Proof Searching in PVS Theorem Prover Using Simulated Annealing

M. Saqib Nawaz[1], Meng Sun[2], and Philippe Fournier-Viger[1(✉)]

[1] School of Humanities and Social Sciences,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
{msaqibnawaz, philfv}@hit.edu.cn
[2] School of Mathematical Sciences, Peking University, Beijing, China
sunmeng@math.pku.edu.cn

Abstract. The proof development process in PVS theorem prover is interactive in nature, that is not only laborious but consumes lots of time. For proof searching and optimization in PVS, a heuristic proof searching approach is provided where simulated annealing (SA) is used to search and optimize the proofs for formalized theorems/lemmas in PVS theories. In the proposed approach, random proof sequence is first generated from a population of frequently occurring PVS proof steps that are discovered with sequential pattern mining. Generated proof sequence then goes through the annealing process till its fitness matches with the fitness of the target proof sequence. Moreover, the performance of SA with a genetic algorithm (GA) is compared. Obtained results suggest that evolutionary/heuristic techniques can be combined with proof assistants to efficiently support proofs finding and optimization.

Keywords: PVS · Simulated annealing · Genetic algorithm · Proof searching · Proof sequences

1 Introduction

Interactive theorem provers (ITPs), also known as proof assistants, such as PVS [1], Coq [2] and HOL4 [3], follow a user driven proof development process. In ITPs, the user guides the proof process by providing the proof goal (theorem or lemma) and by applying proof commands and decision procedures to prove the goal. Generally, one needs to do lots of repetitive work to prove a nontrivial goal, which is laborious and consumes a large amount of time. Thus, proof guidance, proof searching and automation are some extremely desirable features in ITPs.

For proof guidance, a sequential pattern mining (SPM)-based proof process learning approach was proposed [4] for PVS. The approach can be used to find frequent proof steps/patterns that are used in the proofs, their relationships with each other, dependency of new proofs on already proved facts and to predict the next proof step(s). A high-utility itemset mining (HUIM)-based proof process learning approach was also proposed [5] for PVS. Moreover, an evolutionary approach was proposed [6], where a genetic algorithm (GA) with different crossover

Y. Tan and Y. Shi (Eds.): ICSI 2021, LNCS 12690, pp. 253–262, 2021.
https://doi.org/10.1007/978-3-030-78811-7_24

and mutation operators was used for proof searching and optimization in theories available in HOL4 library. This proof searching approach was quite efficient in evolving random proofs. However, there is still room to develop alternative proof searching approaches.

Based on previous studies [4,6], a heuristic-based approach is designed in this paper for proof searching and optimization in PVS theories. The basic idea is to use Simulated Annealing (SA) [7] for proof searching, where an initial population (a set of potential solutions) is first created from frequent PVS proof steps that are discovered with the approach in [4]. A random proof sequence is then generated from the population and is evolved with the annealing process. The annealing process continues till the fitness of the random proof sequence matches that of the original (target) proof for formalized theorems and lemmas. Moreover, the performance of the proposed SA is compared with GA [6] for various parameters. It is found that SA outperforms GA for proof finding and optimization.

In literature, evolutionary algorithms have been used in ITPs [8–10]. However, the approach in [8,9] for Coq can only be used to successfully find the proofs of easy theorems (that contain less number of proof steps). Similarly, the approach in [10] represented Isabelle's proofs with a tree structure, which are linearized. However, linearization leads to the loss of important connections (information) between different branches of the proofs due to which interesting patterns and tactics may be lost in the evolution process. A recent work [11] briefly discussed the applicability of evolutionary computation in improving the heuristics of automatic proof search in Isabelle/HOL theorem prover. Moreover, the framework based on GA in [12] can be used to find good search heuristics in the automated theorem prover (ATP) E. The dataset for the proof sequences in this work contains all the essential information needed to discover frequent PVS proof steps, which are used for the generation of initial population. Moreover, the SA-based proof searching approach can handle formalized proof goals of various length.

The remainder of this paper is organized as follows. Section 2 provides the proof searching approach based on SA for PVS theorem prover. Obtained results are discussed in Sect. 3, followed by the conclusion in Sect. 4.

