Machine Learning Coalgebraic Proofs

Ekaterina Komendantskaya[1]

Department of Computing, University of Dundee, UK *

Abstract. This position paper argues for a novel method to machine learn patterns in formal proofs using statistical machine learning methods. The method exploits coalgebraic approach to proofs. The success of the method is demonstrated on three applications allowing to distinguish well-formed proofs from ill-formed proofs, identify families of proofs and even families of potentially provable goals.
Key words: Logic Programming, Coalgebra, Coinductive Proofs, Statistical Machine Learning, Neural Networks

1 Background

Research in Formal logic has always aspired to design a formal theory based upon a language expressive enough to allow formalisation of complex mathematical theories and (later) programming tasks. Once a given problem is formulated in the language of such a theory, one can use the theory to prove or verify various statements, theorems and properties.

One of the major steps towards this goal was made in the mid'60s, when *Logic Programming* - a family of programming languages based upon first-order logic - was designed and implemented. The two major methods used in logic programming were first-order unification and resolution [22]; first order unification is decidable and SLD-resolution based on it yields efficient implementations. Many state-of-the art tools currently used in formal methods, are based on this methodology.

Over the past decade, higher-order proof assistants (also called **Interactive Theorem Provers, or ITPs**), such as ACL2, AGDA, Coq and Isabelle(HOL) have proven to be suitable for expressing and solving sophisticated mathematical problems (e.g., verification of the Four-Colour Theorem in Coq), and industrial-scale verification of very complex computer systems (e.g., verification of computer micro-processors in ACL2).

However, currently, ITPs require considerable programming skills, constant interaction between the prover and the programmer, and overall, are time-consuming and hence expensive. Some aspects of higher-order theorem proving may never be fully automated due to their hardness: for example, higher-order unification is undecidable; termination of functions cannot be automatically decided in the general case due to the Halting Problem.

* The work was supported by the Engineering and Physical Sciences Research Council, UK; Postdoctoral Fellow research grant EP/F044046/2.

Programs in ITPs contain thousands of lemmas and theorems of variable sizes and complexities; each proof is constructed by combining a finite number of tactics. Some proofs may yield the same pattern of tactics, and can be fully automated, and others may require programmer's intervention. In this case, a manually found proof for one problematic lemma may serve as a template for several other lemmas needing a manual proof. Automated discovery of common proof-patterns using tools of statistical machine learning is the goal of the method ML-CAP (for *Machine-Learning Coalgebraic Automated Proofs*).

Another feature of theorem proving is that unsuccessful attempts at proofs, although discarded when the correct proof is found, play an important role in proof discovery. This kind of "negative" information finds no place in mathematical textbooks or libraries of automated proofs. Conveniently, analysis of both positive and negative examples is inherent in statistical machine learning [3, 8]; and the ML-CAP method we propose exploits this.

Figure 1 shows other AI methods that may eventually serve for various automation or optimisation tasks in ITPs. ML-CAP is one possible *Proof-pattern recognition* method in this figure.

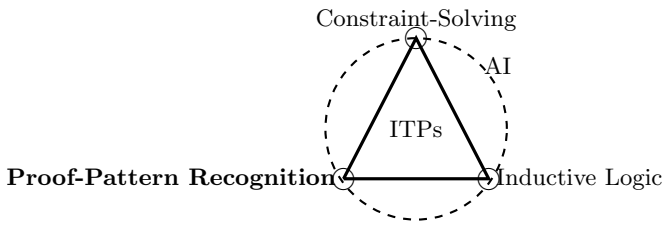

Fig. 1. Possible extensions of formal methods coming from AI

Many related attempts to exploit the inductive nature of formal reasoning are related to methods of *Inductive Logic*, (e.g. [1, 5, 7, 11, 13, 14, 25]). In contrast to them, we propose to enrich the inductive logic methods with the tools of statistical machine learning [8, 3], such as e.g. neural networks or kernels/SVMs, known for applications in statistical pattern-recognition.

Unlike logic programming, that has been successfully adapted to AI purposes, higher-order theorem proving and ITPs are commonly seen as areas completely disjoint from AI. Our ML-CAP method is intended to fill this gap. A related attempt to apply machine learning methods in higher-order logics was the application of *kernel methods* to manipulate terms of λ-calculus [9, 18, 20].

2 Methodology

The method *ML-CAP* we propose here is a pilot attempt to advance implementation of formal proofs by employing methods from Machine Learning and Coalgebra; see Figure 2.

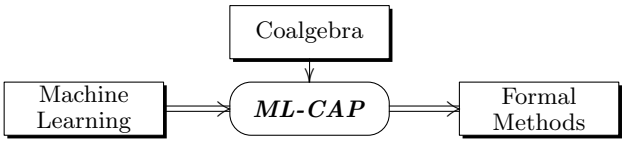

Fig. 2. Methods used in ML-CAP.

