TACTICZERO: LEARNING TO PROVE THEOREMS FROM SCRATCH WITH DEEP REINFORCEMENT LEARNING

Minchao Wu
Australian National University
Minchao.Wu@anu.edu.au

Michael Norrish
Data61, CSIRO
michael.norrish@data61.csiro.au

Christian Walder
Data61, CSIRO
christian.walder@data61.csiro.au

Amir Dezfouli
Data61, CSIRO
amir.dezfouli@data61.csiro.au

ABSTRACT

We propose a novel approach to interactive theorem-proving (ITP) using deep reinforcement learning. Unlike previous work, our framework is able to prove theorems both end-to-end and from scratch (*i.e.*, without relying on example proofs from human experts). We formulate the process of ITP as a Markov decision process (MDP) in which each state represents a set of potential derivation paths. The agent learns to select promising derivations as well as appropriate tactics within each derivation using deep policy gradients. This structure allows us to introduce a novel backtracking mechanism which enables the agent to efficiently discard (predicted) dead-end derivations and restart the derivation from promising alternatives. Experimental results show that the framework provides comparable performance to that of the approaches that use human experts, and that it is also capable of proving theorems that it has never seen during training. We further elaborate the role of each component of the framework using ablation studies.

1 INTRODUCTION

In this work, we propose a novel reinforcement-learning (RL) approach to learning interactive theorem proving in an end-to-end manner and from scratch. The RL agent interacts with the HOL4 interactive theorem prover (Slind & Norrish, 2008) environment to apply tactics. Through this interaction, and receiving feedback from the environment, the agent learns to choose promising subgoals to attack, as well as the tactics and tactic-arguments to apply to those goals.

Our Contributions are as follows.

- We provide a formulation of using ITP as a Markov decision process (MDP) (Bellman, 1957) in which flexible state representations enable tracking multiple subproofs during the search process. The framework supports not only learning of tactic applications (with theorem names or terms as arguments), but also proof search strategies.
- We propose an RL architecture using multiple recurrent and feed-forward neural network modules, to solve the above MDP. We use policy gradients (Williams, 1992) to jointly learn to apply actions at the back-tracking, goal, tactic and argument levels.
- We demonstrate comparable performance to approaches that rely on examples from human experts available in the HOL4 system. We further use ablation studies to establish the role of different components of the architecture in achieving the performance.

2 INTERACTIVE THEOREM PROVING WITH REINFORCEMENT LEARNING

2.1 MODELING ITP WITH MDP

We first introduce the basic components of our MDP in terms of ITP concepts.

States A proof attempt starts with a main goal $g \in \mathbb{G}$ which is a HOL4 proposition. At any point in a traditional HOL4 proof attempt, there is a set of goals that all need to be proved. We refer to these finite sets of goals as *fringes*. In the context of our novel framing as an MDP, multiple fringes will be generated and maintained in such a way that the main goal will be proved if everything in any one fringe is proved. In this way, a fringe represents a particular path along which we may continue in an attempt to complete a single valid HOL4 proof of the main goal. In contrast, always choosing the most recently generated fringe would be equivalent to never *backtracking* in HOL4. The MDP state s is therefore a finite sequence of fringes, the order of which is irrelevant but maintained here for notational convenience, *i.e.*, we denote the i-th fringe in state s by $s(i)$.

Actions An action is a 4-tuple $(i, j, t, \boldsymbol{c}) \in \mathbb{N} \times \mathbb{N} \times \mathcal{T} \times \mathcal{A}^*$. Intuitively, given a state s, i selects the i-th fringe in s and j selects the j-th goal within fringe $s(i)$. Then, t is the HOL4 tactic that we select from the set $\mathcal{T}$ of possible HOL4 tactics, and $\boldsymbol{c}$ is the possibly empty list of arguments that accompany t. An argument in $\mathcal{A}$ is either a theorem name, or a HOL4 term. For example, the tactic `fs` takes a list of theorem names as its argument, and may therefore be applied to the theorem called `listTheory.MEM`, becoming `fs[listTheory.MEM]`. The existence of arguments makes the action space arbitrarily large—terms may be arbitrarily defined, and there are thousands of theorems that can be given as arguments to the tactic. This necessitates an RL algorithm that can handle large action and state spaces, for which we select policy gradients (Williams, 1992).

Rewards When a tactic is applied to a goal g, one receives feedback from HOL4. If successful, one receives a set of new (sub-)goals such that proving all of them proves g; if this set is empty, then g is itself proven. A tactic application may also fail, either with an error indicating that it is not applicable, or with a timeout caused by HOL4 exceeding a maximum computation time which we set. In terms of our framing as an MDP, different numeric rewards are associated with the different cases described above.

MDP State Transition A proof attempt always starts with a single main goal g, and so the initial state is a single fringe containing one element, g. Given a state s, performing an action $(i, j, t, \boldsymbol{c})$ may not change the state. This happens when the tactic times out, has no effect or is not applicable. Otherwise, the application of the tactic generates a set of new subgoals. This set G of new subgoals may be empty, indicating that goal $s(i)(j)$ is immediately proved by tactic t. In any case, a new fringe is then constructed by first copying fringe $s(i)$, and then replacing the goal $s(i)(j)$ with the new subgoals G. Then we construct a new state s' by adding the new fringe to state s. Note that each state constructed in this way maintains all possible partial proof attempts.

Termination The MDP terminates when an empty fringe is constructed, which implies that we can recover a proof of the main goal. In this case, the agent receives a positive reward. The process is also terminated if a prescribed maximum number of steps is exceeded, in which case the proof attempt is unsuccessful and the agent receives a negative reward.

2.2 Reinforcement Learning Agent Algorithm

We now define our novel agent model, for application to our MDP formulation of the ITP problem as decribed in the previous sections. We learn policy networks for selecting fringes, tactics and arguments (including lists of theorems and terms) via policy gradient methods. Each HOL4 expression is represented by a 256-dimensional vector[1]. This representation is pre-trained by a transformer (Vaswani et al., 2017) based autoencoder (Kingma & Welling, 2014) for sequences.

Selecting fringes Let $\mathbb{G} \subseteq \mathbb{R}^{256}$ be the space of goals; we learn a function

$$V_{\text{goal}} : \mathbb{G} \to \mathbb{R}, \tag{1}$$

which, intuitively, represents how easy a goal is to prove. Given a state s we define the score $v_{s(i)}$ of its i-th fringe as

$$v_{s(i)} = \Sigma_{g \in s(i)} V_{\text{goal}}(g). \tag{2}$$

Summing in this way reflects a design choice; namely that *1)* to prove $s(i)$ we must prove all its consituent goals, *2)* that V_{goal} outputs logits, and *3)* as we sum logits that the probability of solving each goal is independent given its numeric representation.

[1]To simplify the notation we occasionally overload our symbols, which depending on the context may either refer to a HOL4 expression, or its latent represesentation.

