well-formed formulas can straight-forwardly be translated. The only interesting case is that of a recursive first-order formula $(\texttt{rec}\,\mathcal{F}(x_1,\dots,x_k \mid y_1,\dots,y_{k'}).\varphi)$. It is equivalent to the HFL^1 formula $\mu\mathcal{F}^\tau.\lambda x_1^\bullet.\dots.\lambda x_k^\bullet.\varphi$ with type annotation $\tau = \bullet^+ \to \dots \to \bullet^+ \to \bullet^- \to \dots \to \bullet^- \to \bullet$. ◀

For the lower bound we need to quickly recall FLC. Its formulas are built from basic literals p, $\neg p$ and the modal $\Diamond$ and $\Box$. Note that they receive no fixed argument, as they are being interpreted not as predicates but as predicate transformers, i.e. a function of type $2^{\mathcal{S}} \to 2^{\mathcal{S}}$ over a state space $\mathcal{S}$ of some LTS.

The syntax has conjunctions and disjunctions and a *chop* operator ";" which is interpreted as the functional composition of two predicate transformers, and another atomic formula τ which is interpreted as the neutral element to composition, i.e. the identity predicate transformer. On top of this, fixpoint quantifiers are added which are interpreted as fixpoints in the complete lattice of pointwise-ordered predicate transformers.

▶ **Theorem 6.** *Every* FLC *formula can equivalently be expressed in* RecCTL*.*

Proof. We devise a translation $\widehat{\cdot}$: FLC $\to$ RecCTL that preserves equivalence using, for every FLC fixpoint variable X, a unique recursion variable $\mathcal{F}_X$.

$$\begin{array}{rlrlrl}
\widehat{p} &:= \texttt{fun}\,x.p & \widehat{\varphi \vee \psi} &:= \texttt{fun}\,x.\widehat{\varphi}(x) \vee \widehat{\psi}(x) & \widehat{\Diamond} &:= \texttt{fun}\,x.\texttt{EX}x \\
\widehat{\neg p} &:= \texttt{fun}\,x.\neg p & \widehat{\varphi \wedge \psi} &:= \texttt{fun}\,x.\widehat{\varphi}(x) \wedge \widehat{\psi}(x) & \widehat{\Box} &:= \texttt{fun}\,x.\texttt{AX}x \\
\widehat{\tau} &:= \texttt{fun}\,x.x & \widehat{\varphi;\psi} &:= \texttt{fun}\,x.\widehat{\varphi}(\widehat{\psi}(x)) & & \\
\widehat{X} &:= \mathcal{F}_X & \widehat{\mu X.\varphi} &:= \texttt{rec}\,\mathcal{F}_X(y).\widehat{\varphi}(y) & \widehat{\nu X.\varphi} &:= \neg\,\texttt{rec}\,\mathcal{F}_X(y).\neg\widehat{\varphi}[\neg\mathcal{F}_X/\mathcal{F}_X](y)
\end{array}$$

where $[\neg\mathcal{F}_X/\mathcal{F}_X]$ denotes the substitution of any $(\mathcal{F}_X(y).\psi)(\psi')$ with $\neg((\mathcal{F}_X(y).\psi)(\psi'))$.

A straight-forward induction on the structure of an FLC formula φ shows that $\widehat{\varphi}$ denotes the same predicate transformer as φ under any variable assignment that maps X and $\mathcal{F}_X$ to the same predicate transformer. We then get that the FLC formula φ is equivalent to the RecCTL formula $\widehat{\varphi}(\texttt{tt})$ by virtue of the way that the semantics of FLC turns a predicate transformer into a predicate. ◀

▶ **Corollary 7.** *Neither* RecCTL *nor* RecLTL *have the finite model property.*

Proof. For RecCTL this is a consequence of Thm. 6 since FLC does not have the finite model property [28]. For RecLTL consider the formula $\varphi_{\mathsf{steps}}(p)$ to be defined next. It is easily seen to be satisfiable, yet unsatisfiable on any finitely represented linear structure. ◀

4 Satisfiability and Model Checking

In this section we investigate the issues of decidability and computational complexity of the two most important reasoning problems for temporal logics: satisfiability and model checking.

