SAT-verifiable LTL Satisfiability Checking via Graph Representation Learning

Weilin Luo[1], Yuhang Zheng [1], Rongzhen Ye [1], Hai Wan [1*],
Jianfeng Du [2, 3,*], Pingjia Liang [1], Polong Chen [1],

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2] Guangdong University of Foreign Studies, Guangzhou, China
[3] Pazhou Lab, Guangzhou, China

ASE NIER 2023

Content

Content

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $(p \wedge q)\ \mathcal{U}\ \bigcirc r$, output: SAT
- widely used in software engineering, *e.g.*, model checking[4], goal-conflict analysis[3,16], and business process[18]
- PSPACE-complete[24]

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $(p \wedge q)\ \mathcal{U}\ \bigcirc r$, output: SAT
- widely used in software engineering, *e.g.*, model checking [4], goal-conflict analysis [3,16], and business process [18]
- PSPACE-complete [24]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking [19,20], tableau [1,8,22,26], temporal resolution [5,21], anti-chain [27], and Boolean satisfiability (SAT) problem [10–15]

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $(p \wedge q)\ \mathcal{U}\ \bigcirc r$, output: SAT
- widely used in software engineering, *e.g.*, model checking [4], goal-conflict analysis [3,16], and business process [18]
- PSPACE-complete [24]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking [19,20], tableau [1,8,22,26], temporal resolution [5,21], anti-chain [27], and Boolean satisfiability (SAT) problem [10–15]
- *neural approaches*: TreeNN [17]

Motivation

LTL satisfiability checking

- neural networks to solve LTL satisfiability checking [17]
  - take only *polynomial time* to check the satisfiability
- weaknesses of current neural approach
  - difficult to match the permutation invariance of atomic propositions(Proposition 1)
    - *e.g.* $(p \wedge r)\ \mathcal{U}\ q$ and $(p \wedge q)\ \mathcal{U}\ r$ are satisfiable
  - limited by number of different atomic propositions

Motivation

LTL satisfiability checking

- neural networks to solve LTL satisfiability checking [17]
  - take only *polynomial time* to check the satisfiability
- weaknesses of current neural approach
  - difficult to match the permutation invariance of atomic propositions(Proposition 1)
    - *e.g.* $(p \land r)\ \mathcal{U}\ q$ and $(p \land q)\ \mathcal{U}\ r$ are satisfiable
  - limited by number of different atomic propositions

LTL satisfiability verification

- *e.g.*, input: $(p \land q)\ \mathcal{U}\ \bigcirc r$, output: $\{p, q\}, (\{r\})^\omega$

Motivation

LTL satisfiability checking

- neural networks to solve LTL satisfiability checking [17]
  - take only *polynomial time* to check the satisfiability
- weaknesses of current neural approach
  - difficult to match the permutation invariance of atomic propositions(Proposition 1)
    - *e.g.* $(p \wedge r)\ \mathcal{U}\ q$ and $(p \wedge q)\ \mathcal{U}\ r$ are satisfiable
  - limited by number of different atomic propositions

LTL satisfiability verification

- *e.g.*, input: $(p \wedge q)\ \mathcal{U}\ \bigcirc r$, output: $\{p, q\}, (\{r\})^{\omega}$

We explore:

- How does our approach compare with SOTA neural approaches?
- How Effective is our network?

Content

One-step Unfloded Graph

Definition 1

Let ϕ be an LTL formula. Its *one-step unfolded graph (OSUG)* is a two-tuple (V, E), where V is a set of vertices and $E \subseteq V \times V$ is a set of undirected edges. V and E are initialized as $\{v_\phi\}$ and $\emptyset$ respectively.

For every sub-formula ϕ_i of ϕ, V and E are computed as follows:

- if $\phi_i = \neg\phi_j$ or $\phi_i = \bigcirc\phi_j$, then $V = V \cup \{v_{\phi_j}\}$, $E = E \cup \{\left(v_{\phi_i}, v_{\phi_j}\right)\}$;
- if $\phi_i = \phi_j \wedge \phi_k$, then $V = V \cup \{v_{\phi_j}, v_{\phi_k}\}$, $E = E \cup \{\left(v_{\phi_i}, v_{\phi_j}\right), (v_{\phi_i}, v_{\phi_k})\}$;
- if $\phi_i = \phi_j \ \mathcal{U} \ \phi_k$, then $V = V \cup \{v_{\phi_j}, v_{\phi_k}, v_{\phi_i'}, v_{\bigcirc\phi_i}\}$,
  $E = E \cup \{(v_{\phi_i}, v_{\phi_k}), \left(v_{\phi_i}, v_{\phi_i'}\right), \left(v_{\phi_i'}, v_{\phi_j}\right), \left(v_{\phi_i'}, v_{\bigcirc\phi_i}\right), (v_{\bigcirc\phi_i}, v_{\phi_i})\}$,

where ϕ_j, ϕ_k are LTL formulae.

