Deep Learning for Temporal Logics

Frederik Schmitt[1], Christopher Hahn[1], Jens U. Kreber[2], Markus N. Rabe[3], and Bernd Finkbeiner[1]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2] Saarland University, Saarbrücken, Germany
[3] Google Research, Mountain View, CA, USA

Abstract

Temporal logics are a well established formal specification paradigm to specify the behavior of systems, and serve as inputs to industrial-strength verification tools. We report on current advances in applying deep learning to temporal logical reasoning tasks, showing that models can even solve instances where competitive classical algorithms timed out.

1 Introduction

The assumption that deep learning is not yet ready to tackle hard reasoning questions has been drawn into question. For example, Transformer models [25] perform surprisingly well on symbolic integration tasks [16], self-supervised training can lead to mathematical reasoning abilities [22], or large-enough language models learn basic arithmetic despite being trained on mostly natural language sources [21]. These success stories and uprising workshops and conferences on this topic (e.g., [5, 24, 26]), pose the question if challenging problems in the area of automated verification lend themselves to a direct learning approach as well. We report on current advances in applying deep learning to involved temporal logical reasoning tasks.

Many approaches in verification are based on temporal logics, a specification paradigm that is the basis for industrial hardware specification languages like the IEEE standard PSL [13]. For example, linear-time temporal logic (LTL) [20] can specify that some proposition P must hold at every point in time ($\square P$) or that P must hold at some future point of time ($\diamond P$). By combining these operators, one can specify that P must occur infinitely often ($\square\diamond P$). *LTL satisfiability* is the (PSPACE-complete) problem of computing a logical solution, i.e., a *trace*, which is sequence of propositions, that satisfies an LTL formula. In applications, solutions to LTL formulas can represent (counter-)examples for a specified system behavior. *LTL synthesis* is the (2EXPTIME-complete) problem of automatically constructing a circuit that satisfies an LTL specification for every input. This is an especially challenging and active research field including an annual tool competition (SYNTCOMP [14]).

Over the last decades, generations of advanced algorithms have been developed to solve temporal reasoning tasks automatically. We show that fully neural approaches are already competitive: By computing satisfying traces to LTL formulas, we show that Transformers generalize to the semantics of LTL (Section 2 and [11]) and based on those findings, we show that Transformers can be used to even synthesize fully functional circuits directly out of LTL specifications (Section 3 and [23]).

2 Transformers Generalize to the Semantics of Temporal Logics

For predicting a satisfying trace to an LTL formula, we generated training data in two different ways: randomly and as conjunctions of patterns typically encountered in practice [7]. We use `spot` [6], that implements a competitive classical algorithm, to generate solutions to formulas from these distribution and train a Transformer model to predict solutions directly. The

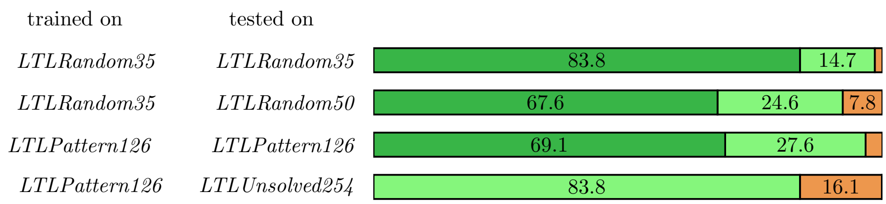

Figure 1: [11] Performance on different datasets, where the number refers to the size of the largest formula in the data set. The percentage of a dark green bar refers to the syntactic accuracy, the percentage of a light green bar to the semantic accuracy without the syntactic accuracy. Incorrect predictions are given in orange.

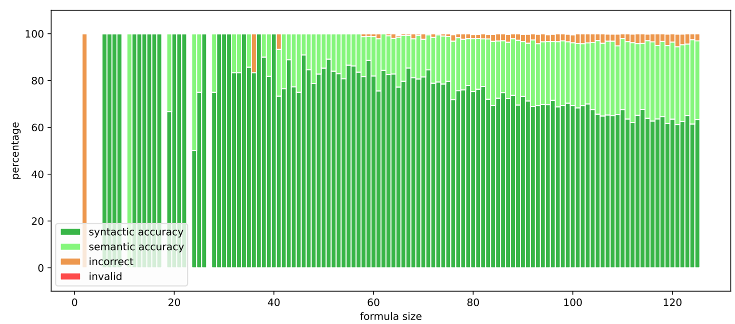

Figure 2: [11] Instances where the model agrees with the generator are displayed in dark green; deviations from the generators output but still correct predictions are displayed in light green; incorrect predictions in orange.

question, however, is whether Transformers learn to imitate the generator of the training data, rather than learn to solve the formulas according to the *semantics* of the logics. We, thus, differentiate between a syntactic accuracy, i.e., where the model predicts the same trace as the classical algorithm and the semantic accuracy, i.e., where the model deviates.

