Report from Dagstuhl Seminar 19202

Approaches and Applications of Inductive Programming

Edited by

Luc De Raedt[1], Richard Evans[2], Stephen H. Muggleton[3], and Ute Schmid[4]

1 **KU Leuven, BE,** luc.deraedt@cs.kuleuven.be
2 **Google DeepMind – London, GB,** richardevans@google.com
3 **Imperial College London, GB,** s.muggleton@imperial.ac.uk
4 **Universität Bamberg, DE,** ute.schmid@uni-bamberg.de

Abstract

In this report the program and the outcomes of Dagstuhl Seminar 19202 "Approaches and Applications of Inductive Programming" is documented. After a short introduction to the state of the art to inductive programming research, an overview of the introductory tutorials, the talks, program demonstrations, and the outcomes of discussion groups is given.

Seminar May 12–17, 2019 – http://www.dagstuhl.de/19202
2012 ACM Subject Classification Computing methodologies → Artificial intelligence, Computer systems organization → Robotics, Computing methodologies → Parallel programming languages, Software and its engineering → Compilers, Human-centered computing → Human computer interaction (HCI)
Keywords and phrases Enduser programming, Explainable AI, Human-like computing, Inductive logic programming, Probabilistic programming
Digital Object Identifier 10.4230/DagRep.9.5.58
Edited in cooperation with Lidia Contreras-Ochando (Technical University of Valencia, ES), liconoc@upv.es

1 Executive Summary

Ute Schmid
Luc De Raedt

Inductive programming addresses the automated or semi-automated generation of computer programs from incomplete information such as input-output examples, constraints, computation traces, demonstrations, or problem-solving experience [11]. The generated – typically declarative – program has the status of a hypothesis which has been generalized by induction. That is, inductive programming can be seen as a special approach to machine learning. In contrast to standard machine learning, only a small number of training examples is necessary. Furthermore, learned hypotheses are represented as logic or functional programs, that is, they are represented on symbol level and therefore are inspectable and comprehensible [36, 15, 37, 29]. On the other hand, inductive programming is a special approach to program synthesis. It complements deductive and transformational approaches [39, 25, 4]. In cases where synthesis of specific algorithm details that are hard to figure out by humans inductive reasoning can be used to generate program candidates from either user-provided data such as test cases or from data automatically derived from a formal specification [35]. Finally, symbolic approaches can be combined with probabilistic methods [8, 9].

Inductive program synthesis is of interest for researchers in artificial intelligence since the late sixties [2]. On the one hand, the complex intellectual cognitive processes involved in producing program code which satisfies some specification are investigated, on the other hand methodologies and techniques for automating parts of the program development process are explored. One of the most relevant areas of application of inductive programming techniques is end-user programming [5, 22, 6]. For example, the Microsoft Excel plug-in Flashfill synthesizes programs from a small set of observations of user behavior [15, 14, 13]. Related applications are in process mining and in data wrangling [19, 21]. Inductive programming in general offers powerful approaches to learning from relational data [30, 23] and to learning from observations in the context of autonomous intelligent agents [28, 20, 36]. Furthermore, inductive programming can be applied in the context of teaching programming [38, 41].

A recent new domain of interest is how to combine inductive programming with blackbox approaches, especially in the context of (deep) neural networks [10] and in data science.

Relation to Previous Dagstuhl-Seminars

The seminar is a continuation Dagstuhl-Seminars 13502, 15442, and 17382. In the first seminar, the focus was on establishing the research community by exploring the different areas of basic research and applications of inductive programming and identifying commonalities and differences in methods and goals. In the second seminar, more in-depth coverage of algorithmic methods was provided and the relation of inductive programming to cognitive modeling was explored. The third seminar had a main focus on applications in data cleansing, teaching programming, and interactive training. Furthermore, first proposals for neural approaches to learning for inductive programming were presented.

Besides many new insights from many discussions, visible outcomes from the previous seminars are:

- Muggleton, S.H., Schmid, U., Zeller, C., Tamaddoni-Nezhad, A. and T. Besold (2019). Ultra-strong machine learning – comprehensibility of programs learned with ILP. Machine Learning, 107(7), 1119–1140.
- Schmid, U., Zeller, C., Besold, T., Tamaddoni-Nezhad, A., & Muggleton, S.H. (2017). How does predicate invention affect human comprehensibility?. In Alessandra Russo and James Cussens, editors, Proceedings of the 26th International Conference on Inductive Logic Programming (ILP 2016), pp. 52-67, Springer.
- Hernández-Orallo, J., Martínez-Plumed, F., Schmid, U., Siebers, M., & Dowe, D. L. (2016). Computer models solving intelligence test problems: Progress and implications. Artificial Intelligence, 230, 74-107.
- Gulwani, S., Hernández-Orallo, J., Kitzelmann, E., Muggleton, S. H., Schmid, U., & Zorn, B. (2015). Inductive programming meets the real world. *Communications of the ACM*, 58(11), 90-99.
- https://en.wikipedia.org/wiki/Inductive_programming
- NIPS'2016 workshop on Neural Nets and Program Induction involving Stephen Muggleton.
- A collaboration in the context of the EPSRC funded Human-Like Computing Network+ headed by Muggleton (see http://hlc.doc.ic.ac.uk/).

For the fourth seminar, we extend our invitation to researchers from deep learning and to researchers addressing statistical machine learning and probabilistic reasoning. **The focus of the fourth seminar has been on the potential of inductive programming for explainable AI, especially in combination with (deep) neural networks and with data science.**

Inductive Programming as a Approach to Explainable AI

Recently it has been recognized that explainability is crucial for machine learning to be usable in real world domains – especially such where erroneous decisions might be harmful to humans. Consequently, interfaces which explain (aspects) of classifier decisions have been proposed – especially in the context of deep learning [32, 3]. The notion of explainability has already been addressed in the early days of machine learning: During the 1980s Michie defined Machine Learning in terms of two orthogonal axes of performance: predictive accuracy and comprehensibility of generated hypotheses. Since predictive accuracy was readily measurable and comprehensibility not so, later definitions in the 1990s, such as that of Mitchell [27], tended to use a one-dimensional approach to Machine Learning based solely on predictive accuracy, ultimately favouring statistical over symbolic Machine Learning approaches.

In [29] a definition was provided of comprehensibility of hypotheses which can be estimated using human participant trials. Experiments were conducted testing human comprehensibility of logic programs. Results show that participants were not able to learn the relational concept on their own from a set of examples but they were able to apply the relational definition provided by the ILP Metagol system correctly. That is, the results demonstrate that ILP systems can fulfill Michie's criterion of operational effectiveness. The findings also imply the existence of a class of relational concepts which are hard to acquire for humans, though easy to understand given an abstract explanation. We believe improved understanding of this class could have potential relevance to contexts involving human learning, teaching and verbal interaction.

Finally, while research in explanations in the context of neural networks is focusing on visualization, ILP learned classifiers allow natural language explanations. In the Dagstuhl seminar we plan to discuss possibilities for combining deep learning approaches and inductive programming such that both modes of explanations can be generated.

Inductive Programming for Support in Data Science

The success of the inductive programming system FlashFill has motivated the developed of several approaches to using inductive programming in the context of data science, more specifically data wrangling. It is well known that in data science and data mining processes, about 80 per cent of the time goes to selecting the right data, and further pre-processing it so that it be input into data mining software. One important step in that process is data wrangling, which is concerned with cleaning up the data and transforming it across different formats. For this step, inductive programs can be used; various approaches are moving in that direction, e.g. FlashRelate [1], Tacle [19] and SYNTH [7]. Furthermore, workshops are being organised on the topic of automating data wrangling (e.g. at ICDM 2016, http://dmip.webs.upv.es/DWA2016/, at ECMLPKDD 2019, https://sites.google.com/view/autods and at Dagstuhl Seminar 18401), and tools such as MagicHaskeller and JailBreakR being used in this context. In the Dagstuhl seminar, we also want to deepen the link between data wrangling and inductive programming.

