Learning Computation Tree Logic

A SAT-based approach

Adrien Pommellet, Daniel Stan, Simon Scatton, LRE

January 16, 2025

Passive Learning

Also known as mining

What? Computing a theoretical model compatible with a labelled dataset (e.g. desirable and undesirable behaviours).

[1] Daniel Neider and Ivan Gavran. "Learning Linear Temporal Properties". In: *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*. Ed. by Nikolaj S. Bjørner and Arie Gurfinkel. IEEE, 2018, pp. 1–10. DOI: 10.23919/FMCAD.2018.8603016. URL: https://doi.org/10.23919/FMCAD.2018.8603016.

Passive Learning

Also known as mining

What? Computing a theoretical model compatible with a labelled dataset (e.g. desirable and undesirable behaviours).

Why? Explainability, extrapolate tests, system design.

[1] Daniel Neider and Ivan Gavran. "Learning Linear Temporal Properties". In: *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*. Ed. by Nikolaj S. Bjørner and Arie Gurfinkel. IEEE, 2018, pp. 1–10. DOI: 10.23919/FMCAD.2018.8603016. URL: https://doi.org/10.23919/FMCAD.2018.8603016.

Passive Learning

Also known as mining

What? Computing a theoretical model compatible with a labelled dataset (e.g. desirable and undesirable behaviours).

Why? Explainability, extrapolate tests, system design.

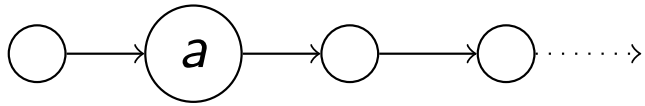
How? Automata, **logical formulas**[1].

[1] Daniel Neider and Ivan Gavran. "Learning Linear Temporal Properties". In: *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*. Ed. by Nikolaj S. Bjørner and Arie Gurfinkel. IEEE, 2018, pp. 1–10. DOI: 10.23919/FMCAD.2018.8603016. URL: https://doi.org/10.23919/FMCAD.2018.8603016.