There are some serious constraints on the way of merging machine-learning and formal methods. The first constraint is zero-tolerance to *"noise"* in the syntax of ITPs (which made them so valuable for formal verification), whereas noise is part of statistical approximation and is endemic in machine-learning methods. This makes for a clash in approaches to the syntax. Neuro-Symbolic systems often solve this problem by *"propositionalising"* the syntax and working with (vectors of) truth values instead. This solution, however, does not work for some fragments of first-order logics or for higher-order logics.

The second obstacle is that many conventional algorithms used in formal methods and involving variable dependencies tend to be implemented sequentially, while statistical machine learning lends itself to concurrency, with many algorithms coming from linear algebra.

These two obstacles can be avoided if we use coalgebraic representation of proofs, [17, 16]. Coalgebraic representation of formal proofs facilitates extraction of important proof features that can be classified and "learned" using statistical machine-learning methods, see also Figure 2.

Coalgebraic methods occur in different areas of computer science, and range from categorical (structural) semantics of programming languages, [12, 23, 26, 21, 17, 16] and models of concurrent systems [19] to programming with infinite data-structures in Type Theory [6, 2] or in Logic Programming [10, 24, 17, 16]. However, the potential of coalgebraic methods in statistical machine learning has not yet received equal attention, but see e.g. [4] for a relation between coalgebra and probabilistic systems.

Generally, the coalgebraic approach to computation brings two advantages: coinductive proofs and programs are often concurrent and/or potentially infinite. Coalgebraic methods offer ways to reform the classical approach to (semantics of) computations in terms of input/output in favor of an approach that pays attention to structure, repeating patterns in computations, and observables of program execution. The main hypothesis of the ML-CAP method is that these properties of coalgebraic methods make it possible to tackle some problematic

aspects of formal reasoning using machinery developed by statistical pattern-recognition [3, 8].

The method ML-CAP arose from the work on coalgebraic semantics for logic programming [17], which gave rise to the novel algorithm for performing coinductive concurrent derivations in logic programs [16]. Our recent experiments have shown that these coalgebraic proofs yield excellent results in statistical pattern-recognition. All experiments we report here concern first-order logic programs corresponding to inductive and coinductive types in ITPs, see also [15].

3 Agenda and preliminary results

In this section, we summarise some of the results concerning data-mining proofs in logic programming; they are also formally stated and explained in [15]. When working with automated proofs, we use their representation as coinductive proof trees, [16]. We have identified four bench-mark problems for machine-learning formal proofs, as follows.

Problem 1. Classification of well-formed and ill-formed proofs. *Given a set of examples of well-formed and ill-formed coinductive trees, classify any new example of a coinductive tree in either of the two classes.*

This task is one of the most difficult for pattern-recognition (see Table 3), and in the same time, perhaps the least significant for practical purposes, as automated proof methods already work well for such tasks.

A more interesting task in practical terms is recognition of various proof-families among well-formed proofs, as this is something that may help to optimize proof-search.

Problem 2. Discovery of proof families. *Given a set of positive and negative examples of well-formed coinductive proof trees belonging to a proof family, classify any new example of a coinductive tree, whether it belongs to the given proof family.*

This problem has practical applications: finding that some unfinished proof belongs to a family of successful proofs may safe intermediate derivation steps; and knowing that some unfinished proof belongs to the family of failed proofs, may serve as a hint to abandon any future attempts.

The next problem is discovery of the proof families comprised of coinductive proof trees that may potentially lead to successful proofs – *success families.*

Problem 3. Discovery of potentially successful proofs. *Given a set of positive and negative examples of well-formed coinductive trees belonging to a success family, classify any new example of a coinductive tree, whether it belongs to the given success family.*

Finally, when it comes to coinductive logic programs defining infinite data structures, such as streams, detection of success families is impossible. In such cases, detection of well-typed and ill-typed proofs within a proof family will be an alternative to determining success families.

Problem 4. Discovery of ill-typed proofs. *Given a set of positive and negative examples of ill-typed coinductive trees belonging to a given proof family,*

classify any new example of a coinductive tree, whether it is ill-typed or well-typed.

To machine-learn classes of proof trees using statistical pattern-recognition, we used several well-known tools, such as neural networks and SVMs. All experiments involving neural networks were made in MATLAB Neural Network Toolbox (*pattern-recognition package*), with a standard three-layer feed-forward network, with sigmoid hidden and output neurons. The network was trained with scaled conjugate gradient back-propagation. Such networks can classify vectors arbitrarily well, given enough neurons in the hidden layer, we used 40, 50, 60, 70, 90 hidden neurons for various experiments. All experiments involving SVMs were performed in MATLAB Bioinformatics toolbox, *SVM package with Gaussian Radial Basis kernel*.