Undecidability results. Consider the RecLTL formula (scheme)

$$\begin{aligned}\varphi_{\mathsf{steps}}(\psi) \;:=\; & \psi \wedge \mathtt{X}\psi \wedge \mathtt{XX}\neg\psi \wedge \mathtt{XXX}\psi \\ \wedge\; & \mathtt{G}\Big(\psi \to \mathtt{X}\big(\underbrace{\mathtt{rec}\,\mathcal{H}(x).(\psi \wedge \mathtt{X}x) \vee (\neg\psi \wedge \mathtt{X}\mathcal{H}(\neg\psi \wedge \mathtt{X}x))}_{\Phi_{\mathcal{H}}}\big)(\neg\psi \wedge \mathtt{X}\psi)\Big)\ .\end{aligned}$$

For brevity, let $\mathtt{X}^+\chi$ abbreviate $\psi \wedge \mathtt{X}\chi$ and $\mathtt{X}^-\chi$ abbreviate $\neg\psi \wedge \mathtt{X}\chi$. Note that the argument to $\Phi_{\mathcal{H}}$ is $\mathtt{X}^-\psi$ in this respect, and that $\Phi_{\mathcal{H}}$ can be written as $\mathtt{rec}\,\mathcal{H}(x).(\mathtt{X}^+x) \vee \mathtt{X}^-\mathcal{H}(\mathtt{X}^-x)$. We also have $\mathtt{X}^-(\chi_1 \vee \chi_2) \equiv \mathtt{X}^-\chi_1 \vee \mathtt{X}^-\chi_2$ in general. Then consider $\Phi_{\mathcal{H}}(\mathtt{X}^-\psi)$. We have

$$\begin{aligned}\Phi_{\mathcal{H}}(\mathtt{X}^-\psi) &\equiv \mathtt{X}^+\mathtt{X}^-\psi \vee \mathtt{X}^-\Phi_{\mathcal{H}}(\mathtt{X}^-\mathtt{X}^-\psi) \\ &\equiv \mathtt{X}^+\mathtt{X}^-\psi \vee \mathtt{X}^-(\mathtt{X}^+\mathtt{X}^-\mathtt{X}^-\psi \vee \mathtt{X}^-\Phi_{\mathcal{H}}(\mathtt{X}^-\mathtt{X}^-\mathtt{X}^-\psi)) \\ &\equiv \mathtt{X}^+\mathtt{X}^-\psi \vee \mathtt{X}^-\mathtt{X}^+\mathtt{X}^-\mathtt{X}^-\psi \vee \mathtt{X}^-\mathtt{X}^-\Phi_{\mathcal{H}}(\mathtt{X}^-\mathtt{X}^-\mathtt{X}^-\psi) \\ &\equiv \mathtt{X}^+\mathtt{X}^-\psi \vee \mathtt{X}^-\mathtt{X}^+\mathtt{X}^-\mathtt{X}^-\psi \vee \mathtt{X}^-\mathtt{X}^-(\mathtt{X}^+\mathtt{X}^-\mathtt{X}^-\mathtt{X}^-\psi \vee \mathtt{X}^-\Phi_{\mathcal{H}}(\mathtt{X}^-\mathtt{X}^-\mathtt{X}^-\mathtt{X}^-\psi)) \\ &\equiv \ldots \equiv \bigvee_{n\geq 0} \underbrace{\mathtt{X}^-\ldots\mathtt{X}^-}_{n}\mathtt{X}^+\underbrace{\mathtt{X}^-\ldots\mathtt{X}^-}_{n+1}\psi\end{aligned}$$

The first and second equivalence and every second after that uses Lemma 3. The others simply use the the commutation of $\mathtt{X}^-$ with disjunctions.

The first conjuncts in $\varphi_{\mathsf{steps}}(\psi)$ fix the values of ψ on a possible model in the first four states, namely to hold at positions $0, 1$ and 3. Since ψ holds at positions 1 and 3 but not at 2, $\mathtt{G}(\psi \to \mathtt{X}\Phi_{\mathcal{H}}(\mathtt{X}^-\psi))$ forces ψ to furthermore hold at position 6 but not at 4 and 5. This can be iterated now with position 3 to see that the next moment at which ψ holds is 10. Hence, $\varphi_{\mathsf{steps}}(\psi)$ forces ψ to hold at the initial point of a model and then at distances increasing by 1 in each step. This can be used in the proof of the next result.