Hence, to choose a fringe to work on, our agent samples from the discrete distribution with probabilities

$$\pi_{\text{fringe}}(s) = \text{softmax}(v_{s(0)}, ..., v_{s(|s|-1)}). \tag{3}$$

After selecting a fringe, by default we select the first goal in that fringe. That is, the j in an action $(i, j, t, \boldsymbol{c})$ is fixed to be 0. This reflects a simplifying design choice based on the observation that all of the goals within a fringe have to be proved in order to prove the main goal, and that the order in which they are proven is irrelevant. Note that in contrast, fringe selection is crucial because only one fringe has to be solved in order to prove the main goal.

Selecting tactics Suppose that we have selected a goal g to work on. We learn a function

$$V_{\text{tactic}} : \mathbb{G} \to \mathbb{R}^D, \tag{4}$$

where D is the total number of tactics allowed, which estimates the effectiveness of each tactic for proving g. To select the tactic, we again sample from the discrete distribution, here with probabilities

$$\pi_{\text{tactic}}(g) = \text{softmax}(V_{\text{tactic}}(g)). \tag{5}$$

Selecting a list of theorems as arguments Suppose that we have selected a goal g and a tactic t. If t takes theorem names as arguments, then we need to select a list of them. This process is naturally modeled by a recurrent neural network architecture, as we now detail. Each theorem selection depends on previously selected theorems. This is because arguments to tactics like `simp` and `metis_tac` generally interact with one another in terms of the resultant behavior.

The argument policy recurrence takes three pieces of data as its input. The first is a set of candidate theorems that the policy can choose from. The second is the previously selected argument, which is initialized to the selected tactic. The third is a latent state variable which carries the sequential information of previously selected arguments over the argument generation process.

The first output of the recurrent module is a set of scores associated with the candidate arguments. We softmax these scores and sample to get one argument. An LSTM also produces a second output which is the latent state variable that carries the sequential information. The recurrent process is terminated when it reaches a pre-defined maximum length L which we set for the argument list. See Figure 4 and Algorithm 1 in Appendix for an illustration and pseudo-code of this process.

Selecting terms as arguments When generating terms, only variables that occur in the selected goal are considered as candidates. The generation of terms as arguments is then similar to that of theorem names—we have a list of candidate terms that plays the same role as that of the list of candidate theorems in generation of theorem names as arguments. An algorithm similar to the one that generates theorems as arguments is applied to the generation of terms.

Computing policy gradients Given a state s, the selection process described above now allows us to define our policy distribution in terms of the following factors.

- $\pi(f|s)$, the probability of selecting fringe f given s induced by π_{fringe} of Equation 3. Recall that, subsequently, the first goal g in f is always selected.
- $\pi(t|s, g)$, the probability of selecting tactic t given s and g, is induced by π_{tactic} of Equation 5.
- $\pi(c_l|s, g, t, \boldsymbol{c}_{<l}); l = 1, 2, \ldots, L$, the probability of selecting the l-th element of the argument list $\boldsymbol{c}$ given s, g, t and the previously generated elements of the argument list. This is induced by our recurrent argument selection algorithm, where L is the fixed length of arguments, and $\boldsymbol{c}_{<l} = (c_1, ..., c_{l-1})$.

Denoting the action by a, the policy therefore factorises as

$$\pi_\theta(a|s) = \pi(f|s)\pi(t|s, g) \prod_{l=1}^{L} \pi(c_l|s, g, t, \boldsymbol{c}_{<l}), \tag{6}$$

where θ represents the parameters of $\{V_{\text{goal}}, V_{\text{tactic}}, V_{\text{arg}}\}$. Denoting the m-th reward by r_m, and the MDP trajectory by $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_M, a_M, r_M)$, our objective function is the usual sum of discounted expected rewards with discount factor $\gamma \in (0, 1]$, *i.e.*,

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[\sum_{m=0}^{M}\gamma^m r_m\Big]. \tag{7}$$

Despite the large action space at hand, optimisation in θ by gradient descent is made tractable by substituting the classic REINFORCE estimator (Williams, 1992) for $\nabla_\theta J$.

3 EXPERIMENTS

The learning task is to train an agent that is capable of proving theorems using a given set of tactics in HOL4's core library. The dataset contains 1342 theorems that are known to be provable using the given set of tactics. No information about the existing human proofs is included in the training set. We reserve 20% of the theorems for evaluation and train the agent to prove the rest of them.

Evaluation If we think of proving the theorems in the training data as a stand-alone reinforcement learning task, it can be seen that the agent learns rather well (see Figure 5a in Appendix). The performance of a trained agent is 5 times better than that of random rollouts. Moreover, a trained agent is also able to prove theorems that it has never seen during training. The first column of Table 1 compares the performance of a trained agent and random rollouts when on the testing set. We also observe that TacticZero remains notably competitive overall with HOL(y)Hammer (Gauthier & Kaliszyk, 2015) on unseen problems, despite the latter enjoying the rather significant advantage of using algorithms and features engineered by highly trained human experts.

Table 1: Percentage of proved theorems by TacticZero trained on different datasets compared to that by the corresponding random rollouts and HOL(y)Hammer (with Vampire (Kovács & Voronkov, 2013)) on testing sets (*i.e.*, unseen theorems). TacticZero is allowed to perform at most 8 proof attempts for each theorem. Note that HOL(y)Hammer is run in default mode and can choose any theorems as premises including the very theorem being proved and even stronger theorems, which are not allowed to be used by TacticZero. The performance of TacticZero is comparable to that of HOL(y)Hammer even with this disadvantage. Also note that although a direct comparison is impossible, TacticZero's performance is similar to that of *TacticToe* which is reported to be 66.3% (Gauthier et al., 2020).

Method	all (%)	set (%)	list (%)	others (%)
Random	14.2	5.0	10.8	21.5
TacticZero	62.3	81.7	61.7	75.0
Hammer	64.5	69.5	62.8	64.1

Generalizability We further demonstrate that our agent does not necessarily need to experience proving many different theorems in order to generalize to unseen theorems. We split the dataset into three parts. One contains all the theorems from set theory, one contains all the theorem from list theory, and the other contains the rest of the theorems. Each part is further split into training and testing sets with a 80:20 ratio on which 3 agents are then trained and tested. The last three columns of Table 1 illustrate the performance of these agents. In spite of each training set being smaller due to the theory-wise splitting, the trained agents still outperform random rollouts by a large margin (see Figure 1).

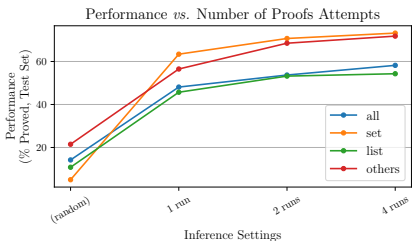

Figure 1: The performance of TacticZero increases when it is allowed to perform multiple proof attempts to a theorem (due to stochastic policy some attempts might be unsuccessful).