2 Proof Searching in PVS with SA

The proof development process in PVS is interactive in nature and it follows the *sequent-style proof representation* [13]. A user first provides the property (in the form of a lemma or theorem) that is called a proof goal. Each proof goal is a *sequent* consisting of a sequence of formulas called *antecedents* (hypotheses) and a sequence of formulas called *consequents* (conclusions).

User then applies proof commands, inference rules and decision procedures to solve the proof goal. The action resulting from a proof command, inference rule or decision procedure is referred to as a PVS proof step (*PPS*) here. A *PPS* may either prove the goal or generates another sequent or divide the main goal into

sub-goals. The proof development process for a theorem or lemma is completed when the sequent or all the sub-goals are proved.

After proof development, PVS saves the proof scripts of a theory in a separate proof file. These files contain *PPS* with some other information related to PVS. Inside a theory, a particular proof goal for a theorem or lemma depends on the specifications and it can be completed by applying the *PPS* in different orders. This makes it difficult to automatically find the proof for a goal or to carry out a brute force or pure random search approach for proof searching. However, evolutionary and heuristic algorithms have the potential to search for the proofs of theorems/lemmas due to their ability to handle black-box search and optimization problems. The schematic of using SA for proof searching in PVS is shown in Fig. 1.

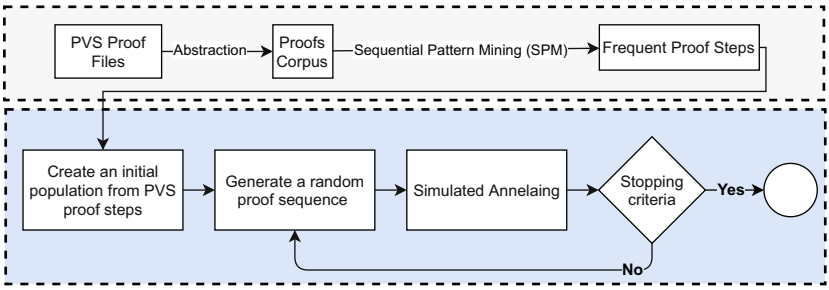

Fig. 1. Proof searching in PVS using SA

The data available in PVS proof files is first converted into a proper computational format so SA algorithms can be used. Moreover, the redundant information (related to PVS) that plays no part in proof searching and evolution is removed from the proof files. The complete proof for a goal can now be considered as a sequence of *PPS*. Let $PS = \{PPS_1, PPS_2, \ldots PPS_m\}$ represent the set of *PPS* proof steps. Let the notation $|PSS|$ denotes the set cardinality. For example, consider that $PS = \{$*skeep, expand, flatten, typepred, split, beta, iff, assert*$\}$. The set $\{$*skeep, expand, flatten, assert*$\}$ is a proof step set that contains four proof steps. A proof sequence is a list of proof step sets $S = \langle PSS_1, PSS_2, ..., PSS_n \rangle$, such that $PSS_i \subseteq PSS$ $(1 \leq i \leq n)$. For example, $\langle \{$*skeep, flatten*$\}, \{$*typepred*$\}, \{$*split, beta, iff, assert*$\}\rangle$ is a proof sequence which has three *PSS* and seven *PPS* that are used to prove a proof goal. A *proof dataset PD* is a list of proof sequences $PD = \langle S_1, S_2, ..., S_p \rangle$, where each sequence has an identifier (ID). For example, Table 1 shows a *PD* that has five proof sequences.

Algorithm 1 presents the pseudocode of the SA that is used to find the proofs in PVS theories. An initial population (*Pop*) is first created from frequent *PPS* (*FPPS*) that are discovered with various SPM techniques. In data mining, SPM can be used to find useful and interesting patterns (information) that are generally hidden in large corpora of sequential data [14]. From population, one

Table 1. A sample proof dataset

ID	Proof sequence
1	⟨{*inst , grind*}⟩
2	⟨{*skosimp, expand, flatten, assert*}⟩
3	⟨{*skosimp, expand, propax*}⟩
4	⟨{*skeep, expand, typepred, expand, flatten, expand, inst , assert*}⟩
5	⟨{*induct*},{*expand, propax*},{*skosimp, expand, assert*}⟩

random proof sequence (PS) is generated. Random proof sequences goes through the annealing process (Steps 9–25 in Algorithm 1), where it is evolved till its fitness is equal to the fitness of the target proof sequence from PD. Besides annealing process, the algorithm consists of two main procedures: (1) *Fitness*, and (2) *Get_ Neighbor* (GN).

