SAT-based Learning of Computation Tree Logic

Adrien Pommellet[1][0000−0002−1825−0097], Daniel Stan[2][0000−0002−4723−5742], and Simon Scatton[3]

[1] LRE, EPITA, France
✉ adrien@lrde.epita.fr
https://www.lrde.epita.fr/ adrien/
[2] LRE, EPITA, France
daniel.stan@epita.fr
https://www.tudo.re/daniel.stan/
[3] LRE, EPITA, France
simon.scatton@lrde.epita.fr

Abstract. The CTL learning problem consists in finding for a given sample of positive and negative Kripke structures a distinguishing CTL formula that is verified by the former but not by the latter. Further constraints may bound the size and shape of the desired formula or even ask for its minimality in terms of syntactic size. This synthesis problem is motivated by explanation generation for dissimilar models, e.g. comparing a faulty implementation with the original protocol. We devise a SAT-based encoding for a fixed size CTL formula, then provide an incremental approach that guarantees minimality. We further report on a prototype implementation whose contribution is twofold: first, it allows us to assess the efficiency of various output fragments and optimizations. Secondly, we can experimentally evaluate this tool by randomly mutating Kripke structures or syntactically introducing errors in higher-level models, then learning CTL distinguishing formulas.

Keywords: Computation Tree Logic · Passive learning · SAT solving.

1 Introduction

Passive learning is the act of computing a theoretical model of a system from a given set of data, without being able to acquire further information by actively querying said system. The input data may have been gathered through monitoring, collecting executions and outputs of systems. Automata and logic formulas tend to be the most common models, as they allow one to better explain systems of complex or even entirely opaque design.

Linear-time Temporal Logic LTL [19] remains one of the most widely used formalisms for specifying temporal properties of reactive systems. It applies to finite or infinite execution traces, and for that reason fits the passive learning framework very well: a LTL formula is a concise way to distinguish between correct and incorrect executions. The LTL learning problem, however, is anything

but trivial: even simple fragments on finite traces are NP-complete [10], and consequently recent algorithms tend to leverage SAT solvers [16].

Computation Tree Logic CTL [8] is another relevant formalism that applies to execution trees instead of isolated linear traces. It is well-known [1, Thm. 6.21] that LTL and CTL are incomparable: the former is solely defined on the resulting runs of a system, whereas the latter depends on its branching structure.

However, the CTL passive learning problem has seldom been studied in as much detail as LTL. In this article, we formalize it on *Kripke structures* (KSs): finite graph-like representations of programs. Our goal is to find a CTL formula (said to be *separating*) that is verified by every state in a positive set S^+ yet rejected by every state in a negative set S^-.

We first prove that an explicit formula can always be computed and we bound its size, assuming the sample is devoid of contradictions. However, said formula may not be minimal. The next step is therefore to solve the bounded learning problem: finding a separating CTL formula of size smaller than a given bound n. We reduce it to an instance Φ_n of the Boolean satisfiability problem whose answer can be computed by a SAT solver; to do so, we encode CTL's bounded semantics, as the usual semantics on infinite executions trees can lead to spurious results. Finally, we use a bottom-up approach to pinpoint the minimal answer by solving a series of satisfiability problems. We show that a variety of optimizations can be applied to this iterative algorithm. These various approaches have been implemented in a `C++` tool and benchmarked on a test sample.

Related work. Bounded model checking harnesses the efficiency of modern SAT solvers to iteratively look for a witness of bounded size that would contradict a given logic formula, knowing that there exists a completeness threshold after which we can guarantee no counter-example exists. First introduced by Biere et al. [3] for LTL formulas, it was later applied to CTL formulas [17, 24, 25].

This approach inspired Neider et al. [16], who designed a SAT-based algorithm that can learn a LTL formula consistent with a sample of *ultimately periodic* words by computing propositional Boolean formulas that encode both the semantics of LTL on the input sample and the syntax of its Directed Acyclic Graph (DAG) representation. This work spurred further SAT-based developments such as learning formulas in the property specification language `PSL` [21] or $\mathsf{LTL_f}$ [7], applying `MaxSAT` solving to noisy datasets [11], or constraining the shape of the formula to be learnt [15]. Our article extends this method to CTL formulas and Kripke structures. It subsumes the original LTL learning problem: one can trivially prove that it is equivalent to learning CTL formulas consistent with a sample of lasso-shaped KSs that consist of a single linear sequence of states followed by a single loop.

Fijalkow et al. [10] have studied the complexity of learning $\mathsf{LTL_f}$ formulas of size smaller than a given bound and consistent with a sample of *finite* words: it is already NP-complete for fragments as simple as $\mathsf{LTL_f}(\wedge, \mathsf{X})$, $\mathsf{LTL_f}(\wedge, \mathsf{F})$, or $\mathsf{LTL_f}(\mathsf{F}, \mathsf{X}, \wedge, \vee)$. However, their proofs cannot be directly extended to samples of infinite but ultimately periodic words.

Browne et al. [6] proved that KSs could be characterized by CTL formulas and that conversely bisimilar KSs verified the same set of CTL formulas. As we will show in Section 3, this result guarantees that a solution to the CTL learning problem actually exists if the input sample is consistent.

Wasylkowski et al. [23] mined CTL specifications in order to explain preconditions of `Java` functions beyond pure state reachability. However, their learning algorithm consists in enumerating CTL templates of the form $\forall \mathsf{F}\, a$, $\exists \mathsf{F}\, a$, $\forall \mathsf{G}(a \implies \forall \mathsf{X} \forall \mathsf{F}\, b)$ and $\forall \mathsf{G}(a \implies \exists \mathsf{X} \exists \mathsf{F}\, b)$ where $a, b \in \mathsf{AP}$ for each function, using model checking to select one that is verified by the Kripke structure representing the aforementioned function.

Two very recent articles, yet to be published, have addressed the CTL learning problem as well. Bordais et al. [5] proved that the passive learning problem for LTL formulas on ultimately periodic words is NP-hard, assuming the size of the alphabet is given as an input; they then extend this result to CTL passive learning, using a straightforward reduction of ultimately periodic words to lasso-shaped Kripke structures. Roy et al. [22] used a SAT-based algorithm, resulting independently to our own research in an encoding similar to the one outlined in Section 4. However, our explicit solution to the learning problem, the embedding of the negations in the syntactic DAG, the approximation of the recurrence diameter as a semantic bound, our implementation of this algorithm, its test suite, and the experimental results are entirely novel contributions.

2 Preliminary Definitions

2.1 Kripke structures

Let AP be a finite set of atomic propositions. A Kripke structure is a finite directed graph whose vertices (called *states*) are labelled by subsets of AP.

Definition 1 (Kripke Structure). *A* Kripke structure *(KS)* $\mathcal{K}$ *on* AP *is a tuple* $\mathcal{K} = (Q, \delta, \lambda)$ *such that:*

- Q *is a finite set of states; the integer* $|Q|$ *is known as the* size *of* $\mathcal{K}$*;*
- $\delta : Q \to (2^Q \setminus \{\emptyset\})$ *is a transition function; the integer* $\max_{q \in Q} |\delta(q)|$ *is known as the* degree *of* $\mathcal{K}$*;*
- $\lambda : Q \to 2^{\mathsf{AP}}$ *is a labelling function.*

An infinite *run* r of $\mathcal{K}$ starting from a state $q \in Q$ is an infinite sequence $r = (s_i) \in Q^\omega$ of consecutive states such that $s_0 = q$ and $\forall\, i \geq 0$, $s_{i+1} \in \delta(s_i)$. $\mathcal{R}_\mathcal{K}(q)$ is the set of all infinite runs of $\mathcal{K}$ starting from q.

The *recurrence diameter* $\alpha_\mathcal{K}(q)$ of state q in $\mathcal{K}$ is the length of the longest finite run $(s_i)_{i=0,\ldots,\alpha_\mathcal{K}(q)}$ starting from q such that $\forall\, i, j \in [0\,..\,\alpha_\mathcal{K}(q)]$, if $i \neq j$ then $s_i \neq s_j$ (i.e. the longest *simple* path in the underlying graph structure). We may omit the index $\mathcal{K}$ whenever contextually obvious.

Note that two states may generate the same runs despite their computation trees not corresponding. It is therefore necessary to define an equivalence relation on states of KSs that goes further than mere run equality.