Inductive Programming and Neural Computation

The deep learning community has been interested in taking on challenging tasks from the artificial intelligence community. It is therefore no surprise that they have also started to look into inductive and automatic programming. In particular, they have contributed several

mixtures of traditional computational models with those of neural networks. For instance, the neural Turing machine [12] integrates a neural network with an external memory and it is able to learn simple algorithms such as copy and sort, the neural program interpreter [31] is a recurrent neural network that learns to represent and execute programs from program traces, while [33] present an end-to-end differentiable interpreter for the programming language Forth and [34, 24] for a declarative Prolog like language. The central goal in these approaches is to obtain an end-to-end differentiable model. While initial results are promising, the approaches still require a lot of data to be trained or need to scale up. This contrasts with the more traditional symbolic approaches to inductive programming and thus a key opportunity is to further cross-fertalize these two approaches.

Inductive Programming and Human-like Computing

The human ability to master complex demands is to a large extend based on the ability to exploit previous experiences. Based on our experience, we are able to predict characteristics or reactions of (natural or man-made, inanimate or animate) objects, we can reason about possible outcomes of actions, and we can apply previously successful routines and strategies to new tasks and problems. In philosophy, psychology and artificial intelligence, researchers proposed that the core process to expand knowledge, that is, construct hypotheses, in such a way that we can transfer knowledge from previous experience to new situations is *inductive* inference [18, 16, 40].

One special aspect of induction is that humans are able to acquire complex, productive rule sets from experience. Following Chomsky, rules are productive when they can be applied in situations of various complexity. Typical examples of such rule sets are knowledge about natural language grammar, recursive concepts such as ancestor and recursive problem solving strategies. For example, if humans have learned how to solve Tower of Hanoi problems with three and four discs, at least some of them are able to generalize the underlying strategy for solving problems with an arbitrary number of discs.

Inductive programming provides mechanisms to generate such productive, recursive rule sets. Examples of recent work on using inductive programming to model this learning-capability of human cognition are [26, 20, 36, 17, 23]. Therefore, it might be fruitful for cognitive scientists to get acquainted with inductive programming as one approach to model the acquisition of complex knowledge structures. On the other hand, knowledge gained from experiments in human problem solving, concept learning and language acquisition can be a source of inspiration for new algorithmic approaches to inductive programming.

Objectives and Expected Outcomes of the Seminar

A long-term objective of the seminar series is to establish inductive programming as a self-contained research topic in artificial intelligence, especially as a field of machine learning and of cognitive modeling. The seminar serves as community building event by bringing together researchers from different areas of inductive programming – especially inductive logic programming and inductive functional programming –, from different application areas such as end-user programming and tutoring, and from cognitive science research, especially from cognitive models of inductive (concept) learning. For successful community building we seek to balance junior and senior researchers and to mix researchers from universities and from industry.

The previous seminars resulted in new collaborations between researchers from different backgrounds as documented in joint publications and we expect that the collaborations will continue, deepen and extend, resulting not only in further joint publications but also in joint research projects.

In the fourth seminar, we continued and extended previous discussions addressing the following aspects:

- Identifying the specific contributions of inductive programming to machine learning research and applications of machine learning, especially identifying problems for which inductive programming approaches are more suited than standard machine learning approaches, including deep learning and probabilistic programming. Focus here is on possibilities of combining (deep) neural approaches or probabilistic programming with (symbolic) inductive programming, especially with respect to new approaches to comprehensibility of machine learned models and on explainable AI.
- Establishing criteria for evaluating inductive programming approaches in comparison to each other and in comparison to other approaches of machine learning and providing a set of benchmark problems.
- Discussing current applications of inductive programming in end-user programming and programming education and identifying further relevant areas of application.
- Establishing stronger relations between cognitive science research on inductive learning and inductive programming under the label of human-like computation.
- Strengthening the relation of inductive programming and data science, especially with respect to data cleansing and data wrangling.

Concluding Remarks and Future Plans

In the wrapping-up section, we decided to move the IP webpage[1] to a Wiki and encouraged all participants to make available their systems, tutorial/lecture slides and publications there.

As the grand IP challenge we came up with 2017 is still up:

> An IP program should invent an algorithm publishable in a serious journal (e.g., an integer factorization algorithm) or win a programming competition!

References

1 Daniel W. Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. Flashrelate: Extracting relational data from semi-structured spreadsheets using examples. *SIGPLAN Not.*, 50(6):218–228, June 2015.
2 A. W. Biermann, G. Guiho, and Y. Kodratoff, editors. *Automatic Program Construction Techniques*. Macmillan, New York, 1984.
3 Alexander Binder, Sebastian Bach, Gregoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for deep neural network architectures. In *Information Science and Applications (ICISA) 2016*, pages 913–922. Springer, 2016.
4 Rastislav Bodik and Emina Torlak. Synthesizing programs with constraint solvers. In *CAV*, page 3, 2012.
5 A. Cypher, editor. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, 1993.

[1] www.inductive-programming.org

6 Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols, editors. *No Code Required: Giving Users Tools to Transform the Web*. Elsevier, 2010.

7 Luc De Raedt, Hendrik Blockeel, Samuel Kolb, Stefano Teso, and Gust Verbruggen. Elements of an automatic data scientist. In *International Symposium on Intelligent Data Analysis*, pages 3–14. Springer, 2018.

8 Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

9 Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

10 Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? *arXiv preprint arXiv:1802.08535*, 2018.

11 P. Flener and U. Schmid. Inductive programming. In C. Sammut and G. Webb, editors, *Encyclopedia of Machine Learning*, pages 537–544. Springer, 2010.

12 Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

13 Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *38th Symposium on Principles of Programming Languages*. ACM, 2011.

14 Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105, 2012.

15 Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, and Benjamin G. Zorn. Inductive programming meets the real world. *Commununications of the ACM*, 58(11):90–99, 2015.

16 Ulrike Hahn, Todd M Bailey, and Lucy BC Elvin. Effects of category diversity on learning, memory, and generalization. *Memory & Cognition*, 33(2):289–302, 2005.

17 J. Hernández-Orallo, D. L. Dowe, and M. V. Hernández-Lloreda. Universal psychometrics: Measuring cognitive abilities in the machine kingdom. *Cognitive Systems Research*, 27(0):50–74, 2014.

18 J.H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R. Thagard. *Induction – Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, MA, 1986.

19 Samuel Kolb, Sergey Paramonov, Tias Guns, and Luc De Raedt. Learning constraints in spreadsheets and tabular data. *Machine Learning*, 106(9-10):1441–1468, 2017.

20 P. Langley and D. Choi. A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, MA, 2006. AAAI Press.

21 Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. *ACM SIGPLAN Notices*, 49(6):542–553, 2014.

22 Henry Lieberman, editor. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, San Francisco, 2001.

23 D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, and S.H. Muggleton. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530, Amsterdam, 2014. IOS Press.

24 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3749–3759. Curran Associates, Inc., 2018.

25 Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

26 G. F. Marcus. *The Algebraic Mind. Integrating Conncetionism and Cognitive Science*. Bradford, Cambridge, MA, 2001.

27 Tom Mitchell. *Machine learning*. McGraw Hill, 1997.

28 Bogdan Moldovan, Plinio Moreno, Davide Nitti, José Santos-Victor, and Luc De Raedt. Relational affordances for multiple-object manipulation. *Auton. Robots*, 42(1):19–44, 2018.

29 S.H. Muggleton, U. Schmid, C. Zeller, A. Tamaddoni-Nezhad, and T. Besold. Ultra-strong machine learning – comprehensibility of programs learned with ILP. *Machine Learning*, 2018.

30 Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

31 Scott E. Reed and Nando de Freitas. Neural programmer-interpreters. *CoRR*, abs/1511.06279, 2015.

32 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

33 Sebastian Riedel, Matko Bosnjak, and Tim Rocktäschel. Programming with a differentiable forth interpreter. *CoRR*, abs/1605.06640, 2016.

34 Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *CoRR*, abs/1705.11040, 2017.

