# FEMaLeCoP: Fairly Efficient Machine Learning Connection Prover

Cezary Kaliszyk[1⋆] and Josef Urban[⋆⋆2]

[1] University of Innsbruck, Austria
[2] Czech Technical University in Prague

**Abstract.** FEMaLeCoP is a connection tableau theorem prover based on leanCoP which uses efficient implementation of internal learning-based guidance for extension steps. Despite the fact that exhaustive use of such internal guidance can incur a significant slowdown of the raw inferencing process, FEMaLeCoP trained on related proofs can prove many problems that cannot be solved by leanCoP. In particular on the MPTP2078 benchmark, FEMaLeCoP adds 90 (15.7%) more problems to the 574 problems that are provable by leanCoP. FEMaLeCoP is thus the first AI/ATP system convincingly demonstrating that guiding the internal inference algorithms of theorem provers by knowledge learned from previous proofs can significantly improve the performance of the provers. This paper describes the system, discusses the technology developed, and evaluates the system.

## 1 Introduction: Guiding Search By Learned Relevance

Intelligent guidance of the proof search is crucial for automated theorem proving (ATP). While complete ATP calculi such as resolution, superposition, and tableau can in principle find a proof of arbitrary length and complexity, the practical strength of state-of-the-art ATP systems is nowhere near the performance of expert mathematicians in most of mathematical domains.

In particular, experiments over large formal mathematical libraries [15,2] show that current ATP calculi have practically no chance to find a more complicated proof in large-theory mathematics unless they are equipped with *external* axiom-selecting AI methods. Such AI methods are based on various ideas estimating the *relevance* of the axioms to the conjecture based on sufficiently descriptive *features* [9] of the axioms and conjectures. The strongest methods are based on *learning* such relevance from the large libraries of previous related proofs. This is not surprising for two reasons. First, mathematicians also gradually *learn* their problem-solving expertise. Second, the chances of completely manually specifying the most efficient proof-search algorithm for all mathematical domains and problems seem very low.

⋆ Supported by the Austrian Science Fund (FWF): P26201.
⋆⋆ Supported by NWO grant nr. 612.001.208 and ERC Consolidator grant nr. 649043 *AI4REASON*.

Despite the ability of the AI/learning methods to focus the proof search towards the most relevant axioms, the power of today's ATPs in most of mathematics is still very limited. The automatically found proofs typically do not go over 20 lines of formal proof-assistant code [8], and are usually easy for trained mathematicians. This limited power is due to fast blow-up of the internal ATP search, which is reminiscent of the blow-up incurred by off-the-shelf ATPs when left to struggle alone with a very large number of axioms.

The success of the axiom-selection AI/learning methods in curbing such search space motivates research in automated learning of smarter guidance of the *internal* search. In the MaLeCoP (Machine Learning Connection Prover) experiment [16] we have shown that in principle it is possible to significantly prune the internal search space of leanCoP (lean Connection Prover) [13] when guiding each extension step by an off-the-shelf machine learner trained on related leanCoP proofs. However, the speed of the guiding machine learner in MaLeCoP was impractically (about 1000 times) slower [16] than the raw leanCoP inferencing process, resulting in MaLeCoP's low real-time performance.

## 2 Contributions

In this work, we devise much stronger learning-based guidance for connection tableau by developing an AI/ATP system where the learning-based guidance is an optimized and tightly integrated part of the core inferencing algorithm and data structures. This in particular involves (i) developing very fast (*online* in the machine-learning terminology) methods for characterizing the current proof state on which the trained learner gives advice to the inferencing process, (ii) suitable modification and integration of a machine learner whose advising speed is comparable to the core deductive inference mechanisms, (iii) designing mechanisms that suitably combine the learning-based guidance with semantic/deductive pruning methods such as discrimination-tree indexing. The main nontrivial concern is to provide strong proof-state characterization and AI/learning methods for guiding the inference steps, while keeping the speed of such methods sufficiently high.