One-step Unfloded Graph

Proposition 1 (permutation invariance of atomic propositions)

Let $\mathbb{P}$ be a set of atomic propositions and ϕ an LTL formula over $\mathbb{P}$. For any $p, q \in \mathbb{P} \cup \{\top\}$, if ϕ is satisfiable, then $\phi[p/\diamond][q/p][\diamond/q]$ is satisfiable, where $\diamond \notin \mathbb{P} \cup \{\top\}$ and $\varphi[\varphi_j/\varphi_i]$ means replacing the sub-formula φ_j of φ with φ_i.

e.g. The one-step unfolded formula of $\phi = (p \wedge q)\ \mathcal{U}\ \bigcirc r$ is $((p \wedge q) \wedge \bigcirc \phi) \vee \bigcirc r$

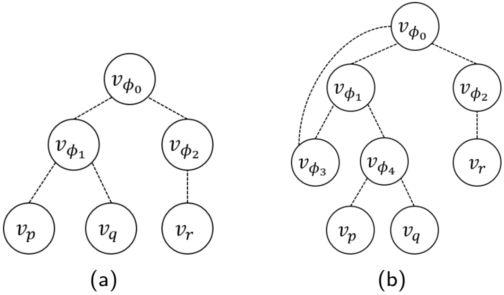

Figure 1: Comparison of Syntax Tree and OSUG of $\phi = (p \wedge q)\ \mathcal{U}\ \bigcirc r$.

Feature Extractor and Satisfiability Checking

Feature Extrator(SAGE)

- sample and aggregate graph convolutional network(GraphSAGE [7]).
- one-step unfolded graph.

Formally, at the t-th iteration, where $t \in [1, T]$, the detailed computation is represented by

$$\mathbf{v}_i^{(t)} = \mathbf{W}_1 \mathbf{v}_i^{(t-1)} + \mathbf{W}_2 \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j^{(t-1)}, \tag{1}$$

where $\mathbf{v}_i^{(t)}$ denotes the vector of vertex i in t-th iteration, $\mathbf{W}_1$ and $\mathbf{W}_2$ are two learnable weight matrices, and $\mathcal{N}(i)$ is set of neighbors of vertex i.

LTL Satisfiability Checking

- obtain a graph feature by (Equation (1))
- obatin probability by a two-layer perceptron and softmax

Trace Generating

deep Q-learning network (DQN)

- State: A state o is a tuple $(\phi, w, loop)$, where $w = \langle w[1], \ldots \rangle$, $w[i] \in 2^{\mathbb{P}}$ is a state of a trace, $loop = \langle b_0, \ldots \rangle$, and $b_i \in \{0, 1\}$ represents whether $w[i]$ is in the loop.
- Action: An action $a = (a_0, a_1) \in 2^{\mathbb{P}} \times \{0, 1\}$, where a_0 is a state of a trace and a_1 is whether a_0 is in the loop.
- Transition: A transition at step t is a tuple $\left(o^{(t)}, a^{(t)}, o^{(t+1)}\right)$, where $o^{(t)} = \left(\phi, w^{(t)}, loop^{(t)}\right)$ and $a^{(t)}$ are a state and an action at time t, respectively.
- Reward: The reward for a transition includes three parts:
  - sequence constraint: 1 for the prefix w^{t+1} not conflict with the ϕ.
  - loop constraint: 1 for $a_1^{t-1} \leq a_1^t$ and 0 otherwise.
  - integrity constraint: 10 for $a_1^{L-1} = 1$ and 10 for predicted trace π satisfies ϕ, respectively.

Trace Generating

Algorithm 1: GENERATE

Input : An LTL formula ϕ, $\mathbf{w}^{(t)}$, and $\mathbf{loop}^{(t)}$.

Output : $a_0^{(t+1)}$ and $a_1^{(t+1)}$.