REFERENCES

Eser Aygün, Zafarali Ahmed, Ankit Anand, Vlad Firoiu, Xavier Glorot, Laurent Orseau, Doina Precup, and Shibl Mourad. Learning to prove from synthetic theorems, 2020.

Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning*, pp. 454–463. PMLR, 2019.

Kshitij Bansal, Christian Szegedy, Markus N. Rabe, Sarah M. Loos, and Viktor Toman. Learning to reason in large theories without imitation, 2020. URL https://arxiv.org/abs/1905.10501.

Richard Bellman. A Markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957. ISSN 0022-2518.

G. Brockman, Vicki Cheung, Ludwig Pettersson, J. Schneider, John Schulman, Jie Tang, and W. Zaremba. OpenAI Gym. *ArXiv*, abs/1606.01540, 2016.

Coq Development Team. *The Coq proof assistant reference manual*, 2004. URL http://coq.inria.fr. Version 8.0.

Maxwell Crouse, Ibrahim Abdelaziz, Bassem Makni, Spencer Whitehead, Cristina Cornelio, Pavan Kapanipathi, Kavitha Srinivas, Veronika Thost, Michael Witbrock, and Achille Fokoue. A deep reinforcement learning approach to first-order logic theorem proving, 2020.

Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: automation for dependent type theory. *J. Automated Reasoning*, 61(1):423–453, 2018.

S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327, 1976. doi: 10.1109/TSMC.1976.5408784.

Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, pp. 49–57, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450332965. URL https://doi.org/10.1145/2676724.2693173.

Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Tactictoe: Learning to prove with tactics. *J. Automated Reasoning*, Aug 2020. ISSN 1573-0670. URL https://doi.org/10.1007/s10817-020-09580-x.

Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoas An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. A formal proof of the Kepler Conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017. doi: 10.1017/fmp.2017.1.

John Harrison. HOL Light: A tutorial introduction. In Mandayam Srivas and Albert Camilleri (eds.), *Formal Methods in Computer-Aided Design*, pp. 265–269, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-49567-3.

Geoffrey Hinton, Nitish Srivatava, and Kevin Swersky. Lecture 6e: rmsprop: Divide the gradient by a running average of its recent magnitude, 2012. Lecture notes available from http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf.

HOL4 Development Team. HOL4 *Documentation*. URL http://hol-theorem-prover.org.

Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=r1xwKoR9Y7.

Cezary Kaliszyk and Josef Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In Jasmin Christian Blanchette and Josef Urban (eds.), *PxTP 2013. Third International Workshop on Proof Exchange for Theorem Proving*, volume 14 of *EPiC Series in Computing*, pp. 87–95. EasyChair, 2013. doi: 10.29007/5gzr. URL https://easychair.org/publications/paper/VZv6.

Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Automated Reasoning*, 53(2):173–213, Aug 2014. ISSN 1573-0670. URL https://doi.org/10.1007/s10817-014-9303-3.

Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov (eds.), *Logic for Programming, Artificial Intelligence, and Reasoning*, pp. 88–96, Berlin, Heidelberg, 2015a. Springer Berlin Heidelberg. ISBN 978-3-662-48899-7.

Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Automated Reasoning*, 55(3):245–256, Oct 2015b. ISSN 1573-0670. URL https://doi.org/10.1007/s10817-015-9330-8.

Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olšák. Reinforcement learning of theorem proving. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31, pp. 8822–8833. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/55acf8539596d25624059980986aaa78-Paper.pdf.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine learning in Proof General: Interfacing interfaces. In Cezary Kaliszyk and Christoph Lüth (eds.), *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, UITP 2012, Bremen, Germany, July 11th, 2012*, volume 118 of *EPTCS*, pp. 15–41, 2012. URL https://doi.org/10.4204/EPTCS.118.2.

Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith (eds.), *Computer Aided Verification*, pp. 1–35, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39799-8.

Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJluxREKDB.

Xavier Leroy. A formally verified compiler back-end. *J. Automated Reasoning*, 43(4):363–446, 2009. URL https://doi.org/10.1007/s10817-009-9155-4.

Reinhold Letz, Klaus Mayr, and Christoph Goller. Controlled integration of the cut rule into connection tableau calculi. *J. Automated Reasoning*, 13(3):297–337, 1994.

Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *ArXiv*, abs/1701.06972, 2017.

Norman D. Megill and David A. Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, Morrisville, North Carolina, 2019. http://us.metamath.org/downloads/metamath.pdf.

Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska (eds.), *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, volume 2 of *EPiC Series in Computing*, pp. 1–11. EasyChair, 2010. URL https://easychair.org/publications/paper/wV.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *ArXiv*, abs/2009.03393, 2020.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 3788–3800. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/b2ab001909a8a6f04b51920306046ce5-Paper.pdf.

Alex Sanchez-Stern, Yousef Alhessi, Lawrence K. Saul, and Sorin Lerner. Generating correctness proofs with neural networks. *ArXiv*, abs/1907.07794, 2019.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *ArXiv*, abs/1802.03685, 2018. URL http://arxiv.org/abs/1802.03685.

Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar (eds.), *Theorem Proving in Higher Order Logics*, pp. 28–32, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-71067-7.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Hao Wang, Xingjian SHI, and Dit-Yan Yeung. Natural-parameter networks: A class of probabilistic neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29, pp. 118–126. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/fe9fc289c3ff0af142b6d3bead98a923-Paper.pdf.

Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 2786–2796, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/18d10dc6e666eab6de9215ae5b3d54df-Abstract.html.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. URL https://doi.org/10.1007/BF00992696.

Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6984–6994. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/yang19a.html.

A APPENDIX

A.1 MORE ON INTERACTIVE THEOREM PROVING

Interactive theorem-proving (ITP) is the process of humans interacting with a computer system to develop formal proofs of mathematical theorems. In ITP, a human user can define mathematical objects, and then guide the computer to prove theorems about them using commands called *tactics*. Tactics are programs that embody high-level proof strategies such as simplification and induction. Successful tactic application converts the current target *goal* (a theorem to be proved) into zero or more subgoals that remain to be proved.

Proofs written in an ITP system are ultimately checked by a machine, and therefore, they are much more trustworthy than pencil-and-paper proofs. For this reason, ITP has gained success in both formalizing mathematics, and in verifying computer programs (Leroy, 2009; Hales et al., 2017). Nevertheless, because a great amount of formal detail needs to be spelled out when using an ITP system, substantial human inputs are required to fill the gaps between steps in a proof. In particular, in terms of tactic-based theorem proving, human guidance in the selection of both tactics, *and* the arguments to tactics is crucial to a successful proof. This requires expert knowledge both of the relevant mathematics, and the particular ITP system being used. This requirement for experts has in turn limited the application of ITP systems.

