ML4PG in Computer Algebra Verification*

Jónathan Heras and Ekaterina Komendantskaya

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

Abstract. ML4PG is a machine-learning extension that provides statistical proof hints during the process of Coq/SSReflect proof development. In this paper, we use ML4PG to find proof patterns in the CoqEAL library – a library that was devised to verify the correctness of Computer Algebra algorithms. In particular, we use ML4PG to help us in the formalisation of an efficient algorithm to compute the inverse of triangular matrices.

Keywords: ML4PG, Interactive Theorem Proving, Coq, SSReflect, Machine Learning, Clustering, CoqEAL.

1 Introduction

There is a trend in interactive theorem provers to develop general purpose methodologies to aid in the formalisation of a family of related proofs. However, although the application of a methodology is straightforward for its developers, it is usually difficult for an external user to decipher the key results to import such a methodology into a new development. Therefore, tools which can capture methods and suggest appropriate lemmas based on proof patterns would be valuable. ML4PG [5] – a machine-learning extension to Proof General that interactively finds proof patterns in Coq/SSReflect – can be useful in this context.

In this paper, we use ML4PG to guide us in the formalisation of a fast algorithm to compute the inverse of triangular matrices using the CoqEAL methodology [4] – a method designed to verify the correctness of efficient Computer Algebra algorithms.

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

2 Combining the CoqEAL Methodology with ML4PG

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

* The work was supported by EPSRC grant EP/J014222/1.

J. Carette et al. (Eds.): CICM 2013, LNAI 7961, pp. 354–358, 2013.

presented in [4], and was implemented as a new library, built on top of the SSReflect libraries, called *CoqEAL*. The approach [4] to formalise efficient algorithms 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* 1*s 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 and to finish the formalisation we should prove the following lemma.

Lemma 1. *Let M be a square triangular matrix of size n with* 1*s in the diagonal; then given M as input,* `fast_invmx` *and* `cfast_invmx` *obtain the same result but with different representations. The statement of this lemma in SSReflect is:*

```
Lemma cfast_invmxP : forall (n : nat) (M : 'M_n),
  seqmx_of_mx (fast_invmx M) = cfast_invmx (seqmx_of_mx M).
```

where the function `seqmx_of_mx` *transforms matrices represented as functions to matrices represented as lists of lists.*

The proof of Lemma 1 for a non-expert user of CoqEAL is not direct, and, after applying induction on the size of the matrix, the developer can get easily stuck when proving such a result.

Problem 1. *Find a method to proceed with the inductive case of Lemma 1.*

In this context, the user can invoke ML4PG to find some common proof-pattern in the CoqEAL library. ML4PG generated solutions is presented in Figure 1.

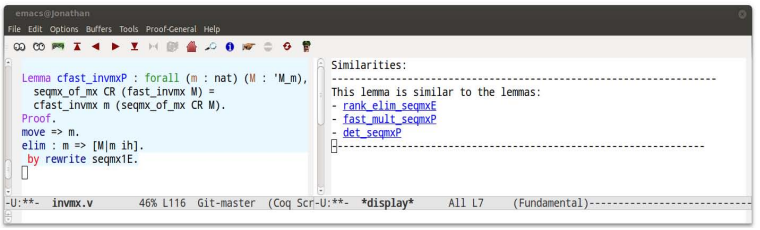

Fig. 1. Suggestions for Lemma `cfast_invmxP`. The Proof General window has been split into two windows positioned side by side: the left one keeps the current proof script, and the right one shows the suggestions provided by ML4PG.

ML4PG suggests three lemmas which are the equivalent counterparts of Lemma 1 for the algorithms computing the rank, the determinant and the fast multiplication of matrices. Inspecting the proof of these three lemmas, the user can find Proof Strategy 1 which is followed by those three lemmas and which can also be applied in Lemma 1.

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.*

It is worth remarking that the user is left with the task of finding a proof strategy from the suggestions provided by ML4PG. In the future, we could apply symbolic machine-learning techniques such as Rippling [1] and Theory Exploration [3] to automatically conceptualise the proof strategies from the suggestions provided by ML4PG.

3 Applying ML4PG to the CoqEAL Library

In the section, we show how ML4PG discovers the lemmas which follow Proof Strategy 1. This process can be split into 4 steps: extraction of significant features from library-lemmas, selection of the machine-learning algorithm, configuration of parameters, and presentation of the output.

Step 1. Feature Extraction. During the proof development, ML4PG works on the background of Proof General, and extracts (using the algorithm described in [5]) some simple, low-level features from interactive proofs in Coq/SSReflect. In addition, ML4PG extends Coq's compilation procedure to extract lemma-features from already-developed libraries.

In the example presented in the previous section, we have extracted the features from the 18 files included in the CoqEAL library (these files involve 720 lemmas). Any number of additional Coq libraries can be be selected using the ML4PG menu. Unlike e.g. [6], scaling is done at the feature extraction stage, rather than on the machine-learning stage of the process.