35 Reudismam Rolim, Gustavo Soares, Loris D'Antoni, Oleksandr Polozov, Sumit Gulwani, Rohit Gheyi, Ryo Suzuki, and Bjoern Hartmann. Learning syntactic program transformations from examples. *arXiv preprint arXiv:1608.09000*, 2016.

36 Ute Schmid and Emanuel Kitzelmann. Inductive rule learning on the knowledge level. *Cognitive Systems Research*, 12(3):237–248, 2011.

37 Ute Schmid, Christina Zeller, Tarek Besold, Alireza Tamaddoni-Nezhad, and Stephen Muggleton. How does predicate invention affect human comprehensibility? In *International Conference on Inductive Logic Programming*, pages 52–67. Springer, 2016.

38 Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices*, 48(6):15–26, 2013.

39 Douglas R. Smith. The synthesis of LISP programs from examples: A survey. In *Automatic Program Construction Techniques*, pages 307–324. Macmillan, 1984.

40 J. Tenenbaum, T.L. Griffiths, and C. Kemp. Theory-based Bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, 10(7):309–318, 2006.

41 Christina Zeller and Ute Schmid. Automatic generation of analogous problems to help resolving misconceptions in an intelligent tutor system for written subtraction. In *Proceedings of the Workshop on Computational Analogy at the 24th International Conference on Case Based Reasoning (ICCBR 2016, Atlanta, GA, 31th October to 2nd November 2016)*, 2016.

2 Table of Contents

3 Introductory Talks

3.1 Introduction to Inductive Functional Programming

Ute Schmid (Universität Bamberg, DE)

Inductive programming (IP) is an area of research concerned with learning recursive programs, typically in some declarative language, from small sets of input/output examples. Since a general program is induced from examples, IP can be seen as a special case of machine learning. In contrast to standard machine learning, induced program hypotheses are required to cover all given examples correctly. In the talk I will give a general introduction to the research area of IP and then focus on induction of functional programs. First, I will present Thesys – the classical approach for learning Lisp programs by regularity detection. Afterwards, I will present the more recent approach IgorII. IgorII has been the first approach realizing necessary function invention on the fly. It integrates the concept of structure-guided program induction from Thesys, concepts of contemporary programming languages, such as pattern matching, as well as concepts proposed in Inductive Logic Programming (ILP), especially allowing to take into account background knowledge. I will point out relations of IP to cognitive models of learning, demonstrating that IgorII can be applied out of the box to learning the recursive rule sets for problems such as Tower of Hanoi, language parsing, and number series induction. Finally, I will present arguments for IP as a highly expressive approach to interpretable ML.

References

1 Gulwani, S., Hernández-Orallo, J., Kitzelmann, E., Muggleton, S., Schmid, U., and Zorn, B. (2015). Inductive Programming Meets the Real World, *Communications of the ACM, 58(11)*, 90-99.
2 Hernández-Orallo, J., Martínez-Plumed, F., Schmid, U., Siebers, M., Dowe, D.L. (2016). Computer Models Solving Intelligence Test Problems: Progress and Implications. *Artificial Intelligence, Vol. 230*, 74–107.

4 Overview of Talks

4.1 Declarative Compression of Imperative Programs

Eli Bingham (Uber AI Labs – San Francisco, US)

Joint work of Eli Bingham, Fritz Obermeyer, Noah D. Goodman

Standard approaches to program synthesis in general-purpose imperative languages like Python often struggle with ambiguous or inconsistent examples or produce large deterministic programs which are difficult to interpret. These limitations can be overcome in principle by adding randomness or other forms of nondeterminism to the language, but support for nondeterminism is limited in existing synthesis tools and the resulting programs may only approximately solve the tasks at hand.

To bridge this gap, we introduce program merging, a simple algorithm for finding short nondeterministic programs that approximate the behavior of a deterministic imperative program. We define a family of compressive program transformations that refactor exactly duplicated program fragments into new subroutines or replace approximately duplicated program fragments with declarative approximations such as random variables, constraints, or learnable parameters. Program merging searches for sequences of such transformations applied to an initial program that maximize the posterior probability of the final program given the original examples or specification.

The resulting programs may mix multiple intractable declarative computations (in the form of nested sums over multiple semirings). As a step towards principled and efficient approximate evaluation of such programs, we introduce Funsor, a Python-embedded domain-specific language for mixed-mode declarative programming with an initial focus on approximate probabilistic inference and gradient-based optimization.

4.2 Difficulty of problems for humans

Ivan Bratko (University of Ljubljana, SI)

Joint work of Ivan Bratko, Matej Guid, Dayana Hristova, Simon Stoiljkovikj

Some questions of interest for Explainable AI are: (1) How difficult a given problem is for humans, and how humans would typically tackle the problem? (2) How can we measure the comprehensibility of a theory or of an explanation? In this talk we discuss one approach to automatic prediction of difficulty for humans of problems that are solved through informed search. Although there are many applications that require prediction of difficulty, for example intelligent tutoring systems, in our case the motivation came from a computer analysis of the quality of play in chess games played by human world chess champions. This analysis aimed at answering questions like who was the best chess player of all time. The quality of play was estimated on the basis of the differences between moves played by humans and moves played in the same positions by a chess playing engine taken as the golden standard. Different chess champions tended to play in different styles: some tended towards simple, quiet and safe positions, while others preferred complicated, aggressive and risky game. As it is easier to play correctly in simple positions than in complex positions, our evaluation method, to be fair, also had to take into account the difficulty of individual positions that occurred in champions games. Our method of assessing the difficulty of chess positions was based on the amount of search needed by the chess engine to find the best move. This enabled a fair comparison of players with different playing styles, and to answer questions like: How well would a player like Capablanca, known for his tendency to simplify the game, do if he played in the style of Tal, known for his tendency toward extremely complicated positions. Although this approach was generally successful in the comparison of world champions, it was later found inappropriate for estimating the difficulty of particular type of positions, called tactical chess positions. In solving tactical positions, humans use pattern-based knowledge to guide their search extremely effectively. As this knowledge has never been sufficiently formalized for use in a chess program, a refinement was needed to simulate human search by a chess program without access to humans' pattern-based tactical knowledge.

References

1 Ivan Bratko, Dayana Hristova, Matej Guid. *Search vs. Knowledge in Human Problem Solving: A Case Study in Chess*. Model-Based Reasoning in Science and Technology: Logical, Epistemological and Cognitive Issues, pp. 569-584. Springer 2016. SAPERE Book Series (Studies in Applied Philosophy, Epistemology and Rational Ethics).

2 Simon Stoiljkovikj, Ivan Bratko, Matej Guid. *A Computational Model for Estimating the Difficulty of Chess Problems*. Proceedings of the Third Annual Conference on Advances in Cognitive Systems, ACS-2015 (Article 7), Cognitive Systems Foundations.

4.3 Trace-Based Programming Method

Maurice Chandoo (Leibniz Universität Hannover, DE)

Main reference Maurice Chandoo: "A Systematic Approach to Programming", CoRR, Vol. abs/1808.08989, 2018.
URL https://arxiv.org/abs/1808.08989

We describe a programming method for systematically implementing sequential algorithms in any imperative or functional programming language. The method is based on the premise that it is easy to write down how an algorithm proceeds on a concrete input. This information–which we call execution trace–is used as a starting point to derive the desired program. In contrast to test-driven development the program is directly constructed from the test cases instead of written separately. The program's operations, control flow and predicates guiding the control flow are worked out separately, which saves the programmer from having to think about them simultaneously. This reduces the likelihood of introducing errors. We demonstrate our method for two examples and discuss its utility. The method only requires pen and paper. However, a computer can facilitate its usage by taking care of certain trivial tasks.

