Proof Construction for Coq by Tactic Learning

Lasse Blaauwbroek*

Czech Institute for Informatics, Robotics and Cybernetics,
Czech Republic
lasse@blaauwbroek.eu

Abstract

We present some early work being done to utilize Artificial Intelligence for proof search in the Coq theorem prover. In a similar vein as the TacticToe project for HOL4 [2], we are working on a system that finds proofs of goals on the tactic level, by learning from previous tactic scripts. Learning on the level of tactics has several advantages over more low-level approaches. First, this allows for much coarser proof steps, meaning that during proof search more complicated proofs can be found. Second, it allows for the usage of custom built, domain specific tactics that where previously defined and used in the development. This will allow for better performance of the system in very specialized domains. The rest of this abstract will describe the required components of our system. Since a number of technical issues need to be addressed, we hope to solicit feedback from the Coq developers at the workshop.

Proof Recording The first component of the system is the recording of previous proofs. As said, this is done on the level of tactics. When a tactic script is executed, we record the goal state before and after the execution of each tactic. The diff between the state before and after the tactic then represents the action that has been performed by a tactic. By recording many of these instances for a tactic, we create a database that contains an approximation of the semantic meaning of tactics.

A major question here is what exactly constitutes a tactic in Coq. One option is to decompose a script into a series of primitive tactic invocations, and record those. On the other side of the extreme, one could view every vernacular command as one whole tactic. The first option means that the advantages of the system are greatly diminished, because then we are working on a very low level and no custom tactics will be recorded. The second option means that almost all tactics will be unique. The best solution is likely to lie somewhere in between. Our current approach is to fully decompose tactic scripts but never to unfold identifiers. Another issue is how tactic parameters are treated. For now, the same tactic invocation with different arguments is seen as a completely different tactic.

Recording the proof states before and after every tactic is a major technical challenge. A very early prototype of our system took the source code of a Coq development and added the recording of tactics before and after every tactic invocation. This approach has many disadvantages. First, correctly parsing a Coq source file is a rather tricky undertaking. Second, the workflow for the end-user is awkward because an external tool must be run to transform the source code, and then the whole development must be re-run through the Coq toplevel. To improve upon this, we are now working to perform the recording via a Coq plugin. This has required us to program a hook into the Coq source code to intercept the execution of each tactic vernacular. However, we can now take advantage of the Coq parser and have a tighter integration.

*This work was supported by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466)

Tactic Prediction After creating a database of tactics and recorded proof states, we now wish to predict the correct tactic to use in order to make progress in a new, unseen proof state. For this, we must find a proper characterization of the proof states. For our initial prototype, we have opted to reuse feature characterization that is already present in the CoqHammer system [1]. CoqHammer characterizes a formula as a vector containing all identifiers and pairs of adjacent identifiers in the abstract syntax tree. To find a list of likely matches for a new goal, a fast k-nearest neighbor algorithm is run on the vectors.

Proof Search It has to be acknowledged that it is unlikely that the tactic prediction will predict the correct tactic every time. For this reason, a proof search must be performed. In the TacticToe system, initially an A*-style algorithm was used to guide the search. Later, taking inspiration from AlphaGo Zero [3] a Monte Carlo Tree Search algorithm was used. In our initial implementation, we are using a simple breadth first search algorithm. However, in the future we intend to move in the same direction as the TacticToe system.

Proof Reconstruction When the proof search successfully finds a proof, the final step is to create a reconstruction of the proof. Conceptually, we just backtrack trough our search tree and list every tactic in our path. However, an important feature of a tactic-level system is that when a proof is found, a tactic script is built that represents this proof. This will allow the user to inspect, understand and modify the proof easily. In order to make this work, the script needs to be short and intuitively understandable. Hence, it is not acceptable if the proof is just a listing of tacticals for each branch and sub-branch of the proof. For this, we must deduplicate subproofs, use short names for tactics and automatically combine tactics into more complicated expressions.

Future Extensions After the initial implementation is finished, we plan on creating several extensions that should improve the effectiveness of the system. In the case of TacticToe, it has been shown that the system is much more effective when combined with a Hammer. We therefore intend to interface our system with CoqHammer [1]. The idea is that during proof search, sometimes a short invocation of the CoqHammer is made to attempt to finish the proof. Another avenue of extension is to try learning the correct parameters that need to be supplied to tactics. In TacticToe this has been an ongoing challenge, so we expect similar difficulties here.

Finally, we would like to investigate the possibility to do speculative proving. The motivation for this idea is that when one tries to prove a theorem, this often requires a lemma that one has forgotten to formalize before starting proof on the theorem. We propose that during the proof search, the system should sometimes make small leaps of faith. For example, when the goal and a assumption look very similar, the system might assume that they can be proven equal. With this assumption, the proof search can then continue. When one or more proofs are found, the user will be presented with these proofs and the holes that still need to be filled to complete the proof.

References

[1] Łukasz Czajka and Cezary Kaliszyk. Hammer for coq: Automation for dependent type theory. *Journal of Automated Reasoning*, pages 1–31, 2018.

[2] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with HOL4 tactics. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, pages 125–143, 2017.

[3] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.