One promising line of work (Komendantskaya et al., 2012; Yang & Deng, 2019; Huang et al., 2019; Gauthier et al., 2020) to address this limitation has been to use machine learning methods to replace the role of human experts in ITP. These attempts, however, have mostly reached only partial success, and the majority of them are directly or indirectly built upon the proofs provided by humans. For example, *TacticToe* (Gauthier et al., 2020) chooses tactics by looking at candidates that have been applied to similar goals recorded from human proofs. This (partial) dependency limits the application of these techniques depending on the availability and quality of the human proofs in the domain that the proof is required.

A.2 RELATED WORK

Here we review the machine learning approaches to theorem proving, which we break into two main categories.

Automated theorem proving (ATP) Unlike ITP, ATP usually focuses on manipulating expressions in first order logic. Machine learning methods for ATP (Kaliszyk & Urban, 2015a; Wang et al., 2016; 2017; Rocktäschel & Riedel, 2017; Loos et al., 2017; Selsam et al., 2018; Kaliszyk et al., 2018; Lederman et al., 2020; Crouse et al., 2020; Aygün et al., 2020) are then applied to guide proof search along with well-developed ATP algorithms such as saturation-based or connection-based algorithms (Letz et al., 1994). One limitation of ATP is that the proofs are often represented in a low-level language and hard to interpret as high-level mathematical concepts, which is in contrast to ITP which uses tactics and embodies more high-level human-like mathematical reasoning.

There are also approaches that work by interfacing ATP with ITP systems. These tools are called *hammers* (Paulson & Blanchette, 2010; Kaliszyk & Urban, 2014; 2015b; Gauthier & Kaliszyk, 2015; Czajka & Kaliszyk, 2018). These tools work by translating goals in an ITP system to the languages in ATP systems. If a proof is found by an ATP system, the tool then tries to reconstruct the corresponding proof in the ITP system. This process, however, might fail in the situations that the derived proof cannot be fully translated back to the ITP high-level representations. Our work focuses on proof search in ITP without invoking any external provers.

Interactive theorem proving Machine learning approaches to ITP are more closely related to our work, although most focus on supervised learning from existing human proofs in the library of an ITP system. *TacticToe* (Gauthier et al., 2020) chooses tactics based on recorded human proofs in HOL4's library, and subsequently expands a search tree by Monte Carlo Tree Search. A distance-weighted k nearest-neighbour classifier (Dudani, 1976) is adapted for lemma selection (Kaliszyk & Urban, 2013).

Other approaches that learn from human proofs include the following. *GPT-f* (Polu & Sutskever, 2020) learns from proofs in the METAMATH (Megill & Wheeler, 2019) library, and uses transformer-

Algorithm 1 Generating arguments

Input:
Goal g, tactic t, number of candidate theorems M, candidate theorems $X \in \mathbb{R}^{M \times 256}$, recurrent (LSTM-based) model $V_{\mathrm{arg}} : \mathbb{R}^{256} \times \mathbb{R}^{M \times 256} \times \mathbb{R}^{512} \to \mathbb{R}^{M} \times \mathbb{R}^{512}$, and number of required arguments L.
Begin
Initialize $h \leftarrow (g, g)$
Initialize $c \leftarrow t$
Initialize an empty argument list $\boldsymbol{c} \leftarrow [\,]$
for $l = 1$ **to** L **do**
  $v, h \leftarrow V_{\mathrm{arg}}(c, X, h)$
  $c \leftarrow$ sample from $\pi_{\mathrm{arg}}(g) = \mathrm{softmax}(v)$
  $\boldsymbol{c} \leftarrow \boldsymbol{c}.\mathrm{append}(c)$
end for
return $\boldsymbol{c}$ and the associated (log) probabilities

based (Vaswani et al., 2017) language models to predict proof steps. That work also demonstrates improved prover performance by using generative pre-training and iterative training of a value function on statements generated by the language model. *GamePad* (Huang et al., 2019) and *CoqGym* (Yang & Deng, 2019) are both learning environments for the COQ theorem prover (Coq Development Team, 2004) and focus on learning from human proofs. *GamePad* targets specific sub-tasks of ITP such as the algebraic rewriting problem. *CoqGym* is more general and comes with a deep learning model that learns from human proofs to generate tactics by expanding abstract syntax trees. *ProverBot9001* (Sanchez-Stern et al., 2019) learns from proofs in the *CompCert* (Leroy, 2009) project of COQ, and uses recurrent neural networks over sequential representations of terms to predict arguments for tactics. The use of the *CompCert* project's proofs is again a dependent on raw material from human experts.

There are also approaches including reinforcement learning components. *HOList*/*DeepHOL* (Bansal et al., 2019; 2020) trains a proof guidance model to prove theorems in the HOL LIGHT theorem prover (Harrison, 1996) by continuously expanding training data. If a proof is found, it is used to generate additional training data, which is used to update the model used for exploration. Although that work refers to that process as a reinforcement learning loop, it uses pre-engineered scoring for premise selection to help find new proofs, and a fixed breadth-first search strategy to find proofs, and it is unclear how this approach may be expressed as an MDP. In our framework, the agent learns arguments (premise) selection as well as tactic selection without pre-engineered scoring, and manages proof search *by itself*, all through deep policy gradient.

A.3 MORE DETAILS ON THE MDP FORMULATION

Figure 2b is an example of proof search in terms of MDP. Figure 3 illustrates the structure of the model. Figure 4 illustrates the recurrent process for generating arguments. Algorithm 1 is pseudo-code of the recurrent argument generation algorithm.

A.4 MORE DETAILS ON EXPERIMENTS

In our setting, the agent is allowed to use the tactics listed in Table 2. The dataset contains 1342 theorems that are known to be provable using the given set of tactics. The theorems come from a wide range of theories, including those of lists (finite sequences), (binary) relations, and sets. For the purpose of reinforcement learning, we implement an environment that an agent can interact with. The API of our environment is inpired by that of Gym (Brockman et al., 2016).

We call a single proof attempt of a theorem an episode. An iteration is completed if the agent has attempted to prove every theorem in the training dataset. For each episode, the limit of MDP timesteps (*i.e.*, calls to the ITP engine, HOL4) is set to 50. The execution time limit of each tactic application is set to 0.1 seconds. With these settings, each proof attempt takes around 5 seconds to complete if we ignore the inference time of the policies. Crucially, our novel scheme of maintaining and selecting

