Checking LTL Satisfiability via End-to-end Learning

Weilin Luo[1], Hai Wan [1*], Delong Zhang [1], Jianfeng Du [2,3,*], Hengdi Su [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 2022

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[7], goal-conflict analysis[6,18], and business process[19]
- PSPACE-complete[26]

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $(p \land q)\ \mathcal{U}\ \bigcirc r$, output: SAT
- widely used in software engineering, *e.g.*, model checking[7], goal-conflict analysis[6,18], and business process[19]
- PSPACE-complete[26]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking[20,21], tableau[3,10,23,30], temporal resolution[8,22], anti-chain[31], and Boolean satisfiability (SAT) problem[11–16]
- sound and complete

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[7], goal-conflict analysis[6,18], and business process[19]
- PSPACE-complete[26]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking[20,21], tableau[3,10,23,30], temporal resolution[8,22], anti-chain[31], and Boolean satisfiability (SAT) problem[11–16]
- sound and complete
- suffer from the efficiency problem

Motivation

End-to-end neural networks to solve SAT problem [4,24]

- take only *polynomial time* to check the satisfiability
- main idea: capture the *permutation invariance* of the Boolean formulae
  - *e.g.*, $(p \vee q) \wedge (\neg q \vee r)$ and $(\neg q \vee r) \wedge (p \vee q)$

Motivation

End-to-end neural networks to solve SAT problem [4,24]

- take only *polynomial time* to check the satisfiability
- main idea: capture the *permutation invariance* of the Boolean formulae
  - *e.g.*, $(p \vee q) \wedge (\neg q \vee r)$ and $(\neg q \vee r) \wedge (p \vee q)$

No sound and *no* complete, is it useful?

Motivation

End-to-end neural networks to solve SAT problem [4,24]

- take only *polynomial time* to check the satisfiability
- main idea: capture the *permutation invariance* of the Boolean formulae
  - *e.g.*, $(p \vee q) \wedge (\neg q \vee r)$ and $(\neg q \vee r) \wedge (p \vee q)$

No sound and *no* complete, is it useful?

- LTL-SAT-heavy tasks such as goal-conflict identification [6,18]
- extract knowledge to guide the practice in LTL satisfiability checking
- SAT-verifiable neural network: give a satisfiable trace as a proof of satisfiability

Motivation

End-to-end neural networks to solve SAT problem [4,24]

- take only *polynomial time* to check the satisfiability
- main idea: capture the *permutation invariance* of the Boolean formulae
  - *e.g.*, $(p \vee q) \wedge (\neg q \vee r)$ and $(\neg q \vee r) \wedge (p \vee q)$

No sound and *no* complete, is it useful?

- LTL-SAT-heavy tasks such as goal-conflict identification [6,18]
- extract knowledge to guide the practice in LTL satisfiability checking
- SAT-verifiable neural network: give a satisfiable trace as a proof of satisfiability

We explore:

- Can end-to-end neural networks check LTL satisfiability?
- Can neural networks capture the semantics of LTL?

Content

Recursive Property

- Recursive property of syntax

Example 1

$(p \wedge q)\ \mathcal{U}\ \bigcirc r$ can be defined as $\phi_1 = \phi_2\ \mathcal{U}\ \phi_3$, $\phi_2 = p \wedge q$, and $\phi_3 = \bigcirc r$.

- Recursive property of semantics

Example 2

Let $\{p, q, r\}$ be a set of atomic propositions. $(p \wedge q)\ \mathcal{U}\ r$ is satisfiable because:

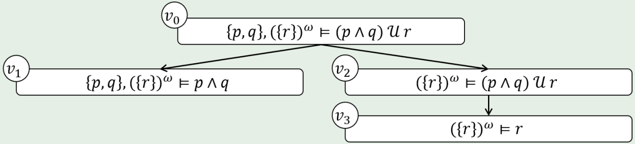

Figure 1: The semantics of $(p \wedge q)\ \mathcal{U}\ r$ is recursive.