Outline of the method. The execution trace of an algorithm can be seen as a table where each column corresponds to a variable. The i-th row contains the values of the variables after i execution steps of the algorithm for a certain input. The first step is to determine the set of variables used by the algorithm. Then an input is chosen and an execution trace for this input is written down. The next step is to generalize the literals in this table. More specifically, one has to determine how each value in row $i+1$ can be expressed in terms of the values in row i. After removing the literals from the table, the sequence of expressions that is left in each row corresponds to an operation executed by the algorithm. Certain rows correspond to the same operation. This information can be used to derive the control flow graph of the desired program. Assume the i-th row corresponds to the operation $\alpha(i)$. Then the graph has a vertex for each operation and for each i there is an edge from $\alpha(i)$ to $\alpha(i+1)$. Stated differently, the sequence of operations in the table describes a path through the control flow graph. By repeating the previous steps for various inputs one will eventually arrive at the complete control flow graph of the program. Finally, it remains to determine the edge predicates, i.e. under what circumstances does the program move from one program state to the next.

The method works in such a way that the constructed program is consistent with all execution traces that were used to build it. A one-to-one correspondence between program states and operations is assumed. Intuitively, this means each line of code in a program must be unique. While this is not necessarily the case, this can always be achieved by adding a state variable.

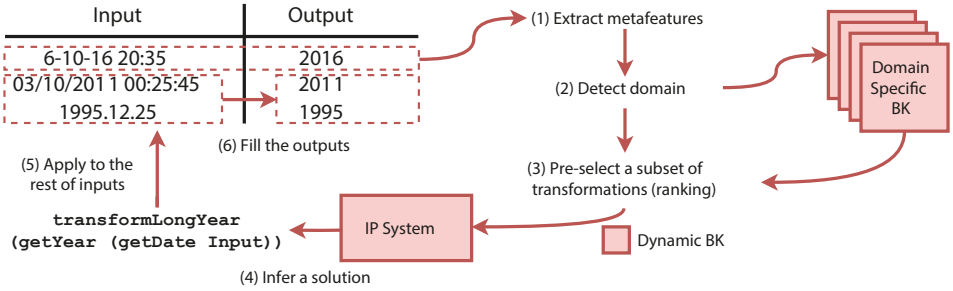

Figure 1 Automating data wrangling with IP: process example. The first row (Input and Output) is used as an input example for the IP system. The function returned is applied to the rest of the instances to obtain the outputs.

4.4 Automated Data Transformation with Inductive Programming and Dynamic Background Knowledge

Lidia Contreras-Ochando (Technical University of Valencia, ES)

Joint work of Lidia Contreras-Ochando, Cesar Ferri Ramirez, José Hernández-Orallo, Fernando Martínez-Plumed, María-José Ramírez-Quintana, Susumu Katayama
Main reference Lidia Contreras-Ochando, Cesar Ferri Ramirez, José Hernández-Orallo, Fernando Martínez-Plumed, María-José Ramírez-Quintana, Susumu Katayama: "Automated data transformation with inductive programming and dynamic background knowledge". In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2019 (to appear). ECML-PKDD'19 (2019)

Data quality is vital for machine learning and data science. Despite the growing amount of data preparation tools, the most tedious data wrangling activities and feature manipulation are still partially resistant to automation because they depend heavily on domain information. For example, if the strings "17th of August of 2017" and "2017-08-17" are to be formatted into "08/17/2017" to be correctly recognised by a data science tool, this is generally processed in two phases: (1) they are recognised as dates and (2) some conversions are applied specific to the date domain. The dates manipulating processes, however, are very distinct from those for manipulating addresses or phone numbers. This needs enormous quantities of background knowledge, which generally becomes a bottleneck as domain and format diversity grows. We assist to relieve this issue by using inductive programming (IP) with dynamic background knowledge (BK) fuelled by a machine learning meta-model that chooses the domain, the primitives (or both) from some descriptive features of the data wrangling problem (see Figure 1). We demonstrate this for the automation of data transformation and we evaluate the approach on an integrated benchmark for data wrangling, which we share publicly for the community.

Acknowledgments. This research was supported by the EU (FEDER) and the Spanish MINECO (RTI2018-094403-B-C32), Universitat Politècnica de València (PAID-06-18), and the Generalitat Valenciana (PROMETEO/2019/098 and BEST/2018/027). L. Contreras-Ochando was also supported by the Spanish MECD (FPU15/03219). J. Hernández-Orallo is also funded by FLI (RFP2-152).

References

1 Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., Ramírez-Quintana, M.J., Katayama, S.: Automated data transformation with inductive programming and dynamic background knowledge. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2019 (to appear). ECML-PKDD '19 (2019)
2 Contreras-Ochando, L.: DataWrangling-DSI: BETA – Extended Results (2019). 10.5281/zenodo.2557385
3 Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., Ramírez-Quintana, M.J., Katayama, S.: General-purpose declarative inductive programming with domain-specific background knowledge for data wrangling automation. arXiv preprint arXiv:1809.10054 (2018)

4.5 Inductive general game playing

Andrew Cropper (University of Oxford, GB), Richard Evans (Google DeepMind – London, GB), and Mark Law (Imperial College London, GB)

Main reference Andrew Cropper, Richard Evans, Mark Law: "Inductive general game playing", CoRR, Vol. abs/1906.09627, 2019.
URL https://arxiv.org/abs/1906.09627

General game playing (GGP) is a framework for evaluating an agent's general intelligence across a wide range of tasks. In the GGP competition, an agent is given the rules of a game (described as a logic program) that it has never seen before. The task is for the agent to play the game, thus generating game traces. The winner of the GGP competition is the agent that gets the best total score over all the games. In this paper, we invert this task: a learner is given game traces and the task is to learn the rules that could produce the traces. This problem is central to *inductive general game playing* (IGGP). We introduce a technique that automatically generates IGGP tasks from GGP games. We introduce an IGGP dataset which contains traces from 50 diverse games, such as *Sudoku*, *Sokoban*, and *Checkers*. We claim that IGGP is difficult for existing inductive logic programming (ILP) approaches. To support this claim, we evaluate existing ILP systems on our dataset. Our empirical results show that most of the games cannot be correctly learned by existing systems. The best performing system solves only 40% of the tasks perfectly. Our results suggest that IGGP poses many challenges to existing approaches. Furthermore, because we can automatically generate IGGP tasks from GGP games, our dataset will continue to grow with the GGP competition, as new games are added every year. We therefore think that the IGGP problem and dataset will be valuable for motivating and evaluating future research.

4.6 Playgol: learning programs through play

Andrew Cropper (University of Oxford, GB)

Main reference Andrew Cropper: "Playgol: learning programs through play", CoRR, Vol. abs/1904.08993, 2019.
URL https://arxiv.org/abs/1904.08993

Children learn though play. We introduce the analogous idea of *learning programs through play*. In this approach, a program induction system (the learner) is given a set of tasks and initial background knowledge. Before solving the tasks, the learner enters an *unsupervised playing* stage where it creates its own tasks to solve, tries to solve them, and saves any solutions (programs) to the background knowledge. After the playing stage is finished, the learner enters the *supervised building* stage where it tries to solve the user-supplied tasks and can reuse solutions learnt whilst playing. The idea is that playing allows the learner to discover reusable general programs on its own which can then help solve the user-supplied tasks. We claim that playing can improve learning performance. We show that playing can reduce the textual complexity of target concepts which in turn reduces the sample complexity of a learner. We implement our idea in *Playgol*, a new inductive logic programming system. We experimentally test our claim on two domains: robot planning and real-world string transformations. Our experimental results suggest that playing can substantially improve learning performance. We think that the idea of playing (or, more verbosely, *unsupervised bootstrapping for supervised program induction*) is an important contribution to the problem of developing program induction approaches that self-discover BK.