$x_i^{(0)} = 1$, for $i \in V_a$; $x_i^{(0)} = 2$, for $i \in V/V_a$; $l^{(0)} = 3$

$\mathbf{h}_i^{(0)} = \mathtt{MLP}_0(\mathtt{CAT}(\mathbf{v}_i^{(0)}, \mathtt{EMB}(x_i^{(0)}) + \mathtt{EMB}(l^{(0)}))$, for $i \in V$

$\{\mathbf{z}_i^{(0)} | i \in V\}, \mathbf{z}_g^{(0)} = \mathrm{EXTRACT}(\phi, \{\mathbf{h}_i^{(0)} | i \in V\})$

foreach $u \in [1, |w|]$ **do**

- $x_i^{(u)} = (\mathbf{w}^{(t)})_{u,i}$, for $i \in V_a$; $x_i^{(u)} = 2$, for $i \in V/V_a$; $l^{(u)} = (\mathbf{loop}^{(t)})_u$
- $\mathbf{h}_i^{(u)} = \mathtt{MLP}_0(\mathtt{CAT}(\mathbf{z}_i^{(u-1)}, \mathtt{EMB}(x_i^{(u)}) + \mathtt{EMB}(l^{(u)})))$, for $i \in V$
- $\{\mathbf{z}_i^{(u)} | i \in V\}, \mathbf{z}_g^{(u)} = \mathrm{EXTRACT}(\phi, \{\mathbf{h}_i^{(u)} | i \in V\})$

return $\{i \in V | \arg\max(\mathtt{MLP}_1(\mathbf{z}_i^{|w|})) = 1\}, \arg\max(\mathtt{MLP}_2(\mathbf{z}_g^{|w|}))$

Figure 2: `TraceGeneration` function.

Content

Competitor, Dataset, and Setup

Competitor

- satisfiability checking: `TreeNN-inv`[17] and `TreeNN-con`[17]
- trace generating: `Transformer`[6]

Competitor, Dataset, and Setup

Competitor

- satisfiability checking: `TreeNN-inv`[17] and `TreeNN-con`[17]
- trace generating: `Transformer`[6]

Dataset

- *SPOT*

Setup

- satisfiability checking
  - train all neural networks on the training set of *SPOT*-$[100, 200)$
  - test all neural networks on the test set of *SPOT*-$[100, 200)$
  - test all neural networks on the *SPOT* with larger formulae
- trace generating: we only use the satisfiable formulae in *SPOT*-$[100, 200)$, which follows the work[6].

How does our approach compare with SOTA neural approaches?

Table 1: Evaluation Results on *SPOT*-$[100, 200)$

approach	satisfiability checking					trace generating	
	acc.	pre.	rec.	F1	time	sem. acc.	time
random	50.00	50.00	50.00	50.00	-	51.73	-
`TreeNN-con`	93.61	97.70	89.32	93.32	5,371.56	-	-
`TreeNN-inv`	93.42	97.45	89.18	93.13	4,968.67	-	-
`Transformer`	-	-	-	-	-	50.52	12,880.00
`OSUGGraphSAGE`	**98.48**	**99.58**	**98.89**	**99.23**	**68.11**	**54.95**	**2,676.12**

- The performance of `OSUGGraphSAGE` in checking LTL satisfiability exceeds that of `TreeNN-con` and `TreeNN-inv` in all the evaluation metrics and has a significant improvement in running speed (70 times faster).
- In trace generating, `Transformer` fails since its performance is close to that of random guessing.The performance of `OSUGGraphSAGE` is slightly higher than random guessing, indicating that our approach captures some features suitable for generating traces.

How does our approach compare with SOTA neural approaches?

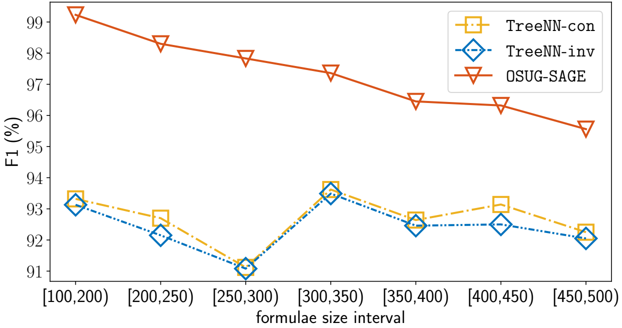

Figure 3: Evaluation results on the larger formulae.

- the performance of OSUGGraphSAGE only slightly decreases when the formula sizes get larger.
- the performance of OSUGGraphSAGE exceeds that of TreeNN-con and TreeNN-inv by a large gap across all distributions.

How Effective is our network?

Table 2: Evaluation Results about Variants of OSUGGraphSAGE

approach		[100, 200)	[200, 250)	[250, 300)	[300, 350)	[350, 400)	[400, 450)	[450, 500)	average
GCN	OSUG	98.66	97.46	96.91	95.75	94.85	94.21	93.36	95.89
	ST	89.14	88.64	88.80	87.42	87.26	88.73	87.23	88.17
GT	OSUG	79.09	78.93	78.05	78.97	79.23	77.86	77.94	78.58
	ST	74.83	74.17	74.33	72.77	72.12	73.66	70.97	73.27
RGGC	OSUG	97.66	95.64	95.40	94.32	91.60	88.58	71.96	90.74
	ST	91.64	90.77	91.19	90.53	90.68	89.53	88.77	90.44
GAT	OSUG	77.88	77.68	78.64	78.31	78.75	78.75	76.35	78.05
	ST	79.72	78.82	79.92	77.34	78.60	79.41	79.52	79.05
SAGE	OSUG	99.23	98.30	97.83	97.36	96.45	96.32	95.56	97.29
	ST	89.76	89.23	89.01	86.94	87.36	87.51	85.78	87.94

