Towards Finding Longer Proofs

Zsolt Zombori[1,2], Adrián Csiszárik[1,2], Henryk Michalewski[3,4], Cezary Kaliszyk[5,3], and Josef Urban[6]

[1] Alfréd Rényi Institute of Mathematics, Budapest
[2] Eötvös Loránd University, Budapest
[3] University of Warsaw
[4] Google Inc.
[5] University of Innsbruck
[6] Czech Technical University in Prague

Abstract. We present a reinforcement learning (RL) based guidance system for automated theorem proving geared towards Finding Longer Proofs (FLoP). Unlike most learning based approaches, we focus on generalising from very little training data and achieving near complete confidence. We use several simple, structured datasets with very long proofs to show that FLoP can successfully generalise a single training proof to a large class of related problems. On these benchmarks, FLoP is competitive with strong theorem provers despite using very limited search, due to its ability to solve problems that are prohibitively long for other systems.

Keywords: automated theorem proving · machine learning · reinforcement learning · connection calculus

1 Introduction

Automated Theorem Proving (ATP) is the study of using machines for formal mathematical reasoning. It is related to general game playing, for example, the game of Go can be viewed as a simple formal system. Building on the recent success of machine learning, a growing trend in this field is to use learning methods to make theorem provers more powerful. Several research projects have shown that learning can be used to replace/surpass human-engineered heuristics. Despite huge improvements, interesting mathematical theorems remain elusive today. One crucial shortcoming of ATP systems is that they can typically find only relatively short proofs.

In this paper, we address this shortcoming and ask the question of how machine learning can be used to solve problems requiring very long inference chains. We argue that the fundamental reason why current ATP systems are limited to short proofs is that they focus on the search aspect of the task in the space of inference steps. It is very natural to see theorem proving as a search problem: each proof step involves a choice from a set of valid inferences, yielding a search space that grows exponentially with the length of the proof. Due to the exponential blowup, the search is bound to fail beyond a certain depth – except for special classes of problems where one of the smart human heuristics of the theorem prover allows for finding the solution without a search. As W. W. Bledsoe observed [7]: "Automated theorem proving . . . is not the beautiful process we know as mathematics. This is 'cover your eyes with blinders and hunt through a cornfield for a diamond-shaped grain of corn'."

Approaches that try to avoid excessive search broadly fall into three categories: 1) Perform large steps, such as the invocation of tactics, decision procedures in SMT solvers [4], or other complex algorithms. This approach is widely used in interactive theorem provers, e.g. [13,5,30]. 2) Perform hierarchical reasoning by first creating a high-level proof plan and then gradually refine it to the calculus level, e.g. [10,29]. 3) Reason by analogy, e.g. [28,8].

Reasoning by analogy involves observing the proof of one problem, extracting the core idea, and successfully applying it to another. Note that using this formulation, success is barely dependent on proof length. On the other hand, establishing mappings between proofs is challenging and depends heavily on a proper data representation, which has been from the beginnings of ATP a major bottleneck for this approach. However, with the advent of machine learning methods capable of automatically discovering good data embeddings, the analogy approach seems worth revisiting.

In this work, we interpret analogical reasoning as building a model that internalises a proof and then successfully applies it to a class of related problems, without relying much on search. The trained model is supposed to know the proof of an unseen, yet familiar problem. This is a highly simplified approach which does not capture the full potential of analogy, but we argue that it is a meaningful start that will hopefully lead to more refined solutions. We select classes of problems where proofs are highly similar and trained humans can often generalize a single demonstration to the entire class, even if proof lengths greatly differ. We explore whether machine learning can yield similar generalization.

Many successful ATP systems, such as [52,17,11,19,56,31,36] implement the MaLARea [50,52] learning/reasoning loop (described later also as the DAgger [43] meta-algorithm). The MaLARea loop interleaves ATP runs based on the current models (*data collection phase*) with a *training phase*, in which these models are updated to fit the collected data.

An alternative family of reinforcement learning methods, including Temporal Difference (TD) learning [49], continuously update their models, allowing the system to bootstrap on itself. Such methods have so far been mostly ignored by the theorem proving community. In these methods, the search is usually replaced by rollouts. While MaLARea has been shown to yield good search heuristics, we argue that rollout based data collection is more suitable when the aim is to fully explore the space around a single problem without overfitting to it. Our work has the following contributions.

- We introduce a new theorem proving algorithm FLoP (Section 4) based on a TD algorithm [7] and the connection tableau calculus [6]. FLoP makes use of the curriculum learning algorithms presented by [40] and [44]. These techniques are well established in RL, however, they have never been applied to theorem proving before.
- We introduce a synthetic dataset of increasingly difficult arithmetic problems, as well as two datasets from the Logical Calculi domain of the TPTP [48] library, augmented with lemmata (Section 5).
- We show that when restricted to single shot evaluation – without search – FLoP performs very well, while another prover based on guided Monte Carlo Tree Search greatly degrades.
- We evaluate FLoP on our arithmetic benchmarks by training it on a single problem and show that it generalizes very well even when evaluated without search, allowing just a few proof attempts. This suggests that it has learned a simple form of reasoning by analogy.
- We use the arithmetic benchmarks to compare FLoP with state-of-the-art provers Vampire [25], E [46], leanCoP [32] guided by human-designed strategies, and with rlCoP [19] – an RL-based connection tableau prover. In the simple setup of unary encoding of numbers, FLoP is only outperformed by a portfolio (multi-strategy) mode of a single manually optimized rewriting-based system and only after trying several of its autoconfiguration heuristics. When using binary encoding, FLoP performs best, demonstrating its ability to generalize to long proofs.

Our datasets presented in Section 5 seem to be particularly suited for machine learning methods: some problems are algorithmically simple, with long solutions and strong shared structure (Robinson Arithmetic) while others are less similar, but hierarchically structured (Logical Calculi). Nevertheless, state-of-the-art systems struggle with solving some of the problems (see Section 6). Furthermore, our problems are much easier to analyze than typical heterogeneous proof corpora, hence promising better understanding of the current limits of theorem provers. The difficulty of our synthetic problems, as well as the proof lengths, are easily adjustable, yielding a scalable RL benchmark with interpretable failure modes.

Our code, datasets and all experiment configuration files are available at http://bit.ly/code_atpcurr[8]. Supplementary materials including screencasts with gameplays performed in our environments are available at the project webpage http://bit.ly/site_atpcurr.

[7] In particular, we use Proximal Policy Optimization [45] (PPO), a variant of the policy gradient method, which uses Temporal Difference learning for optimization of the value function.

[8] This distribution does not include the fCoP theorem prover, which cannot yet be publicly released, however, a binary can be obtained upon request.