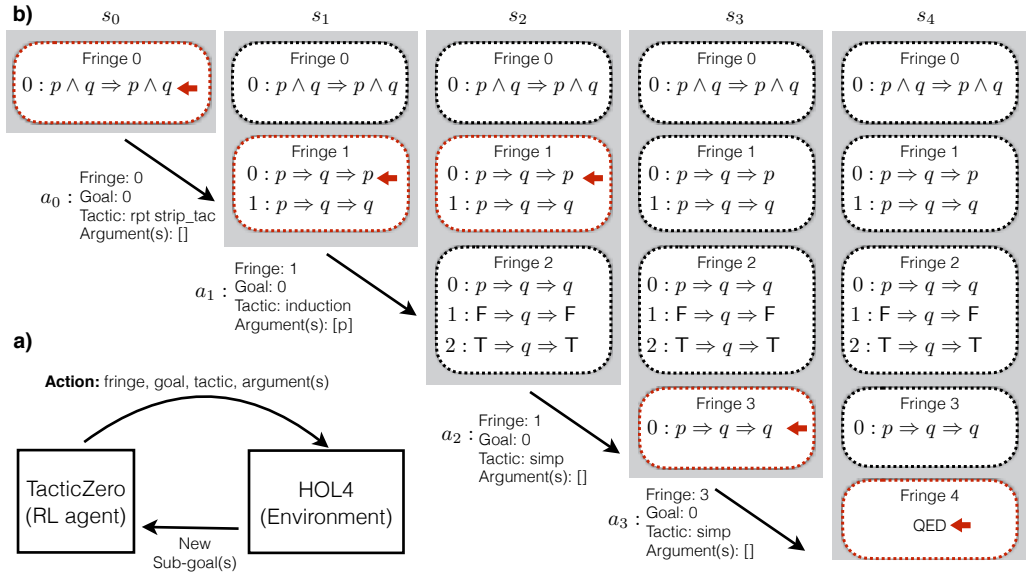

Figure 2: *a)* Interaction between the RL agent (TacticZero) and the environment (HOL4) which is an interactive theorem prover. Given a target theorem (goal) to prove, the agent chooses a tactic and arguments to the tactic and sends them to HOL4. HOL4 in turn provides the agent with new subgoals derived from the application of the tactic. *b)* An example scenario for proving $p \wedge q \implies p \wedge q$. The initial state of the agent (s_0) includes only the target theorem. The agent then takes an action (a_0) which consists of a fringe, a goal, a tactic and its arguments and sends it to HOL4. HOL4 applies that tactic and provides new subgoals which form Fringe 1 (shown by '0 :' and '1 :'); Fringe 1 is then added to the fringes in s_0 which together form s_1. In general, each state is comprised of multiple fringes and each fringe includes a set of goals that are sufficient to be proved to derive the original target theorem. At each state, the agent evaluates goals within each fringe and decides which fringe is more promising, and within each fringe which goal to work on first (shown by red arrows). Given this choice, the agent then selects a tactic and appropriate arguments. In s_1, the agent chooses Fringe 1 (shown in red) and the first subgoal and takes action a_1. The resulting subgoals form Fringe 2 in s_2. At each state the agent can cease working on the newly added fringe, and backtracks to an older fringe if the new one is predicted to be hard. This process continues until an empty fringe is derived (s_4). After taking each action the agent also received a reward signal (not shown in the figure) to update its action selection policies.

Table 2: Tactics that can be used by the agent. These are standard HOL4 tactics, documented at (HOL4 Development Team), *e.g.*, `drule`. Though small, this set of tactics is powerful enough to prove a significant subset of results from HOL4's core libraries.

Tactics	Argument types
`drule`, `irule`	single theorem
`fs`, `metis_tac`, `rw`, `simp`	list of theorems
`Induct_on`	single term
`eq_tac`, `strip_tac`	none

fringes (which models *backtracking* from un-promising derivations) requires only to manipulate the scalar scores $V_{\text{goal}}(g)$, and hence introduces negligible computational overhead.

Rewards During proof search, if an action makes progress, then a reward 0.1 is given to the agent. Making progress means that the tactic is successfully applied by HOL4, yielding new subgoals. If an action immediately proves the goal it is targeting, meaning that no new subgoals are generated, then a reward of 0.2 is given. Note that in both cases a new state is generated. In all other cases including inapplicable actions that do not generate any new states, a reward of -0.1 is given to the agent. If the

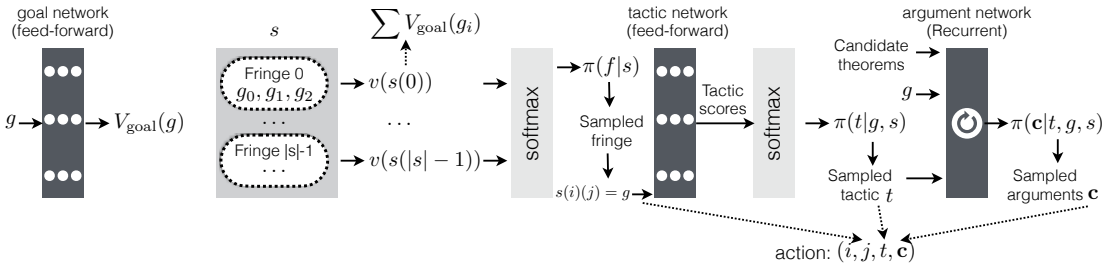

Figure 3: Structure of the model. Given state s the aim of the agent is to select an action which consists of a fringe, a goal within that fringe, a tactic to apply to the goal, and a list of arguments for the tactic. This is achieved using a set of neural networks. The first network which is referred to as goal-network (feed-forward) takes a goal g as input, and returns a score $V_{\text{goal}}(g)$ for the goal. Given state s, the agent calculates the sum of the scores for all the goals in each fringe and obtains a score for each fringe ($v(s(0))$ to $v(s(|s|-1))$). These scores then feed into a softmax layer to determine the probability of selecting each fringe, $\pi(f|s)$. Within the selected fringe a goal is selected ($s(i)(j)$ for goal j in fringe i). This goal is then fed in to the tactic network (feed-forward) which through a softmax layer generates the probabilities of selecting each tactic t, $\pi(t|g,s)$. The selected tactic along with the selected goal and a candidate list of theorems are fed into the argument network (recurrent) which determines the probability of selecting each candidate theorem as an argument for the tactic (see Figure 4). The final action then consists of a chosen fringe i, goal j, tactic t and arguments $\boldsymbol{c}$ which are used to get new sub-goals from HOL4. Based on how successful was the action the agent receives a reward which is used to update the weights in all the network using policy-gradient.

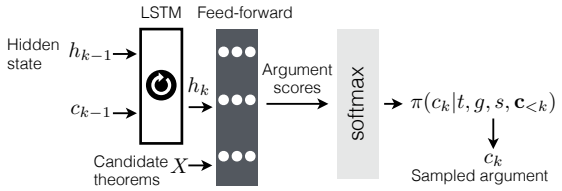

Figure 4: The argument network. The aim is to select arguments from candidates theorems X for a tactic t to be applied on goal g. To achieve this, the hidden state of an LSTM neural network are initialized using g. The LSTM layer takes the previously chosen argument as an input (c_{k-1}) and through a feed-forward and a softmax layer generates the probability of selecting each candidate theorem for the next argument (c_k). c_0 is initialized to t.

agent manages to prove a theorem within 50 timesteps, then it receives a positive terminal reward. This positive terminal reward depends on the difficulty of the target theorem. If the rate at which the theorem is being proved in earlier rollouts is above average, then the reward is 5. If the rate is below average, then the reward is 15. If the proof attempt fails, the agent receives a terminal reward of -5.