The rest of the paper is organized as follows. Section 3 briefly summarizes leanCoP, its recent OCaml implementation, and the MaLeCoP prototype, which are the basis for the current system. Then we describe the main techniques developed and used in FEMaLeCoP (Section 4). In Section 5 we show that the raw inference speed of the resulting AI/ATP system is reasonably high in comparison to unguided leanCoP, and that the system indeed adds 15.7% more MPTP2078 problems to the 574 problems provable by unguided leanCoP.

## 3 Background: leanCoP, MaLeCoP and OCAML-leanCoP

leanCoP [13] is an automated theorem prover implementing connected tableau search with iterative deepening, written very economically in Prolog by Otten. The reduction rule of the connection calculus is applied before the extension rule, and open branches are selected in a depth-first way. Additional inference rules and strategies include *regularity*, *lemmata*, and *restricted backtracking* (cut) [12]. Given the very compact implementation, leanCoP's performance is surprisingly high, regularly outperforming much larger ATPs such as Metis and even Prover9 in the CASC competition and in particular on problems coming from large formal libraries [10]. Its size/performance ratio makes leanCoP suitable for various experiments and integration with other systems. Two such offsprings of leanCoP relevant here are:

1. Its OCaml implementation (OCaml-leanCoP), which has been linked to the HOL Light LCF-style kernel, resulting in the currently strongest internal automation tactic for interactive theorem provers [10].
2. The MaLeCoP prototype [16], providing the original Prolog-based leanCoP with a communication link to an external learning system (the SNoW system [3]) which is trained on previous leanCoP proofs and guides the choice of the extension steps. A large cache and a number of meta-strategies (e.g., advising only when a large branching factor is encountered) were used to combine the (very) slow external advice with the (much) faster raw inference process. Large speed-ups in terms of the abstract time (number of inferences) were measured, however the system was still too slow to be usable in practice.

## 4 FEMaLeCoP

### 4.1 Consistent Clausification, Indexing, and Basic Calculus

The basis of FEMaLeCoP is the OCaml version of leanCoP. As in MaLeCoP, FEMaLeCoP starts by a consistent clausification (with relation to the symbols used) of the FOL problem. This is done by using content-based names for Skolem functions and for the names of the clauses (or rather for the *contrapositives* created from the clauses - see below). For example, formula `?[X]: p(X)` thus becomes `p('skolem(?[A]:p(A),1)')` (involving also variable normalization), and the name of this clause (contrapositive) is just its MD5 hash. Such consistent naming is essential for good recall of similar proof situations and their solutions from the previous problems.

As in leanCoP, the initial clauses and their literals are put into an indexing datastructure – the *lit* matrix. The *lit* matrix keeps all literals $L$ from all input clauses $C$, remembering the rest of the clause $(C - L)$. We call the entries in the *lit* matrix (i.e., the pairs $L$, $C - L$) *contrapositives.* Contrapositives are the main object of leanCoP's search. The *lit* indexing is used for fast Prolog-style unification of literals during the tableau search. While in leanCoP, the indexing of *lit* is done automatically by Prolog, FEMaLeCoP uses indexing by the toplevel predicate and optional discrimination-tree indexing of the literals.

The core theorem-proving function of leanCoP written in Prolog is shown below, with `Cla` being the open subgoal and `Path` being the active path. For simplicity, we omit here the code implementing regularity, lemmata, iterative deepening and restrictive backtracking. The main source of nondeterminism is obviously the tableau extension rule, and this is where we will apply the learning-based guidance.

```
%  prove(Cla,Path)
prove([Lit|Cla],Path) :-
        (-NegLit=Lit;-Lit=NegLit) ->
        (
          member(NegL,Path),
          unify_with_occurs_check(NegL,NegLit)
        ; % extension step
          lit(NegLit,NegL,Cla1,Grnd1),
          unify_with_occurs_check(NegL,NegLit),
          prove(Cla1,[Lit|Path])
        ),
        prove(Cla,Path).
prove([],_,).
```