2 Related work

Theorem Proving by Analogy. Analogy has long been considered one of the most important heuristics in mathematical problem solving, e.g. [38,37]. It also gained attention in automated theorem proving, e.g.[8,28], as an alternative of search-based methods. [8] defines analogical reasoning as "the proof of one theorem is used to guide the proof of a similar theorem by suggesting analogous steps". They rely on a user-provided matching between analogous concepts related to the two theorems and try to reuse the proof steps (adjusted modulo analogy) in the source proof during the construction of the target. [28] aim to achieve this on a higher level of abstraction by matching proof plans of a source and a target problem. As the proof plan is constructed, the plan of the source is searched for steps that can be transformed into a suitable step for the target. The set of allowable transformations are predefined and designed for a narrow domain. For example, the transformations given in [28] aim to carry a result, such as the Heine Borel theorem, stated in $\mathbb{R}^1$ over to $\mathbb{R}^2$. The characteristic feature of these systems is that search is performed on the meta level of plan mappings and proof step transformations. The search space is often defined ad hoc and is much smaller than that given by the inference rules of the calculus.

A machine learning system that is trained to guide a theorem prover is supposed to achieve a similar result, with two important improvements. First transformations are learned, without the need for manual engineering. Second, establishing mappings between proof steps (that can be transformed into each other) should result from learning of flexible and abstract features. The flexibility and abstraction allows for potentially reusing the same proof components several times, as well as using components from different proofs, which goes beyond earlier attempts that only establish direct matching between the two proofs.

Machine learning systems for guiding theorem provers. A large body of research exists that aims to provide guidance for theorem provers via machine learning. FEMaLeCoP [18], rlCoP [19,31] plCoP [56] and lazyCoP [39] guide the leanCoP [32] compact connection tableau prover, which is also the system guided in our project. Learning based guidance is added to the saturation based E prover [46] in [26,16,11,15]. The HOList project [3,33] builds guidance on the tactic level[9] for the HOL Light [13] higher-order theorem prover. A distinctive feature of all these systems is that they rely heavily on an external search procedure, such as Monte Carlo Tree Search [24], or the search engine of the guided prover. Learning is aimed at making search more efficient and it is implemented in alternating iterations of proof search and model fitting, according to the DAgger [43] meta-algorithm, first used in MaLARea [52] for theorem proving. In contrast with the above, we use an algorithm which uses bootstrapping and learns from generated rollouts, aiming to learn to generate entire proof sequences in the leanCoP calculus. Such rollout based learning has so far been barely used in theorem proving, with the noteable exception of [12], developed in parallel with FLoP, which guides a saturation style prover using a simple policy gradient RL algorithm.

[9] A tactic is a human-designed program which aggregates multiple proof steps.

Concurrently with our work, [35,51,36] have used recurrent neural networks, attention and transformers to generate next proof steps. E.g., [36] report generalisation on problems with relatively short proofs. In line with emphasizing analogy over search, their evaluation protocol only allows for limited search in a single proof attempt [10]. Our work employs much smaller neural models and focuses on generalizing to proofs with hundreds and thousands of steps (see Figure 3).

Provers guiding the leanCoP Connection Tableau Calculus. As noted above, a series of learning systems guide the leanCoP connection calculus. Of these, we highlight three systems that use roughly the same learning setup: rlCoP [19], plCoP [56] and graphCoP [31]. In these systems, the value and policy functions of the guided MCTS algorithm are learned similarly to [1,47]. FLoP shares the same manually developed features [21] with rlCoP and plCoP, while graphCoP employs a graph neural network for feature extraction. We use these systems as an important baseline in Section 6. While the differences are important, they play little role in our current investigation and we refer to them jointly as *mcts-CoP*s.

3 The leanCoP Connection Tableau Calculus

FLoP provides guidance for of the very compact leanCoP [32] connection tableau calculus. The calculus was originally implemented in Prolog, but it also has an OCaml reimplementation fCoP [20] and FLoP can be used to guide both systems.

We briefly describe the connection tableau calculus, assuming basic first-order logic and theorem proving terminology [41]. The input is a (mathematical) problem consisting of *axioms* and *conjectures* formally stated in first-order logic (FOL). The calculus searches for *refutational proofs*, i.e. proofs showing that the axioms together with the negated conjectures are *unsatisfiable*. The FOL formulas are first translated to *clause normal form* (CNF), producing a set of first-order *clauses* consisting of *literals*, e.g. $\{\forall X, Y : (f(X)|r(X,Y|\neg f(Y)), f(a)\}$. Proof search starts with a *start clause* as a *goal* and proceeds by building a connection tableau by repeatedly applying *extension steps* and *reduction steps*.

The extension step connects (*unifies*) the *current goal* with a complementary literal of a new clause. This extends the *current branch*, possibly splitting it into several branches if there are more literals in the new clause, and possibly *instantiating* some variables in the tableau. The reduction step connects the current goal to a complementary literal of the *active path*, thus *closing* the current branch. The proof is finished when all branches are closed. The extension and reduction steps are nondeterministic, requiring backtracking in the standard connection calculus. *Iterative deepening* is often used to ensure completeness. The project webpage shows an example *closed connection tableau*, i.e., a finished proof tree where every branch contains *complementary literals* (literals with opposite polarity). This shows that the set of clauses is unsatisfiable.

leanCoP represents theorem proving as a one-person game. The game ends with success if a proof is found. The prover has many choices to make along the

[10] A maximum of 4096 search nodes are allowed.

way, in particular it can select from several valid extension and reduction steps. Whether a step is valid depends on the unification condition, i.e., if the current goal unifies with the negation of a literal in the corresponding clause. The full information about the game state consists of all previous proof steps, the partial proof tree (proof state) and the current goal.

The search space of the prover is exponentially large in the length of the proof. In leanCoP, the action space is roughly correlated with the size of the axiom set. While this can be large for large problems, typically only a few actions are available in any particular state.