Figure 1 shows a subset of experiments conducted in [11] showing that, in fact, the latter holds true. In our experiments, we observed that Transformers predict correct solutions to 98.5% of the random formulas (line 1) and 96.8% of the pattern formulas (line 3) from a held-out test set. Figure 2 shows the performance of a model trained and tested on combinations of specification patterns *LTLPattern126* in more detail. When the formulas become larger, the gap between syntactic and semantic accuracy increases. We also observed that Transformers generalize to larger formulas than seen during training (line 2). Impressive enough, Transformers hold up pretty well and predict correct solutions in 83% of cases on a set of formulas that `spot` could not solve within 60 seconds (line 4). Finally, we performed an out-of-distribution test by predicting traces to held-out specification patterns from the literature ([8, 12, 19]). Trained on *LTLRandom126*, the model achieved an accuracy of 84.4% (62.2% syntactic accuracy) and trained on *LTLPattern126* it achieved 50.0% (11.3%).

3 Transformers Construct Circuits out of Temporal Logical Specifications

To predict a circuit that satisfies an LTL formula for every input, we utilized specification patterns mined from benchmarks of the annual reactive synthesis competition (SYNTCOMP) [14]. An example for such a pattern is a typical request-response property: $\square(\mathit{request} \rightarrow \diamond\, \mathit{grant})$.

(assumptions)
$(\square(\diamond\neg(r_m)))$
$\rightarrow$
(guarantees:)
$(\square((\neg(g_m))\vee(\neg(g_0))))$
$(\square((r_0)\rightarrow(\diamond(g_0))))$
$(\square((r_m)\rightarrow(\bigcirc((\neg(g_0))\,\mathcal{U}(g_m)))))$

```
aag 7 5 1 5 1
2
4
6
8
10
12 4
14
0
13
14
0
14 12 5
```

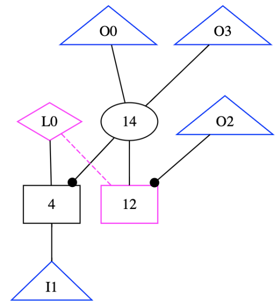

Figure 3: [23] The specification (left), the predicted AIGER circuit (middle) and the visualization of the circuit (right) for a prioritizing arbiter.

The formula states that at every point in time ($\square$) a request must be eventually ($\diamond$) followed by a grant. By combining specification patterns, we generated over 200 000 specifications including both realizable and unrealizable specifications. Using classical LTL synthesis tools [9, 18], we obtained a dataset consisting of formulas and their circuit implementations in AIGER [2]. We represented the specifications and circuits as sequences and trained hierarchical Transformers [17] on the circuit construction task. For example, when given the specification of a prioritizing arbiter that manages a shared resource, the model predicts a correct circuit given as an AIGER file, displayed in Figure 3. The specification consists of a combination of request-response and mutual exclusion patterns.

We ran several experiments [23], where the results of a subset can be found in Table 1. The `Testset` contains held-out data from the training distribution, i.e., combinations of mined patterns, where the model achieved an accuracy of 79.9%. We also tested on the challenging `SYNTCOMP` benchmarks, which contain practical specifications of systems, such as arbiters of the AMBA AHB bus [3, 4, 10] or robotic controllers [15]. For the instances that fit into the size restrictions of the model [23], the model achieved an accuracy of 66.8% (making this already a competitive approach). We stored specifications for which the synthesis tool timed out ($> 120s$) in the dataset `Timeouts`. The Transformer was able to solve 30.1% substantiating the strong generalization of this model. We also performed an out-of-distribution test by predicting circuits for a `Smart Home` benchmark [1] that has only recently been added to the SYNTCOMP competition 2021. Note that we have not mined patterns from this benchmark. We achieved an accuracy of 40.0% for the instances that fit into the size restrictions of the model. This means that, already today, direct machine learning approaches may be useful to augment classical algorithms in verification tasks.

