Exploring Representation of Horn Clauses using GNNs

Chencheng Liang[1], Philipp Rümmer[1,2], and Marc Brockschmidt[3]

[1] Uppsala University, Uppsala, Sweden
[2] University of Regensburg, Regensburg, Germany
[3] Microsoft Research, Cambridge, United Kingdom

Automatic program verification has been used in safety-critical industrial software for decades. Constrained Horn Clauses (CHCs) [7] as an intermediate verification language is a standard representation of program verification problems. The program is safe if and only if the CHCs are satisfied. In practice, it is essential to extract information from program features (e.g., loops, control flow, or data flow) to guide the CHC solvers. For instance, the authors of [9] and [4] perform static analysis systematically to extract semantic program features (e.g., loop variables) to guide refinement process in the counterexample-guided abstraction refinement (CEGAR) [3] based solver. In recent years, along with breakthrough practices in deep learning [10, 8, 6, 20], many studies [19, 2, 14, 15, 19] have introduced deep learning methods to guide program verification and produce promising results. In particular, since graphs can represent highly structured relations naturally, some closely related fields, such as automatic reasoning, theorem proving, and SAT solving, begin to use the graph to represent logic formulas and apply graph neural networks (GNNs) [1] to learn the features to guide the solving process. Works such as FormulaNet [21], LERNA [13], NeuroSAT [17, 18], [12], and [11] have used this graph-based framework to improve their results by various learning tasks, e.g., premise selection and unsat-core prediction. However, to the best of our knowledge, we did not see any study which encodes CHCs to graph representations and use GNNs to learn the program features.

We believe GNNs can learn useful program features from graph represented CHCs to guide CHC solvers. In this work, to evaluate our assumption, we first answer two preliminary questions: (1) What kind of graph representation is suitable for CHCs? (2) Which kind of GNN is suitable for learning CHC graph representations?

To answer the first question, we have designed two graph representations (see Figure 1) of CHCs. Our *constraint graph* (CG) representation emphasizes the syntactic information of CHCs by constructing abstract syntax trees for constraints and building binary connections for relation symbols and their arguments. Our *control- and data-flow hypergraph* (CDHG) emphasizes semantic information of programs by using (ternary) hyperedges to represent the flow of control and data. To better express control- and data-flow, we construct CDHG from normalized CHCs. The normalization adding extra clauses to the original CHC but retains logical meaning.

For the second question, we introduce a new Relational Hypergraph Neural Network (R-HyGNN) architecture which is an extension of a message-passing GNN, namely, Relational Graph Convolutional Networks (R-GCN) [16]. In R-HyGNN, messages exchanged between nodes are computed from the representations of all nodes connected by typed edges. Then, the messages from all typed edges are aggregated to update the node representations.

To evaluate our framework, we introduce five proxy tasks (see Table 1) with increasing difficulties. Task 1 is a trivial sanity check, evaluating whether models can recover information from the initial node features. Task 2 evaluates the ability of models to handle counting problems in the overall graph. Task 3 requires the models to answer basic questions about the wider graph structure. Task 4 is significantly harder than the previous task, requiring the model to infer if a program variable is bounded from below or above. Finally, Task 5 is much harder, as it requires implicitly identifying counter-examples (CEs) traces. Moreover, we hope

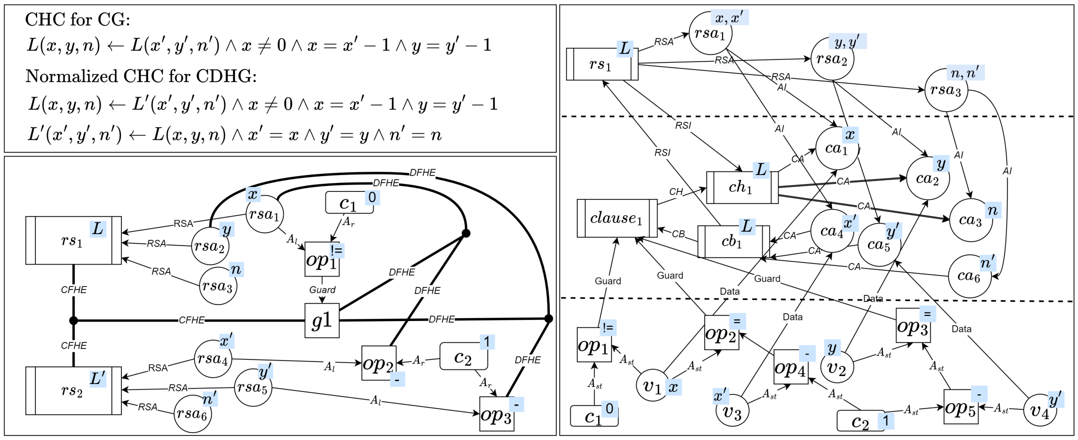

Figure 1: A CHC and the corresponding normalized CHCs are in the left upper corner. The CDHG constructed from the normalized CHCs is in the left lower corner. The CG for the CHC is on the right side. The texts on nodes and edges indicate the types of nodes and edges. To better illustrate the graphs, we add the blue boxes with text on nodes to relate the corresponding concrete symbol names in CHCs.