Computation Tree Logic

Linear time logic

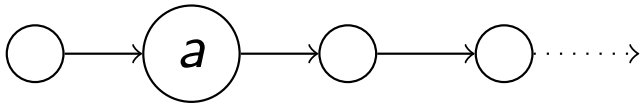
$\mathsf{X}\, a$



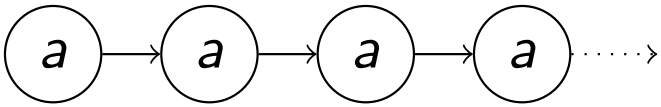
Computation Tree Logic

Linear time logic

X a



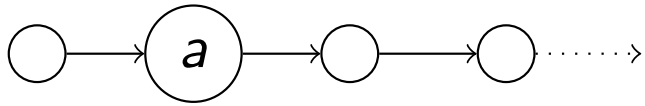
G a



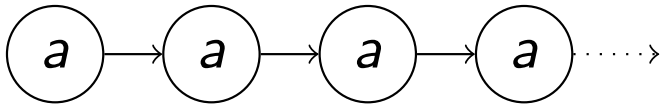
Computation Tree Logic

Linear time logic

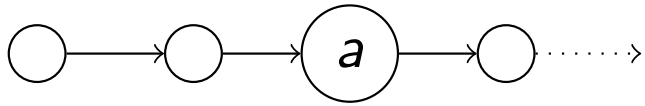
X a



G a



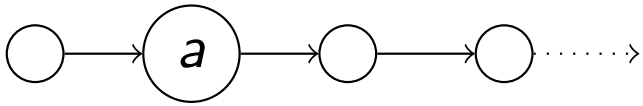
F a



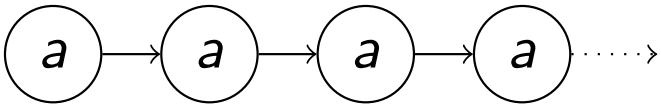
Computation Tree Logic

Linear time logic

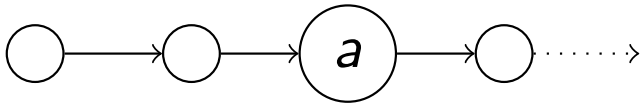
X a



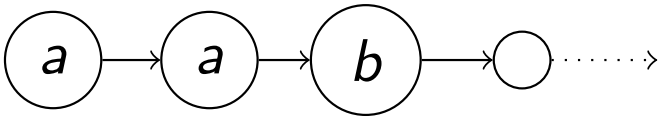
G a



F a

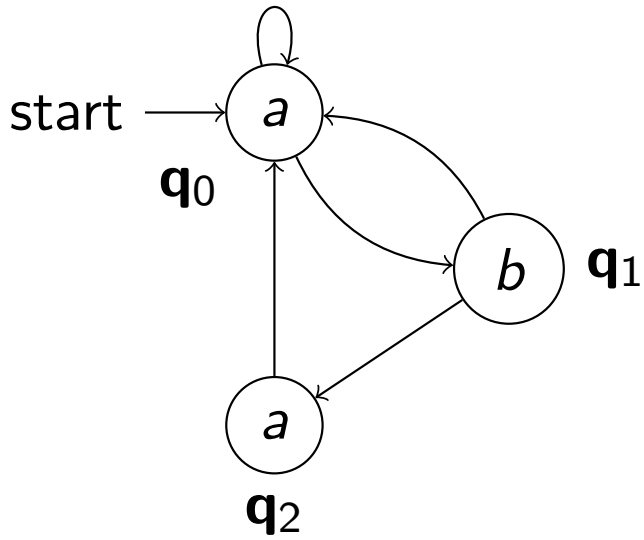


a U b



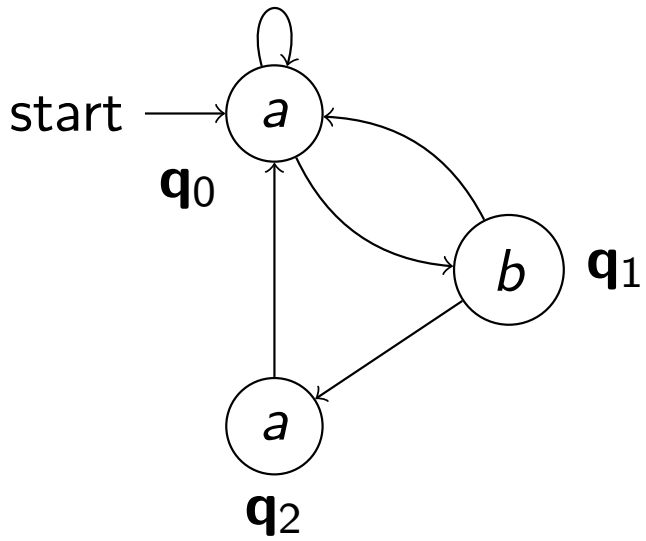
Computation Tree Logic

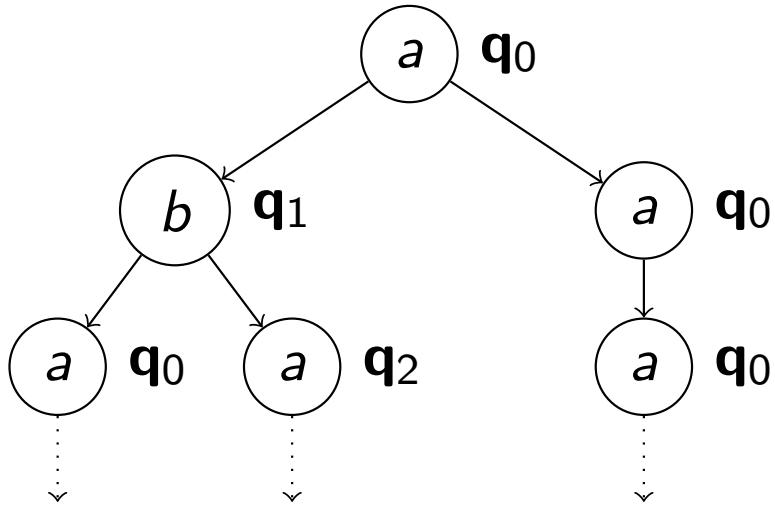
Kripke structures and execution trees



Computation Tree Logic

Kripke structures and execution trees

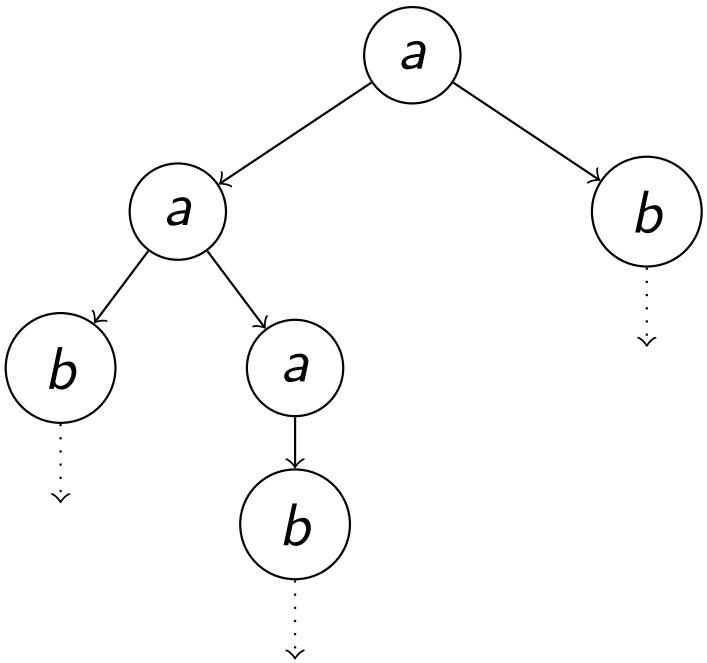




Computation Tree Logic

Universal and existential quantifiers

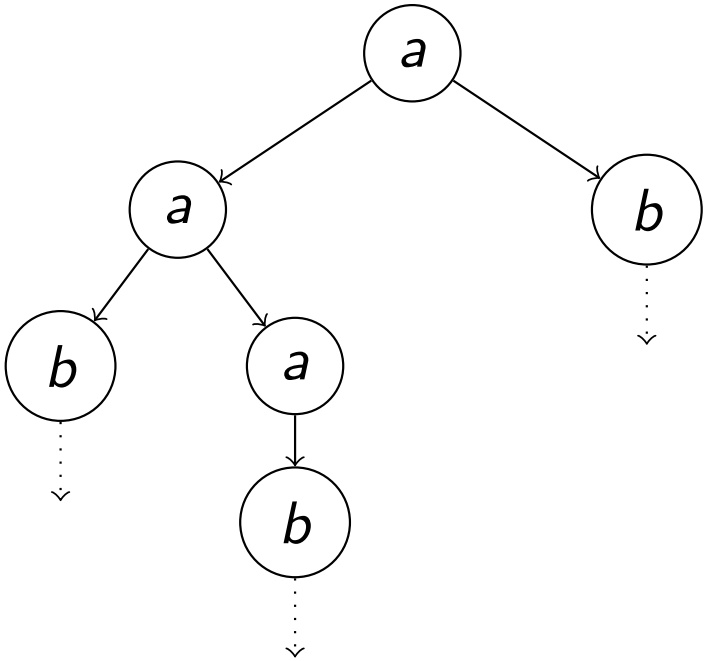
$\forall\, a \,\mathsf{U}\, b$



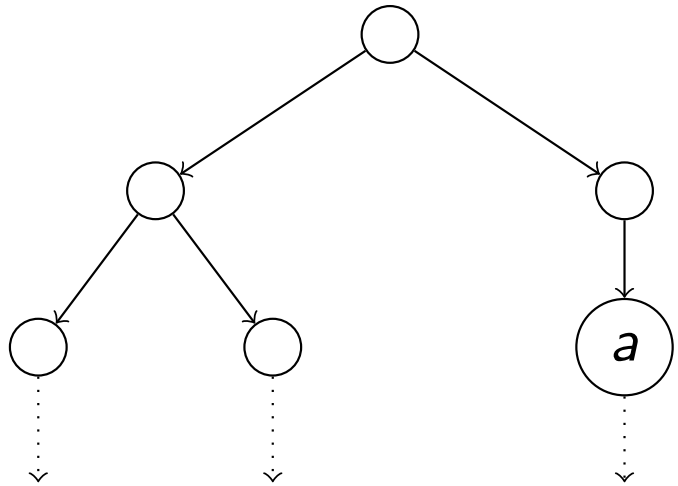
Computation Tree Logic

Universal and existential quantifiers

$\forall\, a \,\mathsf{U}\, b$



$\exists \mathsf{F}\, a$



Computation Tree Logic

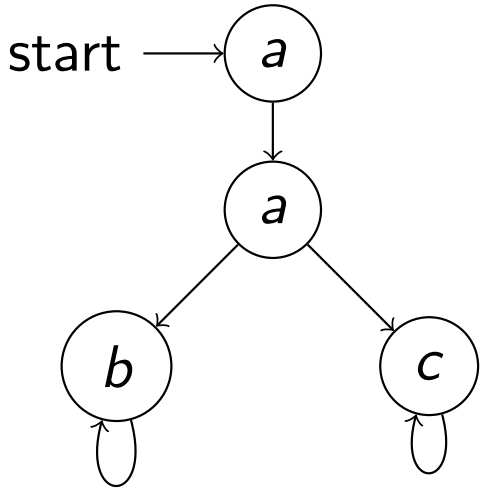
Incomparable logics

$$\forall \mathsf{X} \left[(\exists \mathsf{X} \ b) \wedge (\exists \mathsf{X} \ c) \right]$$