Fitness values guide the SA toward the best solution(s) (proof sequences here). For our case, the fitness value is the total number of PPS in the random proof sequence that matches the PPS in the position of the original (target) proof sequence. Algorithm 2 presents the procedure for calculating the fitness value of a proof sequence. This procedure evaluates how close a given solution is to the optimum solution (in our case, the target solution).

The fitness procedure compares each gene i of a random proof sequence ($Pseq$) with the genes of the target (P). The fitness of $PSeq$ is set to 0, and increased by 1 for each matching gene and if genes in both sequences are equal then fitness of 1 is assigned. For example, consider the random proof sequence (RP) and the target sequence (TP) are:

$$RP = skosimp, expand, lemma, grind, flatten, assert, skolem, assert$$
$$TP = skeep, expand, typepred, expand, flatten, expand, inst, assert$$

The $Fitness$ procedure will return 3 as three PPS are same in both sequences (at positions 2, 5 and 8 respectively).

In the annealing process, a neighbor random sequence is first generated. Algorithm 3 presents the procedure for getting the neighbor solution. The selected location value is changed from its original value in *Get_ Neighbor*. For a proof sequence, a randomly chosen genes value i is replaced by a random PPS from the current population Pop. For example, a neighbor of the proof sequence RP is:

$$RP' = skosimp, expand, lemma, grind, flatten, \boldsymbol{inst}, skolem, assert$$

It is important to point out here that the standard mutation operator of GA and the get neighbor procedure in SA are same. The fitness of the neighbor solution is then calculated. The random generated proof sequence and the neighbor sequence is compared. If the fitness of the neighbor is better, then it is selected. Otherwise, an acceptance rate (Step 19 in Algorithm 1) is used to select one

Algorithm 1. SA proof searching

Input: *FPPS*: Frequent PVS proof steps, *PD*: proof sequences database, Temp, Temp_min, α

Output: Generated proof sequences

1: $Pop \leftarrow FPPS$
2: **for each** $P \in PD$ **do**
3: $OF \leftarrow Fitness(P, P)$
4: $PS \leftarrow$ randomseq(Pop, length(P))
5: $BF \leftarrow Fitness(PS, P)$
6: **if** $BF \geq OF$ **then**
7: return PS
8: **end if**
9: **while** $(Temp > Temp\_min)$ **do**
10: $NS \leftarrow get\_neighbor(PS)$
11: $NF \leftarrow Fitness(NS, P)$
12: **if** $NF == OF$ **then**
13: return NS
14: **end if**
15: **if** $NF > BF$ **then**
16: $PS \leftarrow NS$
17: $BF \leftarrow NF$
18: **end if**
19: $ar \leftarrow exp(\frac{T}{1+T})$
20: **if** $ar >$ randomuniform$(0, 10)$ **then**
21: $PS \leftarrow NS$
22: $BF \leftarrow NF$
23: **end if**
24: $Temp \leftarrow Temp \times \alpha$
25: **end while**
26: return PS
27: **end for**

sequence from the two sequences. The acceptance rate depends on temperature. In last, the temperature *Temp* is decreased with the following formula:

$$Temp = Temp \times \alpha \tag{1}$$

where the value of α is in the range of $0.8 < \alpha < 0.99$. The process of annealing is repeated (Steps 9–24 in Algorithm 1) until the random proof sequence fitness matches with the target proof sequence or *Temp* reaches to the minimum (*Temp_ min*). In our case, we set the value of *Temp* such that the SA always terminates when the random proof sequence matches with the target proof sequence. The process that distinguishes SA from GA is the annealing process.