Table 1: [23] Accuracy reported for 5 runs on `Testset`, `SYNTCOMP` benchmarks, `Timeouts`, and `Smart Home` benchmarks for different beam sizes, including the standard deviation. For the test data we show the syntactic accuracy in parenthesis.

Dataset	Beam Size 1	Beam Size 4	Beam Size 8	Beam Size 16
`Testset`	$53.6(31.1) \pm 2.4$	$70.4(39.0) \pm 2.3$	$75.8(41.9) \pm 2.1$	$79.9(44.5) \pm 2.0$
`SYNTCOMP`	51.9 ± 2.2	60.0 ± 1.5	63.6 ± 1.9	66.8 ± 1.2
`Timeouts`	11.7 ± 1.1	21.1 ± 0.9	25.9 ± 1.0	30.1 ± 1.2
`Smart Home`	22.9 ± 3.6	31.4 ± 7.1	44.8 ± 6.5	40.0 ± 6.5

References

[1] J.A.R.V.I.S. TSL/TLSF benchmark suite. https://github.com/SYNTCOMP/benchmarks/tree/master/tlsf/tsl_smart_home_jarvis, 2021.

[2] A. Biere. The AIGER and-inverter graph (AIG) format version 20071012. *FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr*, 69:4040, 2007.

[3] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1188–1193. IEEE, 2007.

[4] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. *Electron. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007.

[5] A. d'Avila Garcez, D. Silver, P. Hitzler, P. Földiák, K.-U. Kühnberger, L. C. Lamb, and L. de Raedt. Workshop series on neural-symbolic learning and reasoning. http://www.neural-symbolic.org.

[6] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, Oct. 2016.

[7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In M. A. Ardis and J. M. Atlee, editors, *Proceedings of the Second Workshop on Formal Methods in Software Practice*, 1998.

[8] K. Etessami and G. J. Holzmann. Optimizing büchi automata. In *International Conference on Concurrency Theory*, pages 153–168. Springer, 2000.

[9] P. Faymonville, B. Finkbeiner, and L. Tentrup. Bosy: An experimentation framework for bounded synthesis. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, 2017.

[10] Y. Godhal, K. Chatterjee, and T. A. Henzinger. Synthesis of AMBA AHB from formal specification: a case study. *Int. J. Softw. Tools Technol. Transf.*, 2013.

[11] C. Hahn, F. Schmitt, J. U. Kreber, M. N. Rabe, and B. Finkbeiner. Teaching temporal logics to neural networks. *International Conference on Learning Representations, ICLR*, 2021.

[12] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, P. Šimeček, et al. Verification results in liberouter project, 2004.

[13] IEEE-Commission et al. Ieee standard for property specification language (psl). *IEEE Std 1850-2005*, 2005.

[14] S. Jacobs and G. A. Pérez. The reactive synthesis competition. www.syntcomp.org.

[15] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.

[16] G. Lample and F. Charton. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[17] W. Li, L. Yu, Y. Wu, and L. C. Paulson. Modelling high-level mathematical reasoning in mechanised declarative proofs. *CoRR*, abs/2006.09265, 2020.

[18] P. J. Meyer, S. Sickert, and M. Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification - 30th International Conference, CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018.

[19] R. Pelánek. Beem: Benchmarks for explicit model checkers. In *International SPIN Workshop on Model Checking of Software*, pages 263–267. Springer, 2007.

[20] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE

Computer Society, 1977.

[21] S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020.

[22] M. N. Rabe, D. Lee, K. Bansal, and C. Szegedy. Mathematical reasoning via self-supervised skip-tree training. In *International Conference on Learning Representations, ICLR*, 2021.

[23] F. Schmitt, C. Hahn, M. N. Rabe, and B. Finkbeiner. Neural circuit synthesis from specification patterns. *arXiv preprint arXiv:2107.11864*, 2021.

[24] V. Thost, K. Talamadupula, V. Srikumar, C. Zhang, and J. Tenenbaum. Knowledge representation & reasoning meets machine learning. https://kr2ml.github.io/2020/, 2020.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[26] Y. Wu, K. Bansal, W. Li, M. Mitchell, D. McAllester, and J. Harrison. The role of mathematical reasoning in general artificial intelligence. https://mathai-iclr.github.io, 2021.