Computation Tree Logic

Incomparable logics

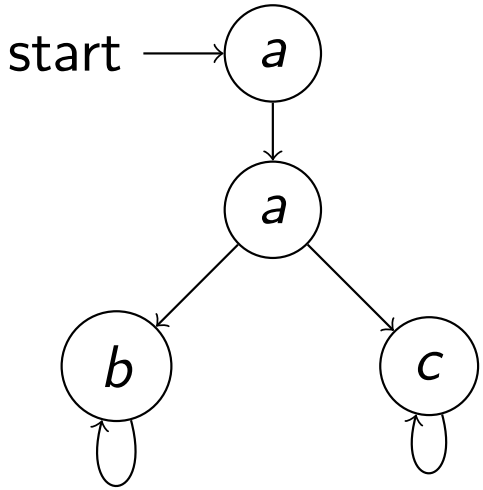
$$\forall \mathsf{X} \left[(\exists \mathsf{X} \ b) \wedge (\exists \mathsf{X} \ c) \right]$$

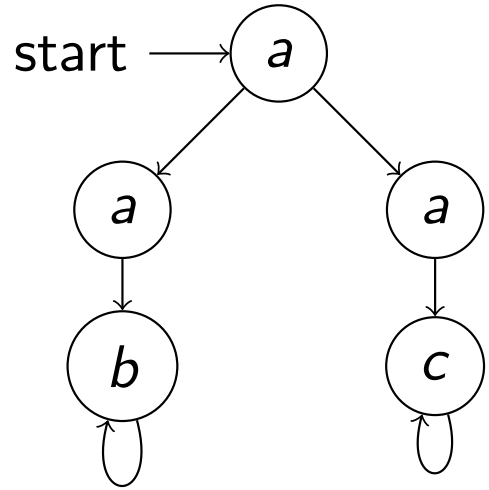


Computation Tree Logic

Incomparable logics

$$\forall \mathsf{X} \left[(\exists \mathsf{X}\ b) \wedge (\exists \mathsf{X}\ c) \right]$$

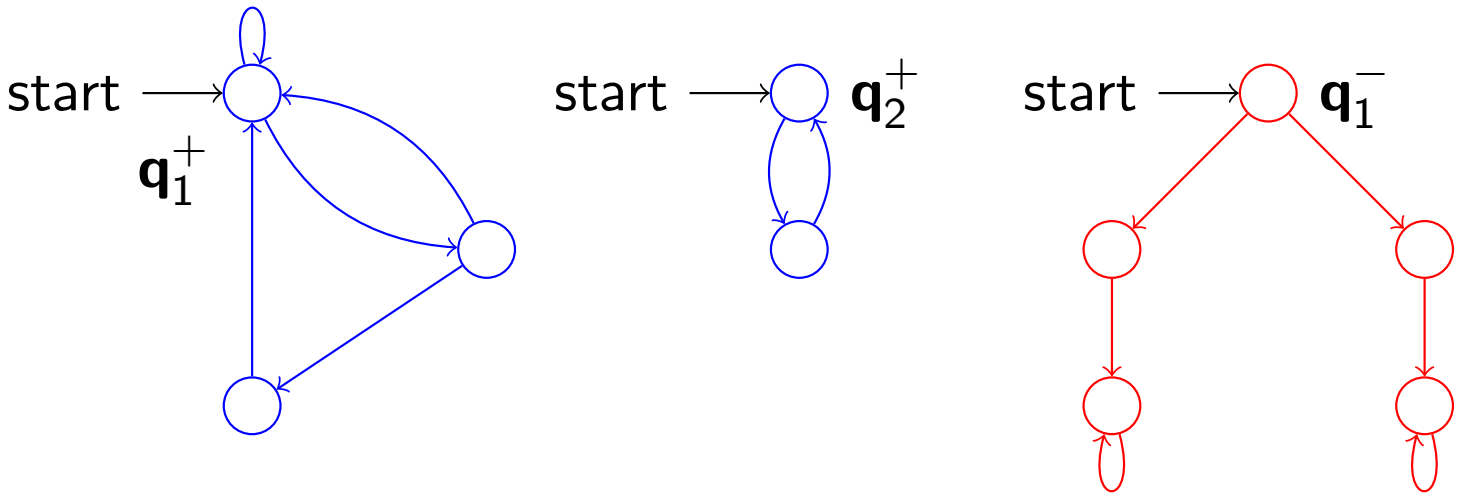




The Learning Problem

Sample of structures

A sample made of **dissimilar** positive and negative structures.



The Learning Problem

Formalizing various problems

The learning problem. Find a CTL formula φ verified by the positive sample and unanimously rejected by the negative sample.

The Learning Problem

Formalizing various problems

The learning problem. Find a CTL formula φ verified by the positive sample and unanimously rejected by the negative sample.

$\mathsf{L}_{\mathsf{CTL}}$. Any compatible formula φ will do.

The Learning Problem

Formalizing various problems

The learning problem. Find a CTL formula φ verified by the positive sample and unanimously rejected by the negative sample.

$\mathsf{L}_{\mathsf{CTL}}$. Any compatible formula φ will do.

$\mathsf{L}_{\mathsf{CTL}}^{\leq n}$. Formula φ must be of size $\leq n$.

The Learning Problem

Formalizing various problems

The learning problem. Find a CTL formula φ verified by the positive sample and unanimously rejected by the negative sample.

$\mathsf{L}_{\mathsf{CTL}}$. Any compatible formula φ will do.

$\mathsf{L}_{\mathsf{CTL}}^{\leq n}$. Formula φ must be of size $\leq n$.

$\mathsf{ML}_{\mathsf{CTL}}$. Formula φ must be of minimal size.

Theorem

L_{CTL} (thus ML_{CTL}) always admits a solution.

[2] M.C. Browne, E.M. Clarke, and O. Grümberg. "Characterizing finite Kripke structures in propositional temporal logic". In: *Theoretical Computer Science* 59.1 (1988), pp. 115–131. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(88)90098-9. URL: https://www.sciencedirect.com/science/article/pii/0304397588900989.

