Tactic Learning for Coq

Lasse Blaauwbroek*

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

Abstract

We present ongoing work being performed to utilize Artificial Intelligence for proof search in the Coq theorem prover. In a similar vein as the TacticToe project for HOL4 [4], 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. Currently our system is not yet capable of proof search but does support live tactic learning and provides feedback to the user in the form of tactic suggestions. The rest of this abstract will describe the required components of our system and an evaluation of our prediction system.

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. On a low level, Coq utilizes a backtracking monad in which tactics can be written [8]. This monad is not immediately accessible by end-users. For that, a number of tactic languages exist that are then compiled into the backtracking monad. The most used one is the Ltac language [3]. In an ideal world, we would record tactics on the level of the proof monad since that would allow us to record tactics from all existing tactic languages. However, this turns out to be a major technological challenge. Therefore we have chosen to only record tactics of the Ltac language.

Withing the Ltac language, it is also not immediately clear what a tactic is. 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. At the moment, we see every vernacular command as one tactic, with the exception of tactic composition and tactic dispatching. In order to record a tactic script, conceptually we replace the tactic with a custom recording tactic that receives the original tactic as an argument. For example, the tactic script `tac1; [tac2 | tac3]; tac4` will be converted to `r (tac1); [r (tac2) | r (tac3)]; r (tac4)`, where `r` is the recording tactic. The recording tactic first records the proof state before the tactic, then executes the original tactic and

*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)

finally records the proof state after the tactic. Our current approach also means that if tactics require arguments, then every instance of this tactic with a different argument will be seen as a unique tactic. In the future we intend to change this by also performing some machine learning on the parameters of tactics.

Tactic recording can happen both in batch mode and interactive mode. When recording in batch mode, an entire Coq file is processed and a file containing the recorded data of all tactic invocations is outputted. This file can later be used for predictions in other Coq files. In interactive mode our system follows the actions of the user. This means that when a tactic is executed, the system immediately learns from this. When a tactic is removed from the script the system also automatically unlearns that tactic. This ensures that what we learn is consistent with what would be learned is the script gets executed in batch mode.

In order to achieve this, our system is rather deeply integrated with the Coq system itself. The code is writted as a Coq plugin, but also requires a few minor modifications to the Coq source code. This level of integration is a distinguishing feature compared to existing machine learning systems for Coq. Our system is fully implemented in Ocaml and has no external dependencies. The ML4PG system [9] also provides tactic predictions and other statistics but is instead integrated with the Proof General [1] proof editor and requires connections to Matlab or Weka to function. The SEPIA system [5] provides proof search using tactic predictions and is also integrated with Proof General. It should be noted, however that their proof search is only based on tactic traces and does not make predictions based on the proof state. Finally, GamePad [6] is a framework that integrates with the Coq codebase and allows machine learning to be performed in Python. Similar to our system this is in the early stages of development and no evaluation of tactic prediction has been performed yet.

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 [2]. 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. We compare neighbors using the Jaccard similarity with TF-IDF weighted features as described by Kaliszyk and Urban [7].

In order to evaluate our predictions, we use Coq's standard library. During the compilation of the library, before every tactic invocation we try to predict the correct tactic using the database collected until that point. We then check if that tactic corresponds with the tactic used in the source file. The standard library consists of 145866 tactic invocations. If one where to predict a random tactic from the database a success rate of 4.9% can be expected. The best possible success rate lies at 67.2%. This is because the correct tactic is not always present in the database. Using the machine learning method described above, we are able to predict the correct tactic 21.7% of the time. When we look for the correct tactic in the top 10 predictions we can increase this number to 47.7%. We also performed an evaluation using a naive Bayes classifier. The results are rather similar with a prediction rate of respectively 20.8% and 45.3%. We expect that even simpler machine learning techniques will also perform reasonably well.

Proof Search From the previous paragraph it becomes clear that it is unlikely that the tactic prediction system will predict the correct tactic every time. For this reason, a proof search must be performed. In the TacticToe system for HOL4, initially an A*-style algorithm was used to guide the search. Later, taking inspiration from AlphaGo Zero [10] a Monte Carlo Tree Search algorithm was used. We are currently working on a similar search system. This has turned out to be a challenge because a tactic that was intended for one location in a script can not always easily be executed in a different location of a script.

References

[1] David Aspinall. Proof general: A generic tool for proof development. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 38–42. Springer, 2000.

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

[3] David Delahaye. A tactic language for the system coq. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000.

[4] 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.

[5] Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. SEPIA: search for proofs using inferred automata. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 246–255. Springer, 2015.

[6] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *CoRR*, abs/1806.00608, 2018.

[7] Cezary Kaliszyk and Josef Urban. Stronger automation for flyspeck by feature weighting and strategy evolution. In Jasmin Christian Blanchette and Josef Urban, editors, *Third International Workshop on Proof Exchange for Theorem Proving, PxTP 2013, Lake Placid, NY, USA, June 9-10, 2013*, volume 14 of *EPiC Series in Computing*, pages 87–95. EasyChair, 2013.

[8] Florent Kirchner and César A. Muñoz. The proof monad. *J. Log. Algebr. Program.*, 79(3-5):264–277, 2010.

[9] Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine learning in proof general: Interfacing interfaces. In Cezary Kaliszyk and Christoph Lüth, editors, *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, UITP 2012, Bremen, Germany, July 11th, 2012.*, volume 118 of *EPTCS*, pages 15–41, 2012.

[10] 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.