Inductive Logic Programming for Interactive Theorem Proving

Liao Zhang[1,2]

[1]University of Innsbruck
[2]Czech Technical University

1 Introduction

Researchers have applied various machine learning techniques to predict tactics and search proofs for interactive theorem proving (ITP). Tactician [2] and TacticToe [4] apply classical statistical machine learning techniques such as k-nearest neighbors (k-NN), random forests, and naive-bayes to predict tactics. CoqGym [8] and HOList [1] use neural networks such as Tree LSTM and GNN for predicting tactics. Recently, large language models have also been applied to making tactic predictions in GPT-f [5].

However, all of the existing approaches suffer from some limitations. First, most existing approaches are unexplainable. Both neural networks and random forests are unexplainable. But explainability is crucial in making tactic suggestions. In many proof assistants, users are able to define their own tactics. It is common that the machine learning model suggests some rarely known tactics from large datasets. Understanding the reasons behind the suggestions enables the users to determine which tactic to choose in order to continue the proof.

Second, propositional features make the learners lose explainability and expressivity. Propositional features are not able to distinguish features that belong to different hypotheses. They also cannot keep the tree structures of the proof state.

Finally, including neural networks, none of the approaches are actually good at proving complicated theorems. Most of the theorems proved by them are relatively trivial. This inspires us to try some new techniques to overcome the limitations of the existing statistical learning techniques and improve the performance.

Given the limitations of existing approaches, we want to investigate a very different machine learning technique called inductive logic programming (ILP). The goal of ILP is to induce a set of logical rules to generalize the training examples. ILP is explainable and can express complicated relationships. We aim at combining ILP and existing statistical learning techniques to develop

```
dif(HypName1, HypName2).
dif(HypPosition1, HypPosition2).
dif(GoalPosition1, GoalPosition2).
position_left(GoalPosition1, GoalPosition2).
position_left(HypPosition1, HypPosition2).
position_above(HypPosition1, HypPosition2).
position_above(GoalPosition1, GoalPosition2).
% For a proof state with ProofStateId, two subterms rooted at
% GoalPosition1 and GoalPosition2 are the same.
eq_subterm(ProofStateId, GoalPosition1, GoalPosition1).
eq_subterm(ProofStateId, HypPosition1, HypPosition2).
eq_subterm(ProofStateId, GoalPosition, HypPosition).
```

Figure 1: The predicates that can be learned by Aleph.

novel machine learning techniques for making tactic predictions for the Coq proof assistant [7].

2 Methods

The goal of ILP is to induce first- or higher-order logic programs (hypotheses) to generalize training examples. In this work, we use the ILP system Aleph [6] to learn hypotheses. The learned hypothesis is conceived as a first-order Prolog clause as depicted below.

`tac(ProofStateId, string of the tactic) :- A1, ..., Am`

Here, the head `tac` is a tactic and the body `A1`, . . . , `Am` consists of atoms. The `tac` takes the identifier of the proof state and the string of the tactic as arguments. Every node in the abstract syntax tree (AST) of the proof state is converted to a fact. Assume a proof state identified with the number 4. A node `ng` in the position `goal_pos` of the goal is converted to `ng(4, goal_pos)`. A node `nh` in the position `hyp_pos` of the hypothesis `h` is converted to `ng(4, h, hyp_pos)`.

Besides using the nodes in the AST as predicates, we also define the predicates in Figure 1. The defined predicates can capture the relative positions of nodes in the AST. The predicate `eq_subterm` can show the equality between subterms. The clause of `assumption` (`simpl`) indicates that if a proof state satisfies this clause, then `assumption` (`simpl`) is appropriate for the proof state and vice versa.

Figure 2 depicts two clauses learned by Aleph. The first clause is learned from the proof state `H : (i < n)%nat, ... |- (i < n)%nat` It shows that if there is a Coq proof state that has `<` both in a hypothesis and the goal, and two subterms rooted with `<` are the same, then assumption may be suitable. The second clause is learned from the state `... H1 : (i <= N)\%nat |- (y = y ^ 1)`. After the execution of `simpl`, it becomes `... H1 : (i <= N)%nat |- (y =`

```
% The predicate coq_Init_Peano_lt denotes the less than identifier (<).
tac(A,"assumption") :-
    coq_Init_Peano_lt(A,B), coq_Init_Peano_lt(A,C,D), eq_subterm(A,B,D).
% The predicate coq_Init_Peano_le denotes the less than or equal to
% identifier (<=). The predicates coq_Init_Datatypes_O and coq_Init_Datatypes_S
% denote two identifiers for defining natural numbers. The predicate
% coq_Reals_Rpow_def_pow denotes the identifier of the power operation (^).
tac(A,"simpl") :-
    coq_Init_Peano_le(A,B,C), coq_Init_Datatypes_O(A,D),
    coq_Reals_Rpow_def_pow(A,E), position_above(E,D),
    coq_Init_Datatypes_S(A,F), position_above(F,D).
```

Figure 2: Two clauses learned for `assumption` and `simpl`.

`y * 1)`. The clause shows that the power of one is likely to be simplified to the multiplication by one.

$$jacard(f_1, f_2) = \frac{f_1 \cap f_2}{f1 \cup f_2}$$

The k-NN classifier is too simple compared to ILP. It merely calculates the distance (such as the Jaccard similarity shown above) between the features of the proof states and cannot recognize the importance of each feature. A proof state may contain more nodes than those presented for the clauses in Figure 2; nevertheless, ILP can try to only keep the important atoms in the clause.

Compared to propositional features, first-order logic clauses are better at expressivity and explainability. Current proof automation systems like Tactician and Enigma [3] merely use propositional features. They use vertical paths and horizontal paths up to a certain length. However, they do not keep the tree structure. A longer path may also be useful. Sometimes we do not need all the nodes in the path. For instance, we may not need the node `b` in the vertical path `a-b-c` and may only want to know that `a` is above `c`. First-order logic can also represent which hypothesis the feature belongs to. However, propositional features merge features from all hypotheses together.

The experiments of the combination of ILP and statistical learning are still being conducted. First, we will generate several rules for each tactic. Then, we will use k-NN, random forests, and GPT-2 to predict a sequence of tactics for a proof state. For each predicted tactic, if the proof state cannot satisfy any rules related to the tactic, we will filter it from the predictions.

References

[1] Bansal, K., Loos, S., Rabe, M., Szegedy, C., and Wilcox, S. Holist: An environment for machine learning of higher order logic theorem proving.

In *International Conference on Machine Learning* (2019), PMLR, pp. 454–463.

[2] Blaauwbroek, L., Urban, J., and Geuvers, H. The tactician: A seamless, interactive tactic learner and prover for coq. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings* (2020), Springer, pp. 271–277.

[3] Chvalovský, K., Jakubův, J., Suda, M., and Urban, J. Enigma-ng: Efficient neural and gradient-boosted inference guidance for e. In *Automated Deduction – CADE 27* (2019), P. Fontaine, Ed.

[4] Gauthier, T., Kaliszyk, C., and Urban, J. Learning to reason with hol4 tactics. *arXiv preprint arXiv:1804.00595* (2018).

[5] Polu, S., and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393* (2020).

[6] Srinivasan, A. The aleph manual.

[7] The Coq Development Team. Coq reference manual 8.11.1, 2020.

[8] Yang, K., and Deng, J. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning* (2019), PMLR, pp. 6984–6994.