Theorem

L_{CTL} (thus ML_{CTL}) always admits a solution.

Proof. Consider a CTL formula that encodes the dissimilarity of the positive and the negative states.

[2] M.C. Browne, E.M. Clarke, and O. Grümberg. "Characterizing finite Kripke structures in propositional temporal logic". In: *Theoretical Computer Science* 59.1 (1988), pp. 115–131. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(88)90098-9. URL: https://www.sciencedirect.com/science/article/pii/0304397588900989.

Theorem

L_{CTL} (thus ML_{CTL}) always admits a solution.

Proof. Consider a CTL formula that encodes the dissimilarity of the positive and the negative states.

This explicit answer is however of size $\mathcal{O}(|S^+| \cdot |S^-| \cdot k^c)$ where k is the degree of $\mathcal{K}$ and c its characteristic number[2].

[2] M.C. Browne, E.M. Clarke, and O. Grümberg. "Characterizing finite Kripke structures in propositional temporal logic". In: *Theoretical Computer Science* 59.1 (1988), pp. 115–131. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(88)90098-9. URL: https://www.sciencedirect.com/science/article/pii/0304397588900989.

The Learning Problem

A hardness result

Bordais et al[3] proved that:

Theorem

$L_{CTL}^{\leq n}$ is NP-complete (assuming an unary encoding of n).

[3] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. *Learning Temporal Properties is NP-hard*. 2023. arXiv: 2312.11403 [cs.LO].

The Learning Problem

A hardness result

Bordais et al[3] proved that:

Theorem

$L_{CTL}^{\leq n}$ is NP-complete (assuming an unary encoding of n).

We will now design a SAT instance Φ_n equivalent to $\mathrm{L}_{\mathrm{CTL}}^{\leq n}$.

[3] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. *Learning Temporal Properties is NP-hard*. 2023. arXiv: 2312.11403 [cs.LO].

The Learning Problem

Solving the minimal learning problem

Input: a KS $\mathcal{K}$ and a sample S.
Output: the smallest CTL 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 $\mathsf{ML}_{\mathsf{CTL}}(\mathcal{K}, S)$.

A Boolean Encoding

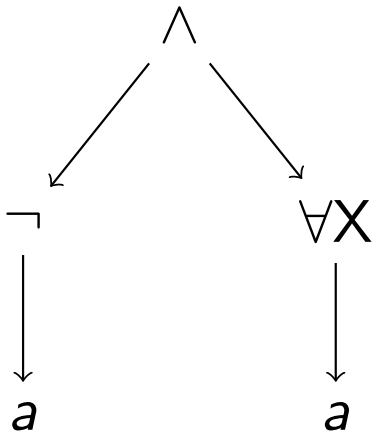
Representing CTL formulas

$$\neg a \wedge \forall \mathsf{X}\, a$$

A Boolean Encoding

Representing CTL formulas

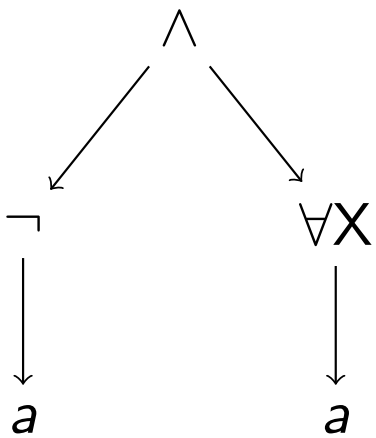
$$\neg a \wedge \forall \mathsf{X}\, a$$

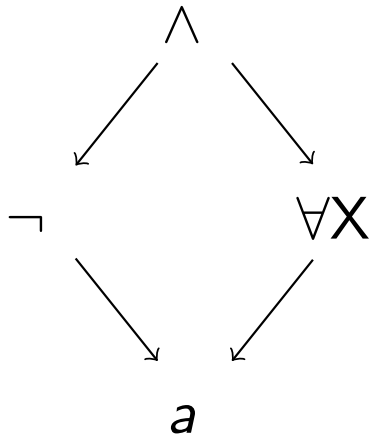


A Boolean Encoding

Representing CTL formulas

$$\neg a \wedge \forall \mathrm{X}\, a$$

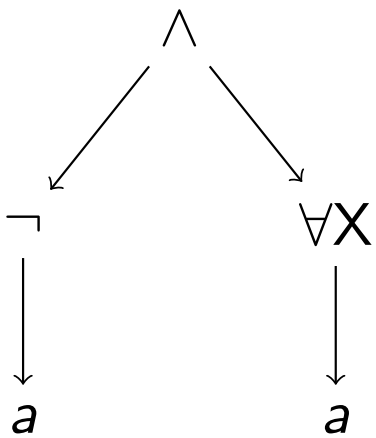


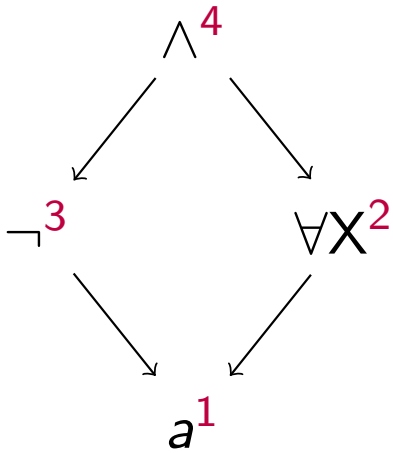


A Boolean Encoding

Representing CTL formulas

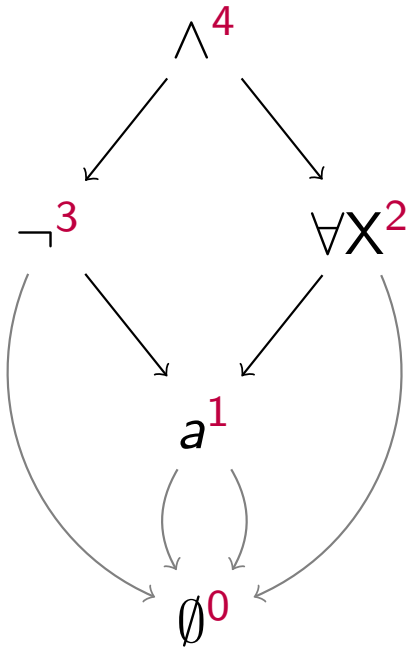
$$\neg a \wedge \forall \mathsf{X}\, a$$





A Boolean Encoding

A binary syntactic DAG



A Boolean Encoding

Encoding the syntactic tree

Variables.

- Node i has label o. τ_i^o

A Boolean Encoding

Encoding the syntactic tree

Variables.