4 FLoP – Main Algorithm

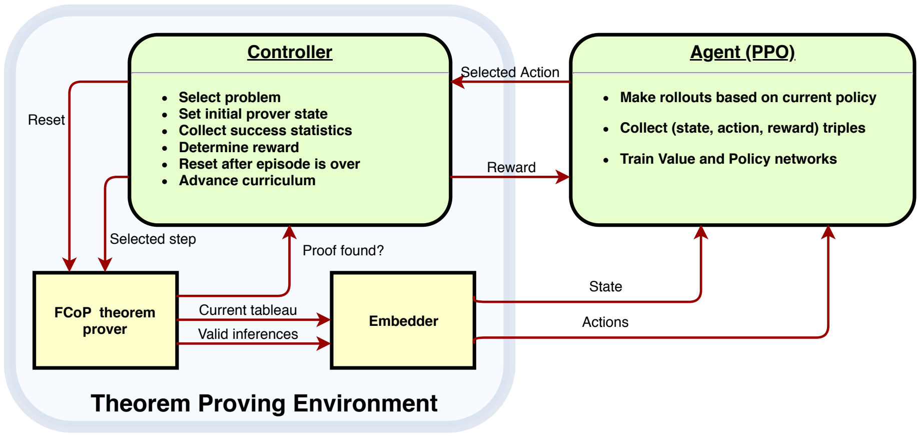

Fig. 1: *Theorem proving as a reinforcement learning environment.*

FLoP combines the connection tableau calculus with guidance based on Temporal Difference and curriculum learning. After each inference step, the prover engine returns its current state as well as the set of valid actions, i.e., valid inference steps that transform the current goal. The prover is encapsulated into a Reinforcement Learning (RL) environment. In the following, we provide a brief summary of the relevant RL techniques.

4.1 Reinforcement Learning fundamentals

Our RL summary is highly selective, aiming to describe Proximal Policy Optimization [45], the method used in FLoP. For further details, see [49].

Markov Decision Process The mathematical foundation of the class of problems that Reinforcement Learning aims to solve is given by Markov Decision Processes (MDP). An MDP$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ describes a dynamic process and consists of the following components: $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{R} : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ is a reward function, $\mathcal{P} : (\mathcal{S} \times \mathcal{A}) \to \mathcal{S}$ is the state transition function and γ is the discount factor. We assume that an agent interacts with this MDP, generating sequences of (s_t, a_t, r_t) state-action-reward tuples, called *trajectories*. The agent is equipped with a *policy* function $\pi : \mathcal{S} \to \mathcal{A}$ which determines which action it selects in a particular state. The aim of the agent is to maximize its total accumulated reward $\sum_{t \geq 0} \gamma^t r_t$. Several components of the model can be stochastic: the reward function, the transition function, as well as the policy. In such settings, the aim of the agent it to find the policy π^* that maximizes its cumulative expected reward, where future rewards are discounted with the γ discount factor:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | \pi]$$

Policy Gradient One successful family of methods solves this task by considering a parametric class of policy functions $\Pi = \{\pi_\Theta, \Theta \in \mathbb{R}^m\}$. We continuously sample trajectories from the current policy and optimize the parameters Θ via gradient descent based on the observed rewards. This is called *policy gradient*, and the RL literature contains numerous variants that differ in the details of optimization.

One well known difficulty of policy gradient is the large variance in the sampled trajectories, which makes convergence slow and requiring large number of training samples. A popular technique to reduce variance is to train a baseline model that esimates the expected reward from a given state and optimize the policy with respect to the excess reward on top of the baseline. This gives rise to the *actor-critic framework*. We train two models jointly: a *critic*: $V_\pi(s)$ that estimates the expected reward of trajectories starting from s given policy π and an *actor*, which is our policy π. Given some state s, we use the policy to sample an action a. We then sample further transitions to estimate the expected reward $Q_\pi(s, a)$ from state s after taking action a. We define *advantage* as the difference between these two expectations: $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$. Our optimization objective is then:

$$\min_{\Theta_V} \max_{\Theta_\pi} A_\pi(s, a)$$

where Θ_V and Θ_π are the parameters of the critic and the actor, respectively.

Proximal Policy Optimization Policy gradient is an *on-policy* method, meaning that it optimizes the parameters of the policy based on trajectories sampled from the same policy. In contrast with *off-policy* methods, which extract samples through some other mechanism, policy gradient learning can be highly unstable. This is because the change in policy potentially invalidates the samples it was trained on. Proximal Policy Optimization [45] (PPO) addresses this problem by introducing a soft constraint on the magnitude of the policy updates.

We maintain two instances of the policy network: π_Θ which we aim to improve and $\pi_{\Theta_{old}}$ which we sample from. The ratio of the two policies gives us a measure of difference:

$$r_t(\Theta) = \frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}(a_t|s_t)}$$

If this ratio lies outside of the range $[1-\epsilon, 1+\epsilon]$, then the advantage function is clipped:

$$r_t^*(\Theta) = \text{clip}(r_t(\Theta), 1-\epsilon, 1+\epsilon)$$

$$A_{\pi_\Theta}^*(s,a) = \min(r_t(\Theta)A_{\pi_{\Theta_{old}}}, r_t^*(\Theta)A_{\pi_{\Theta_{old}}})$$

The two neworks are periodically synchronized to ensure that they are not too different. PPO has been shown to strike a good balance between simplicity and stability and is one of the most popular policy gradient methods.

4.2 Reinforcement Learning in FLoP

Theorem proving can be directly mapped into an MDP by treating prover states as states, inference steps as actions and proof attempts as trajectories. The only missing component is the reward function, which we set to be

$$\mathcal{R}(s,a) = \begin{cases} 1 \text{ if perfoming a in s finishes the proof} \\ 0 \text{ otherwise} \end{cases}$$

Other reward functions are also possible, though we argue that the selected one is most faithful to the task at hand: 1) we know very little about progress before we have found a proof, hence the zero reward for intermediary steps and 2) it is hard to tell if one proof is better than another, hence the binary nature of the rewards.

Reward maximization directly corresponds to finding a proof. Hence, we augment the core connection tableau calculus with a value (critic) and a policy (actor) model trained using PPO. Classical proof search is then replaced with generating proof attempts from the policy. Figure 1 shows the overall architecture of the system and Figure 2 shows the policy and value network architectures. The state and the actions (formulae) are represented using previously developed features [21]. The features include (suitably hashed) triples, pairs, and singletons of adjacent nodes in the formula trees and the partial proof trees, as well as some global features: number of open goals, number of symbols in them, their maximum size and depth, length of the current path, and two most frequent symbols in open goals. This means that the proof states and the actions are presented as (sparse) fixed-length vectors.