Permutation Invariant and Sequentiality

- Permutation invariance of sub-formulae

Example 3

$(p \wedge q)\ \mathcal{U}\ r \equiv (q \wedge p)\ \mathcal{U}\ r$

- Permutation invariance of atomic propositions

Example 4

Both $(p \wedge r)\ \mathcal{U}\ q$ and $(p \wedge q)\ \mathcal{U}\ r$ are satisfiable

Permutation Invariant and Sequentiality

- Permutation invariance of sub-formulae

Example 3

$(p \wedge q)\ \mathcal{U}\ r \equiv (q \wedge p)\ \mathcal{U}\ r$

- Permutation invariance of atomic propositions

Example 4

Both $(p \wedge r)\ \mathcal{U}\ q$ and $(p \wedge q)\ \mathcal{U}\ r$ are satisfiable

- Sequentiality

Example 5

$(r\ \mathcal{U}\ q) \wedge \Box\neg r$ is satisfiable while $(q\ \mathcal{U}\ r) \wedge \Box\neg r$ is unsatisfiable, where $\Box$ is the always operator.

Embedding Based on Transformer

Motivation

- a sequence of tokens
- train a Transformer to generate LTL satisfiable traces [9]

Embedding Based on Transformer

Motivation

- a sequence of tokens
- train a Transformer to generate LTL satisfiable traces[9]

`Transformer`

- use one-hot vectors $\mathbf{x}_p \in \mathbb{R}^{d_m}$ as the initial embedding for atomic propositions and set them to be non-trainable
- use trainable vectors $\mathbf{x}_\top, \mathbf{x}_{op}, \mathbf{x}_{[CLS]}, \mathbf{x}_{[EOS]} \in \mathbb{R}^{d_m}$

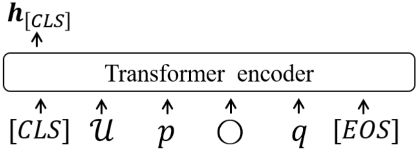

Figure 2: `Transformer` embeds $p \, \mathcal{U} \bigcirc q$.

Embedding Based on GNN

Motivation

- graph neural network (GNN) is used to embed the abstract syntax tree of input programs in programming languages and verification [2,25]
- embedded commands expressed in LTL formulae using relational graph convolutional network (R-GCN) [29]

Embedding Based on GNN

Motivation

- graph neural network (GNN) is used to embed the abstract syntax tree of input programs in programming languages and verification [2,25]
- embedded commands expressed in LTL formulae using relational graph convolutional network (R-GCN) [29]

RGCN

- embed ϕ through its labeled graph
- initialize: one-hot vectors $\mathbf{x}_p \in \mathbb{R}^{d_m}$ and trainable vectors $\mathbf{x}_\top, \mathbf{x}_{op}, \mathbf{x}_g \in \mathbb{R}^{d_m}$
- update: based on the type of the edges through message passings

$$\mathbf{x}_v^{(t+1)} = \sigma\left(\sum_{r \in R_\phi^L \cup \{GV\}} \sum_{u \in \mathrm{N}(v,r)} \frac{1}{|\mathrm{N}(v,r)|} \mathbf{W}_r \mathbf{x}_u^{(t)}\right), \tag{1}$$

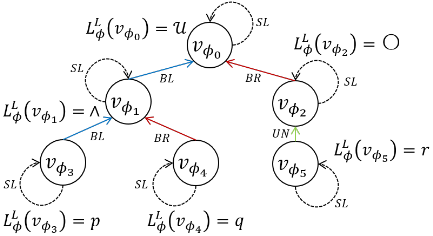

Figure 3: RGCN embeds $(p \wedge q)\ \mathcal{U}\ \bigcirc r$.