▶ **Theorem 8.** *The satisfiability problem for* RecLTL *is undecidable (Σ^1_1-hard).*

Proof. The following problem, known as the recurrent octant tiling problem, is Σ^1_1-hard [14]: given a tiling system $\mathcal{T} = (T, H, V, t_0, t_\infty)$ where T is a finite set of tile types, $H, V \subseteq T^2$ and $t_0, t_\infty \in T$, is there a tiling $\tau : \{(i,j) \mid 0 \leq i \leq j\} \to T$ of the octant plane, such that

- $\tau(0,0) = t_0$, (initial tile set properly)
- for all i, j with $j > i$: $(\tau(i,j), \tau(i+1,j)) \in H$, (no horizontal mismatch)
- for all i, j with $j \geq i$: $(\tau(i,j), \tau(i,j+1)) \in V$, (no vertical mismatch)
- there are infinitely many j such that $\tau(0,j) = t_\infty$? (recurrence)

Such a tiling τ can be represented straight-forwardly as a linear-time model over the set of propositions T by listing it row-wise:

$$\tau(0,0), \tau(0,1), \tau(1,1), \tau(0,2), \tau(1,2), \tau(2,2), \tau(0,3), \ldots, \tau(3,3), \tau(0,4), \ldots$$

Moreover, the conditions on a successful tiling can be formalised in RecLTL as follows, using one additional proposition fst to mark the beginnings of each row.

$$\begin{aligned}\varphi_{\mathcal{T}} \;:=\;& t_0 \wedge \mathtt{G}(\bigwedge_{t\in T} t \to \bigwedge_{t'\neq t} \neg t') \wedge \varphi_{\mathsf{steps}}(\mathsf{fst}) \wedge \mathtt{G}(\mathtt{X}\neg\mathsf{fst} \to \bigvee_{(t,t')\in H} t \wedge \mathtt{X}t') \\ & \wedge\, \mathtt{G}\Big(\mathsf{fst} \to \neg \bigvee_{(t,t')\in T^2\setminus V} \big(\mathtt{rec}\, H(x,y).(x \wedge \mathtt{X}(\neg\mathsf{fst}\,\mathtt{U}\,(\mathsf{fst}\wedge y))) \vee H(\mathtt{X}(\neg\mathsf{fst}\wedge x), \mathtt{X}y)\big)(t,t')\Big) \\ & \wedge\, \mathtt{GF}(\mathsf{fst} \wedge t_\infty)\end{aligned}$$

The first conjunct enforces the initiality condition, the second ensures that each position is occupied by exactly one tile. The third conjunct ensures that exactly the positions of the form $\tau(0,j)$ for any j are marked using fst, using φ_{steps} constructed above. The fourth ensures horizontal matching by comparing adjacent positions apart from those where the succeeding one is the beginning of the next row. The fifth conjunct states that it is impossible to find the beginning of some row j, a vertically non-matching pair of tiles (t,t'), and an $i \leq j$ such that t is the tile i steps after that beginning of the row, and t' is found i steps after the next state satisfying fst. Note that these are exactly the positions that are vertically adjacent in the octant plane.

The last conjunct ensures the recurrence condition. Now a successful tiling τ for $\mathcal{T}$ induces a linear time model for $\varphi_{\mathcal{T}}$ in the shape as described and vice-versa. ◀

An immediate consequence of Thms. 4 and 8 is the (high) undecidability of RecCTL.

▶ **Corollary 9.** *The satisfiability problem for* RecCTL *is* Σ^1_1*-hard.*

Also, it is well-known that model checking for linear-time logics under the usual all-paths-semantics is closely related to the validity problem.

▶ **Corollary 10.** *The model checking problem for* RecLTL *over transition systems is* Π^1_1*-hard.*

In contrast, the model checking problem for RecCTL over finite transition systems is decidable.

▶ **Theorem 11.** *The model checking problem for* RecCTL *over finite transition systems is* EXPTIME*-complete.*

Proof. A deterministic exponential-time upper bound can be derived from a naïve bottom-up algorithm that computes the semantics $[\![\varphi]\!]_{\mathcal{T}}$ of a given formula φ over a given LTS $\mathcal{T}$ using fixpoint iteration. The EXPTIME upper bound also follows from the (linear) embedding of RecCTL into HFL^1 (Thm. 5) whose model checking problem is known to be EXPTIME-complete [4].

A matching lower bound is inherited from FLC using Thm. 6 since the model checking problem for FLC is known to be EXPTIME-complete as well [24]. ◀

A natural question that arises concerns the decidability of model checking RecCTL over classes of infinite-state transition systems. A consequence of Thm. 6 is the negative result that this problem is already undecidable for the class BPA (Basic Process Algebra) [6] – in some sense the smallest class of context-free processes – as this is undecidable for FLC [28, 22].

