Learning to Prove Theorems via Interacting with Proof Assistants Supplementary Material

Kaiyu Yang [1] **Jia Deng** [1]

A. Details on Constructing the Dataset

A.1. Building the Coq projects

We manually compile and install the Coq standard library and a few projects (such as math-comp) that are frequently required by other projects. For the rest, we try compiling them automatically using simple commands such as "`./configure && make`", and we take whatever compiles, ending up with 123 projects and 3,061 Coq files (excluding the files that do not contain any proof).

A.2. Reconstructing the Proof Tree

After applying a tactic, the current goal disappears, and a set of new goals emerge, which become the children of the current goal in the proof tree. We can identify the edges of the tree by tracking how goals emerge during the proof. For example, if the list of goals changes from [2, 7] to [8, 9, 7], we know that node 2 has two children: 8 and 9.

In certain cases, a tactic can affect more than one goal, and it is unclear who should be the parent node. This can happen when a tactic is applied to multiple goals using a language feature called goal selectors (by default, a tactic is applied only to the first goal). However, goal selectors are rarely used in practice. We discard all such proofs and lose only less than 1% of our data. For the remaining data, only one goal disappears at each step, and we can build the proof trees unambiguously.

[1]Department of Computer Science, Princeton University. Correspondence to: Kaiyu Yang <kaiyuy@cs.princeton.edu>, Jia Deng <jiadeng@cs.princeton.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

A.3. Extracting Synthetic Proofs from Intermediate Goals

Given an intermediate goal, it is straightforward to treat it as a theorem by adding its local context to the environment. For example, in Fig. A, the goal G2 can be a theorem $(a + b) + c = a + (b + c)$ in the environment augmented by a b and c. Extracting synthetic proofs for the new theorem requires nontrivial processing. One straightforward proof would be the sequence of tactics that follows G2 in the original human-written proof: "`induction a as [|a']. trivial. simpl; rewrite IHa'. trivial.`". This proof corresponds to the sub-tree rooted at G2.

However, there are potentially shorter proofs for G2 using a trimmed sub-tree. For example, if we only apply the first tactic to generates G3 and G4, then we can treat them as premises H3 and H4, and complete the proof by "`apply H3. apply H4.`". Equivalently, we can also use `auto` to complete the proof. This technique of converting unsolved sub-goals into premises allows us to generate synthetic proofs of controllable lengths, by taking a sequence of tactics from the original proof and appending an `auto` at the end.

We need to take extra care in converting a goal into a premise. For example, it is easy to treat G3 as a premise, but G4 needs some care. G4 depends on `a'`, which is missing in G2's context. In order to convert G4 into a well-formed term in G2's context, we apply the "`generalize dependent`" tactic to push a local premise into the statement of the goal. When applied to G4, it generates H4 in Fig. A, which can be added to G2's local context.

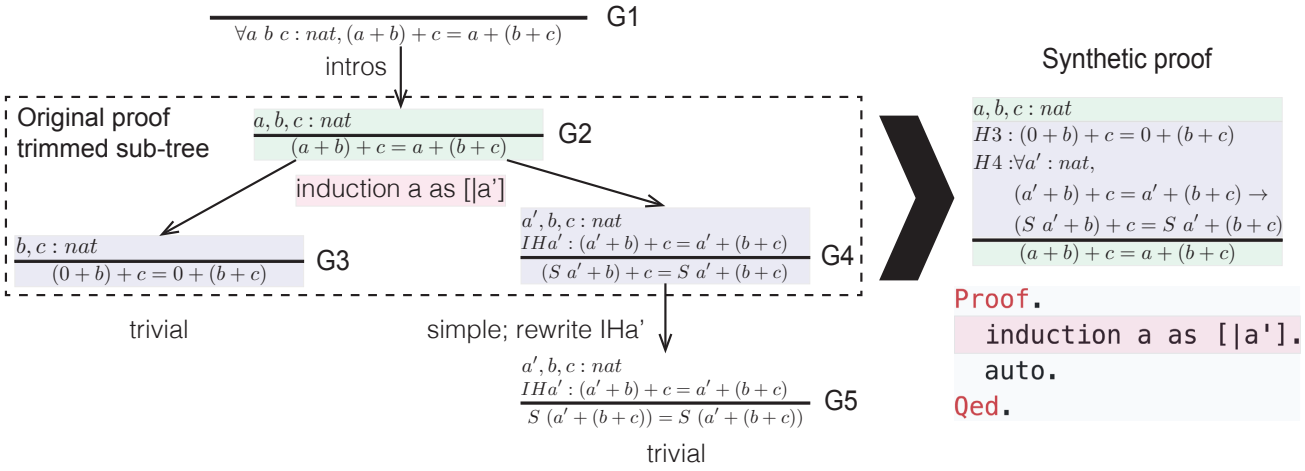

Figure A. Extracting a synthetic proof from the intermediate goal G2. Goals G3 and G4 are converted into premises in G2's local context. The synthetic proof corresponds to a trimmed sub-tree rooted at G2.