ITP settings The length of list of arguments of a tactic is set to 5. For each episode, the set of candidate theorems consists of all the theorems coming from the theories mentioned in the theorem to be proved. To prevent the agent from "cheating" by learning to use the very theorem to prove itself, or learning to use stronger theorems in the library to prove weaker ones, only theorems that come before the theorem that is being proved in the library are allowed in the candidate set.

Simplification lemmas are results that HOL4 adds to its global context to help tactics such as `simp` and `fs` find proofs. When the agent is trying to prove a theorem from theory T during training, only simplification lemmas in theories which T builds upon are allowed to be in the environment. In other words, the tactics cannot invisibly appeal to "prior" knowledge that is actually from the same theory. If such results are to be used, they can be explicitly mentioned, as *per* the previous paragraph.

Replaying One difficulty during training is that positive rewards are sparse in the early training phase. The agent may find a proof of a difficult theorem by accident, but then take many episodes to prove it

again. To help the agent recall its successes, we maintain a replay buffer of earlier successful proofs of each theorem. During training, if the agent fails to prove a theorem that it was previously able to prove, replaying will be triggered so that the agent is walked through one of the 5 most recent successful proof again and parameters updated correspondingly. This represents a departure from pure policy gradients, but worked well in our experiments, presumably because the update remains in the high dimensional direction $\nabla_\theta J$ and therefore differs from a precise *on-policy* update only by the ad-hoc choice of learning rate.

Training We jointly train the policies using RMSProp (Hinton et al., 2012) with a learning rate of 5×10^{-5} for each policy. The discount factor γ is set to be 0.99. The structure of our model is illustrated in Figure 3. Figure 5a shows the number of theorems proved in each iteration (epoch) of training. Figure 6a shows the decrease of average execution time of a proof attempt during training. It takes two weeks with a single Tesla P100-SXM2 GPU and a single Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz to complete 800 iterations.

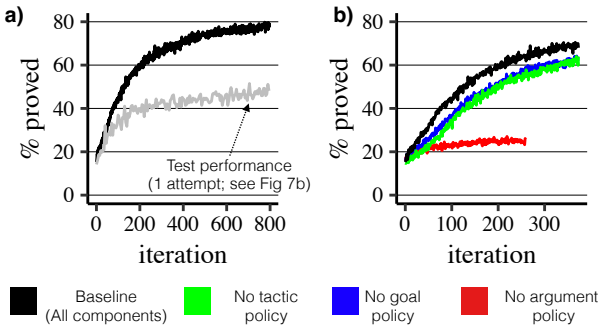

Figure 5: Training progress as percentage of "training" theorems (a slight misnomer—see section 3, *Evaluation*) proved per iteration (epoch), *a)* for our complete agent and *b)* for various ablations of our agent as denoted by the labels above. The degradations at right demonstrate that all components contribute performance, especially the argument policy.

Ablation study We show how each component of the agent's policy contributes to the overall performance by ablation studies. Now, instead of training all the policies modules jointly, we fix some of their parameters during the training to their random initial values. Figure 5b illustrates different training curves when the update of goal policy V_{goal}, tactic policy V_{tactic} and argument policy V_{arg} are turned off, respectively. It can be seen that the argument policy V_{arg} has the most significant impact on performance. This result is intuitive — even if proof states and tactics are well chosen, the failure of finding relative arguments (premises) to the chosen tactic leads to a greater chance of failure of proof attempts.

On the other hand, it can be seen that updating all other policies as usual can compensate the loss of the goal policy or the tactic policy to a certain degree. A possible explanation for this is that because the argument policy takes both the chosen goal and the chosen tactic into consideration in order to predict the arguments, it is able to estimate effective choices for making the randomly chosen goal or tactic work.

A.5 Embedding HOL4 in an MDP — A Sketch of our Software Environment

Rather than interacting directly with the HOL4 ITP engine, our agent acts on the more abstract MDP formulation introduced in main submission. In terms of software, we achieve this by defining a reinforcement learning environment which appropriately wraps HOL4. An instance e of the environment can be created by calling `HolEnv(g)` with an initial goal g (which represents the theorem to be proved). An action a can be taken by executing `e.step(a)` and the return value of this function is a pair consisting of the immediate reward and a boolean indicating whether the proof attempt has finished. Internally, the environment keeps track of the states encountered during the proof search.

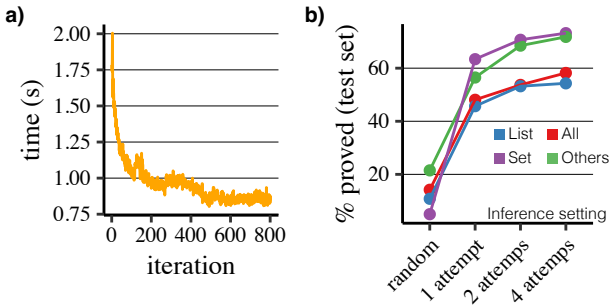

Figure 6: a) The average execution time of a single proof attempt decreases as training goes on. b) The performance of TacticZero increases when it is allowed to perform multiple proof attempts to a theorem (due to stochastic policy some attempts might be unsuccessful). Each proof attempt takes 1 to 1.5 seconds to complete.

Given a goal `g`, a tactic `t` and an argument list `c`, one may call `e.query(g,t,c)` independently of the `e.step(a)` function, to inspect the result of applying a tactic. The execution of a single query takes around 15 milliseconds on a regular laptop on average.

Once the environment detects a successful proof attempt (*i.e.*, an empty fringe), it re-constructs a HOL4 proof script from the MDP state sequence (along with some book-keeping information) and sends it to HOL4 for verification. The implied proof search can also be visualized by our software as an interactive tree which depicts all the information.

Note that the proof search represented by Figure 7 is neither a breadth first search nor a depth first search, but a unique proof search managed by the agent itself, as indicated by the "Step" entry in the edge labels of the interactive version of the plot. Also note that we have separated "assumptions" and "goal" in the fringes for better readability. From the agent's point of view, the "assumptions" and the "goal" are merged into one formula by chained implications.

A.6 Space of Arguments

For each episode, a set of candidate theorems that can be selected as arguments is computed at the beginning of the episode. Duplicates are allowed in the list of arguments. The set of candidate theorems consists of all the theorems coming from the theories mentioned in the theorem to be proved, except for those that come after the theorem that is being proved in the library.

For example, if the agent is trying to prove theorem $\forall x, \bigcup\{x\} = x$ which is a theorem from the `pred_set` theory, the candidate set will contain *a)* all the theorems from the `bool` theory (because the symbol $\forall$ comes from `bool` theory), and *b)* all the theorems whose proof precedes that of $\forall x, \bigcup\{x\} = x$ from within `pred_set` theory.