- Node i has label o. τ_i^o
- The left (resp. right) children of i is j. $\mathsf{l}_{i,j}, \mathsf{r}_{i,j}$

A Boolean Encoding

Encoding the syntactic tree

Variables.

- Node i has label o. τ_i^o
- The left (resp. right) children of i is j. $\mathsf{l}_{i,j}, \mathsf{r}_{i,j}$

Clauses.

- Each node has a single label.

A Boolean Encoding

Encoding the syntactic tree

Variables.

- Node i has label o. τ_i^o
- The left (resp. right) children of i is j. $\mathsf{l}_{i,j}, \mathsf{r}_{i,j}$

Clauses.

- Each node has a single label.
- Each node has at most one left (resp. right) child.

A Boolean Encoding

Encoding the syntactic tree

Variables.

- Node i has label o. τ_i^o
- The left (resp. right) children of i is j. $\mathsf{l}_{i,j}, \mathsf{r}_{i,j}$

Clauses.

- Each node has a single label.
- Each node has at most one left (resp. right) child.
- Binary (resp. unary) operators have 2 (resp. 1) children.

A Boolean Encoding

Encoding the semantics

Variables.

- State q verifies sub-formula φ_i rooted in node i. φ_i^q

A Boolean Encoding

Encoding the semantics

Variables. • State q verifies sub-formula φ_i rooted in node i. φ_i^q

Clauses. • Each state in the positive sample verifies φ.

A Boolean Encoding

Encoding the semantics

Variables.

- State q verifies sub-formula φ_i rooted in node i. φ_i^q

Clauses.

- Each state in the positive sample verifies φ.
- Each state in the negative sample does not verify φ.

A Boolean Encoding

Encoding the semantics

Variables.

- State q verifies sub-formula φ_i rooted in node i. φ_i^q

Clauses.

- Each state in the positive sample verifies φ.
- Each state in the negative sample does not verify φ.
- If state q verifies φ_i, define φ_i's semantics according to the label of node i.

A Boolean Encoding

Encoding ∀X's semantics

$$(\tau_i^{\forall X} \wedge r_{i,j})$$

If node i is labelled by operator ∀X and has right child j

A Boolean Encoding

Encoding ∀X's semantics

$$(\tau_i^{\forall X} \wedge r_{i,j}) \implies \bigwedge_{q \in Q}$$

If node i is labelled by operator ∀X and has right child j then for every state q,

A Boolean Encoding

Encoding ∀X's semantics

$$(\tau_i^{\forall X} \wedge r_{i,j}) \implies \bigwedge_{q \in Q} (\varphi_i^q \qquad)$$

If node i is labelled by operator ∀X and has right child j

then for every state q,

if state q verifies formula φ_i,

A Boolean Encoding

Encoding ∀X's semantics

$$(\tau_i^{\forall X} \wedge r_{i,j}) \implies \bigwedge_{q \in Q} (\varphi_i^q \iff \bigwedge_{q' \in \delta(q)} \varphi_j^{q'})$$

If node i is labelled by operator ∀X and has right child j

then for every state q,

if state q verifies formula φ_i,

then every successor q' of q must verify φ_j.

A Boolean Encoding

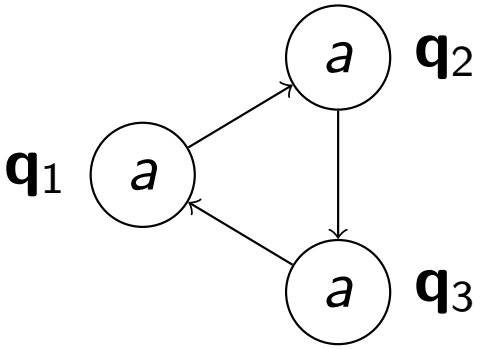
What about temporal operators?

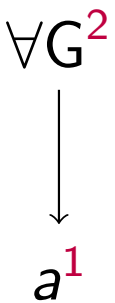
$$(q \models \forall \mathsf{G}\, a) \iff (q \models a) \wedge \bigwedge_{q' \in \delta(q)} (q' \models \forall \mathsf{G}\, a)$$

A Boolean Encoding

What about temporal operators?

$$(\tau_2^{\forall \mathsf{G}} \wedge \mathsf{r}_{2,1}) \implies \bigwedge_{q \in Q} \left(\varphi_2^q \iff \left[\varphi_1^q \wedge \bigwedge_{q' \in \delta(q)} \varphi_2^{q'} \right] \right)$$

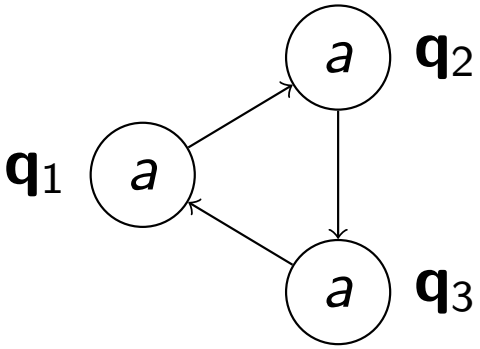


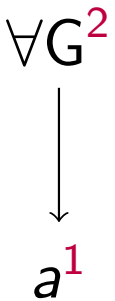


A Boolean Encoding

What about temporal operators?

$$(\tau_2^{\forall \mathsf{G}} \wedge \mathsf{r}_{2,1}) \implies \bigwedge_{q \in Q} \left(\varphi_2^q \iff \left[\varphi_1^q \wedge \bigwedge_{q' \in \delta(q)} \varphi_2^{q'} \right] \right)$$





$$\varphi_1^{q_1} = 1 \quad \varphi_2^{q_1} = 0$$
$$\varphi_1^{q_2} = 1 \quad \varphi_2^{q_2} = 0$$
$$\varphi_1^{q_3} = 1 \quad \varphi_2^{q_3} = 0$$

A Boolean Encoding

A circle of liars

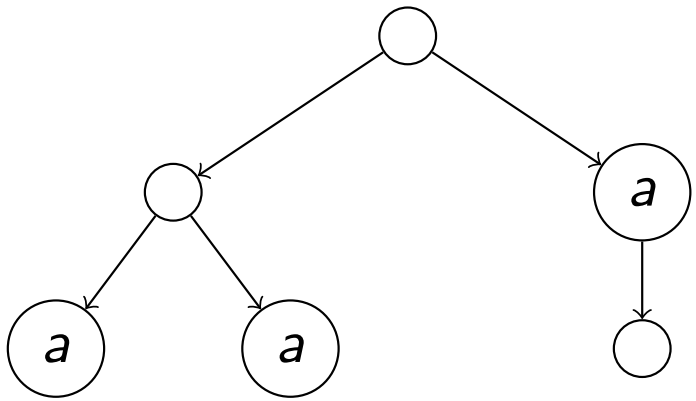
How can we encode CTL's semantics?