4.3 Curriculum Learning

A fundamental challenge for an RL system that learns to prove theorems from its own exploration is that rewards are sparse and binary. In case proofs are long, this makes learning nearly impossible. To tackle this, we use curriculum learning on the length of proofs in case a proof is available. Initially, we start exploration from near the end of the proof, making it easy to succeed and obtain positive reward. As

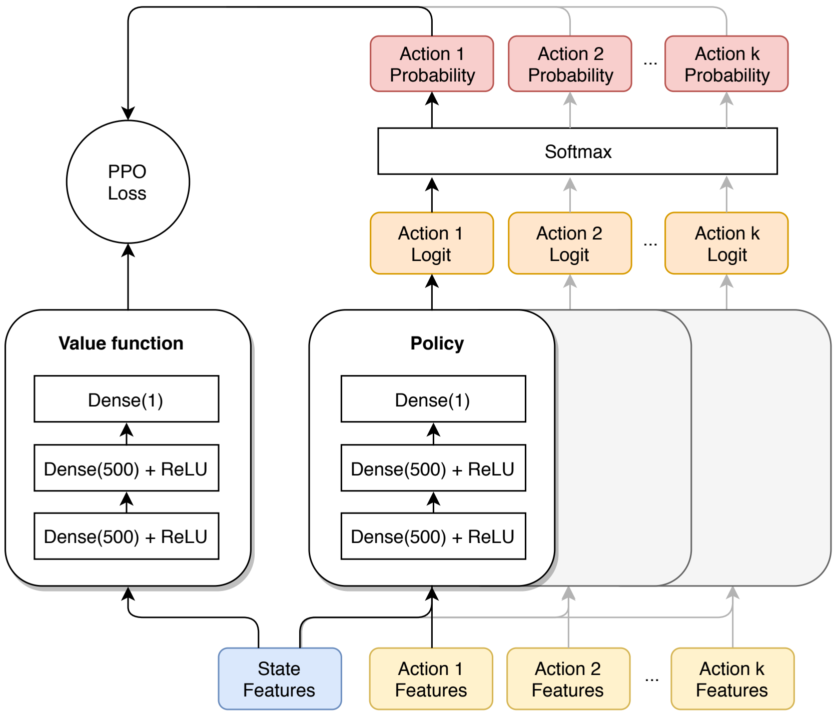

Fig. 2: *Value and Policy network architectures in PPO. Their inputs are state and state-action pair features, respectively. The policy returns a score for each action, which are then normalized to a probability.*

the system gets more confident, we gradually move the starting state backwards along the given proof. This approach has already been successfully applied in many RL experiments. When there is no good alternative to the training proof, the system eventually learns those steps, while random exploration helps to identify alternatives and find novel proofs. Exploration also helps to learn steps that make the proof impossible to finish. We can start learning with or without training proofs. Each training problem can have its own curriculum schedule, which can be restarted when a new proof is found. Curriculum learning is an efficient tool for boosting rewards found during exploration.

4.4 Training algorithm

Algorithm 1 gives an overview of the learning loop. First, in line 5 we sample a problem (in case there are multiple). In lines 6–9 we interact with the prover and ensure that its state corresponds to the one dictated by the current curriculum. In lines 10–15 we generate a proof attempt iterating 1) prover steps, 2) featurization, and 3) sampling a next action according to the policy. If a new problem is solved, we start the curriculum on it in lines 17–19. If performance on a given problem

Algorithm 1 FLoP: Main Learning Loop

```
Require: problems $\mathcal{P}$, policy $\pi$, value $v$, train steps $\in \mathbb{N}$, threshold $\in [0..1]$, episodes
    between updates: k $\in \mathbb{N}$
Ensure: trained policy $\pi$, trained value $v$, possibly proofs for some problems in $\mathcal{P}$
 1: $curriculum \leftarrow$ dictionary such that for each $p \in P$ with proof $Pr$ $curriculum[p] =$
    $\text{len}[Pr] - 1$
 2: steps $\leftarrow 0$
 3: while steps $<$ train steps do
 4:   for j in 1..k do
 5:     $p \leftarrow$ random problem from problem set $\mathcal{P}$ {An episode corresponds to a
        problem}
 6:     initialize prover on problem $p$
 7:     if $p$ has stored proof then
 8:       Take $curriculum[p]$ proof steps according to stored proof
 9:     end if
10:     while not episode over do
11:       $s', a'_1, a'_2 \ldots a'_l \leftarrow$ Query prover for current state and valid actions
12:       $s, a_1, a_2 \ldots a_l \leftarrow \text{feat}(s'), \text{feat}(a'_1), \text{feat}(a'_2) \ldots \text{feat}(a'_l)$ {Extract features}
13:       Take action according to policy $\pi(a|s)$, observe reward $r$
14:       steps $\leftarrow$ steps + 1
15:     end while
16:     update success ratio for $p$
17:     if $p$ is solved with proof $Pr$ and no proof of $p$ was known before then
18:       $curriculum[p] \leftarrow \text{len}(Pr) - 1$ {Start curriculum}
19:     end if
20:     if success rate for $p >$ threshold then
21:       $curriculum[p] \leftarrow curriculum[p] - 1$ {Advance curriculum}
22:     end if
23:   end for
24:   Update policy $\pi$ and value $v$
25: end while
```

and curriculum reaches a threshold, we advance the curriculum in lines 20–22. In line 24 we update the policy and value models.

4.5 Implementation details

Most of the FLoP system is implemented in the Python programming language, using the [14] RL framework. FLoP guides the fCoP [20] system, which is a reimplementation of leanCoP in the OCaml programming language. The communication between the guidance and prover components is provided via the C foreign language interface.

5 Datasets

To evaluate our system, we select simple classes of theorems with strong shared structure, giving a large room for learning-based improvement. Our five datasets are described in Table 1. The datasets are bundled into an OpenAI-gym [9] compliant environment and can be tested with modern RL algorithms.

Table 1: *Three challenges defined in the theory of Robinson Arithmetic (RA) and two challenges from the Logical Calculi (LCL) domain of the TPTP library*

Name	Theory	Size	Description
RA-1	RA	1800	Expressions of the form $N_1 + N_2 = N$, $N_1 \cdot N_2 = N$, where $0 \leq N_i < 30$. (Examples: 3+4=7 or 5·12=60.)
RA-2	RA	1000	$T = N$, where $0 \leq N$, and T is a random expression with 3 operators and operands N_i such that $0 \leq N_i < 10$. (E.g.: ((3+4)·2)+6=20.)
RA-3	RA	1000	$T_1 = T_2$, where T_1 and T_2 are random expressions with 3 operators and operands N_i such that $2 \leq N_i < 10$. E.g. ((3+4)·2)+6=((1+1)·5)·2.)
LCL-Eq	LCL	890	TPTP domain: Logic Calculi (Equivalential) – extended with lemmata from E prover.
LCL-Imp	LCL	1204	TPTP domain: Logic Calculi (Implication/Falsehood 2 valued sentential) – extended with lemmata from E prover.

