LISA: Language models of ISAbelle proofs

Albert Qiaochu Jiang  
University of Oxford  
albert594250@gmail.com

Wenda Li  
University of Cambridge  
w302@cam.ac.uk

Jesse Michael Han  
OpenAI  
jessemichaelhan@gmail.com

Yuhuai Wu  
University of Toronto  
ywu@cs.toronto.edu

ABSTRACT
We introduce an environment that allows interaction with an Isabelle server in an incremental manner. With this environment, we mined the Isabelle standard library and the Archive of Formal Proofs (AFP) and extracted 183K lemmas and theorems. We built language models on this large corpus and showed their effectiveness in proving AFP theorems.

1 INTRODUCTION
There has been a surge of interests recently in applying machine learning models for theorem provers. Examples include [3, 6–8, 12, 14], all of which demonstrate great promises of machine learning models in proving new theorems. In this work, we propose to mine the libraries used by the Interactive Theorem Prover (ITP) Isabelle, namely, the Isabelle standard library and the Archive of Formal Proofs. The libraries have been mined previously for proof method recommendations based on hand-crafted features [9, 10].

Contributions
• We built an environment where agents can interact with the Isabelle theorem prover in an incremental manner. This enables learning-based agents to conjecture in the Isar language.
• We mined the Archive of Formal Proofs and the standard library of Isabelle. We extracted 183K theorems and 2.16M proof steps. This is one of the largest proof corpora for interactive theorem provers.
• We trained large language models on this corpus and obtained the first results of using such models to prove theorems in this new dataset.

2 ENVIRONMENT AND DATASET
We created an environment where theorem proving is modelled as a sequential decision process. Initially, the environment will load a selected theorem and we have access to the top level state. At each time-step, the agent produces a proof step of arbitrary length. The environment then applies the proof step to the top level state and iterates the process if the theorem has not been proved. We show the proof process of a simple theorem in Figure 1. The theorem declaration initialises the first proof state. The proof states in the middle row represent the stage of the proof progress and the proof steps in the bottom row are what the agent should produce. We support three different kinds of inputs: with proof states only, with previous steps only, and with both proof states and previous steps. For example, the previous steps when the agent should produce 

Figure 1: An illustration of the relationship between theorems, proof states, and proof steps.

"done" consist of "apply (rule impI)" and "apply assumption". Because Isabelle provides a Partially Observable Markov Decision Process (POMDP) with the proof states being the observation, conditioning on the previous steps of the proof helps the agent to reconstruct the state of the proof.

The unique feature that Isabelle enables in our system is that we can execute proofs token by token. The benefits brought by this feature include that we can make copies of a certain proof state and try multiple different methods very conveniently. This also allows us to change the order in which a proof is written, which makes proof sketching possible: we can potentially first sketch a proof skeleton containing the keyword "sorry", which assumes that the given statement before it can be proven. Then, by saving all the states before the "sorry" command and attempting them after the skeleton has been completed, we allow a machine to write proofs in the same order a human sometimes would.

With this environment, we mined a total of 183K theorems from the Isabelle standard library [11] and the Archive of Formal Proofs (AFP) [1]. We then extracted a total of 2.16 million pairs of inputs and proof steps. This forms a dataset useful for theorem proving: if an agent can produce the correct proof step when prompted with an arbitrary proof state, it will be able to prove the theorem. We used a 95%/1%/4% random split to divide the proof corpus into the train/valid/test sets. We show some dataset statistics in Table 1.

3 EXPERIMENTS
3.1 Setup
We started by taking a language model pre-trained on the WebMath dataset for 72B tokens, similar to the GPT-f models applied to Metamath [12] and Lean [5]. We then fine-tuned the language
models only on the AFP part of the dataset, due to time constraints. The architecture we chose was a decoder-only transformer similar to GPT-3 [4]. All models have 163M non-embedding parameters. We use the same BPE encoding as GPT-3 [4]. For fine-tuning, we used a batch size of 2048, a learning rate of 0.005, a 100-step ramp-up, and decayed the learning rate according to a cosine schedule over 64B tokens; we early-stopped according to validation perplexity after 35B elapsed tokens.