4.7 Inductive Programming and Constraint Learning for Automated Data Science

Luc De Raedt (KU Leuven, BE)

Joint work of Luc De Raedt, Hendrik Blockeel, Samuel Kolb, Andrea Passerini, Stefano Teso, Gust Verbruggen
Main reference Luc De Raedt, Hendrik Blockeel, Samuel Kolb, Stefano Teso, Gust Verbruggen: "Elements of an Automatic Data Scientist", in Proc. of the Advances in Intelligent Data Analysis XVII – 17th International Symposium, IDA 2018, 's-Hertogenbosch, The Netherlands, October 24-26, 2018, Proceedings, Lecture Notes in Computer Science, Vol. 11191, pp. 3–14, Springer, 2018.
URL https://doi.org/10.1007/978-3-030-01768-2_1

The SYNTH approach on automated data science was presented. It is centred around a simple but non-trivial "autocompletion" setting for automating data science. Given are a set of worksheets in a spreadsheet and the goal is to automatically complete some values. The SYNTH approach has several components that are based on inductive programming. This includes the data wrangling work (that was presented by Gust Verbruggen, this volume) and the work on constraint learning. This talk focussed especially on inductively learning constraints from examples. Although constraints are ubiquitous in artificial intelligence, there are only few approaches that learn them.

References

1 De Raedt, Luc, Hendrik Blockeel, Samuel Kolb, Stefano Teso, and Gust Verbruggen. "Elements of an Automatic Data Scientist." In International Symposium on Intelligent Data Analysis, LNCS 11191, pp. 3-14. Springer, Cham, 2018.

2 Verbruggen, Gust, and De Raedt, Luc. Automatically wrangling spreadsheets into machine learning data formats. In International Symposium on Intelligent Data Analysis, LNCS 11191, pp. 367-379. Springer, Cham, 2018.
3 De Raedt, Luc, Passerini, Andrea and Teso, Stefano. Learning constraints from examples. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018

4.8 Contrastive Explanations: Status and Future

Amit Dhurandhar (IBM TJ Watson Research Center – Yorktown Heights, US)

Main reference Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Pai-Shun Ting, Karthikeyan Shanmugam, Payel Das: "Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives", CoRR, Vol. abs/1802.07623, 2018.
URL https://arxiv.org/abs/1802.07623

With the widespread adoption of AI technologies across society, explainability as a tool to understand and build trustworthy systems has become an endeavor of utmost importance. This has implications in building systems that are not only robust but also fair and accountable. In my talk I will present our recent work on contrastive explanations and along with future directions. An explanation being contrastive is considered to be the most important aspect of an explanation [1] followed by it being selective. Our method is able to achieve both where we generate contrastive explanations that are also sparse. We show how to generate these explanations for images as well as tabular data for complex models such as deep neural networks. We then generalize our approach to be also applicable in model agnostic settings. Generating such explanations for text is part of future work given that we want to change a piece of text by minimal amount, yet create semantically correct text that lies in a different class. An extremely interesting and ambitious direction is to learn boolean features using our explanations and create a simple model using them that can potentially replicate a complex models performance.

References

1 Christoph Molnar. *Interpretable Machine Learning*. Creative Commons License, USA, 2019.
2 Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, Payel Das. *Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives*. Advances of Neural Inf. Proc. Systems, 2018.
3 Ronny Luss, Pin-Yu Chen, Amit Dhurandhar, Prasanna Sattigeri, Chun-Chen Tu and Karthikeyan Shanmugam. *Generating Contrastive Explanations with Monotonic Attribute Functions*. arxiv, 2019.
4 Amit Dhurandhar, Tejaswini Pedapati, Avinash Balakrishnan, Pin-Yu Chen, Karthikeyan Shanmugam and Ruchir Puri. *Model Agnostic Contrastive Explanations Method for Structured Data*. arxiv, 2019.

4.9 Apperception

Richard Evans (Google DeepMind – London, GB) and José Hernández-Orallo (Technical University of Valencia, ES)

This work is an attempt to answer a central question in unsupervised learning: what does it even mean to "make sense" of a sensory sequence? Imagine a machine, equipped with sensors, receiving a stream of sensory information. It must, somehow, make sense of this stream of sensory data. But what does it mean, exactly, to "make sense" of sensory data? We have an intuitive understanding of what is involved in making sense of the sensory stream – but can we specify precisely what is involved? Can this intuitive notion be formalized?

In this paper, we make two contributions. First, we provide a precise formalization of what it means to "make sense" of a sensory sequence. According to our definition, making sense of a sensory sequence involves constructing a symbolic causal theory that explains the sensory sequence and satisfies a set of unity conditions that were inspired by Kant's discussion of the "synthetic unity of apperception" in the Critique of Pure Reason. According to our interpretation, making sense of sensory input is a type of program synthesis, but it is unsupervised program synthesis.

Our second contribution is a computer implementation, the Apperception Engine, that was designed to satisfy our requirements for making sense of a sensory sequence. Our system is able to produce interpretable human-readable causal theories from very small amounts of data, because of the strong inductive bias provided by the Kantian unity constraints. A causal theory produced by our system is able to predict future sensor readings, as well as retrodict earlier readings, and "impute" (fill in the blanks of) missing sensory readings. In fact, it is able to do all three tasks simultaneously.

We tested the engine in a diverse variety of domains, including cellular automata, rhythms and simple nursery tunes, multi-modal binding problems, occlusion tasks, and sequence induction IQ tests. In each domain, we test our engine's ability to predict future sensor values, retrodict earlier sensor values, and impute missing sensory data.The Apperception Engine performs well in all these domains, significantly out-performing neural net baselines. These results are significant because neural nets typically struggle to solve the binding problem (where information from different modalities must somehow be combined together into different aspects of one unified object) and fail to solve occlusion tasks (in which objects are sometimes visible and sometimes obscured from view). We note in particular that in the sequence induction IQ tasks, our system achieved human-level performance. This is notable because the Apperception Engine was not designed to solve these IQ tasks; it is not a bespoke hand-engineered solution to this particular domain. Rather, it is a general purpose apperception system for making sense of *any* sensory sequence, that just happens to be able to solve these IQ tasks out of the box.

4.10 Machine Teaching with P3

Cesar Ferri Ramirez (Technical University of Valencia, ES) and José Hernández-Orallo (Technical University of Valencia, ES)

Joint work of Jan Arne Telle, José Hernández-Orallo, Cèsar Ferri
Main reference Jan Arne Telle, José Hernández-Orallo, Cèsar Ferri: "The teaching size: computable teachers and learners for universal languages", Machine Learning, Vol. 108(8-9), pp. 1653–1675, 2019.
URL http://dx.doi.org/10.1007/s10994-019-05821-2

Machine Teaching is a problem based on finding a minimal subset of examples (or witness set) from which a learner can induce a concept. Traditionally, machine teaching techniques employ the notion of "teaching dimension". This dimension is linked to a concept and is defined as the minimum cardinality of a witness set for learning the concept. In this work, we present some limitations of the notion of teaching dimension, and, motivated by these drawbacks, we introduce the definition of "teaching size" that expresses the the size of the shortest witness set needed to identify the concept. We study some theoretical properties of this definition in the general machine teaching framework. Experiments over as on a simple Turing-complete language are used to show important differences between the teaching dimension and the teaching size. Concretely, we use P3, a version of P", a primitive programming language introduced in 1964 by Corrado Böhm. P3 is Turing complete and uses just 7 instructions. We present some interesting results about the relationship between the teaching dimension and the teaching size when inducing simple concepts in the P3 language.

Acknowledgments. This research was supported by the EU (FEDER) and the Spanish MINECO (RTI2018-094403-B-C32), Universitat Politècnica de València (PAID-06-18), and the Generalitat Valenciana (PROMETEO/2019/098 and BEST/2018/027). J. Hernández-Orallo is also funded by FLI (RFP2-152).

4.11 A potpourri of things we've worked on that at some point seemed to be relevant for this workshop.

Johannes Fürnkranz (TU Darmstadt, DE)

Joint work of Johannes Fürnkranz, Eneldo Loza Mencía, Tomáš Kliegr

Having not been active in inductive programming for several years, I arrived at this workshop with the predisposition to listen and learn. I was surprised to find that several discussion directly related to work that we have been doing in the past, and gave a spontaneous talk that summarized some of these results. The issues that we touched upon included inductive programming, interpretability and rule learning, learning in games, and multi-label rule learning.

To inductive programming, I could not add much, but was surprised that an ancient, long-forgotten paper of mine popped up during dinner conversations [1].