The average size of such a set of candidate theorems for each theorem in our dataset is 468. In other words, the argument policy has to look at 468 theorems on average whenever the agent chooses an argument to a tactic. If the number of required arguments L is n, then the search space of the argument list is 468^n.

A.7 Example Proofs

The subsequent pages include several example proofs found by TacticZero.

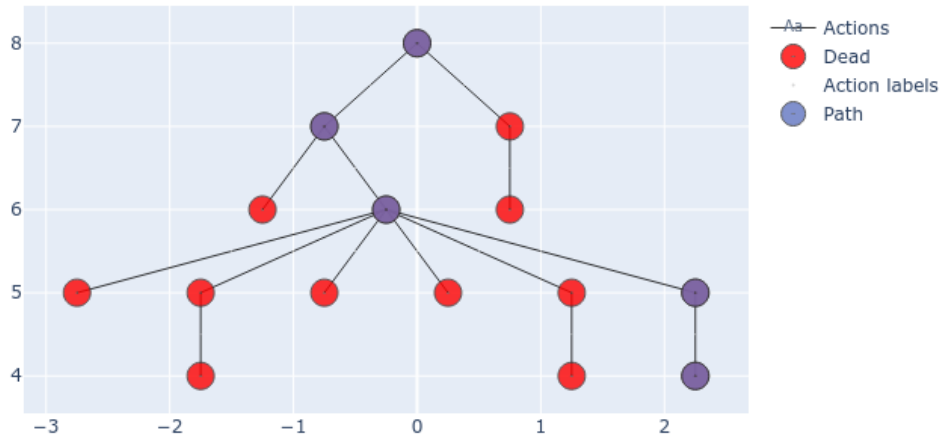

Figure 7: The visualization (in HTML) of a successful proof search of the theorem $\forall\ x\ s.\ x \in s \Rightarrow \forall f.\ f(x) \in \texttt{IMAGE}\ f\ s$. This particular proof was found in 13 steps. Red nodes represent the fringes that never lead to a successful proof, and blue nodes consist of a path from which a valid HOL4 proof can be re-constructed. Detailed information for both the edges and vertices can be revealed by hovering the mouse over the corresponding part of the HTML version of the plot.

```
(* TacticZero proof *)

Theorem EVERY_CONJ:
  ∀P Q l. EVERY (λ(x:'a). (P x) ∧ (Q x)) l = (EVERY P l ∧ EVERY Q l)
Proof
  rw[] >> Induct_on `l`
  >- (rw[listTheory.EVERY_DEF])
  >- (rw[listTheory.EVERY_DEF] >> metis_tac[])
QED
(*
  (* Original human proof *)
   NTAC 2 GEN_TAC THEN LIST_INDUCT_TAC THEN
   ASM_REWRITE_TAC [EVERY_DEF] THEN
   CONV_TAC (DEPTH_CONV BETA_CONV) THEN
   REPEAT (STRIP_TAC ORELSE EQ_TAC) THEN
   FIRST_ASSUM ACCEPT_TAC);
*)


(* TacticZero proof *)

Theorem MONO_EXISTS:
  (∀x. P x ⇒ Q x) ⇒ (EXISTS P l ⇒ EXISTS Q l)
Proof
  rw[listTheory.EXISTS_MEM] >> metis_tac[]
QED
(*
  (* Original human proof *)
  Q.ID_SPEC_TAC `l` THEN LIST_INDUCT_TAC THEN
  ASM_SIMP_TAC (srw_ss()) [DISJ_IMP_THM]);
*)


(* TacticZero proof *)

Theorem FLAT_APPEND:
  ∀l1 l2. FLAT (APPEND l1 l2) = APPEND (FLAT l1) (FLAT l2)
Proof
  strip_tac >> strip_tac >>
  Induct_on `l1`
  >- (rw[listTheory.APPEND, listTheory.FLAT])
  >- (fs[listTheory.APPEND]
      >> fs[listTheory.APPEND_ASSOC, listTheory.FLAT])
QED
(*
  (* Original human proof *)
  LIST_INDUCT_TAC
  THEN REWRITE_TAC [APPEND, FLAT]
  THEN ASM_REWRITE_TAC [APPEND_ASSOC]);
*)
```

```
(* TacticZero proof *)

Theorem MEM_MAP_f:
  ∀f l a. MEM a l ⇒ MEM (f a) (MAP f l)
Proof
  strip_tac >> strip_tac >> strip_tac
  >> rw[listTheory.MEM_MAP] >> (metis_tac[listTheory.MEM_MAP])
QED
(*
  (* Original human proof *)
  PROVE_TAC[MEM_MAP]
*)


(* TacticZero proof *)

Theorem REVERSE_11:
  (∀l1 l2:'a list. (REVERSE l1 = REVERSE l2) ⇔ (l1 = l2)
Proof
  strip_tac >> strip_tac
  >> metis_tac[listTheory.REVERSE_REVERSE]
QED
(*
  (* Original human proof *)
   REPEAT GEN_TAC THEN EQ_TAC THEN1
     (DISCH_THEN (MP_TAC o AP_TERM "REVERSE : 'a list → 'a list") THEN
      REWRITE_TAC [REVERSE_REVERSE]) THEN
   STRIP_TAC THEN ASM_REWRITE_TAC []);
*)


(* TacticZero proof *)

Theorem FILTER_COMM:
  ∀f1 f2 l. FILTER f1 (FILTER f2 l) = FILTER f2 (FILTER f1 l)
Proof
Induct_on 'l'
>- rw[]
>- rw[]
QED
(*
  (* Original human proof *)
  NTAC 2 GEN_TAC
  THEN BasicProvers.Induct
  THEN REWRITE_TAC [FILTER]
  THEN GEN_TAC
  THEN REPEAT COND_CASES_TAC
  THEN ASM_REWRITE_TAC [FILTER]);
*)
```

```
(* TacticZero proof *)

Theorem ABSORPTION:
  ∀x:'a. ∀s. (x IN s) ⇔ (x INSERT s = s)
Proof
  strip_tac
  >> rw[pred_setTheory.INSERT_DEF]
  >> fs[pred_setTheory.GSPEC_ETA, pred_setTheory.INSERT_DEF]
  >> metis_tac[pred_setTheory.SPECIFICATION]
QED
(*
  (* Original human proof *)
  REWRITE_TAC [EXTENSION,IN_INSERT] THEN
  REPEAT (STRIP_TAC ORELSE EQ_TAC) THEN
  ASM_REWRITE_TAC [] THEN
  FIRST_ASSUM (fn th => fn g => PURE_ONCE_REWRITE_TAC [SYM(SPEC_ALL th)] g)
  THEN DISJ1_TAC THEN REFL_TAC
*)


(* TacticZero proof *)

Theorem DISJOINT_INSERT:
  (∀(x:'a) s t. DISJOINT (x INSERT s) t ⇔ DISJOINT s t ∧ x NOTIN t
Proof
  strip_tac >> strip_tac >> strip_tac
  >> fs[pred_setTheory.IN_INSERT, pred_setTheory.INSERT_DEF,
    pred_setTheory.IN_DISJOINT]
  >> metis_tac[]