Encoding Bounded Semantics

Another form of model-checking

Bounded model-checking (BMC): a CTL property holds for all computation trees of depth k.

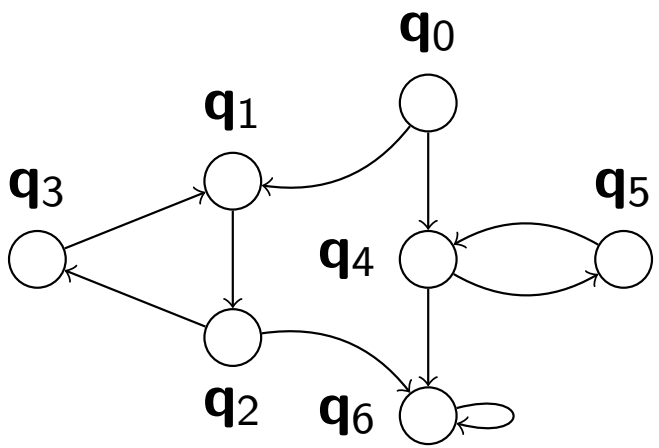
$\forall \mathrm{F}^2\, a$



Encoding Bounded Semantics

An upper bound for finite systems

Recurrence diameter $\alpha(q)$: the length ($\leq |Q|$) of the longest non-repeating sequence of states starting from q.



Encoding Bounded Semantics

Equivalence with model-checking

Model-checking on Kripke structures

is equivalent to

BMC up to the system's recurrence diameter.

Encoding Bounded Semantics

Defining bounded semantics

Base case. $(q \models_{\mathcal{K}} \forall \mathsf{F}^0 \varphi) \iff (q \models_{\mathcal{K}} \varphi)$

Encoding Bounded Semantics

Defining bounded semantics

Base case. $(q \models_{\mathcal{K}} \forall \mathsf{F}^{0}\, \varphi) \iff (q \models_{\mathcal{K}} \varphi)$

Inductive case. $(q \models_{\mathcal{K}} \forall \mathsf{F}^{u+1}\, \varphi) \iff (q \models_{\mathcal{K}} \varphi) \vee \left(\bigwedge_{q' \in \delta(q)} q' \models_{\mathcal{K}} \forall \mathsf{F}^{u}\, \varphi \right)$

Encoding Bounded Semantics

Defining bounded semantics

Base case. $(q \models_{\mathcal{K}} \forall \mathsf{F}^{0}\, \varphi) \iff (q \models_{\mathcal{K}} \varphi)$

Inductive case. $(q \models_{\mathcal{K}} \forall \mathsf{F}^{u+1}\, \varphi) \iff (q \models_{\mathcal{K}} \varphi) \vee \left(\bigwedge_{q' \in \delta(q)} q' \models_{\mathcal{K}} \forall \mathsf{F}^{u}\, \varphi \right)$

Equivalence with MC. $(q \models_{\mathcal{K}} \forall \mathsf{F}\, \varphi) \iff (q \models_{\mathcal{K}} \forall \mathsf{F}^{\alpha(q)}\, \varphi)$

Encoding Bounded Semantics

New variables and clauses for ∀F

New variables. State q verifies the temporal sub-formula φ_i rooted in node i w.r.t. the u-bounded semantics, where $u \leq \alpha(q)$. $\rho^u_{i,q}$

Encoding Bounded Semantics

New variables and clauses for ∀F

New variables. State q verifies the temporal sub-formula φ_i rooted in node i w.r.t. the u-bounded semantics, where $u \leq \alpha(q)$. $\rho^u_{i,q}$

Base case. $\rho^0_{i,q} \iff \varphi^q_j$

Encoding Bounded Semantics

New variables and clauses for ∀F

New variables. State q verifies the temporal sub-formula φ_i rooted in node i w.r.t. the u-bounded semantics, where $u \leq \alpha(q)$. $\rho^u_{i,q}$

Base case. $\rho^0_{i,q} \iff \varphi^q_j$

Inductive case. $\rho^{u+1}_{i,q} \iff \left(\varphi^q_j \vee \bigwedge_{q' \in \delta(q)} \rho^u_{i,q'} \right)$

Encoding Bounded Semantics

New variables and clauses for ∀F

New variables. State q verifies the temporal sub-formula φ_i rooted in node i w.r.t. the u-bounded semantics, where $u \leq \alpha(q)$. $\rho^u_{i,q}$

Base case. $\rho^0_{i,q} \iff \varphi^q_j$

Inductive case. $\rho^{u+1}_{i,q} \iff \left(\varphi^q_j \vee \bigwedge_{q' \in \delta(q)} \rho^u_{i,q'} \right)$

Equivalence with MC. $\tau^{\forall \mathsf{F}}_i \implies \left(\varphi^q_i \iff \rho^{\alpha(q)}_{i,q} \right)$

Encoding Bounded Semantics

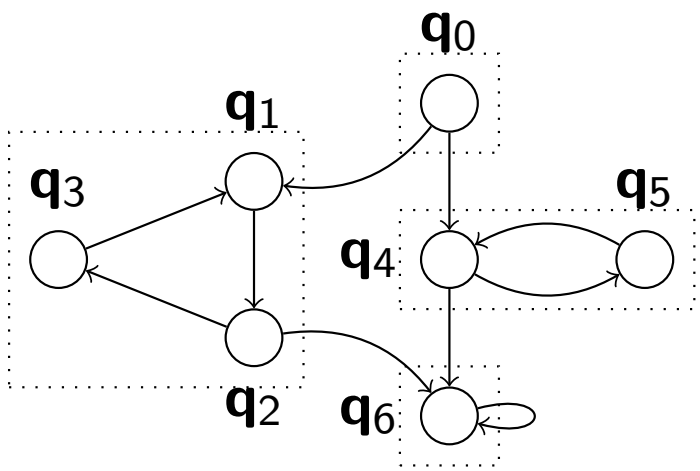
Estimating the instance's complexity

Variables. $\mathcal{O}(n^2 + n \cdot |\mathrm{AP}| + n \cdot |Q| \cdot d)$

Clauses. $\mathcal{O}(n \cdot |\mathrm{AP}| + n^3 \cdot |Q| \cdot d + n \cdot |\mathrm{AP}| \cdot |Q|)$