### 4.2 Overview of the Learning-Based Guidance

We combine the above basic leanCoP algorithm with a learning-based system that advises the inference process. The interesting choices in such AI setup are *what* exactly should be advised, *how* should the advising algorithm work, and in particular *which features* (properties, characteristics) of the proof state are best for recalling similar past proof states that led (typically after some nontrivial search effort) to successfully solved problems and their solutions are thus more likely to lead to successful proof for the current proof state. All these questions open interesting research topics: for example one could advise selection of high-level problem-solving strategies rather than low-level reasoning steps, and the advising algorithm could be interleaved with a gradual computation of more and more advanced features. While such sophisticated designs will certainly be built in the future, our goal here is to develop good-enough first solutions that will show that learning-based guidance leads to significant improvement of the unguided leanCoP. The summary of the choices that we make is as follows:

**What is advised:** We advise the selection of clause for *every* tableau extension step. This means that each time there are multiple clauses (or rather contrapositives) that unify with the current goal, the advise system is called to estimate the candidates' chances (relevance) for leading to a proof. The candidates are then tried (backtracked over) in the order of their relevance. Advising every extension step in this way is quite extreme and ambitious. It requires that the advising system is comparably fast to the standard inference speed, because we cannot assume that there will always be enough previous proof information to completely avoid mistakes and subsequent backtracking.

**How we advise:** We use a fast custom OCaml implementation [8] of the naive Bayes algorithm that learns the association of the features of the proof states (see below) with the contrapositives that were used for the successful tableau extension steps in previous proofs. During each extension step the advising system computes the features of the active proof state, and orders the contrapositives by their estimated relevance for these proof-state features based on the contrapositive's performance on previous similar proof states. The exact computation of the *relevance* and *feature-based similarity* depends on the machine-learning algorithm used. The implementation details of our advising system and the related infrastructure are described below 4.3.

**Features used:** We characterize the proof state as a weighted vector of symbols and/or (possibly generalized) terms extracted from all the literals on the active path. We use frequency-based weighting of such features (the *inverse document frequency* – IDF scheme [6]) which has turned out to work very well in the related large-theory axiom-selection task [7], and we additionally experiment with a simple decay factor (using maximum) for the features depending on the distance of the path literals from the tip of the path. For example, given decay factor of 0.8 and a term feature "$1 + 2$" extracted independently from two path literals $L_1 : 1 + 2 = 3$ and $L_2 : 1 + 2 = 2 + 1$ with $L_1$ being the active goal and $L_2$ being its grandparent on the active path, the (non-IDF) weight of the feature "$1 + 2$" is $w(\text{“}1 + 2\text{”}) = max(0.8^0, 0.8^2) = 1$.

## 4.3 Learning-Based Advising System and Related Infrastructure

*Collecting training data:* First, the advising system needs to collect the training data. To achieve this, FEMaLeCoP stores the complete information about the proof by adding the `prf` argument to the `prove` function and returning and printing it when a proof is found. `prf` is a list of tuples (examples), each consisting of the current literal, the path, and the contrapositive used.

*Data indexing:* The printed `prf` format is very general and verbose, allowing experiments with different features and learning algorithms without re-running the ATP. When extracted from many proofs, the number of printed tuples can easily go over one million. For the naive-Bayes learning and advising we first turn this data by a special program (*hasher*) into an efficient datastructures optimized for the particular choice of features (constants and/or (generalized) subterms – both are used by default). For the particular choice of features hasher extracts the proof-state features from each example and maintains a hashtable `cn_pf_no` keeping for each contrapositive a map of its aggregated (weighted) proof-state feature frequencies. Additionally the following auxiliary data are maintained for fast IDF and naive-Bayes processing: `te_num` – the total number of training examples so far, `pf_no` – a hashtable from features to floats storing the (weighted) sum of occurrences of every feature in all the processed training examples, and `cn_no` – a hashtable storing the total number of occurrences for each contrapositives in all training examples. This data extraction is fast, taking about 30 s for 10000 FEMaLeCoP proofs. Additionally, this also works incrementally, i.e. when a new proof is found, this aggregated information can be very quickly updated by the new training examples.