Definition 2 (Bisimulation relation). *Let $\mathcal{K} = (Q, \delta, \lambda)$ be a KS on AP. The* canonical bisimulation *relation $\sim \subseteq Q \times Q$ is the coarsest (i.e. the most general) equivalence relation such that for any $q_1 \sim q_2$, $\lambda(q_1) = \lambda(q_2)$ and $\forall\, q_1' \in \delta(q_1), \exists\, q_2' \in \delta(q_2)$ such that $q_1' \sim q_2'$.*

Bisimilarity does not only entails equality of runs, but also similarity of shape: two bisimilar states have corresponding computation trees at any depth. A partition refinement algorithm allows one to compute $\sim$ by refining a sequence of equivalence relations $(\sim_i)_{i \geq 0}$ on $Q \times Q$ inductively, where for every $q_1, q_2 \in Q$:

$$\begin{aligned} q_1 \sim_0 q_2 &\iff \lambda(q_1) = \lambda(q_2) \\ q_1 \sim_{i+1} q_2 &\iff (q_1 \sim_i q_2) \wedge (\{[q_1']_{\sim_i} \mid q_1' \in \delta(q_1)\} = \{[q_2']_{\sim_i} \mid q_2' \in \delta(q_2)\}) \end{aligned}$$

Where $[q]_{\sim_i}$ stands for the equivalence class of $q \in Q$ according to the equivalence relation $\sim_i$. Intuitively, $q_1 \sim_i q_2$ if their computation trees are corresponding up to depth i. The next theorem is a well-known result [1, Alg. 31]:

Theorem 1 (Characteristic number). *Given a KS $\mathcal{K}$, there exists $i_0 \in \mathbb{N}$ such that $\forall\, i \geq i_0$, $\sim\, = \sim_i$. The smallest integer i_0 verifying that property is known as the* characteristic number $\mathcal{C}_\mathcal{K}$ *of $\mathcal{K}$.*

Note that Browne et al. [6] introduced an equivalent definition: the characteristic number of a KS is also the smallest integer $\mathcal{C}_\mathcal{K} \in \mathbb{N}$ such that any two states are not bisimilar if and only if their labelled computation trees of depth $\mathcal{C}_\mathcal{K}$ are not corresponding.

2.2 Computation Tree Logic

Definition 3 (Computation Tree Logic). Computation Tree Logic *(CTL) is the set of formulas defined by the following grammar, where $a \in \mathsf{AP}$ is any atomic proposition and $\dagger \in \{\forall, \exists\}$ a quantifier:*

$$\varphi ::= a \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \dagger\, \mathsf{X}\, \varphi \mid \dagger\, \mathsf{F}\, \varphi \mid \dagger\, \mathsf{G}\, \varphi \mid \dagger\, \varphi\, \mathsf{U}\, \varphi$$

Given $E \subseteq \{\neg, \wedge, \vee, \forall\mathsf{X}, \exists\mathsf{X}, \forall\mathsf{F}, \exists\mathsf{F}, \forall\mathsf{G}, \exists\mathsf{G}, \forall\mathsf{U}, \exists\mathsf{U}\}$, we define the (syntactic) fragment *$\mathsf{CTL}(E)$ as the subset of CTL formulas featuring only operators in E.*

CTL formulas are verified against states of KSs (a process known as *model checking*). Intuitively, $\forall$ (*all*) means that all runs starting from state q must verify the property that follows, $\exists$ (*exists*), that at least one run starting from q must verify the property that follows, $\mathsf{X}\,\varphi$ (*next*), that the next state of the run must verify φ, $\mathsf{F}\,\varphi$ (*finally*), that there exists a state of run verifying φ, $\mathsf{G}\,\varphi$ (*globally*), that each state of the run must verify φ, and $\varphi\,\mathsf{U}\,\psi$ (*until*), that the run must keep verifying φ at least until ψ is eventually verified.

More formally, for a state $q \in Q$ of a KS $\mathcal{K} = (Q, \delta, \lambda)$ and a CTL formula φ, we write $(q \models_\mathcal{K} \varphi)$ when $\mathcal{K}$ *satisfies* φ. CTL's semantics are defined inductively on φ (see [1, Def. 6.4] for a complete definition); we recall below the *until* case:

Definition 4 (Semantics of $\forall\mathsf{U}, \exists\mathsf{U}$). *Let $\mathcal{K} = (Q, \delta, \lambda)$ be a KS, φ and ψ two CTL formulas, $q \in Q$, and $\dagger \in \{\forall, \exists\}$. Then:*

$$q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}\, \psi \iff \dagger(s_i) \in \mathcal{R}_{\mathcal{K}}(q), \exists\, i \geq 0, (s_i \models_{\mathcal{K}} \psi) \wedge (\forall\, j < i, s_j \models_{\mathcal{K}} \varphi)$$

Bisimilarity and CTL equivalence coincide [1, Thm. 7.20] on finite KSs. The proof relies on the following concept:

Theorem 2 (Browne et al. [6]). *Given a KS $\mathcal{K} = (Q, \delta, \lambda)$ and a state $q \in Q$, there exists a CTL formula $\varphi_q \in \mathit{CTL}(\{\neg, \wedge, \vee, \forall\mathsf{X}, \exists\mathsf{X}\})$ known as the* master formula *of state q such that, for any $q' \in Q$, $q' \models_{\mathcal{K}} \varphi_q$ if and only if $q \sim q'$.*

To each CTL formula φ, we associate a syntactic tree $\mathcal{T}$. For brevity's sake, we consider a syntactic *directed acyclic graph* (DAG) $\mathcal{D}$ by coalescing identical subtrees in the original syntactic tree $\mathcal{T}$, as shown in Figure 1. The *size* $|\varphi|$ of a CTL formula φ is then defined as the number of nodes of its smallest syntactic DAG. As an example, $|\neg a \wedge \forall\mathsf{X}\, a| = 4$.

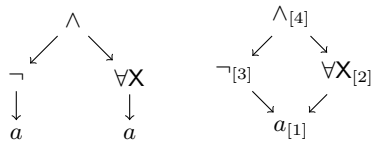


Fig. 1. The syntactic tree and indexed DAG of the CTL formula $\neg a \wedge \forall\mathsf{X}\, a$.

2.3 Bounded semantics

We introduce the bounded temporal operators $\forall\mathsf{F}^u$, $\exists\mathsf{F}^u$, $\forall\mathsf{G}^u$, $\exists\mathsf{G}^u$, $\forall\mathsf{U}^u$, and $\exists\mathsf{U}^u$, whose semantics only applies to the first u steps of a run. Formally:

Definition 5 (Bounded semantics of CTL). *Let $\mathcal{K} = (Q, \delta, \lambda)$ be a KS, φ and ψ two CTL formulas, $u \in \mathbb{N}$ and $q \in Q$. The bounded semantics of CTL of rank u with regards to $\mathcal{K}$ are defined as follows for the quantifier $\dagger \in \{\forall, \exists\}$:*

$$\begin{aligned}
q \models_{\mathcal{K}} \dagger\, \mathsf{F}^u\, \varphi &\iff \dagger(s_i) \in \mathcal{R}_{\mathcal{K}}(q), \exists\, i \in [0\,..\,u], s_i \models_{\mathcal{K}} \varphi \\
q \models_{\mathcal{K}} \dagger\, \mathsf{G}^u\, \varphi &\iff \dagger(s_i) \in \mathcal{R}_{\mathcal{K}}(q), \forall\, i \in [0\,..\,u], s_i \models_{\mathcal{K}} \varphi \\
q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^u\, \psi &\iff \dagger(s_i) \in \mathcal{R}_{\mathcal{K}}(q), \exists\, i \in [0\,..\,u], (s_i \models_{\mathcal{K}} \psi) \wedge (\forall\, j < i, s_j \models_{\mathcal{K}} \varphi)
\end{aligned}$$

Intuitively, the rank u of the bounded semantics acts as a timer: $(q \models_{\mathcal{K}} \forall\mathsf{G}^u\, \varphi)$ means that φ must hold for the next u computation steps; $(q \models_{\mathcal{K}} \forall\mathsf{F}^u\, \varphi)$, that q must always be able to reach a state verifying φ within u steps; $(q \models_{\mathcal{K}} \forall\, \varphi\, \mathsf{U}^u\, \psi)$, that q must always be able to reach a state verifying ψ within u steps, and that φ must hold until it does; etc. This intuition results in the following properties:

Property 1 (Base case). $(q \models_{\mathcal{K}} \psi) \iff (q \models_{\mathcal{K}} \dagger\, \mathsf{F}^0\, \psi) \iff (q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^0\, \psi) \iff (q \models_{\mathcal{K}} \dagger\, \mathsf{G}^0\, \psi)$.