Step 2. Clustering Algorithm. On user's request, ML4PG sends the gathered statistics to a chosen machine-learning interface and triggers execution of a clustering algorithm of the user's choice – clustering algorithms [2] are a family of unsupervised learning methods which divide data into n groups of similar objects (called clusters), where the value of n is provided by the user.

We have integrated ML4PG with several clustering algorithms available in MATLAB (K-means and Gaussian) and Weka (K-means, FarthestFirst and Expectation Maximisation). In the CoqEAL example, ML4PG uses the MATLAB K-means algorithm to compute clusters – this is the algorithm used by default.

Step 3. Configuration of Granularity. The input of the clustering algorithms is a file that contains the information associated with the lemmas to be analysed, and a natural number n, which indicates the number of clusters. The file with the features of the library-lemmas is automatically extracted (see [5]).

To determine the value of n, ML4PG has its own algorithm that calculates the optimal number of clusters interactively, 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. The granularity parameter ranges from 1 to 5, where 1 stands for a low granularity (producing a few large clusters with a low correlation among their elements) and 5 stands for a high granularity (producing many smaller clusters with a high correlation among their elements). By default, ML4PG works with the granularity value of 3 and this is the value presented in the previous section.

Step 4. Presentation of the Results. Clustering algorithms output contains not only clusters but also a measure which indicates the proximity of the elements of the clusters. In addition, results of one run of a clustering algorithm may differ from another; then ML4PG runs the clustering algorithm 200 times, obtaining the frequency of each cluster as a result. These two measures (proximity and frequencies) are used as thresholds to decide on the single "most reliable" cluster to be shown to the user, cf. Figure 1.

These 4 steps are the workflow followed by ML4PG to obtain clusters of similar proofs. Let us present now the results that ML4PG will obtain if the user varies the different parameters – these results are summarised in Table 1.

Table 1. A series of clustering experiments discovering Proof Strategy 1. The table shows the sized of clusters containing: a) Lemma `cfast_invmxP`, b) Lemma about rank (`rank_elim_seqmxE`), c) Lemma about fast multiplication (`fast_mult_seqmxP`), and d) Lemma about determinant (`det_seqmxP`).

Algorithm:	$g = 1$ ($n = 72$)	$g = 2$ ($n = 80$)	$g = 3$ ($n = 90$)	$g = 4$ ($n = 102$)	$g = 5$ ($n = 120$)
Gaussian	$24^{a,b,c,d}$	$12^{a,b,c,d}$	$10^{a,b,c,d}$	$10^{a,b,c,d}$	$10^{a,b,c,d}$
K-means (Matlab)	$20^{a,b,c,d}$	$14^{a,b,c,d}$	$\mathbf{4}^{a,b,c,d}$	0	0
K-means (Weka)	$16^{a,b,c,d}$	$11^{a,b,c,d}$	$\mathbf{4}^{a,b,c,d}$	0	0
Expectation Maximisation	$52^{a,b,c,d}$	$45^{a,b,c,d}$	$43^{a,b,c,d}$	$39^{a,b,c,d}$	$14^{a,b,c,d}$
FarthestFirst	$30^{a,b,c,d}$	$27^{a,b,c,d}$	$27^{a,b,c,d}$	$26^{a,b,c,d}$	$20^{a,b,c,d}$

As can be seen in Table 1, the clusters obtained by almost all variations of the learning algorithms and parameters include the lemmas which led us to formulate Proof Strategy 1. However, there are some remarkable differences among the results. First of all, the results obtained with the Expectation Maximisation and FarthestFirst algorithms include several additional lemmas that make difficult the discovery of a common pattern. The same happens with the other algorithms for granularity values 1 and 2; however the clusters can be refined when increasing the granularity value. The results are clusters of a sensible size which contain lemmas with a high correlation; allowing us to spot Proof Strategy 1.

References

1. Basin, D., Bundy, A., Hutter, D., Ireland, A.: Rippling: Meta-level Guidance for Mathematical Reasoning. Cambridge University Press (2005)
2. Bishop, C.: Pattern Recognition and Machine Learning. Springer (2006)
3. Claessen, K., Johansson, M., Rosén, D., Smallbone, N.: Automating inductive proofs using theory exploration. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 392–406. Springer, Heidelberg (2013)
4. Dénès, M., Mörtberg, A., Siles, V.: A Refinement Based Approach to Computational Algebra in Coq. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 83–98. Springer, Heidelberg (2012)
5. Heras, J., Komendantskaya, E.: ML4PG: downloadable programs, manual, examples (2012-2013), `www.computing.dundee.ac.uk/staff/katya/ML4PG/`
6. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: MaSh: Machine Learning for Sledgehammer. In: Proceedings of ITP 2013. LNCS (2013)