ML4PG: proof-mining in Coq[⋆]

Jónathan Heras and Ekaterina Komendantskaya

School of Computing, University of Dundee, UK
{jonathanheras,katya}@computing.dundee.ac.uk

Abstract. In this paper, we present ML4PG – a machine-learning extension to Proof General. ML4PG is an interactive tool that provides statistical proof hints during the process of Coq/SSReflect proof development. To this purpose, it automatically gathers statistics from proofs written in Coq/SSReflect. ML4PG clusters the gathered data using the state-of-the-art machine learning algorithms available in MATLAB and Weka, and post-processes the results to display families of related proofs to the user. We illustrate ML4PG's functionality with an example coming from the formalisation of a Computer Algebra algorithm using the CoqEAL library.
Keywords: Interactive Theorem Proving, Coq, SSReflect, Machine Learning, Clustering, CoqEAL.

1 Introduction

In recent years, development of interactive theorem provers (ITPs) has led to the creation of big data sets of libraries and varied infrastructures for formal mathematical proofs and software/hardware verification. However, these frameworks usually involve thousands of definitions and theorems. Then, it becomes difficult to trace reusable proof patterns across the libraries.

A research trend has recently emerged to deal with this problem: it applies machine-learning methods to help in the development of interactive proofs. Examples of using machine-learning techniques in ITPs concerned proof strategy discovery [1], hints in lemma generation [6], statistical tactic analysis [4] and speed-up in proof automation [7].

In this paper, we present ML4PG [5] – an extension to Proof General that uses state-of-the-art machine learning techniques to interactively find proof patterns from Coq/SSReflect proofs. As a running example to illustrate ML4PG's functionality, we use this tool to guide us in the formalisation of a fast algorithm to compute the inverse of triangular matrices using the CoqEAL methodology [3].

Availability. ML4PG is accessible from [5], where the reader can find related papers (currently under review), examples, the links to download ML4PG and all libraries and proofs we mention here.

[⋆] The work was supported by EPSRC grant EP/J014222/1.

2 The CoqEAL methodology

Most algorithms in modern Computer Algebra systems are designed to be efficient, and this usually means that their verification is not an easy task. In order to overcome this problem, a methodology based on the idea of *refinements* was presented in [3], and was implemented as a new library, built on top of the SSReflect libraries, called *CoqEAL*. The approach to formalise efficient algorithms followed in [3] can be split into three steps:

S1. define the algorithm relying on rich dependent types, as this will make the proof of its correctness easier;
S2. refine this definition to an efficient algorithm described on high-level data structures; and,
S3. implement it on data structures which are closer to machine representations.

The CoqEAL methodology is clear and the authors have shown that it can be extrapolated to different problems. Nevertheless, this library contains approximately 400 definitions and 700 lemmas; and the search of proof strategies inside this library is not a simple task if undertaken manually. Intelligent proof-pattern recognition methods could help with such a task.

In order to show this, let us consider the formalisation of a fast algorithm to compute the inverse of triangular matrices over a field with 1s in the diagonal using the CoqEAL methodology. SSReflect already implements the matrix inverse relying on rich dependent types using the `invmx` function; then, we only need to focus on the second and third steps of the CoqEAL methodology. We start defining a function called `fast_invmx` using high-level data structures.

Algorithm 1 Let M be a square triangular matrix of size n with 1s in the diagonal; then `fast_invmx(M)` is recursively defined as follows.

- If $n = 0$, then `fast_invmx(M)=1%M` (where `1%M` is the notation for the identity matrix in SSReflect).
- Otherwise, decompose M in a matrix with four components: the top-left element, which is 1; the top-right line vector, which is null; the bottom-left column vector C; and the bottom-right $(n-1) \times (n-1)$ matrix N; that is, $M = \left(\begin{array}{c|c} 1 & 0 \\ \hline C & N \end{array}\right)$. Then define `fast_invmx(M)` as:

$$\texttt{fast\_invmx(M)} = \left(\begin{array}{c|c} 1 & 0 \\ \hline -\texttt{fast\_invmx(N) *m C} & \texttt{fast\_invmx(N)} \end{array}\right)$$

  where `*m` is the notation for matrix multiplication in SSReflect.

