Exploring Representation of Horn Clauses using GNNs

Chenceng Liang\textsuperscript{1}, Philipp Rümmer\textsuperscript{1,2}, and Marc Brockschmidt\textsuperscript{3}

\textsuperscript{1} Uppsala University, Uppsala, Sweden
\textsuperscript{2} University of Regensburg, Regensburg, Germany
\textsuperscript{3} Microsoft Research, Cambridge, United Kingdom

Automatic program verification has been used in safety-critical industrial software for decades. Constrained Horn Clauses (CHCs)\textsuperscript{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\textsuperscript{9} and\textsuperscript{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)\textsuperscript{3} based solver. In recent years, along with breakthrough practices in deep learning\textsuperscript{10, 8, 6, 20}, many studies\textsuperscript{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)\textsuperscript{1} to learn the features to guide the solving process. Works such as FormulaNet\textsuperscript{21}, LERNA\textsuperscript{13}, NeuroSAT\textsuperscript{17, 18},\textsuperscript{12}, and\textsuperscript{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 \textsuperscript{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)\textsuperscript{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 \textsuperscript{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
Exploring Representation of Horn Clauses using GNNs

Liang, Rümmer and Brockschmidt

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.

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

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