Three datasets are built on the theory of Robinson Arithmetic [42], which defines addition and multiplication on the nonnegative integers. Despite its relative simplicity, this theory seems to be particularly suited for machine learning methods: solutions are long and repetitive, while also challenging for state-of-the-art systems (see Section 6). We examine both unary (24 actions) and binary (40 actions) encoding of numbers. The axioms of Robinson Arithmetic are given on the project webpage. Increasing the numbers in the conjecture greatly increases the length of the proof, making this dataset suitable for detecting the length boundary of various theorem provers.

Two datasets are extracted from the TPTP library, from the domain of Logical Calculi with condensed detachment (LCL). These theorems have been extensively studied from the early days of automated theorem proving, e.g. [27,34,22,55]. We run E prover with a large time limit on the problems and augment the dataset with lemmata extracted by E. As a result, many proofs of simpler problems can be directly used as parts of the proofs of harder problems. A direct analogy from one problem to the other is usually not possible, however, shallow search is often sufficient to connect the proofs of easier problems to the proof of harder ones.

The LCL domain in the TPTP [48] library consists of statements about various formal inference systems. LCL-Eq and LCL-Imp formalize properties of the Equivalential Calculus and the Implication and Falsum Calculus, respectively. Both are subsystems of the classical propositional calculus, restricting the set of allowed connectives to $\{\equiv\}$ and $\{\implies, \bot\}$. For both subsystems, the appropriate variant of the *condensed detachment* inference rule ($A, A \equiv B \vdash B$ and $A, A \implies B \vdash B$) constitutes a *strongly complete* inference system, i.e., whenever a formula semantically follows from a set of premises, it also follows from the set syntactically. A number of complete axiomatizations of both the Equivalential Calculus and the Implication and Falsum Calculus exist and the theorems in our datasets establish connections between them.

All arithmetic problems in our dataset are quite simple for humans, but in the case of logical calculi, some of the problems were posing a challenge for mathematicians (see [54]).

6 Experiments

Our experiments with Robinson arithmetic aim to demonstrate that in this highly structured dataset FLoP is capable of extracting a general proof pattern from one or two proofs and generalizing to related proofs of arbitrary length, using a restricted few-shot evaluation method (see below). Experiments 1, 2, and 3 compare FLoP with strong theorem provers using different fragments of the arithmetic dataset, varying the complexity of the axiomatization (unary vs. binary encoding of numbers) and the complexity of the target theorems (RA-1, RA-2, RA-3). FLoP is either the best or the second-best in each experiment. In each of these experiments, FLoP is allowed 100 proof attempts without backtracking: the first attempt is a deterministic run with a high time limit (1000 sec) that always selects the action maximizing the policy and the remaining 99 runs are stochastic samples from the policy with a time limit of 60 sec.

The LCL problems used in our experiments are less structured and success is dependent on search, even if the hierarchical composition of problems ensures that a relatively small search is sufficient to generalize from easier problems to harder ones. Consequently, we expect that search-based methods are better in this domain. However, when search is completely disallowed during evaluation, we show in Experiment 4 that FLoP performs much better than the mcts-CoPs. In Experiments 5 and 6 we demonstrate the benefit of using curriculum learning.

Our hyperparameters were selected using small grid searches. We checked standard RL parameters (e.g., the discount factor), parameters related to curriculum scheduling (e.g., local vs. global), neural network architectures (1–5 layers with 128–1024 neurons), feature sizes (64–1024) and training steps (10^5 – 10^8). Parameters used in the experiments are described in configuration files which are accessible along with the shared codebase.

Experiment 1: Comparison with other provers. We compare FLoP with a random model, two state-of-the-art saturation-style theorem provers (E 2.4, Vampire 4.3.0), a heuristic guided connection tableau prover (leanCoP 2.1), and rlCoP (one of the mcts-CoPs). Vampire, E, and leanCoP use human-designed strategies instead of learning. We use these provers in the configuration used for CASC, the yearly competition of fully automated theorem provers, employing a time limit of 60 sec. per problem. For E, we also report the results of the *auto-schedule* mode. For rlCoP we used the hyperparameters described in [19], only modifying the policy temperature from 2.5 to 1.5, as this works better with the Robinson datasets. The number of inferences in MCTS was limited to 200000. rlCoP was trained on the whole evaluation set, while FLoP was trained on a single problem: $1 \cdot 1 = 1$ and $1 \cdot 1 \cdot 1 = 1$ for RA-1 and RA-2, respectively.[11]

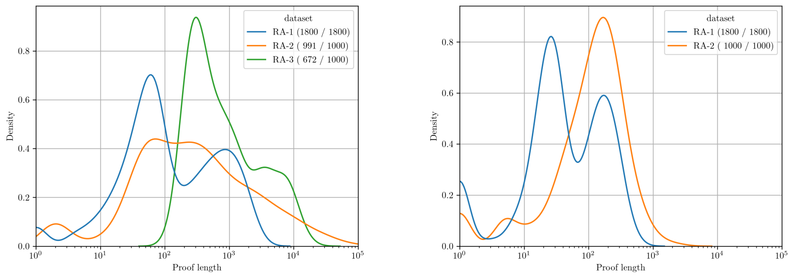

Fig. 3: *Distributions of length of proofs found by FLoP. Note the logarithmic scale.* **Left***: RA-1, RA-2 and RA-3 with average proof lengths 367, 2082, and 1864.* **Right***: binary RA-1 and binary RA-2 with average proof lengths 85 and 179.*

Success ratios are given in Table 2. FLoP is only outperformed by E's *auto-schedule*, which tries multiple strategies and finds one with the left-to-right ordering of all the addition and multiplication axioms. This solves all of our problems immediately without proof search by only rewriting to a normal form [2]. This demonstrates the power of equational theorem proving when a suitable term ordering exists and can be found by human-designed heuristics. This is, however, far from guaranteed in general even in such simple domains, as witnessed by Vampire's failure to find this ordering. To evaluate E without access to its built-in rewriting capability, we have renamed the equality to a new predicate 'eq' axiomatized exactly in the same way as in leanCoP. The auto-schedule mode then becomes somewhat weaker than the auto mode.