Interpretability is currently a very fashionable topic in machine learning. Our work particularly focuses on the aspect that although rules are inherently comprehensible, in the sense that they can be read and interpreted, their interpretability should nevertheless not be taken for granted and requires a deeper investigation. In particular, we have challenged the

view that shorter rules should be preferred and argued instead that in many cases, longer rules may yield better explanations, even in cases when the discriminative power of both rules is approximately the same [8]. We have shown in a crowd-sourcing study that humans do not necessarily prefer shorter rules [3], and have argued for an interpretability bias in rule learning algorithms [2].

In multi-label rule learning, the key challenge is that labels can occur both in the head of the rules, but also as potential inputs in the body of the rules [5]. This gives rise to several interesting challenges that appear in similar form in ILP, such as multi-predicate learning and the challenge of learning mutually recursive rules. Our work so far includes adaptations of the covering strategy [4] as well as pruning techniques for efficiently determining the best multi-label head to a rule body [7].

Finally, a significant portion of the discussion revolved about chess endgame and strategy learning as a test bed for inductive programming, which reminded me of (unpublished) work I did in the mid-90s. Since then, we have continued to work on strategy learning in chess, albeit not relying on IP. For example, we have attempted to learn evaluation functions for players of different strengths, where some of the qualitative results seemed reasonable (important positional features received more recognition by stronger players, whereas material values were approximately evaluated the same), but the overall playing strength was found to be dominated by the search algorithm [6]. The adaptation of search strategies to different playing strengths is still an open problem. In another work, we used preference learning to learn evaluation functions from annotated chess games [9].

References

1 Johannes Fürnkranz: Dimensionality reduction in ILP: A call to arms. In L. De Raedt and S. Muggleton (eds.) Proceedings of the IJCAI-97 Workshop on Frontiers of Inductive Logic Programming, pp. 81–86, Nagoya, Japan, 1997.

2 Johannes Fürnkranz, Tomáš Kliegr: The Need for Interpretability Biases. Proc. IDA 2018: 15-27

3 Johannes Fürnkranz, Tomáš Kliegr, Heiko Paulheim: On Cognitive Preferences and the Interpretability of Rule-based Models. CoRR abs/1803.01316 (2018)

4 Eneldo Loza Mencía, Frederik Janssen: Learning rules for multi-label classification: a stacking and a separate-and-conquer approach. Machine Learning 105(1): 77-126 (2016)

5 Eneldo Loza Mencía, Johannes Fürnkranz, Eyke Hüllermeier, Michael Rapp: Learning Interpretable Rules for Multi-label Classification. In Escalante H. J., Escalera S., Guyon I., Baró X., Güclütürk Y., Güclü U., van Gerven M. A. J., van Lier R. (eds.) Explainable and Interpretable Models in Computer Vision and Machine Learning, Springer Series on Challenges in Machine Learning, Springer-Verlag, 2018

6 Philipp Paulsen and Johannes Fürnkranz: A moderately successful attempt to train chess evaluation functions of different strengths. In Proceedings of the ICML-10 Workshop on Machine Learning in Games, Haifa, Israel, 2010.

7 Michael Rapp, Eneldo Loza Mencía, Johannes Fürnkranz: Exploiting Anti-monotonicity of Multi-label Evaluation Measures for Inducing Multi-label Rules. Proc. PAKDD (1) 2018: 29-42

8 Julius Stecher, Frederik Janssen, Johannes Fürnkranz: Shorter Rules Are Better, Aren't They? Proc. DS 2016: 279-294

9 Christian Wirth, Johannes Fürnkranz: On Learning From Game Annotations. IEEE Trans. Comput. Intellig. and AI in Games 7(3): 304-316 (2015)

4.12 What is (Machine) Teaching with Universal Languages? PbE, IP or more?

José Hernández-Orallo (Technical University of Valencia, ES)

Joint work of José Hernández-Orallo, Jan Arne Telle
Main reference José Hernández-Orallo, Jan Arne Telle: "Finite Biased Teaching with Infinite Concept Classes", CoRR, Vol. abs/1804.07121, 2018.
URL http://arxiv.org/abs/1804.07121

One classical view of Inductive Logic Programming (ILP) places it at the intersection between Logic and Learning, while Inductive Programming (IP) would be at the intersection between Programming and Learning. In both cases, there are two remarkable advantages over some other inductive techniques and general machine learning methods: learning can work from very few examples and the models are interpretable. However, when we look at the range of learning scenarios, they go from the common situation where there is no control of the data by the learner, as in many machine learning applications, to situations where the learner has some control (active learning and reinforcement learning), but also to situations where there is a teacher that can select these examples. (Note that learning by demonstration can fall in any of these three categories). Interestingly, when we have a teacher (or user) that wants to demonstrate a concept or a procedure (to a machine), we have a teaching situation, and if coupled with a representation based on programs, we end up in the area of Programming by Example (PbE), the learner is able to write the program that has the behaviour that is shown by a few examples carefully selected by a teacher (or user). In the presentation I gave a short introduction to the area of machine teaching, which has derived theoretical results about the efficiency of teaching in terms of the teaching dimension, defined as the minimum number of examples that are required so that the learner identifies the concept unequivocally. I argued that in the machine teaching field there were no results for rich languages, defined as those that manipulate structured objects and recursions, which are those that are common in PbE, ILP and IP. I then presented the key ideas of a recent paper [1] that shows results for Turing Machines and Finite State Automata for the expected value of a variant of the teaching dimension that assumes a strong learning prior based on simplicity and a similar sampling prior for the teacher. We make the priors explicit and show why they are important, arguing that we will not teach computers efficiently if we cannot align priors. As potential applications, we do not need to consider both learner and teacher as machines. So we could look at teaching from a more general perspective: theoretical or computational teaching rather than machine teaching. If the learner is a machine and the teacher is a human, we have a common situation where humans have to teach computers, and in the case of inductive programming this is fully identified with PbE, where we can now better understand what examples the human should select and why. If the learner is a human and the teacher is a machine, we are in an explanation situation, and this can be used to understand how key examples must be chosen so that the human can identify the concept or behaviour that is hidden behind the machine. Finally, some of the problems that we found appear because we ignore the size of the examples, which finally leads us to the new notion of teaching size, introduced in a different paper [2] and presentation.

Acknowledgments. This research was supported by the EU (FEDER) and the Spanish MINECO (RTI2018-094403-B-C32), Universitat Politècnica de València (PAID-06-18), and the Generalitat Valenciana (PROMETEO/2019/098 and BEST/2018/027). J. Hernández-Orallo is also funded by FLI (RFP2-152).

References

1 José Hernández-Orallo and Jan Arne Telle: "Finite Teaching in Expectation with Infinite Concept Classes", 2019

2 Jan Arne Telle, José Hernández-Orallo and Cesar Ferri: "The Teaching Size: Computable Teachers and Learners for Universal Languages", 2019

4.13 Can Meta-Interpretive Learning outperform Deep Reinforcement Learning of Evaluable Game Strategies?

Céline Hocquette (Imperial College London, GB) and Stephen H. Muggleton (Imperial College London, GB)

Main reference Stephen H. Muggleton, Céline Hocquette: "Machine Discovery of Comprehensible Strategies for Simple Games Using Meta-interpretive Learning", New Generation Comput., Vol. 37(2), pp. 203–217, 2019.
URL https://doi.org/10.1007/s00354-019-00054-2

World-class human players have been outperformed in a number of complex two person games (Go, Chess, Checkers) by Deep Reinforcement Learning systems. However, owing to tractability considerations minimax regret of a learning system cannot be evaluated in such games. In this work we consider simple games (Noughts-and-Crosses and Hexapawn) in which minimax regret can be efficiently evaluated. We use these games to compare Cumulative Minimax Regret for variants of both standard and deep reinforcement learning against two variants of a new Meta-Interpretive Learning system called MIGO. In our experiments all tested variants of both normal and deep reinforcement learning have worse performance (higher cumulative minimax regret) than both variants of MIGO on Noughts-and-Crosses and Hexapawn. Additionally, MIGO's learned rules are relatively easy to comprehend, and are demonstrated to achieve significant transfer learning in both directions between Noughts-and-Crosses and Hexapawn.