Property 2 (Induction). $(q \models_{\mathcal{K}} \dagger\, \mathsf{F}^{u+1}\, \varphi) \iff (q \models_{\mathcal{K}} \varphi) \vee \underset{q' \in \delta(q)}{\triangle} (q' \models_{\mathcal{K}} \dagger\, \mathsf{F}^u\, \varphi)$,

$(q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^{u+1}\, \psi) \iff (q \models_{\mathcal{K}} \psi) \vee \left[(q \models_{\mathcal{K}} \varphi) \wedge \underset{q' \in \delta(q)}{\triangle} (q' \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^u\, \psi)\right]$, and

$(q \models_{\mathcal{K}} \dagger\, \mathsf{G}^{u+1}\, \varphi) \iff (q \models_{\mathcal{K}} \varphi) \wedge \underset{q' \in \delta(q)}{\triangle} (q' \models_{\mathcal{K}} \dagger\, \mathsf{G}^{u}\, \varphi)$, where $\triangle = \wedge$ if $\dagger = \forall$ and $\triangle = \vee$ if $\dagger = \exists$.

Property 3 (Spread). $(q \models_{\mathcal{K}} \dagger\, \mathsf{F}^{u}\, \varphi) \implies (q \models_{\mathcal{K}} \dagger\, \mathsf{F}^{u+1}\, \varphi)$, $(q \models_{\mathcal{K}} \dagger\, \mathsf{G}^{u+1}\, \varphi) \implies (q \models_{\mathcal{K}} \dagger\, \mathsf{G}^{u}\, \varphi)$, and $(q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^{u}\, \psi) \implies (q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^{u+1}\, \psi)$.

Bounded model checking algorithms [25] rely on the following result, as one can then restrict the study of CTL semantics to finite and fixed length paths.

Theorem 3. *Given $q \in Q$, for $\dagger \in \{\forall, \exists\}$ and $\triangleright \in \{\mathsf{F}, \mathsf{G}\}$, $q \models_{\mathcal{K}} \dagger \triangleright \varphi$ (resp. $q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}\, \psi$) if and only if $q \models_{\mathcal{K}} \dagger \triangleright^{\alpha(q)} \varphi$ (resp. $q \models_{\mathcal{K}} \dagger\, \varphi\, \mathsf{U}^{\alpha(q)}\, \psi$).*

A full proof of this result is available in Appendix A.

3 The Learning Problem

We consider the synthesis problem of a distinguishing CTL formula from a sample of positive and negative states of a given KS.

3.1 Introducing the problem

First and foremost, the sample must be self-consistent: a state in the positive sample cannot verify a CTL formula while another bisimilar state in the negative sample does not.

Definition 6 (Sample). *Given a KS $\mathcal{K} = (Q, \delta, \lambda)$, a* sample *of $\mathcal{K}$ is a pair $(S^+, S^-) \in 2^Q \times 2^Q$ such that $\forall\, q^+ \in S^+$, $\forall\, q^- \in S^-$, $q^+ \not\sim q^-$.*

We define the *characteristic number* $\mathcal{C}_{\mathcal{K}}(S^+, S^-)$ of a sample as the smallest integer $c \in \mathbb{N}$ such that for every $q^+ \in S^+$, $q^- \in S^-$, $q^+ \not\sim_c q^-$.

Definition 7 (Consistent formula). *A CTL formula φ is said to be* consistent *with a sample (S^+, S^-) of $\mathcal{K}$ if $\forall\, q^+ \in S^+$, $q^+ \models_{\mathcal{K}} \varphi$ and $\forall\, q^- \in S^-$, $q^- \not\models_{\mathcal{K}} \varphi$.*

The rest of our article focuses on the following passive learning problems:

Definition 8 (Learning problem). *Given a sample (S^+, S^-) of a KS $\mathcal{K}$ and $n \in \mathbb{N}^*$, we introduce the following instances of the CTL learning problem:*

$\mathsf{L}_{\mathsf{CTL}(E)}(\mathcal{K}, S^+, S^-)$**.** *Is there $\varphi \in \mathsf{CTL}(E)$ consistent with (S^+, S^-)?*
$\mathsf{L}^{\leq n}_{\mathsf{CTL}(E)}(\mathcal{K}, S^+, S^-)$**.** *Is there $\varphi \in \mathsf{CTL}(E)$, $|\varphi| \leq n$, consistent with (S^+, S^-)?*
$\mathsf{ML}_{\mathsf{CTL}(E)}(\mathcal{K}, S^+, S^-)$**.** *Find the* smallest *$\varphi \in \mathsf{CTL}(E)$ consistent with (S^+, S^-).*

Theorem 4. *$\mathsf{L}_{\mathsf{CTL}}(\mathcal{K}, S^+, S^-)$ and $\mathsf{ML}_{\mathsf{CTL}}(\mathcal{K}, S^+, S^-)$ always admit a solution.*

Proof. Consider $\psi = \bigvee_{q^+ \in S^+} \varphi_{q^+}$. This formula ψ is consistent with (S^+, S^-) by design. Thus $\mathsf{L}_{\mathsf{CTL}}(\mathcal{K}, S^+, S^-)$ always admits a solution, and so does the problem $\mathsf{ML}_{\mathsf{CTL}}(\mathcal{K}, S^+, S^-)$, although ψ is unlikely to be the minimal solution. □

Bordais et al. [5] proved that $\mathsf{L}^{\leq n}_{\mathsf{CTL}}(\mathcal{K}, S^+, S^-)$ is NP-hard, assuming the set of atomic propositions AP is given as an input as well.

3.2 An explicit solution

We must find a formula consistent with the sample (S^+, S^-), an easier problem than Browne et al. [6]'s answer to Theorem 2 that subsumes bisimilarity with an entire KS. As we know that every state in S^- is dissimilar to every state in S^+, we will try to encode this fact in CTL form, then use said encoding to design a formula consistent with the sample.

Definition 9 (Separating formula). *Let (S^+, S^-) be a sample of a KS $\mathcal{K} = (Q, \delta, \lambda)$. Assuming that AP and Q are ordered, and given $q_1, q_2 \in Q$ such that $q_1 \not\sim q_2$, formula D_{q_1,q_2} is defined inductively w.r.t. $c = \mathcal{C}_\mathcal{K}(\{q_1\}, \{q_2\})$ as follows:*

- *if $c = 0$ and $\lambda(q_1) \setminus \lambda(q_2) \neq \emptyset$ has minimal element a, then $D_{q_1,q_2} = a$;*
- *else if $c = 0$ and $\lambda(q_2) \setminus \lambda(q_1) \neq \emptyset$ has minimal element a, then $D_{q_1,q_2} = \neg a$;*
- *else if $c \neq 0$ and $\exists\, q_1' \in \delta(q_1)$, $\forall\, q_2' \in \delta(q_2)$, $q_1' \not\sim_{c-1} q_2'$, then $D_{q_1,q_2} = \exists \mathsf{X} \left(\bigwedge_{q_2' \in \delta(q_2)} D_{q_1',q_2'} \right)$, picking the smallest q_1' verifying this property;*
- *else if $c \neq 0$ and $\exists\, q_2' \in \delta(q_2)$, $\forall\, q_1' \in \delta(q_1)$, $q_1' \not\sim_{c-1} q_2'$, then $D_{q_1,q_2} = \forall \mathsf{X} \neg \left(\bigwedge_{q_1' \in \delta(q_1)} D_{q_2',q_1'} \right)$, picking the smallest q_2' verifying this property.*

The formula $\mathcal{S}_\mathcal{K}(S^+, S^-) = \bigvee_{q^+ \in S^+} \bigwedge_{q^- \in S^-} D_{q^+,q^-} \in \mathsf{CTL}(\{\neg, \wedge, \vee, \forall \mathsf{X}, \exists \mathsf{X}\})$ is then called the separating formula *of sample (S^+, S^-).*

Intuitively, the CTL formula D_{q_1,q_2} merely expresses that states q_1 and q_2 are dissimilar by negating Definition 2; it is such that $q_1 \models_\mathcal{K} D_{q_1,q_2}$ but $q_2 \not\models_\mathcal{K} D_{q_1,q_2}$. Either q_1 and q_2 have different labels, q_1 admits a successor that is dissimilar to q_2's successors, or q_2 admits a successor that is dissimilar to q_1's. The following result is proven in Appendix B:

Theorem 5. *The separating formula $\mathcal{S}_\mathcal{K}(S^+, S^-)$ is consistent with (S^+, S^-).*

As proven in Appendix C, we can bound the size of $\mathcal{S}_\mathcal{K}(S^+, S^-)$:

Corollary 1. *Assume the KS $\mathcal{K}$ has degree k and $c = \mathcal{C}_\mathcal{K}(S^+, S^-)$, then:*

- *if $k \geq 2$, then $|\mathcal{S}_\mathcal{K}(S^+, S^-)| \leq (5 \cdot k^c + 1) \cdot |S^+| \cdot |S^-|$;*
- *if $k = 1$, then $|\mathcal{S}_\mathcal{K}(S^+, S^-)| \leq (2 \cdot c + 3) \cdot |S^+| \cdot |S^-|$.*

4 SAT-based Learning

The *universal* fragment $\mathsf{CTL}_\forall = \mathsf{CTL}(\{\neg, \wedge, \vee, \forall \mathsf{X}, \forall \mathsf{F}, \forall \mathsf{G}, \forall \mathsf{U}\})$ of CTL happens to be as expressive as the full logic [1, Def. 6.13]. For that reason, we will reduce a learning instance of $\mathsf{CTL}_\forall$ of rank n to an instance of the SAT problem. A similar reduction has been independently found by Roy et al. [22].