Table 2: *Comparing a random model, Vampire, E, leanCoP, rlCoP and FLoP, with respect to success ratio for RA-1, RA-2 and RA-3 problems. Our method (FLoP) is marked in grey. E_1 – auto mode, E_2 – auto-schedule mode, E_3 – auto-schedule with renamed equality. The reason why FLoP did not reach 100% on RA-2 is that a few problems timeouted.*

Dataset	Random	Vampire	E_1	E_2	E_3	leanCoP	rlCoP	FLoP
RA-1	0.04	0.60	0.60	**1.0**	0.54	0.22	0.86	**1.0**
RA-2	0.05	0.40	0.39	**1.0**	0.25	0.14	0.74	0.99
RA-3	0.00	0.34	0.28	**1.0**	0.22	0.01	0.41	0.67

Experiment 2: Harder Arithmetic Expressions. RA-3 consists of arithmetic equalities with random expressions on both sides. This dataset is significantly more complex because there are many ways of proving the same problem. Proofs are longer, too. For FLoP, we examined various training sets and found that the system is very prone to overfitting. Most problems can be proven in many

[11] For a description of RA-3 training problems, see Experiment 2.

different ways, that vary greatly in terms of how well they foster generalization. It is true especially of easier problems that they can be proven with "shortcuts" that hinder generalization (see more on this on the project webpage). The harder the problems, the less likely they can be solved with such heuristic approaches, hence harder training problems promise better training signal. We demonstrate this by training FLoP on a few harder problems with proofs provided, making use of curriculum learning described in Section 4. A single longer training proof is sufficient to yield meaningful generalization. Adding one more training problem helps even more, as shows Table 3.

Table 3: *Curriculum learning for RA-3 on two harder problems with proofs of* 113 *and* 108 *steps. We report success ratios and average proof lengths, based on* 3 *runs. Standard deviations are given in parenthesis.*

Training problem	Succ.	Len.
$1 \cdot 2 + 1 + 1 = (1+1) \cdot 1 \cdot 2$	0.32(0.05)	566(14)
$1 \cdot 2 + 1 + 1 = (1+1) \cdot 1 \cdot 2$ $(1+1+1) \cdot 2 = 2 \cdot 1 + 2 + 2$	**0.67** (0.03)	1864(54)

Figure 3 shows the distribution of the length of proofs found by FLoP. We can see that a large part of the problems requires thousands of steps to solve, highlighting the need to avoid search.

For rlCoP, all RA-3 problems are too hard to solve without guidance within the inference limit, so we started with the version trained on the solutions of RA-2. Table 2 shows that FLoP is only outperformed by E's auto-schedule mode, which again finds the rewrite ordering that solves all problems without search.

Experiment 3: Binary Number Encoding. We experiment with Robinson Arithmetic using binary encoding of numbers. This makes the domain theory more complex: the total number of actions increases from 24 to 40. [12] On the other hand, proofs get shorter, as shows Figure 3. Again, we train FLoP on a single proof: $3 \cdot 3 = 9$ and $(1 \cdot 2 + 1) \cdot 3 = 9$ for RA-1 and RA-2, respectively. Table 4 shows that provers get weaker, except for Vampire and FLoP. In particular, E is no longer capable of solving the problems with rewriting only. FLoP manages to generalize from a single proof to the whole dataset despite the increased action space and performs best in this experiment.

Table 4: *Comparing Vampire, E (auto-schedule mode), leanCoP, rlCoP and FLoP, using binary encoding of numbers.*

Dataset	Vampire	E	leanCoP	rlCoP	FLoP
RA-1	0.67	0.81	0.19	0.56	**1.0**
RA-2	0.62	0.62	0.13	0.12	**1.0**

Experiment 4: Search vs. Eager Evaluation We compare FLoP with plCoP (one of the mcts-CoPs) using two different evaluation methods. After training both systems on the whole dataset, we evaluate them using 1) MCTS and 2) eager evaluation, i.e. always select the action with the highest probability according to the policy model. Table 5 shows that plCoP performs better when search is allowed, especially for the more heterogeneous LCL problems. However, FLoP takes the upper hand in eager evaluation. For the LCL problems, plCoP collapses while FLoP is unaffected. This suggests that plCoP depends heavily on the search procedure it used for training. FLoP cannot make good use of MCTS, which is

[12] Note that only a subset of these is applicable in a given state.

Table 5: *Comparing FLoP and plCoP using two different evaluation methods: 1) guided MCTS and 2) eager evaluation based on the policy model (Eager Policy). For plCoP we also evaluate based on the value model (Eager Value)*

Prover	**Eval**	**LCL-Eq**	LCL-Imp	RA-1	RA-2
plCoP	MCTS	**47%**	**61%**	65%	48%
plCoP	Eager Policy	5%	5%	82%	49%
plCoP	Eager Value	1%	1%	3%	5%
FLoP	MCTS	19%	24%	61%	31%
FLoP	Eager Policy	19%	27%	**100%**	**99%**

somewhat expected, since its policy and value networks were not trained for that purpose. For the arithmetic datasets, both systems benefit from not doing search because they reach proofs that are longer than what MCTS can reach. For FLoP, the removal of the depth limit reveals that it fully mastered the two problem classes, regardless of depth.

The performance of plCoP gets even worse if the eager evaluation is based on the value model, i.e., when we select the action whose successor state has the highest value score. We conjecture that this is because assigning a value to a never observed state is much harder than selecting from a smaller set of actions. These results are in line with our conjecture that the DAgger approach of plCoP is better for learning good search heuristics, while FLoP is better at internalizing a full proof pattern.

Experiment 5: Curriculum Learning vs only Exploration Based Learning. When training proofs are not available, the positive reward signal only occurs after the system solves a problem through exploration. Afterward, curriculum learning ensures that the system is continuously faced with a "reasonably" hard problem, alleviating the sparse reward challenge of theorem proving. We demonstrate this on the two LCL datasets. Here, before generating each rollout, we randomly select a problem from the entire dataset. We report the number of proofs found during training in Table 6. Curriculum learning brings a small, but consistent improvement when compared with only exploration-based learning.

Table 6: *Curriculum Learning compared with only exploration based learning, on the LCL-Eq and LCL-Imp datasets, using 10M and 30M inference limit, respectively. We report the ratio of proofs found during training. The results are averages of 2 runs.*

Dataset	Curriculum	No curriculum
LCL-Eq	**0.24** (0)	0.23 (0.001)
LCL-Imp	**0.51** (0.002)	0.45 (0.003)

Experiment 6: Curriculum Learning vs. Supervised Learning When training proofs are available, a natural baseline of curriculum learning is supervised learning on the proof steps. While such behavioral cloning sometimes leads to great performance, we show in Table 7 that it greatly depends on the quality of the given proof. We train RA-1 and RA-2 using the following training problems:

1. **RA-1** $1 + 1 = 2$, $1 \cdot 1 = 1$
2. **RA-2** $1 + 1 = 2$, $1 \cdot 1 = 1$, $1 \cdot 1 \cdot 1 = 1$

We take the “nice” proofs (5, 9 and 23 steps) of these problems and construct variants with 2-3 extra steps added. We observe that supervised learning degrades as superfluous steps are introduced, while FLoP’s exploration allows the system to recover and find the original proofs.