4.14 DreamCoder: Growing Deep Domain Expertise with Wake/Sleep Program Learning

Kevin Ellis (MIT – Cambridge, US)

Joint work of Catherine Wong, Max Nye, Mathias Sablé Meyer, Lucas Morales, Luke Hewitt, Joshua B. Tenenbaum, Armando Solar-Lezama

Human domain expertise hinges upon both explicit, declarative concepts and implicit, procedural skill in deploying those concepts to solve new problems. We present a computational model that jointly acquires both these kinds of domain expertise. The model represents solutions to problems as programs, meaning that its domain-expertise takes the form of knowledge about how to write code. It learns this domain expertise through a wake/sleep algorithm, alternating between writing code (during waking) and improving its declarative knowledge (during sleep) by growing out a library of reused library routines. During sleep the model also trains a neural network on randomly generated programs, or "dreams", where the network is trained to guide program search, capturing features of implicit procedural skill in writing code. The learned library routines build on one another, forming a deep hierarchy of explicit declarative knowledge.

4.15 Facilitating research on explainability of (RDF) rule learning: Interactive software systems and crowdsourcing studies

Tomáš Kliegr (University of Economics – Prague, CZ)

Joint work of TomášKliegr, Václav Zeman, Stanislav Vojir, Vojtech Svátek, Jaroslav Kuchar, Stepan Bahnik, Heiko Paulheim
Main reference Stanislav Vojír, Vaclav Zeman, Jaroslav Kuchar, Tomás Kliegr: "EasyMiner.eu: Web framework for interpretable machine learning based on rules and frequent itemsets", Knowl.-Based Syst., Vol. 150, pp. 111–115, 2018.
URL https://doi.org/10.1016/j.knosys.2018.03.006

This talk covered several interactive academic machine learning software systems based on association rule learning, which is an algorithmic approach for generating exhaustive sets of rules from data that meet user-specified quality requirements. We also report on algorithmic research aimed at reduction of size of rule models, and on cognitive experiments focused on understanding interpretability of rules discovered with association rule learning. Finally, subject of ongoing work is finalization of a rule editor aimed at cognitive science experiments with rules. This will allow to transition from questionnaire-based data elicitation setup to a more realistic setting.

EasyMiner [6] is an interactive rule learning system for discovering association rules. EasyMiner uses the Classification-based on Associations algorithm (CBA) [1] to generate rule-based classification models from tabular data. A comparison of CBA with follow-up association rule classification algorithms shows that CBA provides very good balance between speed of learning, predictive performance, and understandability of the resulting models. Possibly the biggest limitation of CBA is that it tends to generate larger models in terms of rule count than some related algorithms. We presented the Quantitative CBA algorithm (QCBA) [3], which makes CBA models smaller by recovering some of the information lost during discretization of numeric attributes.

RDF Rules [7] is a framework for extracting horn clauses from RDF-style knowledge bases that is based on the the AMIE+ [2] algorithm. Experiments presented in [2] show that AMIE+, an association rule learning approach, can be orders of magnitude faster than ALEPH, which is based on principles of Inductive Logic Programming (ILP). The main enhancements in RDF Rules compared to AMIE+ include support for preprocessing of numerical data, top-k approach and a new pattern language for limiting the search space. These new features make the framework more practical to use and even faster on some types of tasks. The RDF Rules reference implementation also contains a web-based user interface.

The second part of the presentation was devoted to on-going experimental research on explainability of rule models. We briefly summarized two working papers. The first paper [5] reviews possible effects of about twenty cognitive biases on interpretation of rule-based machine learning models. The second paper [4] reports on several user studies aimed at assessing effect of selected psychological phenomena on plausibility of learnt rules. There was also a brief demo of currently developed functionality of EasyMiner, which is a rule editor with features specifically designed for crowdsourced experiments on explainability.

A discussion followed on the possibilities for adapting presented association rule learning frameworks for current challenges facing ILP.

References

1 Ma, Bing Liu Wynne Hsu Yiming, Bing Liu, and Yiming Hsu. "Integrating classification and association rule mining." Proceedings of the fourth international conference on knowledge discovery and data mining. 1998.

2 Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. The VLDB Journal **24**(6), 707–730 (2015)
3 Kliegr, Tomáš. "Quantitative CBA: Small and Comprehensible Association Rule Classification Models." arXiv preprint arXiv:1711.10166 (2017).
4 Fürnkranz, Johannes, Tomáš Kliegr, and Heiko Paulheim. "On Cognitive Preferences and the Plausibility of Rule-based Models." arXiv preprint arXiv:1803.01316 (2018).
5 Kliegr, Tomáš, Štěpán Bahník, and Johannes Fürnkranz. "A review of possible effects of cognitive biases on interpretation of rule-based machine learning models." arXiv preprint arXiv:1804.02969 (2018).
6 Vojir, Stanislav, Zeman, Vaclav, Kuchar, Jaroslav, Kliegr, Tomáš: Easyminer.eu: Web framework for interpretable machine learning based on rules and frequent itemsets. Knowledge-Based Systems **150**, 111–115 (2018)
7 Zeman, Václav, Tomáš Kliegr, and Vojtech Svátek.: RdfRules Preview: Towards an Analytics Engine for Rule Mining in RDF Knowledge Graphs. No. 478. CEUR-WS, 2018.

4.16 Inductive Learning of Answer Set Programs

Mark Law (Imperial College London, GB)

Joint work of Mark Law, Alessandra Russo, Krysia Broda
Main reference Mark Law: "Inductive learning of answer set programs", 2018.
URL https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.762179

In recent years, non-monotonic Inductive Logic Programming (ILP) has received growing interest. Specifically, several new learning frameworks and algorithms have been introduced for learning under the answer set semantics, allowing the learning of common-sense knowledge involving defaults and exceptions, which are essential aspects of human reasoning.

The first part of this talk will present our recent advances which have extended the theory of ILP and yielded a new collection of algorithms, called ILASP (Inductive Learning of Answer Set Programs), which are able to learn ASP programs consisting of normal rules, choice rules and both hard and weak constraints. Learning such programs allows ILASP to be applied in settings which had previously been outside the scope of ILP. In particular, weak constraints represent preference orderings, and so learning weak constraints allows ILASP to be used for preference learning.

In the second part, we will present a recent noise-tolerant version of ILASP, which has been successful at learning both from synthetic and from real data sets. In particular, we have shown that on many of the data sets ILASP achieves a higher accuracy than other ILP systems that have previously been applied to those same data sets.

4.17 Representing and Learning Grammars in Answer Set Programming

Mark Law (Imperial College London, GB)

Joint work of Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, Jorge Lobo
Main reference Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, Jorge Lobo: "Representing and Learning Grammars in Answer Set Programming", in Proc. of the The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 – February 1, 2019., pp. 2919–2928, AAAI Press, 2019.
URL https://aaai.org/ojs/index.php/AAAI/article/view/4147

In this paper, we introduce an extension of context-free grammars called answer set grammars (ASGs). These grammars allow annotations on production rules, written in the language of Answer Set Programming (ASP), which can express context-sensitive constraints. We investigate the complexity of various classes of ASG with respect to two decision problems: deciding whether a given string belongs to the language of an ASG and deciding whether the language of an ASG is non-empty. Specifically, we show that the complexity of these decision problems can be lowered by restricting the subset of the ASP language used in the annotations. To aid the applicability of these grammars to computational problems that require context-sensitive parsers for partially known languages, we propose a learning task for inducing the annotations of an ASG. We characterise the complexity of this task and present an algorithm for solving it. An evaluation of a (prototype) implementation is also discussed.