Besides the annealing process, another main concept in SA is the acceptance probability. SA checks whether the new neighbor solution is better than the present best solution. If the new neighbor solution is worse than the present best solution, SA may still select the news solution with probability known as

Algorithm 2. Fitness

Input: *Pseq*: A proof sequence, *P*: The current target proof sequence
Output: Integer that represents the fitness of a proof sequence (*Pseq*)

```
1: procedure FITNESS(Pseq, P)
2:     i, f ← 0
3:     while (i ≤ length(Pseq) - 1) do
4:         if (Pseq[i] = P[i]) then
5:             f ← f + 1
6:         end if
7:         i ← i + 1
8:     end while
9:     return f
10: end procedure
```

Algorithm 3. Get Neighbor

Input: P_1: A proof sequence
Output: A neighbor of P_1

```
1: procedure GN(P1)
2:     ind ← randomint(1, length(P1))
3:     alter ← randomsample(Pop, 1)        ▷ (1-length proof sequence form Pop)
4:     P1[ind] ← alter                     ▷ (P1[ind] ≠ alter)
5:     return P1
6: end procedure
```

acceptance probability. *Acceptance probability* recommends whether to switch to the worst solution or not. This enable SA to avoid the local optimum (maybe) by exploring other solutions. This idea may seem unacceptable or worse at the start, but it could lead SA towards the global optimum. In the proposed SA for proof searching, the *acceptance probability* is calculated by using acceptance rate (AR) formula, that is:

$$AR = exp(\frac{Temp}{1 + Temp}) \tag{2}$$

3 Results and Discussion

The proposed SA for proof searching is implemented in Python. To evaluate the proposed approach, experiments were carried on a computer equipped with a fifth generation Core *i5* processor and 4 GB of RAM. For all experiments with SA, the values for *temp*, *min_temp* and α were set to 50,000, 0.0002, and 0.9954 respectively. Moreover, AR value (using Eq. 2) is compared with a random number generated within the range (2.71825400004040, 2.71825464604849). This range is selected after experimenting with the values of *temp*, *min_temp* and α.

We investigate the performance of the proposed SA for finding the proofs of theorems, lemmas and proof obligations in two PVS theories [15,16]. In total, we have 41 proof sequences and 23 distinct *PPS* in the *PD*. The SA was run ten times on considered theorems and obtained results are listed in Table 2. As discussed

in previous section, SA can select the new solution (proof sequence in this work) obtained with the *GN* procedure that is not better than the present solution with the *acceptance probability*. The reason for this is that there is always a possibility that the new solution could lead the SA to the global optimum. In SA, we chose the *acceptance probability* (AR) using Eq. (2), which is then compared with a random number generated within the range (2.71825400004040, 2.71825464604849). If the value of AR is greater than the random number generated within the range, then the new solution (that is not better than present) is selected. We used a counter called acceptance rate counter (ARC) that keeps track of how many times the new neighbor solution that is not better than the present solution is picked. By simulation, it was found that this factor does not play huge role in the overall generation count or time. This is due to the fact that in our case, we do not have any local optimum. SA finds only one global solution for each random proof sequence based on the fitness value. The average generation count of SA for all proof sequences in the *PD* with and without the acceptance rate is listed in Table 2. The obtained result for ARC indicates that SA selected neighbor solution that is not better than previous solution 565 times.

Table 2. Performance of SA for proof searching in PVS theories

Results without AR		Results with AR ($ARC = 565$)	
Avg. Gen. Count	Time (S)	Avg. Gen. Count	Time (S)
41,569	0.71 s	42,134	0.72 s

Recently, a GA with different crossover and mutation operators was used for proof searching and optimization in HOL4 proof assistant [6]. Same like SA, an initial population for GA was first created using the SPM-based learning approach [4]. Random proof sequences from the population were then evolved by applying two GA operators (crossover and mutation). Both operators randomly evolve the random proof sequences by shuffling and changing the proof steps at particular points. This process of crossover and mutation continues till the fitness of random proof sequences matches the fitness of original (target) proofs for the different theorems and lemmas. Three crossover operators (single point crossover (SPC), multi point crossover (MPC) and uniform Crossover (UCO)) and two mutation operators (standard mutation (SM) and modified pairwsie interchange mutation (MPIM)) were used. The reason to use more than one crossover and mutation operators was to compare their effect on the overall performance of GAs in proof searching. We modified the GA [6] and run it on *PD* to compare its performance with SA. The average generations for the SA and GA with different crossover and mutation operators to reach the target proof sequences in the whole dataset (*PD*) are shown in Table 3. The time taken by algorithms and memory used is also listed in Table 3.

GA with different crossover and *MPIM* operator is approximately five times faster than the GA with different crossover operator and *SM*. Whereas, SA is

four times faster than GA with *MPIM* and different crossover operators. One of the possible reason for this is that in SA, one procedure (*GN*) is called. On the other hand, in GA, two procedures (crossover and mutation) are called.