Subsequently, we should prove the equivalence between the functions `invmx` and `fast_invmx` – Step S2 of the CoqEAL methodology. Once this result is proven, we can focus on the third step of the CoqEAL methodology. It is worth mentioning that neither `invmx` nor `fast_invmx` can be used to actually compute the inverse of matrices. These functions cannot be executed since the definition of matrices is locked in SSReflect to avoid the trigger of heavy computations during deduction steps. Using Step S3 of the CoqEAL methodology, we can overcome this pitfall. In our case, we implement the function `cfast_invmx` using lists of lists as the low level data type for representing matrices.

Let us focus on the proof of the equivalence, module a change of representation, of `fast_invmx` and `cfast_invmx` – this lemma is called `cfast_invmxP`. The suggestions provided by ML4PG when proving this result are related to Step S3 of the formalisation of the rank, the determinant and the fast multiplication of matrices algorithms. These lemmas follow the same proof strategy which can be applied in our concrete case.

Proof Strategy 1 Apply the *morphism lemma* to change the representation from abstract matrices to executable ones. Subsequently, apply the *translation lemmas* of the operations involved in the algorithm – translation lemmas are results which state the equivalence between the executable and the abstract counterparts of several operations related to matrices.

3 Applying ML4PG to the CoqEAL library

In the section, we show how ML4PG discovers the lemmas which follow Proof Strategy 1; to this aim, we provide a brief introduction to proof-pattern recognition with ML4PG.

The discovery, and acquisition, of statistically significant features in data is the first and, probably, the most critical step when using machine-learning methods – critical in the sense that a proper feature extraction algorithm is the basis to obtain sensible results when using machine-learning techniques, see [2].

ML4PG has its own method of feature extraction, called the *proof-trace method*. This method works on the background of Proof General automatically extracting some simple, low-level features from interactive proofs in Coq and SSReflect. In this way, ML4PG captures a potentially infinite variety of lemma shapes, by means of gathering a fixed number of statistical features.

Example 1. Table 1 shows a fragment of the proof for Lemma `cfast_invmxP`, and Table 2 shows the statistical features gathered from its proof according to the proof-trace method.

The proof-trace method lets the lemma structure "shows itself" through some simple statistics of the proof steps it induces. The two dimensions of tables like Table 2 gather statistics both dynamically (considering correlation of features within several proof steps) and relationally (tracking correlation of subgoal shapes and user actions).

Let us note that most pattern recognition tools require the numbers of features to be limited and fixed. However, Coq proofs may have varied length. To tackle this problem, we have developed a technique called *proof-patch* method, which extends the proof-trace method. The underlying idea of the proof-patch method is that one small proof may potentially resemble a fragment of a bigger

Goals and Subgoals	Applied Tactics

`forall (m : nat)(M : 'M_m),`
`seqmx_of_mx (fast_invmx M)= cfast_invmx m (seqmx_of_mx M)`
	`move => m.`

`forall (M : 'M_m),`
`seqmx_of_mx (fast_invmx M)= cfast_invmx m (seqmx_of_mx M)`
	`elim : m => [M | m IH M].`

`seqmx_of_mx (fast_invmx M)= cfast_invmx 0 (seqmx_of_mx M)`	`by rewrite seqmx1E`

Table 1. *Fragment of the proof for Lemma* `cfast_invmxP` *in SSReflect.*

	tactics	*N tactics*	*arg type*	*tactic arg is hyp?*	*top symbol*	*subgoals*
g1	`move`	1	*nat*	no	forall	1
g2	`elim; move`	2	*nat*, M_0, *nat*, Prop, $M_{m.+1}$	yes	forall	2
g3	`rewrite`	1	Prop	seqmx1E	equal	1
g4						
g5						

Table 2. *Feature table for Lemma* `cfast_invmxP`*. Parameters inside the double lines are the extracted features. Notation g1–g5 is used to denote five consecutive subgoals in the derivation. Columns are the properties of subgoals the feature extraction method of ML4PG will track: names of applied tactics, their number, arguments, link of the proof step to a hypothesis, inductive hypothesis or a library lemma; top symbol of the current goal, and the number of the generated subgoals.*