Optimizations and Implementation

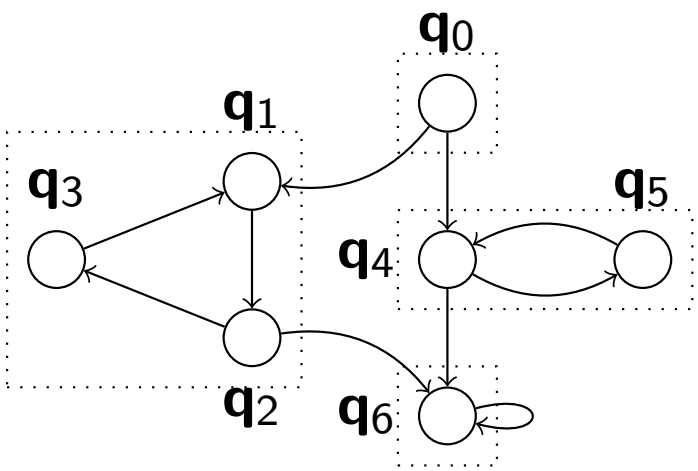
Approximating the recurrence diameter



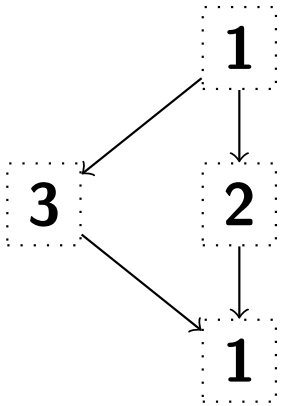
$$\alpha(q_0) = 3$$
$$\alpha(q_1) = 2$$
$$\alpha(q_2) = 1$$
$$\alpha(q_3) = 3$$
$$\alpha(q_4) = 1$$
$$\alpha(q_5) = 2$$
$$\alpha(q_6) = 0$$

Optimizations and Implementation

Approximating the recurrence diameter

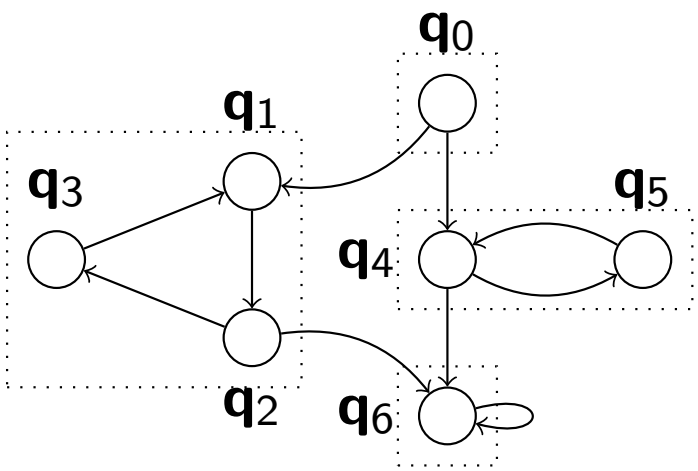


$$\alpha(q_0) = 3$$
$$\alpha(q_1) = 2$$
$$\alpha(q_2) = 1$$
$$\alpha(q_3) = 3$$
$$\alpha(q_4) = 1$$
$$\alpha(q_5) = 2$$
$$\alpha(q_6) = 0$$

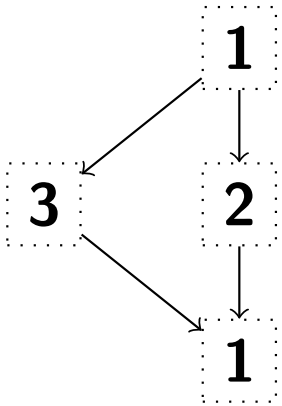


Optimizations and Implementation

Approximating the recurrence diameter



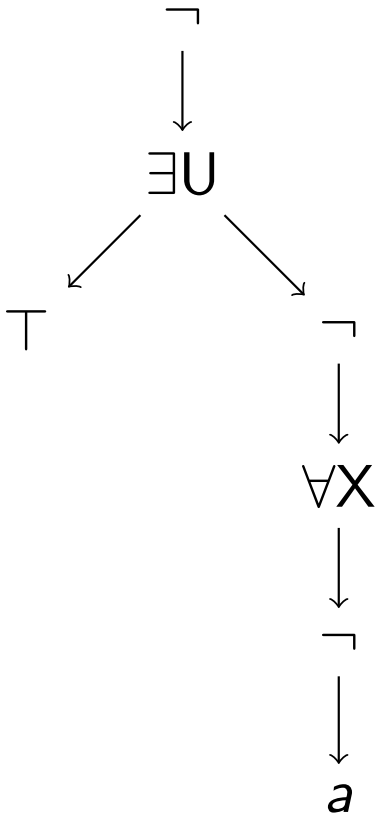
$$\alpha(q_0) = 3 \quad \alpha(q_4) = 1$$
$$\alpha(q_1) = 2 \quad \alpha(q_5) = 2$$
$$\alpha(q_2) = 1 \quad \alpha(q_6) = 0$$
$$\alpha(q_3) = 3$$



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

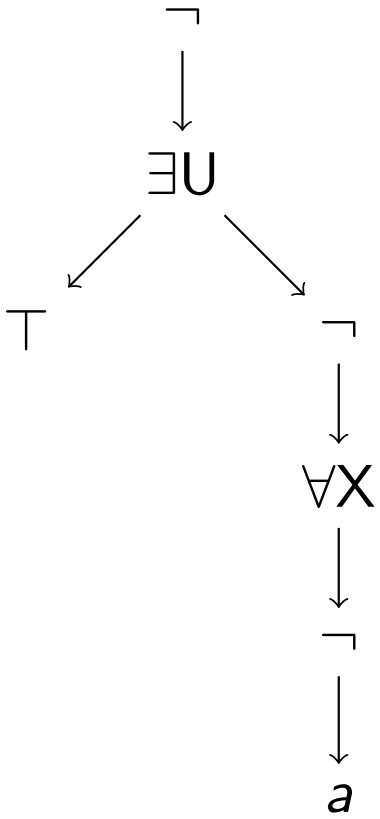
Optimizations and Implementation

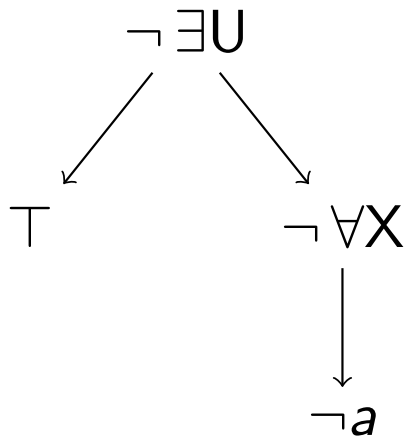
Embedding negations



Optimizations and Implementation

Embedding negations





Optimizations and Implementation

Testing various fragments

$$\mathrm{CTL} = \{\neg, \wedge, \vee, \forall\mathsf{X}, \exists\mathsf{X}, \forall\mathsf{F}, \exists\mathsf{F}, \forall\mathsf{G}, \exists\mathsf{G}, \forall\mathsf{U}, \exists\mathsf{U}\}.$$