Table 3. Average total generation count for SA and GA

	Avg. generation count	Time	Memory
SA	41,569	0.71 s	3,614 Mb
GA (SPC/SM)	902,772	21.06 s	3,555 Mb
GA (MPC/SM)	851,609	20.67 s	4,085 Mb
GA (UCO/SM)	916,746	21.78 s	3,660 Mb
GA (SPC/MPIM)	182,679	9.85 s	3,617 Mb
GA (MPC/MPIM)	177,697	9.21 s	3,579 Mb
GA (UCO/MPIM)	173,055	9.05 s	3,618 Mb

In last, non-parametric statistical tests are used to compare the performance of evolutionary/heuristic-based proof searching approaches. First, the Friedman test [17] is used to statistically compare SA with different versions of GA for proof searching in PVS. The following hypotheses are tested:

H0*: The means for the generations taken by the proof searching approaches for each proof sequence in* PD *are same.*
H1*: The mean number of generations for at least one algorithm is different from the others.*

The results of the Friedman test indicate that the null hypothesis ($H0$) should be rejected and $H1$ should be accepted ($p < 0.05$) with result of 187.41. The Wilcoxon test [18] is further used to investigate SA and GAs more. The Wilcoxon test results for the number of generations for the two algorithms are listed in Table 4. The results in table show that SA is significantly better than different versions of GA. It can bee seen that GA with SM and different crossover operators (GA (SPC/SM, MPC/SM and UCO/SM)) are not significantly better than each other. The same is the case with GA with MPIM and different crossover operators (GA (SPC/MPIM, MPC/MPIM and UCO/MPIM)). Combining these statistical results with previous results suggests that SA is indeed significantly better than GAs. Moreover, the Wilcoxon test results on the means for the times taken by the SA and GAs to find the correct proofs for target proof sequences in the *PD* also indicated that SA is significiantly better than different versions of GA.

Overall, it was observed through various experiments that the SA is able to quickly optimize and automatically find the correct proofs for formalized theorems/lemmas in PVS theories. Obtained results in this paper are also consisted with the results for proof searching in HOL4 with SA [19,20]. This clearly indicates that the developed approaches is not limited to only one proof assistant and can be used easily and effectively in different proof assistants, such as PVS, Coq, HOL4, Isabelle/HOL etc.

Table 4. Wilcoxon p-value matrix for generations

	SPC/SM	MPC/SM	UC/SM	SPC/MPIM	MPC/MPIM	UC/MPIM	SA
SPC/SM	—	4.88E−1	7.90E−1	2.61E−8	2.42E−8	2.42E−8	2.42E−8
MPC/SM	4.88E−1	—	4.88E−1	2.42E−08	2.42E−8	2.81E−8	2.42E−8
UC/SM	7.90E−1	4.88E−1	—	2.61E−8	2.42E−8	2.42E−8	2.42E−8
SPC/MPIM	2.61E−8	2.42E−08	2.61E−8	—	8.91E−1	4.64E−1	1.02E−5
MPC/MPIM	2.42E−8	2.42E−8	2.42E−8	8.91E−1	—	6.30E−1	6.05E−7
UC/MPIM	2.42E−8	2.81E−8	2.42E−8	4.64E−1	6.30E−1	—	2.46E−7
SA	2.42E−8	2.42E−8	2.42E−8	1.02E−5	6.05E−7	2.46E−7	—

4 Conclusion

Developing proofs for unproved theorems and lemmas (also known as conjectures) in PVS requires our interaction with the proof assistant. For long and complex proofs, the proof development process becomes a cumbersome and time consuming activity. This paper provides a proof searching approach, where a SA is used to optimize and find the correct proofs in PVS theories. Moreover, the performance of SA was compared with GA for proof searching and optimization. Obtained results suggests that SA was faster than GA and also indicate that the research direction of linking and integrating evolutionary and heuristic algorithms with proof assistants is worth pursuing.