▶ **Corollary 12.** *The model checking problem for* RecCTL *over classes of infinite-state transition systems subsuming BPA is undecidable.*

A decidable fragment of RecCTL. The undecidability of the satisfiability problem for temporal logics beyond regularity can often be seen by observing that such a logic, for example FLC, can both express context-free properties, and is closed under conjunctions. Since closure under Boolean operators is highly desirable, a restriction of the recursion process to a class strictly below the context-free languages is unavoidable if one wants to recuperate decidability. A natural candidate are the *visibly pushdown languages* (VPL) [2], which are closed under intersection and complement and, hence, not problematic if mixed with full closure under Boolean operators. In particular, it is known that PDL[VPL] is decidable and 2EXPTIME-complete [26]. We refer to the literature for a detailed introduction into VPL and PDL[VPL].

▶ **Definition 13.** *Let $\mathcal{P}$ be a set of propositions. We write $\mathbb{B}(\mathcal{P})$ for the set of all Boolean combinations of these variables. Let $\mathcal{B}_\mathsf{c}, \mathcal{B}_\mathsf{r}, \mathcal{B}_\mathsf{i} \subseteq \mathbb{B}(\mathcal{P})$ be mutually exclusive, i.e. the conjunction of two formulas from these sets is satisfiable only if they are from the same set.*

Formulas of the fragment vpRecCTL *of* RecCTL *are given by the grammar*

$$\varphi \quad ::= \quad \texttt{tt} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \texttt{rec}\,_i \begin{pmatrix} \mathcal{F}_1(x_1) & . & \psi_1 \\ & \vdots & \\ \mathcal{F}_n(x_n) & . & \psi_n \end{pmatrix} (\varphi)$$

where each ψ_j is a disjunction of formulas of the forms x_j, $\texttt{EX}(\beta_\mathsf{i} \wedge \mathcal{F}_k(x_j))$ or $\texttt{EX}\big(\beta_\mathsf{c} \wedge \mathcal{F}_k(\texttt{EX}(\beta_\mathsf{r} \wedge \mathcal{F}_{k'}(x_j)))\big)$, with $\beta_\mathsf{c} \in \mathcal{B}_\mathsf{c}$, $\beta_\mathsf{r} \in \mathcal{B}_\mathsf{r}$ and $\beta_\mathsf{i} \in \mathcal{B}_\mathsf{i}$. Note that formulas derived from φ contain no free variables from $\mathcal{V}_2$.

▶ **Theorem 14.** *The satisfiability problem for* vpRecCTL *is 2*EXPTIME*-complete.*

Proof. (sketch) It is possible to devise a satisfiability-preserving linear translation from vpRecCTL into PDL[VPL]. From $\mathcal{B}_\mathsf{c}, \mathcal{B}_\mathsf{r}, \mathcal{B}_\mathsf{i}$ we construct a visibly pushdown alphabet $\mathcal{A} := \mathcal{A}_\mathsf{c} \cup \mathcal{A}_\mathsf{r} \cup \mathcal{A}_\mathsf{i}$ with $\mathcal{A}_\mathsf{c} := \{a_\beta \mid \beta \in \mathcal{B}_\mathsf{c}\}$ and likewise for $\mathcal{A}_\mathsf{r}$ and $\mathcal{A}_\mathsf{i}$.

The key is then to see that the structure of a functional formula $\Phi = (\texttt{rec}\,_i\mathcal{F}_1(x_1).\psi_1, \dots, \mathcal{F}_n(x_n).\psi_n)$ in vectorial form resembles a context-free grammar G with $\texttt{EX}\big(\beta_\mathsf{c} \wedge \mathcal{F}_k(\texttt{EX}(\beta_\mathsf{r} \wedge \mathcal{F}_{k'}(x_j))))$ in the definition of some $\mathcal{F}_i$ for instance corresponding to a production of the form $\mathcal{F}_i \to a_{\beta_\mathsf{c}}\mathcal{F}_k a_{\beta_\mathsf{r}}\mathcal{F}_{k'}$. The structure of Φ then ensures that the resulting grammar is in fact a visibly-pushdown grammar G_Φ [2], and so $\Phi(\varphi)$ can be translated into $\langle G_\Phi \rangle(\widehat{\varphi})$ where $\widehat{\varphi}$ is the translation of φ.