Lemma 1. *There exists a Boolean propositional formula Φ_n such that the instance $\mathsf{L}^{\leq n}_{\mathsf{CTL}_\forall}(\mathcal{K}, S^+, S^-)$ of the learning problem admits a solution φ if and only if the formula Φ_n is satisfiable.*

4.1 Modelling the formula

Assume that there exists a syntactic DAG $\mathcal{D}$ of size smaller than or equal to n representing the desired CTL formula φ. Let us index $\mathcal{D}$'s nodes in $[1 \,..\, n]$ in such a fashion that each node has a higher index than its children, as shown in Figure 1. Hence, n always labels a root and 1 always labels a leaf.

Let $\mathcal{L} = \mathsf{AP} \cup \{\top, \neg, \wedge, \vee, \forall\mathsf{X}, \forall\mathsf{F}, \forall\mathsf{G}, \forall\mathsf{U}\}$ be the set of labels that decorates the DAG's nodes. For each $i \in [1 \,..\, n]$ and $o \in \mathcal{L}$, we introduce a Boolean variable τ_i^o such that $\tau_i^o = 1$ if and only if the node of index i is labelled by o.

For all $i \in [1 \,..\, n]$ and $j \in [0 \,..\, i-1]$, we also introduce a Boolean variable $\mathsf{l}_{i,j}$ (resp. $\mathsf{r}_{i,j}$) such that $\mathsf{l}_{i,j} = 1$ (resp. $\mathsf{r}_{i,j} = 1$) if and only if j is the left (resp. right) child of i. Having a child of index 0 stands for having no child at all in the actual syntactic DAG $\mathcal{D}$.

Three mutual exclusion clauses guarantee that each node of the syntactic DAG has exactly one label and at most one left child and one right child. Moreover, three other clauses ensure that a node labelled by an operator of arity x has exactly x actual children (by convention, if $x = 1$ then its child is to the right). These simple clauses are similar to Neider et al.'s encoding [16] and for that reason are not detailed here.

4.2 Applying the formula to the sample

For all $i \in [1 \,..,n]$ and $q \in Q$, we introduce a Boolean variable φ_i^q such that $\varphi_i^q = 1$ if and only if state q verifies the sub-formula φ_i rooted in node i. The next clauses implement the semantics of the true symbol $\top$, the atomic propositions, and the CTL operator $\forall\mathsf{X}$.

$$\bigwedge_{\substack{i\in[1\,..n]\\ q\in Q}} (\tau_i^{\top} \implies \varphi_i^q) \tag{$\mathtt{sem}_{\top}$}$$

$$\bigwedge_{\substack{i\in[1\,..n]\\ q\in Q}} \left[\left(\bigwedge_{a\in\lambda(q)} (\tau_i^a \implies \varphi_i^q)\right) \wedge \left(\bigwedge_{a\notin\lambda(q)} (\tau_i^a \implies \neg\varphi_i^q)\right)\right] \tag{$\mathtt{sem}_a$}$$

$$\bigwedge_{\substack{i\in[2\,..n]\\ k\in[1\,..i-1]}} \left[(\tau_i^{\forall\mathsf{X}} \wedge \mathsf{r}_{i,k}) \implies \bigwedge_{q\in Q}\left(\varphi_i^q \iff \bigwedge_{q'\in\delta(q)} \varphi_k^{q'}\right)\right] \tag{$\mathtt{sem}_{\forall\mathsf{X}}$}$$

Semantic clauses are structured as follows: an antecedent stating node i's label and its possible children implies a consequent expressing φ_i^q's semantics for each $q \in Q$. Clause $\mathtt{sem}_{\top}$ states that $q \models \top$ is always true; clause $\mathtt{sem}_a$, that $q \models a$ if and only if q is labelled by a; and clause $\mathtt{sem}_{\forall\mathsf{X}}$, that $q \models \forall\mathsf{X}\,\psi$ if and only if all of q's successors verify ψ. Similar straightforward clauses encode the semantics of the Boolean connectors $\neg$, $\wedge$ and $\vee$.

CTL semantics are also characterized by fixed points, whose naive encoding might however capture spurious (least vs greatest) sets: we resort to the bounded semantics $\forall\mathsf{F}^u$, $\forall\mathsf{U}^u$ and $\forall\mathsf{G}^u$. For all $i \in [1 \,..\, n]$, $q \in Q$, and $u \in [0 \,..\, \alpha(q)]$, we introduce a Boolean variable $\rho_{i,q}^u$ such that $\rho_{i,q}^u = 1$ if and only if q verifies the

sub-formula rooted in i according to the CTL bounded semantics of rank u (e.g. $q \models \forall \mathsf{F}^u \psi$, assuming sub-formula $\forall \mathsf{F} \psi$ is rooted in node i).

Thanks to Theorem 3 we can introduce the following equivalence clause:

$$\bigwedge_{i\in[2..n]} \left[\left(\bigvee_{o\in\{\forall \mathsf{F},\forall \mathsf{G},\forall \mathsf{U}\}} \tau_i^o \right) \implies \bigwedge_{q\in Q} (\varphi_i^q \iff \rho_{i,q}^{\alpha(q)}) \right] \qquad (\mathtt{sem}_\rho)$$

Property 3 yields two other clauses whose inclusion is not mandatory (they were left out by Roy et al. [22]) that further constrains the bounded semantics:

$$\bigwedge_{i\in[2..n]} \left[(\tau_i^{\forall \mathsf{F}} \vee \tau_i^{\forall \mathsf{U}}) \implies \bigwedge_{\substack{q\in Q \\ u\in[1..\alpha(q)]}} (\rho_{i,q}^{u-1} \implies \rho_{i,q}^{u}) \right] \qquad (\mathtt{ascent}_\rho)$$

$$\bigwedge_{i\in[2..n]} \left[\tau_i^{\forall \mathsf{G}} \implies \bigwedge_{\substack{q\in Q \\ u\in[1..\alpha(q)]}} (\rho_{i,q}^{u} \implies \rho_{i,q}^{u-1}) \right] \qquad (\mathtt{descent}_\rho)$$

The next clause enables variable $\rho_{i,q}^u$ for temporal operators only:

$$\bigwedge_{i\in[2..n]} \left[\left(\bigwedge_{o\in\{\forall \mathsf{F},\forall \mathsf{G},\forall \mathsf{U}\}} \neg\tau_i^o \right) \implies \bigwedge_{\substack{q\in Q \\ u\in[0..\alpha(q)]}} \neg\rho_{i,q}^{u} \right] \qquad (\mathtt{no}_\rho)$$

Properties 1 and 2 yield an inductive definition of bounded semantics. We only explicit the base case $\mathtt{base}_\rho$ and the semantics $\mathtt{sem}_{\forall \mathsf{U}}$ of $\forall \mathsf{U}^u$, but also implement semantic clauses for the temporal operators $\forall \mathsf{F}$ and $\forall \mathsf{G}$.

