Analyzing Proof Components*

Karel Chvalovský and Josef Urban

Czech Technical University in Prague, Czech republic,
karel@chvalovsky.cz and josef.urban@gmail.com

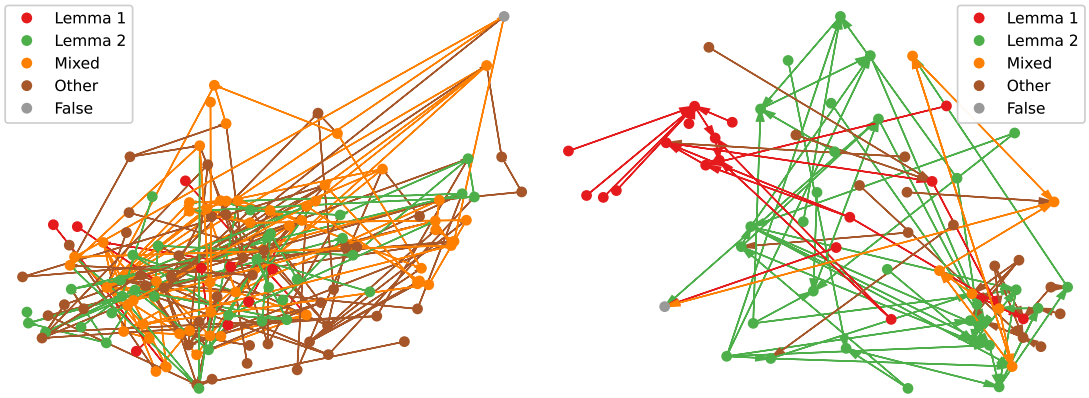
Modern automated theorem provers (ATPs) for first-order logic, such as E [6] or Vampire [5], usually start a proof search by trying to classify the input problem so that they can use strategies that fit the problem best. However, not only are there different problems, each problem also commonly consists of various (semi)independent parts that should be treated differently. The provers are able to treat the most common special cases, like equalities or arithmetic, specifically, but no general approach exists. Although a guidance based on various machine learning models provides a partial solution to these problems, because such models can, in principle, dynamically adapt their guidance as the proof search evolves, it has only been of limited success so far. Therefore, it seems natural to study these issues directly to better understand them.

In [2], we made some initial steps in this direction; we try to detect components in unsuccessful proof attempts, run these components individually, and then use the best obtained clauses from these individual runs to complete the proof. This approach seems promising, but it is hard to analyze what is actually happening inside. A key issue being that detecting components in our setting is an unsupervised task with no ground truth available. For that reason, we decided to create a dataset that allows us to better analyze this behavior with the available training and testing examples.

Although there are various mathematically well-defined ways to combine two (or more) components into one problem, we take advantage of already available mathematical formal libraries and created a natural dataset with possibly many types of components. We take the Mizar Mathematical Library, which can be exported to first-order logic, and hence have a long list of problems (theorems or lemmata) with their dependencies. Many of these problems can be proved by ATPs (various versions of E, ENIGMA, Vampire,...) using a reasonable premise selection. Moreover, these proofs can be analyzed, and we obtain for each problem a minimized set (or more sets) of dependencies required for the proof. For example, we have a theorem T (exported from Mizar) that is automatically provable from the lemmata $L_1, \ldots, L_n$ (also exported from Mizar) by an ATP. Assume that L_1 and L_2 are two lemmata from different mathematical libraries. Furthermore, L_1 is provable from $L_1^1, \ldots, L_1^m$ and L_2 is provable from $L_2^1, \ldots, L_2^k$, where $L_1^1, \ldots, L_1^m$ and $L_2^1, \ldots, L_2^k$ have no overlap or insignificant overlap. Then we can expand L_1 and L_2 in T, so a new expanded problem is to prove T from $L_1^1, \ldots, L_1^m, L_2^1, \ldots, L_2^k, L_3, \ldots, L_n$. This new problem has well-defined components (the expansions of L_1 and L_2) as we wanted. In fact, we can produce many such datasets depending on requirements like number of components, overlap of components, size of expansions, ...

It is possible to understand exported Mizar problems as a large graph in which the nodes are first-order formulae and the edges show dependencies; an edge from L to T means that L was used in a proof of T. In fact, we usually have more types of edges because we have more minimized solutions to a problem. With such a graph, we can easily produce the expansions described previously. In our particular case, we obtain a graph with 67795 nodes, where 29687 nodes are leaves (having no dependencies) and 19334 nodes are roots (not used in other proofs). Although the number of roots may seem a bit high, it is because we are only interested in

*Supported by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.



(a) A common situation. (b) A good situation.

Figure 1: Processed clauses (nodes) and their derivations (arrows) in two successful runs are shown. Red (green) nodes correspond to clauses obtained from expanded Lemma 1 (2) including derived clauses that use only expanded Lemma 1 (2) as their dependencies. Brown clauses depend only on clauses different from Lemma 1 and 2 expansions in their derivations. Orange clauses depend on a mixture of previous types of clauses. An arrow from c_1 to c_2 means that c_1 was used in the derivation of c_2. The final contradiction is the gray node.

problems that can be proved by ATPs, and hence many real dependencies may be lost. An advantage of such a graph is that train/test splits are easy to define. One way is that we, for example, randomly split root nodes into two parts—train and test roots. All nodes having a path to a train root (or are train roots themselves) are now called train nodes. On the other hand, remaining nodes, which have no path to a train node (and hence they are test roots or have a path to a test root), are test nodes. Clearly, there are more train nodes than test nodes. We obtain a fair split of train and test datasets by expanding only train and test nodes, respectively. In our example, we may obtain 275K possible expansions for the train dataset and 5K possible expansions for the test dataset; however, they correspond to approximately 3K unique problems on the train dataset and roughly 100 unique problems on the test dataset, because there are many possible ways to expand a problem.

Having such a dataset enables further analysis of algorithms used to obtain components in [2] and compare them with intended components (our expansions). A clause selection guidance based on GNNs[1], see [4], allows one to extract representations of individual clauses in a latent space and then identify components there, moreover, we can visualize them and also display intended components, see Figure 1. In Figure 1a we see a common situation where it is hard to distinguish both expansions. Sometimes, like in 1b, they are reasonably separated. However, the former situation is much more common than the latter. This improves when we retrain our algorithms using the new dataset, however, preliminary results show that further analysis of our pipeline is still necessary.

Clearly, our dataset is useful for many other experiments, including conjecturing (reconstructing L_1 and L_2), and we will present some preliminary results in this direction.

[1]Note that GNNs are useful for similar problems that involve components, as was recently observed [7] that GNNs align with dynamic programming. Instead of learning algorithms, for example, in combinatorial optimization, from scratch, it is beneficial to learn their individual subroutines (modules) separately, cf. [1, 3].

References

[1] Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In Zhi-Hua Zhou, editor, *IJCAI-21*, pages 4348–4355. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

[2] Karel Chvalovský, Jan Jakubův, Miroslav Olšák, and Josef Urban. Learning theorem proving components. In Anupam Das and Sara Negri, editors, *TABLEAUX 2021*, pages 266–278, Cham, 2021. Springer International Publishing.

[3] Andrew Dudzik and Petar Velickovic. Graph neural networks are dynamic programmers. *CoRR*, abs/2203.15544, 2022.

[4] Jan Jakubův, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[5] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[6] Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[7] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, K. Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *ICLR*, 2020.