Optimizations and Implementation

Testing various fragments

$$\begin{aligned} \mathrm{CTL} &= \{\neg, \wedge, \vee, \forall \mathsf{X}, \exists \mathsf{X}, \forall \mathsf{F}, \exists \mathsf{F}, \forall \mathsf{G}, \exists \mathsf{G}, \forall \mathsf{U}, \exists \mathsf{U}\}. \\ \mathrm{CTL}_{\forall} &= \{\neg, \wedge, \vee, \forall \mathsf{X}, \forall \mathsf{F}, \forall \mathsf{G}, \forall \mathsf{U}\}. \end{aligned}$$

Optimizations and Implementation

Testing various fragments

$$\begin{aligned} \mathrm{CTL} &= \{\neg, \wedge, \vee, \forall\mathsf{X}, \exists\mathsf{X}, \forall\mathsf{F}, \exists\mathsf{F}, \forall\mathsf{G}, \exists\mathsf{G}, \forall\mathsf{U}, \exists\mathsf{U}\}. \\ \mathrm{CTL}_{\forall} &= \{\neg, \wedge, \vee, \forall\mathsf{X}, \forall\mathsf{F}, \forall\mathsf{G}, \forall\mathsf{U}\}. \\ \mathrm{CTL}_{\mathsf{U}} &= \{\neg, \vee, \exists\mathsf{X}, \exists\mathsf{G}, \exists\mathsf{U}\}. \end{aligned}$$

Optimizations and Implementation

Testing various fragments

$$\begin{aligned} \mathrm{CTL} &= \{\neg, \wedge, \vee, \forall\mathsf{X}, \exists\mathsf{X}, \forall\mathsf{F}, \exists\mathsf{F}, \forall\mathsf{G}, \exists\mathsf{G}, \forall\mathsf{U}, \exists\mathsf{U}\}. \\ \mathrm{CTL}_{\forall} &= \{\neg, \wedge, \vee, \forall\mathsf{X}, \forall\mathsf{F}, \forall\mathsf{G}, \forall\mathsf{U}\}. \\ \mathrm{CTL}_{\mathsf{U}} &= \{\neg, \vee, \exists\mathsf{X}, \exists\mathsf{G}, \exists\mathsf{U}\}. \end{aligned}$$

Succinctness **vs** Number of clauses and variables

Optimizations and Implementation

A C++ implementation using Z3

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

Timeouts (over 234 samples) | Arithmetic mean (ms)

Further improvements

Topology-guided solving. Parallel enumeration of DAG shapes[4].

[4] Heinz Riener. "Exact Synthesis of LTL Properties from Traces". In: *2019 Forum for Specification and Design Languages (FDL)*. 2019, pp. 1–6. DOI: 10.1109/FDL.2019.8876900.

[5] Nathanaël Fijalkow and Guillaume Lagarde. "The complexity of learning linear temporal formulas from examples". In: *Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event*. Ed. by Jane Chandlee et al. Vol. 153. Proceedings of Machine Learning Research. PMLR, 2021, pp. 237–250. URL: https://proceedings.mlr.press/v153/fijalkow21a.html.

Further improvements

Topology-guided solving. Parallel enumeration of DAG shapes[4].

Hardness bounds. What about the minimal learning problem[5]?

[4] Heinz Riener. "Exact Synthesis of LTL Properties from Traces". In: *2019 Forum for Specification and Design Languages (FDL)*. 2019, pp. 1–6. DOI: 10.1109/FDL.2019.8876900.

[5] Nathanaël Fijalkow and Guillaume Lagarde. "The complexity of learning linear temporal formulas from examples". In: *Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event*. Ed. by Jane Chandlee et al. Vol. 153. Proceedings of Machine Learning Research. PMLR, 2021, pp. 237–250. URL: https://proceedings.mlr.press/v153/fijalkow21a.html.

Further improvements

Topology-guided solving. Parallel enumeration of DAG shapes[4].

Hardness bounds. What about the minimal learning problem[5]?

State-of-the-art SAT solving. Looking at the SAT competition.

[4] Heinz Riener. "Exact Synthesis of LTL Properties from Traces". In: *2019 Forum for Specification and Design Languages (FDL)*. 2019, pp. 1–6. DOI: 10.1109/FDL.2019.8876900.

[5] Nathanaël Fijalkow and Guillaume Lagarde. "The complexity of learning linear temporal formulas from examples". In: *Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event*. Ed. by Jane Chandlee et al. Vol. 153. Proceedings of Machine Learning Research. PMLR, 2021, pp. 237–250. URL: https://proceedings.mlr.press/v153/fijalkow21a.html.

That's all folks!

Thank you!

An Example

Working on an exclusion protocol

```
bool turn, flag[2];
// t = "turn"; m = "ncrit>1"; c = "ncrit>0"
byte ncrit;
active [2] proctype user()
{
        assert(_pid == 0 || _pid == 1);
        again:
        flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);

        ncrit++;
        assert(ncrit == 1);  /* critical section */
        ncrit--;

        flag[_pid] = 0;
        goto again
}
```

An Example

Introducing mutations

```
bool turn, flag[2];
// t = "turn"; m = "ncrit>1"; c = "ncrit>0"
byte ncrit;
active [2] proctype user()
{
        assert(_pid == 0 || _pid == 1);
        again:
        flag[_pid] = 1; m1
        turn = _pid; m2
        (flag[1 - _pid] == 0 || turn == 1 - _pid); m3

        ncrit++; m4
        assert(ncrit == 1);  /* critical section */
        ncrit--;

        flag[_pid] = 0; m5
        goto again m6
}
```

An Example

Characterizing errors

```
bool turn, flag[2];
// t = "turn"; m = "ncrit>1"; c = "ncrit>0"
byte ncrit;
active [2] proctype user()
{
        assert(_pid == 0 || _pid == 1);
        again:
        flag[_pid] = 1;                          m1
        turn = _pid;                             m2
        (flag[1 - _pid] == 0 || turn == 1 - _pid); m3

        ncrit++;                                 m4
        assert(ncrit == 1);  /* critical section */
        ncrit--;

        flag[_pid] = 0;                          m5
        goto again                               m6
}
```