3.2 Evaluation
We used a best-first search strategy at evaluation, similar to that of [5, 12]. We initialise and maintain a priority queue of top level states, sorted by their cumulative log probability. The cumulative log probability of a top level state is the sum of log probabilities of all the previous proof steps the agent takes to arrive at the current state. Initially, the priority queue contains only the top level state right after the declaration of the theorem, with a cumulative log probability of 0. At each search step, we pop the head of the priority queue to retrieve the top level state with the highest probability. We then query the language model for a set of 16 proof step candidates, with a temperature of 1.0. For each of the candidates, we duplicate the top level state, apply the candidate to it, and calculate the updated cumulative log probability. If the application of the candidate is successful, we add the resulted top level state to the queue. The queue has a length of 16 (i.e., it only maintains 16 entries with the highest cumulative log probabilities). If one of the resulted top level state shows that the proof is complete, we consider the proof attempt successful. If the queue is empty, or a timeout of 120s is spent on one attempt, or the number of queries exceeds 100, we consider the attempt a failure.

3.3 Results
We evaluated our language model with the best-first search strategy on a test set of 4000 theorems. 33.2% of the theorems were successfully proved. We analysed the failure causes of the rest of the theorems. 59.1% of the attempts failed because of the time limitation, 0.2% of the attempts failed because of the query number limitation and 7.6% of the attempts failed because the priority queue was empty at some point in the proving process. We show two successful proofs generated by our language model, and contrast them with the proofs in the AFP.

### Theorem 1

**Lemma foldr_foldr_concat:**

\[
\text{foldr (foldr f) m a = foldr f (concat m) a}
\]

**Original proof**

\begin{verbatim}
proof (induct m arbitrary: a)
  case Nil show ?case by simp
next
  case (Cons v m a)
  show ?case
  unfolding concat.simps foldr_Cons o_def Cons
  unfolding foldr_append by simp
qed
\end{verbatim}

**Our proof**

\begin{verbatim}
by (induct m arbitrary: a) simp_all
\end{verbatim}

### Theorem 2

**Lemma scast_ucast_1:**

\[
\forall \text{is_down (ucast :: 'a word \Rightarrow 'b word); is_down (ucast :: 'b word \Rightarrow 'c word)} \Rightarrow (scast (ucast (a :: 'a::len word) :: 'b::len word) :: 'c::len word) = ucast a)
\]

**Original proof**

\begin{verbatim}
by (metis down_cast_same ucast_eq ucast_down_wi)
\end{verbatim}

**Our proof**

\begin{verbatim}
using unat_ucast apply
  apply (simp add: ucast_def unat_ucast)+
  apply (subst down_cast_same[symmetric])
  apply (simp add: is_down)+
  apply (rule word_eqI)
  apply (simp add: nth_ucast)
  apply safe
  apply simp
  done
\end{verbatim}

As a baseline, we also considered using greedy search. This is equivalent to best-first search with the queue length = 1. This agent, as a consequence, only proved 28.3% of the theorems.

4 CONCLUSIONS AND FUTURE WORK
We extracted a large corpus from Isabelle proofs and examined the performance of language models in proving theorems on the dataset. We showed that a non-trivial proportion of problems on AFP can be solved by the application of a language model and a
best-first search. The successful proofs demonstrated the language model’s ability to compose succinct, or novel proofs.

The proof assistant Isabelle provides a very convenient command that allows users to conjecture (‘have’). With our environment that interacts with the proof assistant in a very flexible manner, and our rich dataset, we can set out to further study how machines can learn to conjecture, and to reason about the proof construction more generally. Specifically, by learning from human conjectures, computer-assisted theorem provers are endowed with the ability to sketch proofs. This can be organically integrated with symbolic methods such as “nitpick” and “sledgehammer”.

REFERENCES