Table 7: *Curriculum Learning vs Supervised Learning trained on proofs with extra steps added for distraction.*

Data	Proof Lengths	Supervised Succ.	Curriculum Succ.
RA-1	5, 9	0.98(0.04)	**1(0.01)**
	9, 11	0.52(0.08)	**0.98(0.01)**
RA-2	5, 9, 23	**0.85(0.04)**	0.76(0.02)
	9, 11, 25	0.59(0.08)	**0.76(0.01)**

7 Conclusion and Future Work

We have built FLoP, a proof guidance system based on a variant of temporal difference reinforcement learning, addressing the problem of finding long proofs in an exponential search space. Previous work [53,23] focused on finding long proofs with the help of human-designed heuristics. We showed that FLoP is capable of extracting proof patterns via learning and can generalise to much longer proofs, implementing a simple form of reasoning by analogy. We believe that mastering analogical reasoning is an important step in creating human-level automated mathematicians. We presented a set of theorem proving datasets that are suitably challenging for existing learning methods and are intended to become a general-purpose testing ground for reinforcement learning methods. We showed that FLoP can outperform strong theorem provers on some of these datasets. We find that curriculum learning is a useful component of the learning algorithm as it allows for amplifying training signal when proofs are long.

8 Acknowledgments

Adrián Csiszárik and Zsolt Zombori were supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008 and by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program. Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959. Cezary Kaliszyk was supported by ERC grant no. 714034 *SMART*. Josef Urban was supported by the *AI4REASON* ERC Consolidator grant number 649043, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

This research was supported by the PL-Grid Infrastructure. In particular, quantitative results of FLoP reported in this paper were performed using the Prometheus supercomputer, located in the Academic Computer Center Cyfronet in the AGH University of Science and Technology in Kraków, Poland.

References

1. Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. CoRR **abs/1705.08439** (2017), http://arxiv.org/abs/1705.08439
2. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1998)
3. Bansal, K., Loos, S.M., Rabe, M.N., Szegedy, C., Wilcox, S.: HOList: An environment for machine learning of higher-order theorem proving (extended version). CoRR **abs/1904.03241** (2019), http://arxiv.org/abs/1904.03241
4. Barrett, C.W., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 305–343. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_11, https://doi.org/10.1007/978-3-319-10575-8_11
5. Bertot, Y., Castran, P.: Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions. Springer Publishing Company, Incorporated, 1st edn. (2010)
6. Bibel, W., Eder, E., Fronhöfer, B.: Towards an advanced implementation of the connection method. In: Bundy, A. (ed.) Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983. pp. 920–922. William Kaufmann (1983), http://ijcai.org/Proceedings/83-2/Papers/072.pdf
7. Bledsoe, W.W.: Some thoughts on proof discovery. In: Proceedings of the 1986 Symposium on Logic Programming, Salt Lake City, Utah, USA, September 22-25, 1986. pp. 2–10. IEEE-CS (1986)
8. Brock, B., Cooper, S., Pierce, W.: Analogical reasoning and proof discovery. In: Lusk, E., Overbeek, R. (eds.) 9th International Conference on Automated Deduction. pp. 454–468. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
9. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI gym. CoRR **abs/1606.01540** (2016), http://arxiv.org/abs/1606.01540
10. Bundy, A.: The use of explicit plans to guide inductive proofs. In: Lusk, E.L., Overbeek, R.A. (eds.) 9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 23-26, 1988, Proceedings. Lecture Notes in Computer Science, vol. 310, pp. 111–120. Springer (1988). https://doi.org/10.1007/BFb0012826, https://doi.org/10.1007/BFb0012826
11. Chvalovský, K., Jakubuv, J., Suda, M., Urban, J.: ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. CoRR **abs/1903.03182** (2019), http://arxiv.org/abs/1903.03182
12. Crouse, M., Abdelaziz, I., Makni, B., Whitehead, S., Cornelio, C., Kapanipathi, P., Srinivas, K., Thost, V., Witbrock, M., Fokoue, A.: A deep reinforcement learning approach to first-order logic theorem proving. arXiv: Artificial Intelligence (2019)
13. Harrison, J.: HOL Light: A tutorial introduction. In: Srivas, M.K., Camilleri, A.J. (eds.) Formal Methods in Computer-Aided Design, First International Conference, FMCAD '96, Palo Alto, California, USA, November 6-8, 1996, Proceedings. LNCS, vol. 1166, pp. 265–269. Springer (1996). https://doi.org/10.1007/BFb0031814, https://doi.org/10.1007/BFb0031814
14. Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: Stable baselines. https://github.com/hill-a/stable-baselines (2018)
15. Jakubův, J., Chvalovský, K., Olšák, M., Piotrowski, B., Suda, M., Urban, J.: Enigma anonymous: Symbol-independent inference guiding machine (system description).

In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning. pp. 448–463. Springer International Publishing, Cham (2020)

16. Jakubuv, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10383, pp. 292–302. Springer (2017). https://doi.org/10.1007/978-3-319-62075-6_20, https://doi.org/10.1007/978-3-319-62075-6_20
17. Jakubuv, J., Urban, J.: Hammering Mizar by learning clause guidance. In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA. LIPIcs, vol. 141, pp. 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019), https://doi.org/10.4230/LIPIcs.ITP.2019.34
18. Kaliszyk, C., Urban, J.: FEMaLeCoP: Fairly efficient machine learning connection prover. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, 2015, Proceedings. LNCS, vol. 9450, pp. 88–96. Springer (2015). https://doi.org/10.1007/978-3-662-48899-7_7, https://doi.org/10.1007/978-3-662-48899-7_7
19. Kaliszyk, C., Urban, J., Michalewski, H., Olsák, M.: Reinforcement learning of theorem proving. In: NeurIPS. pp. 8836–8847 (2018)
20. Kaliszyk, C., Urban, J., Vyskočil, J.: Certified connection tableaux proofs for HOL Light and TPTP. In: Proceedings of the 2015 Conference on Certified Programs and Proofs. pp. 59–66. CPP '15, ACM (2015). https://doi.org/10.1145/2676724.2693176, http://doi.acm.org/10.1145/2676724.2693176
21. Kaliszyk, C., Urban, J., Vyskočil, J.: Efficient semantic features for automated reasoning over large theories. In: Yang, Q., Wooldridge, M. (eds.) Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 3084–3090. AAAI Press (2015)
22. Kalman, J.A.: A shortest single axiom for the classical equivalential calculus. Notre Dame Journal of Formal Logic **19**(1), 141–144 (1978). https://doi.org/10.1305/ndjfl/1093888216
23. Kinyon, M.K., Veroff, R., Vojtechovský, P.: Loops with abelian inner mapping groups: An application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics - Essays in Memory of William W. McCune. LNCS, vol. 7788, pp. 151–164. Springer (2013). https://doi.org/10.1007/978-3-642-36675-8_8, https://doi.org/10.1007/978-3-642-36675-8_8
24. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006. pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
25. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: CAV (2013)
26. Loos, S.M., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. In: 21st International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) (2017)
27. McCune, W., Wos, L.: Experiments in automated deduction with condensed detachment. In: Kapur, D. (ed.) Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings. Lecture Notes in Computer Science, vol. 607, pp. 209–223. Springer (1992). https://doi.org/10.1007/3-540-55602-8_167, https://doi.org/10.1007/3-540-55602-8_167