$$\bigwedge_{\substack{i\in[2..n] \\ k\in[1..i-1]}} \left[\left(\left[\bigvee_{o\in\{\forall \mathsf{F},\forall \mathsf{G},\forall \mathsf{U}\}} \tau_i^o \right] \wedge \mathsf{r}_{i,k} \right) \implies \bigwedge_{q\in Q} (\rho_{i,q}^0 \iff \varphi_k^q) \right] \qquad (\mathtt{base}_\rho)$$

$$\bigwedge_{\substack{i\in[2..n] \\ j,k\in[1..i-1]}} \left[(\tau_i^{\forall \mathsf{U}} \wedge \mathsf{l}_{i,j} \wedge \mathsf{r}_{i,k}) \implies \bigwedge_{\substack{q\in Q \\ u\in[1..\alpha(q)]}} \left(\rho_{i,q}^u \iff \left[\varphi_k^q \vee \left(\varphi_j^q \wedge \bigwedge_{q'\in\delta(q)} \rho_{i,q'}^{\min(\alpha(q'),u-1)} \right) \right] \right) \right] \qquad (\mathtt{sem}_{\forall \mathsf{U}})$$

Finally, the last clause ensures that the full formula φ (rooted in node n) is verified by the positive sample but not by the negative sample.

$$\left(\bigwedge_{q^+\in S^+} \varphi_n^{q^+} \right) \wedge \left(\bigwedge_{q^-\in S^-} \neg\varphi_n^{q^-} \right) \qquad (\mathtt{sem}_\varphi)$$

4.3 Solving the SAT instance

We finally define the formula Φ_n as the conjunction of all the aforementioned clauses. Assuming an upper bound d on the KS's recurrence diameter, this encoding requires $\mathcal{O}(n^2 + n \cdot |\mathsf{AP}| + n \cdot |Q| \cdot d)$ variables and $\mathcal{O}(n \cdot |\mathsf{AP}| + n^3 \cdot |Q| \cdot d + n \cdot |\mathsf{AP}| \cdot |Q|)$ clauses, not taking transformation to conjunctive normal form into account. By design, Lemma 1 holds.

Proof. The syntactic clauses allow one to infer the DAG of a formula $\varphi \in \mathsf{CTL}$ of size smaller than or equal to n from the valuations taken by the variables (τ_i^o), $(\mathsf{l}_{i,j})$, and $(\mathsf{r}_{i,j})$. Clauses $\mathtt{sem}_a$ to $\mathtt{sem}_\varphi$ guarantee that the sample is consistent with said formula φ, thanks to Theorem 3 and Properties 1, 2, and 3. □

5 Algorithms for the Minimal Learning Problem

We introduce in this section an algorithm to solve the minimum learning problem $\mathsf{ML}_{\mathsf{CTL}_\forall}(\mathcal{K}, S^+, S^-)$. Remember that it always admits a solution if and only if the state sample is consistent by Theorem 4.

5.1 A bottom-up algorithm

By Theorem 4, there exists a rank n_0 such that the problem $\mathsf{L}^{\leq n_0}_{\mathsf{CTL}_\forall}(\mathcal{K}, S)$ admits a solution. Starting from $n = 0$, we can therefore try to solve $\mathsf{L}^{\leq n}_{\mathsf{CTL}_\forall}(\mathcal{K}, S)$ incrementally until a (minimal) solution is found, in a similar manner to Neider and Gavran [16]. Algorithm 1 terminates with an upper bound n_0 on the number of required iterations.

Input: a KS $\mathcal{K}$ and a sample S.
Output: the smallest $\mathsf{CTL}_\forall$ formula φ consistent with S.
$n \leftarrow 0$;
repeat
  $n \leftarrow n + 1$;
  compute Φ_n;
until Φ_n *is satisfiable by some valuation* v;
from v build and **return** φ.

Algorithm 1: Solving $\mathsf{ML}_{\mathsf{CTL}_\forall}(\mathcal{K}, S)$.

5.2 Embedding negations

The CTL formula $\exists \mathsf{F}\, a$ is equivalent to the $\mathsf{CTL}_\forall$ formula $\neg\, \forall \mathsf{G}\, \neg a$, yet the former remains more succinct, being of size 2 instead of 4. While $\mathsf{CTL}_\forall$ has been proven to be as expressive as CTL, the sheer amount of negations needed to express an equivalent formula can significantly burden the syntactic DAG. A possible optimization is to no longer consider the negation $\neg$ as an independent operator but instead embed it in the nodes of the syntactic DAG, as shown in Figure 2.

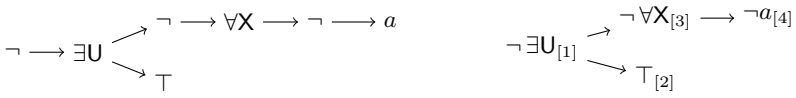


Fig. 2. The syntactic DAG of $\neg\exists \top\, \mathsf{U}\, \neg\, \forall \mathsf{X}\, \neg a$, before and after embedding negations.

Note that such a definition of the syntactic DAG alters one's interpretation of a CTL formula's size: as a consequence, under this optimization, Algorithm 1 may yield a formula with many negations that is no longer minimal under the original definition of size outlined in Section 2.2.

Formally, for each $i \in [1 \,..\, n]$, we introduce a new variable ν_i such that $\nu_i = 0$ if and only if the node of index i is negated. As an example, in Figure 2, $\nu_1 = \nu_3 = \nu_4 = 0$, but $\nu_2 = 1$ and the sub-formula rooted in node 3 is $\neg\forall\mathsf{X}\neg a$.

We then change the SAT encoding of $\mathsf{CTL}_\forall$'s semantics accordingly. We remove the $\neg$ operator from the syntactic DAG clauses and the set $\mathcal{L}$ of labels. We delete its semantics and update the semantic clauses of the other operators. Indeed, the right side of each equivalence expresses the semantics of the operator rooted in node i *before* applying the embedded negation; we must therefore change the left side of the semantic equivalence accordingly, replacing the Boolean variable φ_i^q with the formula $\tilde{\varphi}_i^q = (\neg\nu_i \wedge \neg\varphi_i^q) \vee (\nu_i \wedge \varphi_i^q)$ that is equivalent to φ_i^q if $\nu_i = 1$ and $\neg\varphi_i^q$ if $\nu_i = 0$.

5.3 Optimizations and alternatives

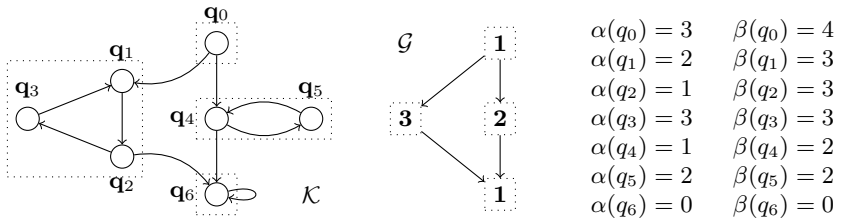
Minimizing the input KS. In order to guarantee that an input S is indeed a valid sample, one has to ensure no state in the positive sample is bisimilar to a state in the negative sample. To do so, one has to at least partially compute the bisimilarity relation $\sim$ on $\mathcal{K} = (Q, \delta, \lambda)$. But refining it to completion can be efficiently performed in $\mathcal{O}(|Q| \cdot |\mathsf{AP}| + |\delta| \cdot \log(|Q|))$ operations [1, Thm. 7.41], yielding a bisimilar KS $\mathcal{K}_{\min}$ of minimal size.

Minimizing the input KS is advantageous as the size of the semantic clauses depends on the size of $\mathcal{K}$, and the SAT solving step is likely to be the computational bottleneck. As a consequence, we always fully compute the bisimulation relation $\sim$ on $\mathcal{K}$ and minimize said KS.

Approximating the recurrence diameter. Computing the recurrence diameter of a state q is unfortunately an NP-hard problem that is known to be hard to approximate [4]. A coarse upper bound is $\alpha(q) \leq |Q| - 1$: it may however result in a significant number of unnecessary variables and clauses. Fortunately, the decomposition of a KS $\mathcal{K}$ into strongly connected components (SCCs) yields a finer over-approximation shown in Figure 3 that relies on the ease of computing α in a DAG. It is also more generic and suitable to CTL than existing approximations dedicated to LTL bounded model checking [14].

Contracting each SCC to a single vertex yields a DAG known as the *condensation* of $\mathcal{K}$. We weight each vertex of this DAG $\mathcal{G}$ with the number of vertices in the matching SCC. Then, to each state q in the original KS $\mathcal{K}$, we associate the weight $\beta(q)$ of the longest path in the DAG $\mathcal{G}$ starting from q's SCC, minus one (in order not to count q). Intuitively, our approximation assumes that a simple path entering a SCC can always visit every single one of its states once before exiting, a property that obviously does not hold for two of the SCCs shown here.