Task description	CG		CDHG	
1. If a node is an argument of a relation symbol	100% (95%)		99% (73%)	
2. How many times a relation symbol occurs in all clauses	1.0		4.2	
3. If a typed node is in a cycle	96% (70%)		99% (51%)	
4. If a relation symbol argument has **upper** and **lower** bound	**upper** 91% (80%)	**lower** 91% (75%)	**upper** 94% (75%)	**lower** 94% (68%)
5. If a clause occurs in **some** or **all** minimum CEs	**some** 95% (85%)	**all** 84% (53%)	**some** 96% (86%)	**all** 90% (55%)

Table 1: Description and experimental results for five proxy tasks. Task 2 performs regression task on nodes and is measured by mean square error, while other tasks perform binary classification task on nodes and are measured by accuracy. Both the fourth and fifth task consists of two independent binary classification tasks. The values in parentheses are the ratios of the dominant labels in the binary data distribution. Note that the label distribution differs for the two graph representations, as CDHGs are constructed from normalized CHCs.

that learning models on the five representative proxy tasks can reduce the bias from adapting to a particular application.

The test data is extracted from 8705 linear and 8425 non-linear Linear Integer Arithmetic (LIA) problems in CHC-COMP repository (see Table 1 in the competition report [5]). We divide the extracted dataset to train, valid, and test set by 60%, 20%, and 20%, respectively. The experimental results on the test set are shown in Table 1. As expected, for both graph representations, the performance of GNN models decreases along with the increasing difficulty of the tasks. However, even for the hardest (fifth) task, the accuracy is far higher than predicting the data distribution (values in the parentheses in Table 1), indicating that the models learn more than trivial patterns. In particular, we see a slight advantage of using the hypergraph representation (CDHG) comparing with binary graph representation (CG). We plan to use this framework to support predicate selection of CEGAR-based program verification.

References

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[2] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. Neural code comprehension: A learnable representation of code semantics. *CoRR*, abs/1806.07336, 2018.

[3] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[4] Yulia Demyanova, Philipp Rümmer, and Florian Zuleger. Systematic predicate abstraction using variable roles. In *NASA Formal Methods*, pages 265–281, Cham, 2017. Springer International Publishing.

[5] Grigory Fedyukovich and Philipp Rümmer. Competition report: CHC-COMP-21, 2021.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[7] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[9] Jérôme Leroux, Philipp Rümmer, and Pavle Subotić. Guiding Craig interpolation with domain-specific abstractions. *Acta Informatica*, 53(4):387–424, Jun 2016.

[10] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Honza Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

[11] Miroslav Olsak and Josef Urban Cezary Kaliszyk. Property invariant embedding for automated reasoning. *arXiv*, 2019.

[12] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *CoRR*, abs/1905.10006, 2019.

[13] Michael Rawson and Giles Reger. *A Neurally-Guided, Parallel Theorem Prover*, pages 40–56. 08 2019.

[14] Cedric Richter, Eyke Hüllermeier, Marie-Christine Jakobs, and Heike Wehrheim. Algorithm selection for software validation based on graph kernels. *Automated Software Engineering*, 27(1):153–186, June 2020.

[15] Cedric Richter and Heike Wehrheim. Attend and represent: A novel view on algorithm selection for software verification. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1016–1028, 2020.

[16] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[17] Daniel Selsam and Nikolaj Bjørner. Neurocore: Guiding high-performance SAT solvers with unsat-core predictions. *CoRR*, abs/1903.04671, 2019.

[18] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *CoRR*, abs/1802.03685, 2018.

[19] Xujie Si, Aaditya Naik, Hanjun Dai, Mayur Naik, and Le Song. Code2inv: A deep learning framework for program verification. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part II*, page 151–164, Berlin, Heidelberg, 2020. Springer-Verlag.

[20] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[21] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Advances in Neural Information Processing Systems 30*, pages 2786–2796. Curran Associates, Inc., 2017.