Embedding Based on TreeNN

Motivation

- general framework – Recursive Neural Network (TreeNN)[27,28] to capture the recursive property of LTL

Embedding Based on TreeNN

Motivation

- general framework – Recursive Neural Network (TreeNN)[27,28] to capture the recursive property of LTL

TreeNN

- recursively aggregating and combining the embeddings of the sub-formulae
- use a non-trainable one-hot vector $\mathbf{v}_p \in \mathbb{R}^{d_m}$
- project the $\mathbf{v}_p \in \mathbb{R}^{d_m}$ to $\mathbf{r}_p \in \mathbb{R}^{d_h}$ by a trainable projection matrix

Algorithm 3: COMBINE

Input : An aggregation $\mathbf{r}$ of the embeddings of sub-formulae and a logical operator op.

Output : An embedding $\mathbf{r}_{out}$.

$\mathbf{r}' \leftarrow \sigma\left(\mathbf{W}_{0,op} \cdot \mathbf{r}\right)$ /* σ is the ReLU activation function. */

$\mathbf{r}_{out} \leftarrow \mathbf{W}_{1,op} \cdot \mathbf{r}' + \mathbf{W}_{2,op} \cdot \mathbf{r}$

return $\mathbf{r}_{out} / \|\mathbf{r}_{out}\|_2$

Figure 4: The combination function of TreeNN.

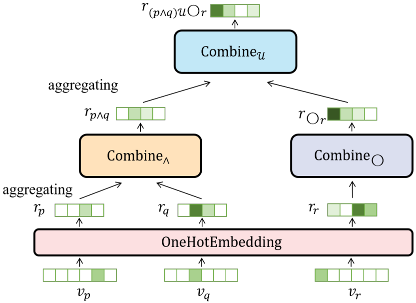

Figure 5: TreeNN embeds $(p \wedge q)\ \mathcal{U}\ \bigcirc r$.

Embedding Based on TreeNN

`TreeNN-MP`

- performing an mean pooling (MP)

Embedding Based on TreeNN

`TreeNN-MP`

- performing an mean pooling (MP)

`TreeNN-con`

- concatenating the embeddings of the sub-formulae in order

Embedding Based on TreeNN

`TreeNN-MP`

- performing an mean pooling (MP)

`TreeNN-con`

- concatenating the embeddings of the sub-formulae in order

`EQNET` [1]

- based on `TreeNN-con`
- add a new loss function to reduce the dependence on surface-level syntactic

---

Algorithm 4: SubexpAe

Input : A formula embedding $\mathbf{r}_\phi$ of an LTL formula ϕ, the embedding $\mathbf{r}_c$ by concatenating the embeddings of all sub-formulae of ϕ, and a logical operator *op*, where $\phi = \phi_i \; op \; \phi_j$ or $\phi = op \; \phi_i$.

Output : A loss value.

1 $\tilde{\mathbf{r}_c} \leftarrow \tanh\left(\mathbf{W}_d \cdot \tanh\left(\mathbf{W}_{e,op} \cdot [\mathbf{r}_\phi, \mathbf{r}_c] \cdot \mathbf{n}\right)\right)$

2 $\tilde{\mathbf{r}_c} \leftarrow \tilde{\mathbf{r}_c} \cdot \|\mathbf{r}_c\|_2 / \|\tilde{\mathbf{r}_c}\|_2$

3 $\tilde{\mathbf{r}_\phi} \leftarrow \textsc{Combine}(\tilde{\mathbf{r}_c}, op)$

4 **return** $-\left(\tilde{\mathbf{r}_c}^{\mathsf{T}}\mathbf{r}_c + \tilde{\mathbf{r}_\phi}^{\mathsf{T}}\mathbf{r}_\phi\right)$

Figure 6: SubexpAe function.