Encoding the full logic. $\mathsf{CTL}_\forall$ is semantically exhaustive but the existential temporal operators commonly appear in the literature; we can therefore consider the learning problem on the full CTL logic by integrating the operators $\exists\mathsf{X}$, $\exists\mathsf{F}$, $\exists\mathsf{G}$, and $\exists\mathsf{U}$ and provide a Boolean encoding of their semantics. We also consider the fragment $\mathsf{CTL}_\mathsf{U} = \{\neg, \vee, \exists\mathsf{X}, \exists\mathsf{G}, \exists\mathsf{U}\}$ used by Roy et al. [22].



$\alpha(q_0) = 3 \quad \beta(q_0) = 4$
$\alpha(q_1) = 2 \quad \beta(q_1) = 3$
$\alpha(q_2) = 1 \quad \beta(q_2) = 3$
$\alpha(q_3) = 3 \quad \beta(q_3) = 3$
$\alpha(q_4) = 1 \quad \beta(q_4) = 2$
$\alpha(q_5) = 2 \quad \beta(q_5) = 2$
$\alpha(q_6) = 0 \quad \beta(q_6) = 0$

Fig. 3. An approximation β of the recurrence diameter α relying on SCC decomposition that improves upon the coarse upper bound $\alpha(q) \leq |Q| - 1 = 6$.

6 Experimental Implementation

We implement our learning algorithm in a `C++` prototype tool `LearnCTL`[4], relying on Microsoft's `Z3` due to its convenient `C++` API. It takes as an input a sample of positive and negative KSs with initial states, then coalesced into a single KS, and a sample of states compatible with the theoretical framework we described. It finally returns a separating $\mathsf{CTL}_\forall$, CTL, or CTL_U formula after a sanity check performed by model-checking the input KSs against the learnt formula, using a simple algorithm based on Theorem 3.

6.1 Benchmark Collection

We intend on using our tool to generate formulas that can explain flaws in faulty implementations of known protocols. To do so, we consider structures generated by higher formalisms such as program graphs: a single mutation in the program graph results in several changes in the resulting KS. This process has been achieved manually according to the following criteria:

- The mutations only consist in deleting lines.
- The resulting KS should be *small*, less than ~ 1000 states.
- Any mutation should result in a syntactically correct model.

We collected program graphs in a toy specification language for a CTL model checker class implemented in Java. Furthermore, we also considered PROMELA models from the `Spin` model-checker [12] repository. Translations were then performed through the Python interface of `spot/ltsmin` [9, 13].

Example 1. Consider the mutual exclusion protocol proposed by [18] and specified in PROMELA in Figure 4 that generates a KS with 55 states. We generate mutants by deleting no more than one line of code at a time, ignoring variable and process declarations as they are necessary for the model to be compiled and the two assertion lines that are discarded by our KS generator, our reasoning being that subtle changes yield richer distinguishing formulas.

[4] publicly available at https://gitlab.lre.epita.fr/adrien/learnctl

Furthermore, removing the instruction `ncrit--` alone would lead to an infinite state space; thus, its deletion is only considered together with the instruction `ncrit++`. Finally, we set some atomic propositions of interest: c stands for at least one process being in the critical section (`ncrit>0`), m for both processes (`ncrit>1`), and t for process 0's turn. An extra *dead* atomic proposition is added by Spot/LTSMin to represent deadlocked states.

As summarized on Figure 4, every mutated model, once compared with the original KS, lead to distinguishing formulas characterizing Peterson's protocol: mutations `m1`, `m2`, and `m3` yield a mutual exclusion property, `m4` yields a liveness property, `m5` yields a fairness property, and `m6` yields global liveness formula.

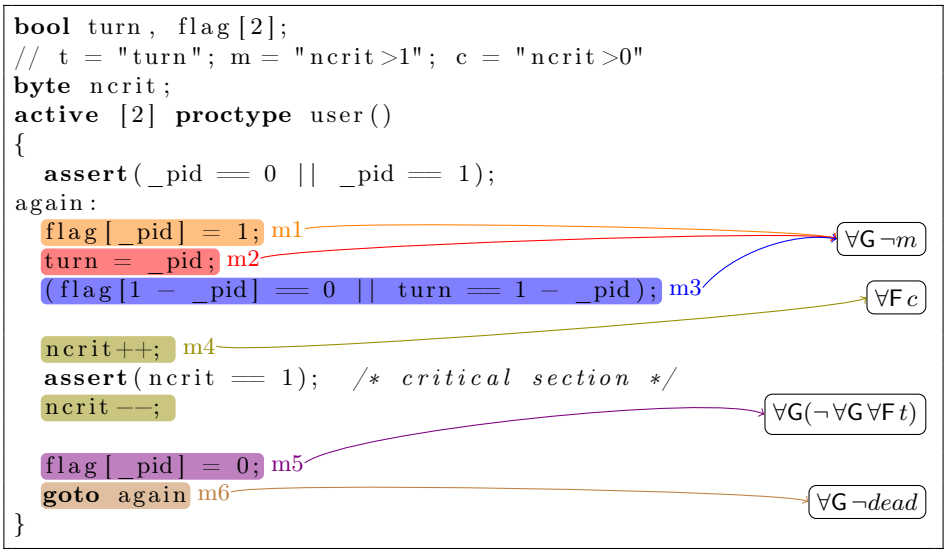


Fig. 4. Peterson's mutual exclusion protocol in PROMELA and learnt formulas for each deleted instruction.

6.2 Quantitative evaluation

We quantitatively assess the performance of the various optimizations and CTL fragments discussed previously. To do so, we further enrich the benchmark series through the use of random mutations of hard-coded KSs: these mutations may alter some states, re-route some edges, and spawn new states. We consider a total of 234 test samples, ranging from size 11 to 698 after minimization. We perform the benchmarks on a GNU/Linux Debian machine (`bullseye`) with 24 cores (Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz) and 256Go of RAM, using version `4.8.10` of `libz3` and `1.0` of `LearnCTL`.

Table 1 displays a summary of these benchmarks: β stands for the refined approximation of the recurrence diameter described in Section 5.3; $\neg$, for the embedding of negations in the syntactic tree introduced in Section 5.2. The average size of the syntactic DAGs learnt is 4.14.

Option β yields the greatest improvement, being on average at least 6 times faster than the default configuration; option $\neg$ further divides the average runtime by at least 2. These two optimizations alone speed up the average runtime by a factor of 12 to 20. The CTL fragment used, all other options being equal, does not influence the average runtime as much (less than twofold in the worst case scenario); $(\mathsf{CTL}_\mathsf{U}, \beta, \neg)$ is the fastest option, closely followed by $(\mathsf{CTL}_\forall, \beta, \neg)$.

Intuitively, approximating the recurrence diameter aggressively cuts down the number of SAT variables needed: assuming that α has upper bound d, we only need $n \cdot |Q| \cdot d$ Boolean variables ($\rho^u_{i,q}$) instead of $n \cdot |Q|^2$. Moreover, embedding negations, despite requiring more complex clauses, results in smaller syntactic DAGs with "free" negations, hence faster computations, keeping in mind that the last SAT instances are the most expensive to solve, being the largest.

	-	β	$\beta, \neg$
$\mathsf{CTL}_\forall$	50 | 46870	14 | 6493	4 | 2271
CTL	50 | 42658	8 | 5357	5 | 3370
CTL_U	46 | 31975	28 | 5064	4 | 1987

Table 1. Number of timeouts at ten minutes | arithmetic mean (in milliseconds) on the 178 samples that never timed out of various options and fragments.

Figure 5 further displays a log-log plot comparing the runtime of the most relevant fragments and options to $(\mathsf{CTL}_\mathsf{U}, \beta, \neg)$. For a given set of parameters, each point stands for one of the 234 test samples. Points above the black diagonal favour $(\mathsf{CTL}_\mathsf{U}, \beta, \neg)$; points below, the aforementioned option. Points on the second dotted lines at the edges of the figure represent timeouts.

Unsurprisingly, $(\mathsf{CTL}_\forall, \beta, \neg)$ and $(\mathsf{CTL}, \beta, \neg)$ outperform $(\mathsf{CTL}_\mathsf{U}, \beta, \neg)$ when a minimal distinguishing formula using the operator $\forall\mathsf{U}$ exists: the duality between $\forall\mathsf{U}$ and $\exists\mathsf{U}$ is complex and, unlike the other operators, cannot be handled at no cost by the embedded negation as it depends on the *release* operator.