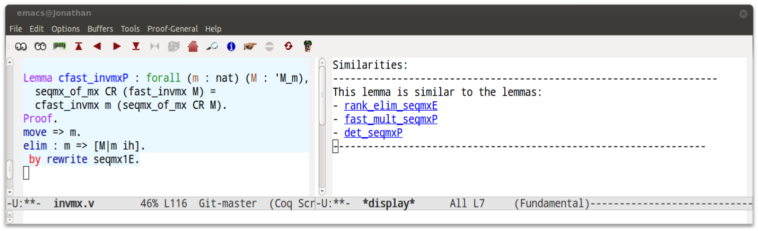
one; also, various small "patches" of big proofs may resemble. The proof-patch method automatically analyses many fragments of a big proof, using the feature extraction mechanism of Table 2. If the proof is larger than 5 steps, it will form separate feature vectors for proof steps 6–10, 11–15 and so on. Then, our feature-extraction method can be applied uniformly to any Coq library, irrespective of proof complexity or length.

Once all proof-features are extracted, and on user's request, ML4PG sends the gathered statistics (converted to a suitable format) to a chosen machine-learning interface: in our case, MATLAB or Weka; but, the list of engines could be extended just defining new translators from ML4PG's internal representation to the concrete format of the engine.

ML4PG is only connected to clustering algorithms [2] – a family of unsupervised learning methods. Clustering techniques divide data into n groups of similar objects (called clusters), where the value of n is provided by the user. There are several clustering algorithms available in MATLAB (K-means and Gaussian) and Weka (K-means, FarthestFirst and Expectation Maximisation).

Various numbers of clusters can be useful for proof-mining: this may depend on the size of the data set, and on existing similarities between the proofs. ML4PG has its own algorithm that determines the optimal number of clusters interactively, and based on the library size. As a result, the user does not provide the value of n directly, but just decides on granularity in the ML4PG menu, by selecting a value between 1 and 5, where 1 stands for a low granularity (producing a few large clusters) and 5 stands for a high granularity (producing many smaller clusters).

Example 2. ML4PG will show the suggestions provided in Figure 1 when proving Lemma `cfast_invmxP`, if we ask it to consider the CoqEAL library (this library contains 720 lemmas) for clustering, 4 as granularity value and MATLAB's K-means algorithm.



```
Lemma cfast_invmxP : forall (m : nat) (M : 'M_m),
  seqmx_of_mx CR (fast_invmx M) =
  cfast_invmx m (seqmx_of_mx CR M).
Proof.
move => m.
elim : m => [M|m ih].
 by rewrite seqmx1E.
```

```
Similarities:
------------------------------------------------------------------
This lemma is similar to the lemmas:
- rank_elim_seqmxE
- fast_mult_seqmxP
- det_seqmxP
-----------------------------------------------------------------
```

```
-U:**-  invmx.v       46% L116  Git-master  (Coq Scr-U:**-  *display*      All L7      (Fundamental)------------------------
```

Fig. 1. Suggestions for Lemma `cfast_invmxP`. The Proof General window has been split into two windows positioned side by side: the left one keeps the current proof script, and the right one shows several suggestions provided by ML4PG.

It is the nature of statistical methods to produce results with some "probability", and not being able to provide "guarantees" that a certain cluster will be found for a certain library. However, ML4PG ensures quality of the output in two different ways. First, the ML4PG output shows the digest of clustering results coming from 200 runs of the clustering algorithm. Only clusters that appear frequently in different runs of the algorithm are displayed. There is another measure of statistical nature – relative proximity of examples in a cluster, which is also taken into account by ML4PG before the results are shown, see [5].

References

1. D. Basin, A. Bundy, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning.* Cambridge University Press, 2005.
2. C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.
3. M. Dénès, A. Mörtberg, and V. Siles. A Refinement Based Approach to Computational Algebra in Coq. In *Proceedings ITP'12*, volume 7406 of *LNCS*, pages 83–98, 2012.
4. Hazel Duncan. *The use of Data-Mining for the Automatic Formation of Tactics.* PhD thesis, University of Edinburgh, 2002.
5. J. Heras and E. Komendantskaya. ML4PG: downloadable programs, manual, examples, 2012–2013. `www.computing.dundee.ac.uk/staff/katya/ML4PG/`.
6. M. Johansson, L. Dixon, and A. Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.
7. D. Kühlwein et al. Learning2Reason. In *Proceedings CICM'11*, volume 6824 of *LNCS*, pages 298–300, 2011.