We used data sets of coinductive proof trees of various sizes - from 120 to 400 examples of coinductive derivation trees for various experiments; we tested all four problems stated above using proof trees for two distinct logic programs – `ListNat` defining lists of natural numbers and `Stream` defining infinite streams of bits. The table below shows the results.

	`ListNat` best test	`ListNat` best average	`Stream` best test	`Stream` best average	SVM best test	SVM best average
Problem 1	88.2%	76.4%	85	84.3 %	100 %	89 %
Problem 2	100 %	92.3%	100 %	99.1 %	n/a	88 %
Problem 2'	n/a	n/a	100 %	90.6 %	n/a	88%
Problem 3	100 %	99 %	n/a	n/a	n/a	n/a
Problem 4	n/a	n/a	100 %	85.7 %	n/a	90%

Fig. 3. Summary of the results of classification of coinductive derivation trees for the four main classification problems, performed in neural networks (first four columns), and in SVMs with kernel functions (the columns 5 and 6); the latter is only tested for the derivation trees for `Stream`.

Overall, the success rate of the classification results well exceeded our initial expectations. Only **Problem 1** results were somewhat disconcerting, but we think this problem has less practical impact. The results indicate that well-founded coinductive proof trees possess a number of strongly correlated features that determine the variety of meta-properties of the trees given by **Problems 1 - 4**, and such properties can be detected by the state-of-the-art pattern-recognition methods.

References

1. D. Basin, A. Bundy, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning.* Cambridge University Press, 2005.

2. Y. Bertot and E. Komendantskaya. Inductive and coinductive components of corecursive functions in coq. *Electr. Notes Theor. Comput. Sci.*, 203(5):25–47, 2008.
3. C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
4. P. Chaput, V. Danos, P. Panangaden, and G. D. Plotkin. Approximating labelled markov processes again! In *CALCO*, volume 5728 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2009.
5. S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.
6. T. Coquand. Infinite objects in type theory. In *Types for Proofs and Programs, Int. Workshop TYPES'93*, volume 806 of *LNCS*, pages 62–78. Springer-Verlag, 1994.
7. L. de Raedt. *Logical and Relational Learning*. 2008.
8. R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley, 2001.
9. T. Gartner, J. Lloyd, and P. Flach. Kernels and distances for structured data. *Machine Learning*, 3(57):205–232, 2004.
10. G. Gupta, A. Bansal, R. Min, L. Simon, and A. Mallya. Coinductive logic programming and its applications. In *ICLP 2007*, volume 4670 of *LNCS*, pages 27–44. Springer, 2007.
11. A. Ireland, G. Grov, and M. Butler. Reasoned modelling critics: Turning failed proofs into modelling guidance. In *ASM*, pages 189–202, 2010.
12. B. Jacobs and J. Rutten. A tutorial on coalgebras and coinduction. *EATCS Bulletin*, (62), 1997.
13. M. Johansson, L. Dixon, and A. Bundy. Case-analysis for rippling and inductive proof. In *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 291–306. Springer, 2010.
14. K. Kersting, L. D. Raedt, and T. Raiko. Logical hidden markov models. *J. Artif. Intell. Res. (JAIR)*, 25:425–456, 2006.
15. E. Komendantskaya and R. Alamghairbe. Machine-learning coalgebraic automated proofs, submitted, 2011.
16. E. Komendantskaya and J. Power. Coalgebraic derivations in logic programming. In *CSL'11*, 2011.
17. E. Komendantskaya and J. Power. Coalgebraic semantics for derivations in logic programming. In *CALCO'11*, 2011.
18. J. Lloyd. *Logic for Learning: Learning Comprehensible Theories from Structured Data*. Springer, Cognitive Technologies Series, 2003.
19. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
20. A. Passerini, P. Frasconi, and L. D. Raedt. Kernels on prolog proof trees: Statistical learning in the ilp setting. *Journal of Machine Learning Research*, 7:307–342, 2006.
21. G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.
22. J. Robinson. A machine-oriented logic based on resolution principle. *Journal of ACM*, 12(1):23–41, 1965.
23. J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 2000.
24. L. Simon, A. Bansal, A. Mallya, and G. Gupta. Co-logic programming: Extending logic programming with coinduction. In *ICALP*, volume 4596 of *LNCS*, pages 472–483. Springer, 2007.
25. V. Sorge, A. Meier, R. L. McCasland, and S. Colton. Automatic construction and verification of isotopy invariants. *J. Autom. Reasoning*, 40(2-3):221–243, 2008.
26. D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291, 1997.