B. The Space of Tactics for ASTactic

Below is the context-free grammar in extended Backus-Naur form for the tactic space. The start symbol is $tactic\_expr$.

```
tactic_expr : intro
            | 'apply' term_commalist1 reduced_in_clause
            | 'auto' using_clause with_hint_dbs
            | 'rewrite' rewrite_term_list1 in_clause
            | 'simpl' in_clause
            | 'unfold' qualid_list1 in_clause
            | destruct
            | induction
            | 'elim' QUALID
            | 'split'
            | 'assumption'
            | trivial
            | 'reflexivity'
            | 'case' QUALID
            | clear
            | 'subst' local_ident_list
            | 'generalize' term_list1
            | 'exists' LOCAL_IDENT
            | 'red' in_clause
            | 'omega'
            | discriminate
            | inversion
            | simple_induction
            | constructor
            | 'congruence'
            | 'left'
            | 'right'
            | 'ring'
            | 'symmetry'
            | 'f_equal'
            | 'tauto'
            | 'revert' local_ident_list1
            | 'specialize' '(' LOCAL_IDENT QUALID ')'
            | 'idtac'
            | 'hnf' in_clause
            | inversion_clear
            | contradiction
            | 'injection' LOCAL_IDENT
            | 'exfalso'
            | 'cbv'
            | 'contradict' LOCAL_IDENT
            | 'lia'
            | 'field'
            | 'easy'
            | 'cbn'
            | 'exact' QUALID
            | 'intuition'
            | 'eauto' using_clause with_hint_dbs

LOCAL_IDENT : /[A-Za-z_][A-Za-z0-9_']*/
```

```
QUANTIFIED_IDENT : /[A-Za-z_][A-Za-z0-9_']*/

INT : /1|2|3|4/

QUALID : /([A-Za-z_][A-Za-z0-9_']*\.)*[A-Za-z_][A-Za-z0-9_']*/

HINT_DB : /arith|zarith|algebra|real|sets|core|bool|datatypes|coc|set|zfc/

local_ident_list :
                  | LOCAL_IDENT local_ident_list

local_ident_list1 : LOCAL_IDENT
                  | LOCAL_IDENT local_ident_list1

qualid_list1 : QUALID
             | QUALID ',' qualid_list1

term_list1 : QUALID
           | QUALID term_list1

term_commalist1 : QUALID
                | QUALID ',' term_commalist1

hint_db_list1 : HINT_DB
              | HINT_DB hint_db_list1

reduced_in_clause :
                  | 'in' LOCAL_IDENT

in_clause :
          | 'in' LOCAL_IDENT
          | 'in' '|-' '*'
          | 'in' '*'

at_clause :
          | 'at' INT

using_clause :
             | 'using' qualid_list1

with_hint_dbs :
              | 'with' hint_db_list1
              | 'with' '*'

intro : 'intro'
      | 'intros'

rewrite_term : QUALID
             | '→' QUALID
             | '←' QUALID

rewrite_term_list1 : rewrite_term
                   | rewrite_term ',' rewrite_term_list1
```

```
destruct : 'destruct' term_commalist1

induction : 'induction' LOCAL_IDENT
          | 'induction' INT

trivial : 'trivial'

clear : 'clear'
      | 'clear' local_ident_list1

discriminate : 'discriminate'
             | 'discriminate' LOCAL_IDENT

inversion : 'inversion' LOCAL_IDENT
          | 'inversion' INT

simple_induction : 'simple induction' QUANTIFIED_IDENT
                 | 'simple induction' INT

constructor : 'constructor'
            | 'constructor' INT

inversion_clear : 'inversion_clear' LOCAL_IDENT
                | 'inversion_clear' INT

contradiction : 'contradiction'
              | 'contradiction' LOCAL_IDENT
```