*Problem-specific data preparation:* Upon start, FEMaLeCoP reads the problem to solve and the aggregated training data. The first task is to select only the parts of these data that are relevant for the current problem. For this, after the (consistent – Section 4.1) clausification the contrapositives and their features are extracted and used for filtering out unnecessary parts of the aggregated training data, resulting in the localized version of the aggregated data structures. The `cn_pf_no` and `cn_no` hashtables are then combined with the *lit* indexing (based on the toplevel predicate or using a discrimination tree) of contrapositives. This makes the aggregated previous proof-use information for each contrapositive accessible right when the contrapositive is accessed in the main `prove` function through the *lit* indexing. This typically allows reasonably fast computation of the naive-Bayes score of the contrapositives that are considered by the *lit* indexing.

An optional problem-specific data-filtering step is to use the k-nearest neighbor (k-NN) algorithm for further restriction of the relevant training data. If this is used, we first find the $k$ solved problems whose conjectures are (in the feature metric) closest to the current conjecture, and extract the training examples only from such problems. Such filtering introduces further parameters to optimize and is not yet used in the Evaluation (Section 5).

*Efficient approximate feature and relevance computation :* To avoid costly recomputation of the features of the path for each extension step, the `prove` function passes the proof-state features computed so far as an additional argument, and the feature vector is only updated incrementally when an extension step is to be performed. This means that the features may occasionally be approximate, because the substitutions performed with the literals of the path (line 6 of the simplified leanCoP algoprithm in Section 4.1) might not be taken into account. This optimization may lose some constant features (e.g., if induced by a unification at a reduce step), however it very significantly speeds up the advising. Given that the features of the current path are $f$, the relevance of the eligible contrapositives (pre-selected by the *lit* indexing) is then computed according to the following modified naive-Bayes score (used by us for axiom selection in [11]):

$$r(t,s) = \sigma_1 \ln t + \sum_{f \in (\overline{f} \cap \overline{s})} i(f) \ln \frac{\sigma_2 s(f)}{t} + \sigma_3 \sum_{f \in (\overline{f} - \overline{s})} i(f) + \sigma_4 \sum_{f \in (\overline{s} - \overline{f})} i(f) \ln(1 - \frac{s(f)}{t})$$

Here $t$ is the total number of times the contrapositive was used, $s$ is its aggregated feature vector, and $i$ is the vector of IDF weights of all features. The score function is parameterized by the following constants (chosen experimentally): $\sigma_1$ – weight of the total number of uses (default $= 2$), $\sigma_2$ – weight of the overlapping features (default $= 2$), $\sigma_3$ – weight of the path-only features (default $= -6$), $\sigma_4$ – weight of the contrapositive-only features (default $= -0.05$).

## 5 Evaluation

The system's main evaluation is done on the 2078 related problems coming from the MPTP2078 large-theory benchmark [1] exported from Mizar. This benchmark has two categories: large (chainy) problems containing many redundant axioms, and small (bushy) problems that contain only the axioms used explicitly in the Mizar proofs plus additional "background" formulas encoding the (typically typing) reasoning steps done by Mizar implicitly.

As explained in Section 1, in FEMaLeCoP we are interested in the problem of guiding the *internal* ATP search once the right axioms have been (approximately) chosen by one of today's reasonably good (external) AI systems used for axiom selection. This is why we evaluate FEMaLeCoP on the bushy (small) problems rather than on the chainy (large) ones. Because the external axiom-selectors are not perfect, it makes sense to evaluate FEMaLeCoP on problems that still contain some redundant axioms, rather than evaluating it on problems where the set of axioms is minimized in some way (see [1] for some discussion of the minimization techniques and issues). The MPTP2078 bushy problems fit this evaluation scenario quite well, because the "background" formulas included in the problems are typically quite redundant [1].

The results are show in Table 1. Unaided OCaml-leanCoP is first run on all the 2078 bushy problems with a time limit of 60 s.[3] This solves 574 problems. From the proofs of these problems we collect the training data from the successful path decisions and preprocess them as described above. This step is done once for all proofs and takes seconds. In the second round we run FEMaLeCoP with these training data loaded, again with a time limit of 60 s, again attacking all the 2078 problems. While the inference speed drops to about 40% (for a sample problem: 305098 inferences per second instead of 772208) of the unadvised OCaml-leanCoP, the advised system solves 635 problems, adding 90 (15.7% more) problems to the original solutions. This is a considerable improvement of the ATP performance. As the union gives 664 solved problems, a portfolio approach might also prove to be effective.