7 Conclusion and Further Developments

We explored in this article the CTL learning problem: we first provided a direct explicit construction before relying on a SAT encoding inspired by bounded model-checking to iteratively find a minimal answer. We also introduced in Section 3 an explicit answer to the learning problem that belongs to the fragment $\mathsf{CTL}(\neg, \wedge, \vee, \forall\mathsf{X}, \exists\mathsf{X})$. It remains to be seen if a smaller formula can be computed using a more exhaustive selection of CTL operators. A finer grained explicit solution could allow one to experiment with a top-down approach as well.

Moreover, we provided a dedicated `C++` implementation, and evaluated it on models of higher-level formalisms such as PROMELA. Since the resulting KSs have large state spaces, further symbolic approaches are to be considered for future work, when dealing with program graphs instead of Kripke structures.

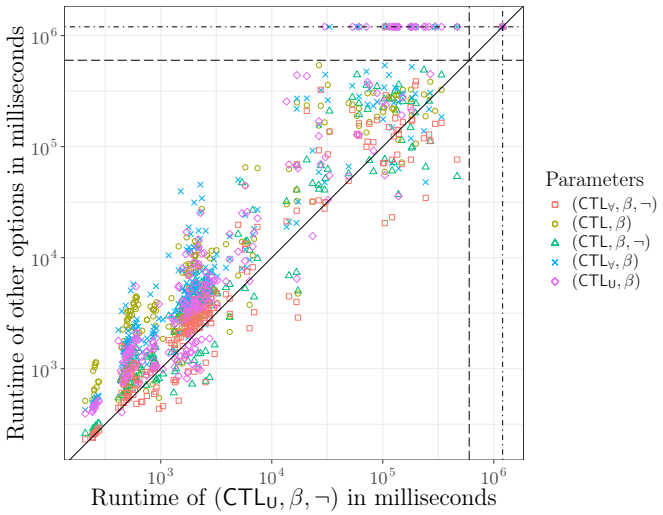


Fig. 5. Comparing $(\mathsf{CTL}_\mathsf{U}, \beta, \neg)$ to other options on every sample.

In this setting, one might also consider the synthesis problem of the relevant atomic propositions from the exposed program variables. Nevertheless, the experiments on Kripke structures already showcase the benefits of the approximated recurrence diameter computation and of our extension of the syntactic DAG definition, as well as the limited relevance of the target CTL fragment.

Another avenue for optimizations can be inferred from the existing SAT-based LTL learning literature: in particular, Rienier et al. [20] relied on a topology-guided approach by explicitly enumerating the possible shapes of the syntactic DAG and solving the associated SAT instances in parallel. Given the small size on average of the formulas learnt so far and the quadratic factor impacting the number of semantic clauses such as $\mathtt{sem}_{\forall\mathsf{U}}$ due to the structural variables $\mathsf{l}_{i,j}$ and $\mathsf{r}_{i,k}$, this approach could yield huge performance gains in CTL's case as well.

We relied on Z3's convenient C++ API, but intuit that we would achieve better performance with state-of-the-art SAT solvers such as the winners of the yearly SAT competition [2]. We plan on converting our Boolean encoding to the DIMACS CNF format in order to interface our tool with modern SAT solvers.

Finally, it is known that the bounded learning problem is NP-complete, but we would also like to find the exact complexity class of the minimal CTL learning problem. We intuit that it is not, Kripke structures being a denser encoding in terms of information than lists of linear traces: as an example, one can trivially compute an LTL formula (resp. a CTL formula) of polynomial size that distinguishes a sample of ultimately periodic words (resp. of finite computation trees with lasso-shaped leaves), but the same cannot be said of a sample of Kripke structures. It remains to be seen if this intuition can be confirmed or infirmed by a formal proof.

References

1. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
2. Balyo, T., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.): Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions. Department of Computer Science Series of Publications B, Department of Computer Science, University of Helsinki, Finland (2023)
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bdds. In: Cleaveland, W.R. (ed.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 193–207. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
4. Bjørklund, A., Husfeldt, T., Khanna, S.: Approximating longest directed paths and cycles. In: Automata, Languages and Programming. pp. 222–233. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
5. Bordais, B., Neider, D., Roy, R.: Learning temporal properties is np-hard (2023)
6. Browne, M., Clarke, E., Grümberg, O.: Characterizing finite kripke structures in propositional temporal logic. Theoretical Computer Science **59**(1), 115–131 (1988). https://doi.org/https://doi.org/10.1016/0304-3975(88)90098-9, https://www.sciencedirect.com/science/article/pii/0304397588900989
7. Camacho, A., McIlraith, S.A.: Learning interpretable models expressed in linear temporal logic. Proceedings of the International Conference on Automated Planning and Scheduling **29**(1), 621–630 (May 2021). https://doi.org/10.1609/icaps.v29i1.3529, https://ojs.aaai.org/index.php/ICAPS/article/view/3529
8. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. pp. 52–71. Springer Berlin Heidelberg, Berlin, Heidelberg (1982)
9. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22). Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (Aug 2022). https://doi.org/10.1007/978-3-031-13188-2_9
10. Fijalkow, N., Lagarde, G.: The complexity of learning linear temporal formulas from examples. In: Chandlee, J., Eyraud, R., Heinz, J., Jardine, A., van Zaanen, M. (eds.) Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 153, pp. 237–250. PMLR (2021), https://proceedings.mlr.press/v153/fijalkow21a.html
11. Gaglione, J.R., Neider, D., Roy, R., Topcu, U., Xu, Z.: Learning linear temporal properties from noisy data: A maxsat-based approach. In: Hou, Z., Ganesh, V. (eds.) Automated Technology for Verification and Analysis. pp. 74–90. Springer International Publishing, Cham (2021)
12. Holzmann, G.J.: The SPIN Model Checker - primer and reference manual. Addison-Wesley (2004)
13. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: Ltsmin: High-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of System - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. Lecture Notes in Computer Science,

vol. 9035, pp. 692–707. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_61, https://doi.org/10.1007/978-3-662-46681-0_61
14. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: Zuck, L.D., Attie, P.C., Cortesi, A., Mukhopadhyay, S. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 298–309. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
15. Lutz, S., Neider, D., Roy, R.: Specification sketching for linear temporal logic. In: André, É., Sun, J. (eds.) Automated Technology for Verification and Analysis. pp. 26–48. Springer Nature Switzerland, Cham (2023)
16. Neider, D., Gavran, I.: Learning linear temporal properties. In: Bjørner, N.S., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018. pp. 1–10. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8603016, https://doi.org/10.23919/FMCAD.2018.8603016
17. Penczek, W., Woźna, B., Zbrzezny, A.: Bounded model checking for the universal fragment of ctl. Fundam. Inf. **51**(1–2), 135–156 (jan 2002)
18. Peterson, G.L.: Myths about the mutual exclusion problem. Inf. Process. Lett. **12**(3), 115–116 (1981). https://doi.org/10.1016/0020-0190(81)90106-X, https://doi.org/10.1016/0020-0190(81)90106-X
19. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp. 46–57 (1977). https://doi.org/10.1109/SFCS.1977.32
20. Riener, H.: Exact synthesis of ltl properties from traces. In: 2019 Forum for Specification and Design Languages (FDL). pp. 1–6 (2019). https://doi.org/10.1109/FDL.2019.8876900
21. Roy, R., Fisman, D., Neider, D.: Learning interpretable models in the property specification language. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. IJCAI'20 (2021)
22. Roy, R., Neider, D.: Inferring properties in computation tree logic (2023)
23. Wasylkowski, A., Zeller, A.: Mining temporal specifications from object usage. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. pp. 295–306 (Nov 2009). https://doi.org/10.1109/ASE.2009.30
24. Xu, L., Chen, W., Xu, Y.Y., Zhang, W.H.: Improved bounded model checking for the universal fragment of ctl. Journal of Computer Science and Technology **24**(1), 96–109 (Jan 2009). https://doi.org/10.1007/s11390-009-9208-5, https://doi.org/10.1007/s11390-009-9208-5
25. Zhang, W.: Bounded semantics of ctl and sat-based verification. In: Breitman, K., Cavalcanti, A. (eds.) Formal Methods and Software Engineering. pp. 286–305. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

A Proof of Theorem 3

$\forall \mathsf{F}$. Assume that $q \models \forall \mathsf{F}\, \varphi$. Let us prove that $q \models \forall \mathsf{F}^{\alpha(q)}\, \varphi$. Consider a run $r = (s_i) \in \mathcal{R}(q)$. By hypothesis, we can define the index $j = \min\{i \in \mathbb{N} \mid s_i \models \varphi\}$.