The lower bound follows equally from PDL[VPL]'s 2EXPTIME-completeness [26], as the translation can easily be reversed into one from PDL[VPL] to vpRecCTL. ◀

An equivalence-preserving translation from vpRecCTL into PDL[VPL] is not possible since PDL[VPL] is defined over edge-labelled LTS, and the partition of edge labels into a visibly pushdown alphabet plays a key role in the definition of the logic. Without the shift from node- to edge-labels – keeping each β as a propositional formula rather than transforming it into an alphabet symbol a_β – the translation would indeed be equivalence preserving but the resulting formula would "only" be in PDL[CFL].

▶ **Corollary 15.** *Every* vpRecCTL *can equivalently be expressed in* PDL[CFL].

5 Conclusion & Further Work

We have presented an expressive extension of the framework of standard temporal logics. The aim is to make the specification and verification of complex systems and properties beyond regular ones more accessible through temporal logics with a reasonably intuitive syntax and semantics, such as CTL and LTL are.

High expressive power is achieved through the introduction of a recursion operator which takes formulas as arguments. The mathematical concepts underlying the formal semantics are borrowed from higher-order logics like HFL without having to involve rather cumbersome tools like proof systems. Instead, only a relatively simple monotonicity requirement for recursion variables has to be obeyed. This way, RecCTL and RecLTL achieve a reasonable balance between expressive power and pragmatic usability.

We have studied the computational complexities of the most important decision problems of model and satisfiability checking of these logics. The increase compared to CTL and LTL is in line with what one can expect to pay for additional expressiveness.

There are various routes for further work on such logics. For instance, model checking procedures that are optimised for practical purposes need to be sought. There is also potential in extending the fragment vpRecCTL by large amounts without losing the decidability property. Take for instance the infinitary modal formula $\bigvee_{n \geq i} \Diamond^n \Box^n q$, stating that for some n there is a path of length n such that all successive paths of that same length end in a q-state. This cannot be stated in PDL[VPL] even though it intuitively uses the VPL $\{a^n b^n \mid n \geq 1\}$, but PDL-based logics can only combine languages with a single modality. The property is, however, easily formalisable in RecCTL as $\mathtt{EX}(\mathtt{rec}\,\mathcal{F}(x).x \vee \mathtt{EX}\,\mathcal{F}(\mathtt{AX}\,x))(\mathtt{AX}\,q)$.

We conjecture that it is possible to allow mixtures of EX- and AX-operators in vpRecCTL, achieving higher expressiveness and yet not losing decidability. We believe that satisfiability in such an extended fragment can also be reduced to the problem of solving a visibly-pushdown game [27].

References

1 R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

2 R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Annual ACM Symp. on Theory of Computing, STOC'04*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

3 A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

4 R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3:1–33, 2007.

5 H. Bekić. *Programming Languages and Their Definition, Selected Papers*, volume 177 of *LNCS*. Springer, 1984.

6 J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1–3):109–137, 1984.

7 E. A. Emerson. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 24(2):77–79, 1987.

8 E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

9 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.

10 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. doi:10.1145/4904.4999.

11 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.

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

13 H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.

14 D. Harel. Recurring dominoes: Making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

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

16 D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In *CONCUR*, pages 263–277, 1996. `doi:10.1007/3-540-61604-7_60`.

17 B. Knaster. Un théorème sur les fonctions d'ensembles. *Annals Soc. Pol. Math*, 6:133–134, 1928.

18 R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, November 1990.

19 D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983. `doi:10.1007/BFb0012782`.

20 D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47(3):233–241, 1988.

21 L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

22 M. Lange. Local model checking games for fixed point logic with chop. In *Proc. 13th Conf. on Concurrency Theory, CONCUR'02*, volume 2421 of *LNCS*, pages 240–254. Springer, 2002.

23 M. Lange. Temporal logics beyond regularity, 2007. Habilitation thesis, University of Munich, BRICS research report RS-07-13.

24 M. Lange. Three notes on the complexity of model checking fixpoint logic with chop. *R.A.I.R.O. – Theoretical Informatics and Applications*, 41:177–190, 2007.

25 M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Information Processing Letters*, 100(2):72–75, 2006.

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

28 M. Müller-Olm. A modal fixpoint logic with chop. In *STACS'99*, volume 1563 of *LNCS*, pages 510–520. Springer, 1999.

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

30 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

31 C. Stirling. Comparing linear and branching time temporal logics. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Conf. on Temporal Logic in Specification*, volume 398 of *LNCS*, pages 1–20, Berlin, 1989. Springer.

32 A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–309, 1955.

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. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004.