There are several directions for future work. For example, making the proof searching process more general in nature to evolve PVS frequent proof steps to a compound proof strategies for guiding the proofs of new conjectures. Moreover, creating a large dataset by including more formalized theorems/lemmas for other PVS theories. We believe that such dataset can be used for the development of learning environments to search for proofs and to automate the interaction in different ITPs. In this regard, some work has been done in [21–24]. Last but not the least, by exploiting Curry-Howard isomorphism for sequent calculus [25], a SA or GA can be used for program (proof) writing and proof assistant can be used for simplification and verification by computationally evaluating the program, that represents the corresponding proof.

References

1. Owre, S., et al.: PVS system guide, PVS prover guide. PVS language reference. Technical report, SRI International, USA (2001)
2. Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Construction. Springer (2003). https://doi.org/10.1007/978-3-662-07964-5
3. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71067-7_6
4. Nawaz, M.S., Sun, M., Fournier-Viger, P.: Proof guidance in PVS with sequential pattern mining. In: Hojjat, H., Massink, M. (eds.) FSEN 2019. LNCS, vol. 11761, pp. 45–60. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31517-7_4

5. Nawaz, M.S., et al.: Proof learning in PVS with utility pattern mining. IEEE Access **8**, 119806–119818 (2020)
6. Nawaz, M.Z., et al.: Proof searching in HOL4 with genetic algorithm. In: Proceedings of SAC, pp. 513–520. ACM (2020)
7. Delahaye, D., Chaimatanan, S., Mongeau, M.: Simulated annealing: from basics to applications. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. ISORMS, vol. 272, pp. 1–35. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_1
8. Huang, S.I., Chen, Y.P.: Proving theorems by using evolutionary search with human involvement. In: Proceedings of CEC, pp. 1495–1502. IEEE (2017)
9. Yang, L.A., et al.: Automatically proving mathematical theorems with evolutionary algorithms and proof assistants. In: Proceedings of CEC, pp. 4421–4428. IEEE (2016)
10. Duncan, H.: The use of data-mining for the automatic formation of tactics. PhD thesis, University of Edinburgh, UK (2007)
11. Nagashima, Y.: Towards evolutionary theorem proving for Isabelle/HOL. In: Proceedings of GECCO (Poster), pp. 419–420. ACM (2019)
12. Schafer, S., Schulz, S.: Breeding theorem proving heuristics with genetic algorithms. In: Proceedings of GCAI, pp. 263–274. EasyChair (2015)
13. Nawaz, M.S., et al.: A survey on theorem provers in formal methods. arXiv:cs.SE/1912.03028 (2019)
14. Fournier-Viger, P., et al.: A survey of sequential pattern mining. Data Sci. Patt. Recogn. **1**(1), 54–77 (2017)
15. Nawaz, M.S., Sun, M.: Reo2PVS: formal specification and verification of component connectors. In: Proceedings of SEKE, pp. 391–396 (2018)
16. Nawaz, M.S., Sun, M.: Using PVS for modeling and verification of probabilistic connectors. In: Hojjat, H., Massink, M. (eds.) FSEN 2019. LNCS, vol. 11761, pp. 61–76. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31517-7_5
17. Friedman, M.: A comparison of alternative tests of significance for the problem of m rankings. Ann. Math. Stat. **11**(1), 86–92 (1940)
18. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bull. **1**(6), 80–83 (1945)
19. Nawaz, M.S., et al.: Proof searching and prediction in HOL4 with evolutionary/heuristic and deep learning techniques. Appl. Intell. **51**(3), 1580–1601 (2021)
20. Nawaz, M.S., et al.: An evolutionary/heuristic-based proof searching framework for interactive theorem prover. Appl. Soft Comput. (2021). https://doi.org/10.1016/j.asoc.2021.107200
21. Huang, D., et al.: GamePad: a learning environment for theorem proving. In: Proceedings of ICLR (Poster) (2019)
22. Yang, K., Deng, J.: Learning to prove theorems via interacting with proof assistants. In: Proceedings of ICML, pp. 6984–6994. PMLR (2019)
23. Bansal, K., et al.: HOList: an environment for machine learning of higher order logic theorem proving. In: Proceedings of ICML, pp. 454–463, PMLR (2019)
24. Sanchez-Stern, A., et al.: Generating correctness proofs with neural networks. In: Proceedings of MAPL@PLDI, pp. 1–10. ACM (2020)
25. Santo, J.E.: Curry-howard for sequent calculus at last! In: Proceedings of TLCA, pp. 165–179 (2015)