`TreeNN-inv`

- based on `TreeNN-con`
- add a new loss function to keep the permutation invariance

$$
\begin{aligned}
L_{inv}(\phi) &= \sum_{\phi_i = \phi_j \wedge \phi_k \in \mathtt{sub}(\phi)} \left(1 - \mathsf{CS}(\mathbf{r}_{\phi_j,\phi_k}, \mathbf{r}_{\phi_k,\phi_j})\right), \\
\mathbf{r}_{\phi_j,\phi_k} &= \textsc{Combine}([\mathbf{r}_{\phi_j}, \mathbf{r}_{\phi_k}], \wedge), \\
\mathbf{r}_{\phi_k,\phi_j} &= \textsc{Combine}([\mathbf{r}_{\phi_k}, \mathbf{r}_{\phi_j}], \wedge), \\
\mathsf{CS}(\mathbf{x}_1, \mathbf{x}_2) &= \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \, \|\mathbf{x}_2\|},
\end{aligned}
\tag{2}
$$

Summary of Embedding Approaches

Why consider Transformer, R-GCN, and TreeNN?

- Transformer and R-GCN: usage in embedding LTL has been verified [9,29]
- TreeNN: in line with the recursive property of LTL

Summary of Embedding Approaches

Why consider Transformer, R-GCN, and TreeNN?

- Transformer and R-GCN: usage in embedding LTL has been verified [9,29]
- TreeNN: in line with the recursive property of LTL

Varying degrees for keeping the logical properties of LTL for three classes of neural networks

Summary of Embedding Approaches

Why consider Transformer, R-GCN, and TreeNN?

- Transformer and R-GCN: usage in embedding LTL has been verified [9,29]
- TreeNN: in line with the recursive property of LTL

Varying degrees for keeping the logical properties of LTL for three classes of neural networks

- Transformer
  - not keep any logical properties
  - powerful multi-head self-attention mechanism [4]

Summary of Embedding Approaches

Why consider Transformer, R-GCN, and TreeNN?

- Transformer and R-GCN: usage in embedding LTL has been verified [9,29]
- TreeNN: in line with the recursive property of LTL

Varying degrees for keeping the logical properties of LTL for three classes of neural networks

- Transformer
  - not keep any logical properties
  - powerful multi-head self-attention mechanism [4]
- R-GCN
  - sequentiality (Proposition 1), not permutation invariance, and not recursive property
  - not distinguish semantics of different logical operators

Proposition 1

Let ϕ be an LTL formula and $\mathbf{W}_{BL}$ and $\mathbf{W}_{BR}$ two trainable parameters of a R-GCN. If $\phi_i = \phi_j \ op \ \phi_k \in \mathsf{sub}(\phi)$ and $\mathbf{W}_{BL} \neq \mathbf{W}_{BR}$, then $\mathbf{x}^{(t+1)}_{v_{\phi_j \ op \ \phi_k}} = \mathbf{x}^{(t+1)}_{v_{\phi_k \ op \ \phi_j}}$ if and only if $\phi_j = \phi_k$, where op is a binary operator.

Summary of Embedding Approaches

Varying degrees for keeping the logical properties of LTL for three classes of neural networks

- TreeNN
  - recursive property
  - `TreeNN-MP`: not sequentiality and permutation invariance (Proposition 2)
  - `TreeNN-con` and `EQNET`: sequentiality (Proposition 3) and not permutation invariance
  - `TreeNN-inv`: sequentiality (Proposition 3) and permutation invariance (Equation (2))

Proposition 2

Let ϕ be an LTL formula. If $\phi_i = \phi_j \; op \; \phi_k \in \mathtt{sub}(\phi)$, then $\textsc{Combine}\;(\mathtt{MP}(\mathbf{r}_{\phi_j}, \mathbf{r}_{\phi_k}), op) = \textsc{Combine}\;(\mathtt{MP}(\mathbf{r}_{\phi_k}, \mathbf{r}_{\phi_j}), op)$, where op is a binary operator and MP is a mean pooling function.