QED
(*
  (* Original human proof *)
  REWRITE_TAC [IN_DISJOINT,IN_INSERT] THEN
  CONV_TAC (ONCE_DEPTH_CONV NOT_EXISTS_CONV) THEN
  REWRITE_TAC [DE_MORGAN_THM] THEN
  REPEAT GEN_TAC THEN EQ_TAC THENL
  [let val v = genvar (==`:'a`==)
       val GTAC = X_GEN_TAC v
   in DISCH_THEN (fn th => CONJ_TAC THENL [GTAC,ALL_TAC] THEN MP_TAC th)
      THENL [DISCH_THEN (STRIP_ASSUME_TAC o SPEC v) THEN ASM_REWRITE_TAC [],
             DISCH_THEN (MP_TAC o SPEC (“x:'a”)) THEN REWRITE_TAC[]]
   end,
   REPEAT STRIP_TAC THEN ASM_CASES_TAC (“x':'a = x”) THENL
   [ASM_REWRITE_TAC[], ASM_REWRITE_TAC[]]]
*)
```

```
(* TacticZero proof *)

Theorem INSERT_INTER:
       (∀x:'a. ∀s t. (x INSERT s) INTER t =
         (if x IN t then x INSERT (s INTER t) else s INTER t)
Proof
strip_tac >> strip_tac >>
rw[pred_setTheory.INSERT_DEF, pred_setTheory.SPECIFICATION,
  pred_setTheory.INTER_DEF]
>- (rw[pred_setTheory.GSPEC_ETA] >> metis_tac[])
>- (rw[] >> (rw[pred_setTheory.GSPEC_ETA] >> metis_tac[]))
QED
(*
  (* Original human proof *)
  REPEAT GEN_TAC THEN COND_CASES_TAC THEN
  ASM_REWRITE_TAC [EXTENSION,IN_INTER,IN_INSERT] THEN
  GEN_TAC THEN EQ_TAC THENL
  [STRIP_TAC THEN ASM_REWRITE_TAC [],
   STRIP_TAC THEN ASM_REWRITE_TAC [],
   PURE_ONCE_REWRITE_TAC [CONJ_SYM] THEN
   DISCH_THEN (CONJUNCTS_THEN MP_TAC) THEN
   STRIP_TAC THEN ASM_REWRITE_TAC [],
   STRIP_TAC THEN ASM_REWRITE_TAC []]);
*)


(* TacticZero proof *)

Theorem SET_MINIMUM:
  (∀s:'a → bool. ∀M. (∃x. x IN s) ⇔ ∃x. x IN s ∧ ∀y. y IN s ⇔ M x <= M y
Proof
rw[]
>> fs[boolTheory.IMP_CONG, boolTheory.EQ_TRANS, boolTheory.EQ_IMP_THM]
>> rw[arithmeticTheory.WOP_measure, boolTheory.COND_ABS]
>> metis_tac[boolTheory.ONE_ONE_THM]
QED
(*
  (* Original human proof *)
  REPEAT (STRIP_TAC ORELSE EQ_TAC) THENL
  [IMP_RES_THEN (ASSUME_TAC o ISPEC ("M:'a→num")) lemma THEN
   let val th = SET_SPEC_CONV ("(n:num) IN M x | (x:'a) IN s")
   in IMP_RES_THEN (STRIP_ASSUME_TAC o REWRITE_RULE [th]) NUM_SET_WOP
   end THEN EXISTS_TAC ("x':'a") THEN CONJ_TAC THENL
   [FIRST_ASSUM ACCEPT_TAC,
    FIRST_ASSUM (SUBST_ALL_TAC o SYM) THEN
    REPEAT STRIP_TAC THEN FIRST_ASSUM MATCH_MP_TAC THEN
    EXISTS_TAC ("y:'a") THEN CONJ_TAC THENL
    [REFL_TAC, FIRST_ASSUM ACCEPT_TAC]],
   EXISTS_TAC ("x:'a") THEN FIRST_ASSUM ACCEPT_TAC]
*)
```

```
(* TacticZero proof *)

Theorem INJ_DELETE:
  (∀f s t. INJ f s t ==> ∀e. e IN s ==> INJ f (s DELETE e) (t DELETE (f e))
Proof
strip_tac >> strip_tac >> strip_tac
>> fs[] >> rw[]
>> (fs[pred_setTheory.INJ_DEF] >>
    (strip_tac >> fs[pred_setTheory.IN_DELETE, boolTheory.IMP_DISJ_THM]
     >- (metis_tac[pred_setTheory.IN_APP])
     >- (fs[] >> (fs[] >> (fs[] >> (metis_tac[]))))))
QED
(*
  (* Original human proof *)
  RW_TAC bool_ss [INJ_DEF, DELETE_DEF] THENL
  [`~(e = x)` by FULL_SIMP_TAC bool_ss
                 [DIFF_DEF,DIFF_INSERT, DIFF_EMPTY, IN_DELETE] THEN
  FULL_SIMP_TAC bool_ss [DIFF_DEF,DIFF_INSERT, DIFF_EMPTY, IN_DELETE] THEN
  METIS_TAC [],
  METIS_TAC [IN_DIFF]]);
*)


(* TacticZero proof *)

Theorem IMAGE_SURJ:
  (∀f:'a->'b. ∀s t. SURJ f s t = ((IMAGE f s) = t)
Proof
strip_tac >> strip_tac >> rw[pred_setTheory.SURJ_DEF]
>> fs[] >> fs[pred_setTheory.EXTENSION]
>> fs[pred_setTheory.SPECIFICATION] >> fs[]
>> fs[pred_setTheory.IMAGE_applied]
>> fs[pred_setTheory.IN_APP, boolTheory.RES_EXISTS_THM]
>> metis_tac[]
QED
(*
  (* Original human proof *)
  PURE_REWRITE_TAC [SURJ_DEF,EXTENSION,IN_IMAGE] THEN
  REPEAT GEN_TAC THEN EQ_TAC THENL
  [REPEAT (STRIP_TAC ORELSE EQ_TAC) THENL
   [RES_TAC THEN ASM_REWRITE_TAC [],
    MAP_EVERY PURE_ONCE_REWRITE_TAC [[CONJ_SYM],[EQ_SYM_EQ]] THEN
    FIRST_ASSUM MATCH_MP_TAC THEN FIRST_ASSUM ACCEPT_TAC],
   DISCH_THEN (ASSUME_TAC o CONV_RULE (ONCE_DEPTH_CONV SYM_CONV)) THEN
   ASM_REWRITE_TAC [] THEN REPEAT STRIP_TAC THENL
   [EXISTS_TAC (“x:'a”) THEN ASM_REWRITE_TAC [],
    EXISTS_TAC (“x':'a”) THEN ASM_REWRITE_TAC []]])
*)
```