28. Melis, E.: Theorem proving by analogy - A compelling example. In: Pinto-Ferreira, C.A., Mamede, N.J. (eds.) Progress in Artificial Intelligence, 7th Portuguese Conference on Artificial Intelligence, EPIA '95, Funchal, Madeira Island, Portugal, October 3-6, 1995, Proceedings. Lecture Notes in Computer Science, vol. 990, pp. 261–272. Springer (1995). https://doi.org/10.1007/3-540-60428-6_22, https://doi.org/10.1007/3-540-60428-6_22
29. Melis, E., Siekmann, J.H.: Knowledge-based proof planning. Artif. Intell. **115**(1), 65–105 (1999). https://doi.org/10.1016/S0004-3702(99)00076-4, https://doi.org/10.1016/S0004-3702(99)00076-4
30. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer-Verlag, Berlin, Heidelberg (2002)
31. Olšák, M., Kaliszyk, C., Urban, J.: Property invariant embedding for automated reasoning. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 1395–1402. IOS Press (2020). https://doi.org/10.3233/FAIA200244, https://doi.org/10.3233/FAIA200244
32. Otten, J., Bibel, W.: leanCoP: lean connection-based theorem proving. J. Symb. Comput. **36**, 139–161 (2003)
33. Paliwal, A., Loos, S.M., Rabe, M.N., Bansal, K., Szegedy, C.: Graph representations for higher-order logic and theorem proving. CoRR **abs/1905.10006** (2019), http://arxiv.org/abs/1905.10006
34. Peterson, J.G.: Shortest single axioms for the classical equivalential calculus. Notre Dame J. Formal Log. **17**(2), 267–271 (1976). https://doi.org/10.1305/ndjfl/1093887534, https://doi.org/10.1305/ndjfl/1093887534
35. Piotrowski, B., Urban, J.: Guiding inferences in connection tableau by recurrent neural networks. In: Benzmüller, C., Miller, B.R. (eds.) Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12236, pp. 309–314. Springer (2020). https://doi.org/10.1007/978-3-030-53518-6_23, https://doi.org/10.1007/978-3-030-53518-6_23
36. Polu, S., Sutskever, I.: Generative language modeling for automated theorem proving. CoRR **abs/2009.03393** (2020), https://arxiv.org/abs/2009.03393
37. Polya, G.: Mathematics and Plausible Reasoning, Volume 1: Introduction and Analogy in Mathematics. Princeton University Press (1954)
38. Polya, G.: How to Solve It. Princeton University Press (November 1971), http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0691023565
39. Rawson, M., Reger, G.: lazycop 0.1. EasyChair Preprint no. 3926 (EasyChair, 2020)
40. Resnick, C., Raileanu, R., Kapoor, S., Peysakhovich, A., Cho, K., Bruna, J.: Backplay: "Man muss immer umkehren". CoRR **abs/1807.06919** (2018), http://arxiv.org/abs/1807.06919
41. Robinson, A., Voronkov, A. (eds.): Handbook of Automated Reasoning. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands (2001)
42. Robinson, R.M.: An essentially undecidable axiom system. Proceedings of the International Congress of Mathematics pp. 729–730 (1950)

43. Ross, S., Gordon, G., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Gordon, G., Dunson, D., Dudik, M. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 15, pp. 627–635. PMLR, Fort Lauderdale, FL, USA (11–13 Apr 2011), http://proceedings.mlr.press/v15/ross11a.html
44. Salimans, T., Chen, R.: Learning Montezuma's Revenge from a single demonstration. CoRR **abs/1812.03381** (2018), http://arxiv.org/abs/1812.03381
45. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017)
46. Schulz, S.: System Description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Proc. of the 19th LPAR. LNCS, vol. 8312. Springer (2013)
47. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of go without human knowledge. Nature **550**, 354– (Oct 2017), http://dx.doi.org/10.1038/nature24270
48. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)
49. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (2018), http://incompleteideas.net/book/the-book-2nd.html
50. Urban, J.: MaLARea: a Metasystem for Automated Reasoning in Large Theories. In: Sutcliffe, G., Urban, J., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, Bremen, Germany, 17th July 2007. CEUR Workshop Proceedings, vol. 257. CEUR-WS.org (2007), http://ceur-ws.org/Vol-257/05_Urban.pdf
51. Urban, J., Jakubuv, J.: First neural conjecturing datasets and experiments. In: Benzmüller, C., Miller, B.R. (eds.) Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12236, pp. 315–323. Springer (2020). https://doi.org/10.1007/978-3-030-53518-6_24, https://doi.org/10.1007/978-3-030-53518-6_24
52. Urban, J., Sutcliffe, G., Pudlák, P., Vyskocil, J.: MaLARea SG1- machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. LNCS, vol. 5195, pp. 441–456. Springer (2008). https://doi.org/10.1007/978-3-540-71070-7_37, https://doi.org/10.1007/978-3-540-71070-7_37
53. Veroff, R.: Using hints to increase the effectiveness of an automated reasoning program: Case studies. J. Autom. Reasoning **16**(3), 223–239 (1996). https://doi.org/10.1007/BF00252178, https://doi.org/10.1007/BF00252178
54. Wos, L., Winker, S., Smith, B., Veroff, R., Henschen, L.: A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains. Artificial Intelligence **22**(3), 303 – 356 (1984). https://doi.org/https://doi.org/10.1016/0004-3702(84)90054-7, http://www.sciencedirect.com/science/article/pii/0004370284900547
55. Wos, L.: Meeting the challenge of fifty years of logic. J. Autom. Reason. **6**(2), 213–232 (1990). https://doi.org/10.1007/BF00245821, https://doi.org/10.1007/BF00245821
56. Zombori, Zs., Urban, J., Brown, C.E.: Prolog technology reinforcement learning prover. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning. pp. 489–507. Springer International Publishing, Cham (2020)