Proposition 3

Let ϕ be an LTL formula. If $\phi_i = \phi_j \; op \; \phi_k \in \mathtt{sub}(\phi)$, then $\textsc{Combine}\;([\mathbf{r}_{\phi_j}, \mathbf{r}_{\phi_k}], op) = \textsc{Combine}\;([\mathbf{r}_{\phi_k}, \mathbf{r}_{\phi_j}], op)$ if and only if $\phi_j = \phi_k$, where op is a binary operator.

Synthetic Dataset

Generate random formulae

- *randltl* tool in the SPOT framework to generate random formulae
- label them (satisfiable or unsatisfiable) using nuXmv [5]
- the set of atomic propositions: 1024

SPOT

- sizes are in $[100, 200)$
- 160K/20K/20K formulae in training/validation/test set
- other 6 test sets with different size intervals: $[200, 250)$, $[250, 300)$, $[300, 350)$, $[350, 400)$, $[400, 450)$, and $[450, 500)$ (2K formulae for each)
- balance the numbers of satisfiable and unsatisfiable formulae
- ensure that formulae in the training, validation and test set are not repeated

Content

Competitor, Dataset, and Setup

Competitor

- 6 neural networks `Transformer`, `RGCN`, `TreeNN-MP`, `TreeNN-con`, `EQNET`, and `TreeNN-inv`
- 2 logical approaches
  - `nuXmv`: SOTA approach for model checking
  - `Aalta`: SOTA approach for checking LTL satisfiability

Competitor, Dataset, and Setup

Competitor

- 6 neural networks `Transformer`, `RGCN`, `TreeNN-MP`, `TreeNN-con`, `EQNET`, and `TreeNN-inv`
- 2 logical approaches
  - `nuXmv`: SOTA approach for model checking
  - `Aalta`: SOTA approach for checking LTL satisfiability

Dataset

- *SPOT*
- Large-scale datasets [17]
  - *LTL-as-LTL$_f$*: 4668 formulae coming from LTL satisfiability checking
  - *LTL$_f$-Specific*: 1700 formulae generated by common LTL$_f$ patterns
  - *NASA-Boeing*: real-world LTL$_f$ specifications
  - *DECLARE*: 112 LTL$_f$ patterns widely used in the business process management

Setup

- 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
- evaluate all approaches on the large scale datasets

Can neural networks capture features for checking LTL satisfiability?

Table 1: Evaluation results on the test set of *SPOT*-$[100, 200)$, where "acc." means the accuracy (%), "pre." means the precision (%), "rec." means the recall (%), "F1" means the F1 score (%), and "time" indicates the sum of the running time for all formulae (seconds). **Bold** numbers mark better results.

approach	acc.	pre.	rec.	F1	time
`Transformer`	70.60	71.02	69.61	70.31	**57.09**
`RGCN`	65.42	71.06	52.01	60.06	3,642.99
`TreeNN-MP`	86.15	90.55	80.73	85.36	1,792.11
`TreeNN-con`	**93.76**	**98.17**	**89.19**	**93.47**	1,814.88
`EQNET`	90.73	94.87	86.13	90.29	485.08
`TreeNN-inv`	91.79	96.23	87.00	91.38	416.96

- Neural networks are capable to capture inductive biases for more accurately checking LTL satisfiability.

How well do neural networks generalize across formula sizes and distributions?

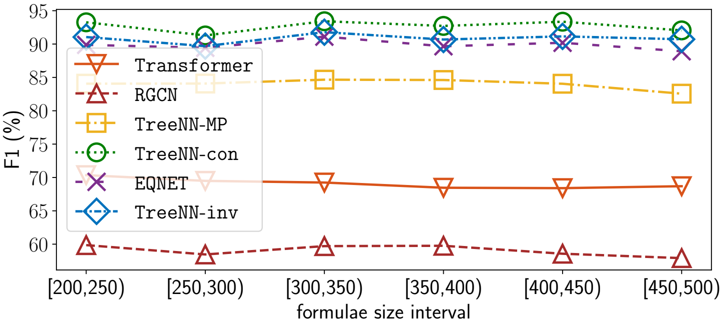

Figure 7: Evaluation results on the larger formulae. The accuracy, precision, and recall for neural networks have the same rankings and trends as F1 score.