We compare variants of `OSUGGraphSAGE` powered by different GNNs and different graph representations. The GNNs include graph convolutional network (GCN)[9], graph Transformer (GT)[23], residual gated graph convolutional network (RGGC)[2], and graph attention network (GAT)[25]. The graph representations include syntax tree (ST) and OSUG.

- All five different GNNs achieve better performance when learning on OSUG, except GAT.

Content

Conclusion and Future Work

Conclusion

1. proposed a new graph representation, OSUG, to achieve SOTA performance
2. made SAT-verifiable checking by additionally generating a satisfiable trace

Conclusion and Future Work

Conclusion

1. proposed a new graph representation, OSUG, to achieve SOTA performance
2. made SAT-verifiable checking by additionally generating a satisfiable trace

Future work

1. extend our approach to validate the effectiveness on intractable industrial instances
2. improve the performance of trace generating
3. propose SAT-UNSAT-verifiable end-to-end neural networks for checking LTL satisfiability

References I

[1] Matteo Bertello, Nicola Gigante, Angelo Montanari, and Mark Reynolds. Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In *IJCAI*, pages 950–956, 2016.

[2] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *CoRR*, abs/1711.07553, 2017.

[3] Renzo Degiovanni, Facundo Molina, Germán Regis, and Nazareno Aguirre. A genetic algorithm for goal-conflict identification. In *ASE*, pages 520–531, 2018.

[4] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and ω -automata manipulation. In *ATVA*, pages 122–129, 2016.

[5] Michael Fisher, Clare Dixon, and Martin Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001.

[6] Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *ICLR*, 2021.

[7] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[8] Yonit Kesten, Zohar Manna, Hugh McGuire, and Amir Pnueli. A decision algorithm for full propositional temporal logic. In *CAV*, volume 697, pages 97–109, 1993.

[9] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[10] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. LTL satisfiability checking revisited. In *TIME*, pages 91–98, 2013.

[11] Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. Aalta: an LTL satisfiability checker over infinite/finite traces. In *FSE*, pages 731–734, 2014.

References II

[12] Jianwen Li, Shufang Zhu, Geguang Pu, and Moshe Y. Vardi. Sat-based explicit LTL reasoning. In *HVC*, volume 9434, pages 209–224, 2015.

[13] Jianwen Li, Geguang Pu, Lijun Zhang, Moshe Y. Vardi, and Jifeng He. Accelerating LTL satisfiability checking by SAT solvers. *J. Log. Comput.*, 28(6):1011–1030, 2018.

[14] Jianwen Li, Lijun Zhang, Shufang Zhu, Geguang Pu, Moshe Y. Vardi, and Jifeng He. An explicit transition system construction approach to LTL satisfiability checking. *Formal Aspects Comput.*, 30(2):193–217, 2018.

[15] Jianwen Li, Shufang Zhu, Geguang Pu, Lijun Zhang, and Moshe Y. Vardi. Sat-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods in System Design*, 54(2):164–190, 2019.

[16] Weilin Luo, Hai Wan, Xiaotong Song, Binhao Yang, Hongzhen Zhong, and Yin Chen. How to identify boundary conditions with contrasty metric? In *ICSE*, pages 1473–1484, 2021.

[17] Weilin Luo, Hai Wan, Delong Zhang, Jianfeng Du, and Hengdi Su. Checking LTL satisfiability via end-to-end learning. In *ASE*, pages 21:1–21:13, 2022.

[18] Fabrizio Maria Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. Discovering data-aware declarative process models from event logs. In *BPM*, volume 8094, pages 81–96, 2013.

[19] Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In *SPIN*, volume 4595, pages 149–167, 2007.

[20] Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. *International Journal on Software Tools for Technology Transfer*, 12(2):123–137, 2010.

[21] Viktor Schuppan. Towards a notion of unsatisfiable cores for LTL. In *FSEN*, volume 5961, pages 129–145. Springer, 2009.

[22] Stefan Schwendimann. A new one-pass tableau calculus for PLTL. In *TABLEAUX*, volume 1397, pages 277–292, 1998.

References III

[23] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *IJCAI*, pages 1548–1554, 2021.

[24] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[25] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[26] Pierre Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, pages 119–136, 1985.

[27] Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In *TACAS*, volume 4963, pages 63–77, 2008.

Thank you for your listening!