Now, assume that $j > \alpha(q)$. By definition of the recurrence diameter α, $\exists\, k_1, k_2 \in [0\,..\,j-1]$ such that $k_1 \leq k_2$ and $s_{k_1} = s_{k_2}$. Consider the finite runs $u = (s_i)_{i \in [0\,..\,k_1]}$ and $v = (s_i)_{i \in [k_1+1\,..\,k_2]}$. We define the infinite, ultimately periodic run $r' = u \cdot v^\omega = (s'_i)$. By definition of j, $\forall\, i \in \mathbb{N}$, $s'_i \not\models \varphi$ in order to preserve the minimality of j. Yet $r' \in \mathcal{R}(q)$ and $q \models \forall \mathsf{F}\, \varphi$. By contradiction, $j \leq \alpha(q)$. As consequence, $(q \models \forall \mathsf{F}\, \varphi) \implies (q \models \forall \mathsf{F}^{\alpha(q)}\, \varphi)$ holds.

Trivially, $(q \models \forall \mathsf{F}^{\alpha(q)}\, \varphi) \implies (q \models \forall \mathsf{F}\, \varphi)$ holds. Hence, we have proven both sides of the desired equivalence for $\forall \mathsf{F}$.

$\forall \mathsf{G}$. Assume that $q \models \forall \mathsf{G}^{\alpha(q)}\, \varphi$. Let us prove that $q \models \forall \mathsf{G}\, \varphi$. Consider a run $r = (s_i) \in \mathcal{R}(q)$ and $j \in \mathbb{N}$. Let us prove that $s_j \models \varphi$.

State s_j is obviously reachable from q. Let us consider a finite run without repetition $u = (s'_i)_{i \in [0\,..\,k]}$ such that $s_0 = q$ and $s'_k = s_j$. By definition of the recurrence diameter, $k \leq \alpha(q)$. We define the infinite runs $v = (s_i)_{i > j}$ and $r' = u \cdot v$. Since $r' \in \mathcal{R}(q)$ and $q \models \forall \mathsf{G}^{\alpha(q)}\, \varphi$, $s_k \models \varphi$, hence $s_j \models \varphi$. As a consequence, $(q \models \forall \mathsf{G}^{\alpha(q)}\, \varphi) \implies (q \models \forall \mathsf{G}\, \varphi)$.

Trivially, $(q \models \forall \mathsf{G}\, \varphi) \implies (q \models \forall \mathsf{G}^{\alpha(q)}\, \varphi)$ holds. Hence, we have proven both sides of the desired equivalence for $\forall \mathsf{G}$.

$\exists \mathsf{F}$ *and* $\exists \mathsf{G}$. Formula $\exists \mathsf{F}\, \varphi$ (rep. $\exists \mathsf{G}\, \varphi$) being equivalent to the dual formula $\neg \forall \mathsf{G}\, \neg \varphi$ (resp. $\neg \forall \mathsf{F}\, \neg \varphi$), the previous proofs immediately yield the desired equivalences.

$\forall \mathsf{U}$ *and* $\exists \mathsf{U}$. We can handle the case of $\forall\, \varphi\, \mathsf{U}\, \psi$ in a manner similar to $\forall \mathsf{F}$: we prove by contradiction that the first occurrence of ψ always happens in less than $\alpha(q)$ steps. And the semantic equivalence for $\exists\, \varphi\, \mathsf{U}\, \psi$ can be handled in a fashion similar to $\forall \mathsf{G}$: an existing infinite run yields a conforming finite prefix without repetition of length lesser than or equal to $\alpha(q)$.

B Proof of Theorem 5

Given two dissimilar states $q_1, q_2 \in Q$, let us prove by induction on the characteristic number $c_{q_1,q_2} = \mathcal{C}_{\mathcal{K}}(\{q_1\}, \{q_2\})$ that $q_1 \models D_{q_1,q_2}$ and $q_2 \not\models D_{q_1,q_2}$.

Base case. If $c_{q_1,q_2} = 0$, then by definition $\lambda(q_1) \neq \lambda(q_2)$ and by design D_{q_1,q_2} is a literal (i.e. an atomic proposition or the negation thereof) verified by q_1 but not by q_2.

Inductive case. Assume that the property holds for all characteristic numbers smaller than or equal to $c \in \mathbb{N}$. Consider two states $q_1, q_2 \in Q$ such that $c_{q_1,q_2} = c + 1$. By definition of the refined equivalence relation $\sim_{c+1}$, $\exists\, q'_1 \in \delta(q_1)$, $\forall\, q'_2 \in \delta(q_2)$, $q'_1 \not\sim_c q'_2$, hence $c_{q'_1,q'_2} \leq c$.

By induction hypothesis, $D_{q'_1,q'_2}$ is well-defined, $q'_1 \models D_{q'_1,q'_2}$ and $q'_2 \not\models D_{q'_1,q'_2}$. As a consequence, D_{q_1,q_2} is well-defined, $q_1 \models \exists \mathsf{X} \left(\bigwedge_{q'_2 \in \delta(q_2)} D_{q'_1,q'_2} \right)$, and $q_2 \not\models \exists \mathsf{X} \left(\bigwedge_{q'_2 \in \delta(q_2)} D_{q'_1,q'_2} \right)$.

We handle the case where $\exists\, q'_2 \in \delta(q_2)$, $\forall\, q'_1 \in \delta(q_1)$, $q'_1 \not\sim_c q'_2$ in a similar fashion. As a consequence, the property holds at rank $c+1$.

Therefore, for each $q^+ \in S^+$ and each $q^- \in S^-$, $q^+ \models D_{q^+,q^-}$ and $q^- \not\models D_{q^+,q^-}$. Hence, $q^+ \models \mathcal{S}_{\mathcal{K}}(S^+, S^-)$ and $q^- \not\models \mathcal{S}_{\mathcal{K}}(S^+, S^-)$. □

C Proof of Corollary 1

First, given $q^+ \in S^+$ and $q^- \in S^-$, let us bound the size of D_{q^+,q^-} based on their characteristic number $c_{q_1,q_2} = \mathcal{C}_{\mathcal{K}}(\{q_1\}, \{q_2\})$.

$$
\begin{aligned}
c_{q_1,q_2} = 0 &\implies |D_{q^+,q^-}| \leq 2 && \text{as } \lambda(q_1) \neq \lambda(q_2) \\
c_{q_1,q_2} \geq 1 &\implies |D_{q^+,q^-}| \leq (k+1) + \sum_{q'_2 \in \delta(q_2)} |D_{q'_1,q'_2}| && \text{for some } q'_1 \in \delta(q_1) \\
&\quad \text{or } |D_{q^+,q^-}| \leq (k+1) + \sum_{q'_1 \in \delta(q_1)} |D_{q'_1,q'_2}| && \text{for some } q'_2 \in \delta(q_2)
\end{aligned}
$$

We are looking for an upper bound $(U_n)_{n \geq 0}$ such that $\forall\, n \in \mathbb{N}$, $\forall\, q^+ \in S^+$, $\forall\, q^- \in S^-$, if $c_{q^+,q^-} \leq n$, then $|D_{q^+,q^-}| \leq U_n$. We define it inductively:

$$
\begin{aligned}
U_0 &= 2 \\
U_{n+1} &= k \cdot U_n + k + 1
\end{aligned}
$$

Assuming $k \geq 2$, we explicit the bound $U_n = (2 + \frac{k+1}{k-1}) \cdot k^n - \frac{k+1}{k-1} \leq 5 \cdot k^n$. As $(\{q^+\}, \{q^-\})$ is a sub-sample of S, $c_{q^+,q^-} \leq c$ and $|D_{q^+,q^-}| \leq U_c \leq 5 \cdot k^c$. We can finally bound the size of $\mathcal{S}_{\mathcal{K}}(S^+, S^-)$:

$$
\begin{aligned}
|\mathcal{S}_{\mathcal{K}}(S^+, S^-)| &\leq (|S^+| - 1) \cdot (|S^-| - 1) + \sum_{\substack{q^+ \in S^+ \\ q^- \in S^-}} |D_{q^+,q^-}| \\
&\leq |S^+| \cdot |S^-| + |S^+| \cdot |S^-| \cdot U_c \\
&\leq (5 \cdot k^c + 1) \cdot |S^+| \cdot |S^-|
\end{aligned}
$$

Yielding the aforementioned upper bound. If $k = 1$, then $U_n = 2 \cdot n + 2$ and the rest of the proof is similar to the previous case. □