- The more a neural network keeps the logical properties of LTL, the more effective it is to capture the inductive biases that are beneficial to classification.

Are neural networks and SOTA logical approaches comparable on large scale datasets?

Table 2: Evaluation results on the large scale datasets.

approach	*LTL-as-LTL$_f$*			*LTL$_f$-Specific*			*NASA-Boeing*			*DECLARE*		
	acc.	F1	time	acc.	F1	time	acc.	F1	time	acc.	F1	time
`nuXmv`	100.00	100.00	8,483.02	100.00	100.00	2,584.66	100.00	100.00	95.72	100.00	100.00	7,296.68
`Aalta`	100.00	100.00	352,224.95	100.00	100.00	1,148,839.93	100.00	100.00	9.37	100.00	100.00	356,043.51
`Transformer`	67.40	78.81	28.72	67.00	40.63	6.08	32.26	48.78	1.95	0.00	0.00	5.12
`RGCN`	73.39	83.31	6,021.31	91.00	88.69	3,145.00	37.10	54.12	220.10	0.00	0.00	5,250.53
`TreeNN-MP`	88.80	94.02	437.78	44.82	61.62	146.43	98.39	99.19	27.93	100.00	100.00	437.78
`TreeNN-con`	88.67	93.90	2,524.40	99.53	99.47	1,370.97	96.77	98.36	179.80	23.85	38.52	7,136.23
`EQNET`	86.37	92.58	2,481.76	98.76	98.62	1,404.96	53.23	69.47	193.44	96.33	98.13	7,182.01
`TreeNN-inv`	87.00	92.92	1,994.57	94.82	94.48	1,214.01	85.48	92.17	153.76	93.58	96.68	5,630.03

- Neural networks are much faster than logical approaches in most datasets.
- Designing an architecture keeping logical properties, especially the recursive property, has a positive effect in improving the generalization ability.

Content

Conclusion and Future Work

Conclusion

1. explore a new paradigm for checking LTL satisfiability to outperform SOTA approaches
2. designing neural networks matching the recursive property, permutation invariance, and sequentiality is positive to check LTL satisfiability
3. make it possible to obtain highly confident results for LTL satisfiability checking in polynomial time, which will benefit LTL-SAT-heavy tasks a lot

Conclusion and Future Work

Conclusion

1. explore a new paradigm for checking LTL satisfiability to outperform SOTA approaches
2. designing neural networks matching the recursive property, permutation invariance, and sequentiality is positive to check LTL satisfiability
3. make it possible to obtain highly confident results for LTL satisfiability checking in polynomial time, which will benefit LTL-SAT-heavy tasks a lot

Future work

1. validate the effectiveness of neural networks on intractable industrial instances
2. design highly confident and verifiable end-to-end neural networks for checking LTL satisfiability

References I

[1] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In *ICML*, volume 70, pages 80–88, 2017.

[2] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *ICLR*, 2018.

[3] 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.

[4] Chris Cameron, Rex Chen, Jason S. Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *AAAI*, pages 3324–3331, 2020.

[5] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, pages 334–342, 2014.

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

[7] 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.

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

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

[10] 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.

References II

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] Jianwen Li, Geguang Pu, Yueling Zhang, Moshe Y. Vardi, and Kristin Y. Rozier. Sat-based explicit ltlf satisfiability checking. *Artif. Intell.*, 289:103369, 2020.

[18] 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.

[19] 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.

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

References III

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

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

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

[24] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *ICLR*, pages 1–11, 2019.

[25] Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *NeurIPS*, pages 7762–7773, 2018.

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

[27] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, pages 1201–1211, 2012.

[28] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642, 2013.

[29] Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. Ltl2action: Generalizing LTL instructions for multi-task RL. In *ICML*, volume 139, pages 10497–10508, 2021.

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

[31] 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!