Table 1: OCaml-leanCoP and trained FEMaLeCoP on bushy problems in 60 s.

| Prover | Proved (%) |
|---|---|
| OCaml-leanCoP | 574 (27.6%) |
| FEMaLeCoP | 635 (30.6%) |
| together | 664 (32.0%) |

## 6 Conclusion and Future Work

To the best of our knowledge, FEMaLeCoP is the first ATP system with efficiently integrated internal learning-based guidance that convincingly shows the feasibility and benefits of such exhaustive knowledge re-use when compared to the standard unguided ATP. While the MaLeCoP prototype has provided evidence that large pruning of the ATP search space is possible *in principle* when using such internal guidance, FEMaLeCoP shows that this is possible *in practice*, adding 15.7% solutions to unguided OCaml-leanCoP in a fair evaluation scenario.

We believe that this is a rather important step towards producing smart integrated AI/ATP systems that do not try to attack each problem in complete isolation, but instead re-use the vast problem-solving knowledge accumulated in the formal ITP libraries by human mathematicians and machines. The immediate future work includes similar modification of more complicated state-of-the-art

[3] The hardware used is Intel Xeon E7-4870 2.30GHz with 256GB RAM.

ATP systems based on resolution/superposition, developing better proof-state features, more general learning setups, and combining with external axiom selection. For example, while the current learning is done on the (MD5) names of normalized contrapositives, better transfer of knowledge (and thus recall) will likely be achieved by abstracting away the symbol names, and advising also the resulting abstract clause patterns [14,4]. Integrated machine learning could also be used to reorder subgoals [5]. Similarly, it seems straightforward to modify FEMaLeCoP for learning and advising the choice of higher-level tactics in ITP systems.

## References

1. J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.
2. J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *J. Formalized Reasoning*, 2015. in press.
3. A. Carlson, C. Cumby, J. Rosen, and D. Roth. The SNoW Learning Architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science, 1999.
4. T. Gauthier and C. Kaliszyk. Matching concepts across HOL libraries. In S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, editors, *CICM'15*, volume 8543 of *LNCS*, pages 267–281. Springer, 2014.
5. O. Ibens and R. Letz. Subgoal alternation in model elimination. In D. Galmiche, editor, *TABLEAUX'97*, volume 1227 of *LNCS*, pages 201–215. Springer, 1997.
6. K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 28:11–21, 1972.
7. C. Kaliszyk and J. Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series*, pages 87–95. EasyChair, 2013.
8. C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *J. Automated Reasoning*, 2015. in press.
9. C. Kaliszyk, J. Urban, and J. Vyskocil. Efficient semantic features for automated reasoning over large theories. In Q. Yang and M. Wooldridge, editors, *IJCAI'15*, pages 3084–3090. AAAI Press, 2015.
10. C. Kaliszyk, J. Urban, and J. Vyskočil. Certified connection tableaux proofs for HOL Light and TPTP. In X. Leroy and A. Tiu, editors, *Proc. of the 4th Conference on Certified Programs and Proofs (CPP'15)*, pages 59–66. ACM, 2015.
11. D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban. MaSh: Machine learning for Sledgehammer. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP 2013*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013.
12. J. Otten. Restricting backtracking in connection calculi. *AI Commun.*, 23(2-3):159–182, 2010.
13. J. Otten and W. Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.
14. S. Schulz. *Learning search control knowledge for equational deduction*, volume 230 of *DISKI*. Infix Akademische Verlagsgesellschaft, 2000.
15. J. Urban, K. Hoder, and A. Voronkov. Evaluation of automated theorem proving on the Mizar Mathematical Library. In *ICMS*, pages 155–166, 2010.
16. J. Urban, J. Vyskočil, and P. Štěpánek. MaLeCoP: Machine learning connection prover. In K. Brünnler and G. Metcalfe, editors, *TABLEAUX*, volume 6793 of *LNCS*, pages 263–277. Springer, 2011.