4.18 Using Association Rule Learning for Verification of Configuration Files

Ruzica Piskac (Yale University – New Haven, US)

Joint work of Mark Santolucito, Ennan Zhai, Rahul Dhodapkar, Aaron Shim, Ruzica Piskac
Main reference Mark Santolucito, Ennan Zhai, Rahul Dhodapkar, Aaron Shim, Ruzica Piskac: "Synthesizing configuration file specifications with association rule learning", PACMPL, Vol. 1(OOPSLA), pp. 64:1–64:20, 2017.
URL https://doi.org/10.1145/3133888

Traditionally software synthesis is trying to automatically derive code that corresponds to some specification. The specification can be given explicitly, or it is stated in the form of given input-output examples illustrating the intentional code behavior. However, sometimes the main challenge is actually to derive the specification itself.

Using verification for configuration files as our main motivation, we learn specification from a given set of configuration files. This set might also contain faulty configuration files. Software failures resulting from configuration errors have become commonplace as modern software systems grow increasingly large and more complex.

We describe a framework which analyzes data sets of correct configuration files and learns rules for building a language model from the given data set. Our framework is based on a generalized association rule learning.

4.19 Generating Explanations for IP Learned Models

Ute Schmid (Universität Bamberg, DE)

Inductive (logic) programming provides interpretable, human-inspectable, explicitly represented models. However, a symbolic model per se might not be easily comprehensible for humans, especially such humans without a background in programming. To make models comprehensible to domain experts and other end-users, it is therefore necessary to explain model decisions. I will argue that a variety of explanation modes – verbal, visual, and example-based – should be made available to accomodate different personal and situational needs. A current challenge is to combine verbal explanations with visual information. Here we explore different strategies to automatically infer perceptual/spatial features and relations which can be used in the context of symbolic models.

References

1 Schmid, Ute (2018). Inductive Programming as Approach to Comprehensible Machine Learning. Proceedings of the 6th Workshop KI & Kognition (KIK-2018), co-located with 41st German Conference on Artificial Intelligence (KI 2018), Berlin, Germany, September 25, 2018. http://ceur-ws.org/Vol-2194/schmid.pdf

4.20 Semantically Aware Data Wrangling

Gust Verbruggen (KU Leuven, BE) and Luc De Raedt (KU Leuven, BE)

Joint work of Gust Verbruggen, Luc De Raedt, Lidia Contreras-Ochando, Cesar Ferri, José Hernández-Orallo
Main reference Gust Verbruggen, Luc De Raedt: "Automatically Wrangling Spreadsheets into Machine Learning Data Formats", in Proc. of the Advances in Intelligent Data Analysis XVII – 17th International Symposium, IDA 2018, 's-Hertogenbosch, The Netherlands, October 24-26, 2018, Proceedings, Lecture Notes in Computer Science, Vol. 11191, pp. 367–379, Springer, 2018.
URL http://dx.doi.org/10.1007/978-3-030-01768-2_30

We show how to use predictive synthesis for automatically wrangling spreadsheets into machine learning formats, introduce semantic spreadsheet segmentation as a way to provide feedback during wrangling and explore how such a predictive synthesizer can be trained from scratch.

5 Discussion groups

5.1 Neuro-symbolic integration

Richard Evans (Google DeepMind – London, GB), Eli Bingham (Uber AI Labs – San Francisco, US), Eneldo Loza Mencía (TU Darmstadt, DE), Harald Ruess (fortiss GmbH – München, DE), Johannes Rabold (Universität Bamberg, DE), Kevin Ellis (MIT – Cambridge, US), Ute Schmid (Universität Bamberg, DE)

We discussed two different approaches to neuro-symbolic integration. In the first, we try to extract interpretable features from standard neural networks (MLPs, CNNs, RNNs). In the second, we implement a hybrid neuro-symbolic architecture that was designed in advance to allow extraction of interpretable features.

When considering the first approach, standard neural networks tend to suffer from what McCarthy called the "propositional fixation": insofar as they learn rules at all, they tend to be rules of propositional logic, not first-order logic. This means the rules do not generalize well. E.g. you train an RNN to reverse a list of length 5, and test it on lists of length 7, and it fails. This means that if we want to extract logical information from a standard neural network, it should be propositional logic, not first-order logic. So we should look at extracting propositional rules, decision-trees, or state-machines – not full first-order rules.

Related work on this approach includes: LIME, LRP (Layerwise Relevance Propagation), www.heatmapping.org, prototype-based neural network layers: https://arxiv.org/abs/1812.01214, neural stethoscopes: https://arxiv.org/abs/1806.05502, interpretable CNNs: https://arxiv.org/abs/1901.02413, and extracting decision trees from NNs: https://papers.nips.cc/paper/1152-extracting-tree-structured-representations-of-trained-networks.pdf.

In the second approach, the key question is: what form should the hybrid architecture take? One homogenous system for both low-level features and high-level rules, or two distinct systems that communicate? We considered various options. (1) End-to-end differentiable. Create a differentiable ILP system and learn the high-level rules and the low-level classifiers jointly using SGD. Example: https://arxiv.org/abs/1711.04574. (2) End-to-end SMT. Model the neural classifiers in SMT and learn the high-level rules and low-level classifiers jointly using SMT. Example: https://homes.cs.washington.edu/~bornholt/post/nnsmt.html. (3) End-to-end SAT solver. Binarize the neural net classifier and learn the high-level rules and low-level classifiers jointly using SAT. Example of using SAT for binarized networks: https://arxiv.org/pdf/1710.03107.pdf. (4) Two distinct systems: we have a neural net for classifiers and a separate symbolic ILP system for learning rules. We hope the classifications learned by the net are useful for the ILP system.

Related work for the second approach includes:

- DeepProbLog: https://arxiv.org/pdf/1805.10872.pdf
- Learning Explanatory Rules from Noisy Data: https://arxiv.org/abs/1711.04574
- The Neural Theorem Prover: https://arxiv.org/pdf/1705.11040.pdf
- Neural-symbolic integration: https://arxiv.org/abs/1711.03902
- NeSy: http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=74066&copyownerid=108715
- Neural Logic Machines: https://openreview.net/forum?id=B1xY-hRctX

- Combining LIME with ILP for relational explanations: https://link.springer.com/chapter/10.10072F978-3-319-99960-9_7
- Investigating human priors for general game playing: https://arxiv.org/abs/1802.10217

5.2 Language primitive bias engineering for generality

José Hernández-Orallo (Technical University of Valencia, ES), Eli Bingham (Uber AI Labs – San Francisco, US), Lidia Contreras-Ochando (Technical University of Valencia, ES), Andrew Cropper (University of Oxford, GB), Kevin Ellis (MIT – Cambridge, US), Tomáš Kliegr (University of Economics – Prague, CZ), Michael Siebers (Universität Bamberg, DE), and Gust Verbruggen (KU Leuven, BE)

In this working group we explored different ways in which a knowledge base or library of predicates or functions (we will use the term primitive) can be improved so that we get results for a general range of problems, by the introduction or invention of new ones, their abstraction or by the selection (or forgetting) of those that are least useful. Note that this is different from the common situation in which we choose the appropriate domain knowledge for a particular example, or the most appropriate schemas or meta-rules for one or more problems. Here we want to explore the way in which the primitives that can be used to reduce the depth of the search space in a learning problem, while keeping breadth at a reasonable level. All this is related to some ideas in psychology or AI (empowerment, curiosity, intrinsic motivation, meta-learning, self-play, etc.) and compositionality (and of course learning to learn).

We started with a discussion on some novel ways in which the use of self-generation or playing with tasks and examples can make these libraries improve. Note that by sampling tasks and "playing" with them we can improve the bias, even if we are not given domain knowledge. Basically we can create new more abstract primitives that we can reuse and the system is better learning new (and especially more complex) problems. This is possible because there are many redundant programs and a few components appear in many programs, especially those that compress the examples, and also because search cost has breadth as the base and depth in the exponent.

For the approach of generating a sample and use learning as a way of improving the library, we addressed a few issues during the discussion:

- What we sample. There are two main alternatives: to sample on the data (problem space) or by sampling programs (solution space) and using them to get the data. If we sample the data, we can use a uniform distribution on limited-size inputs and outputs (e.g., Andrew's paper [1]), or some other distribution for more complex cases (interactive settings). If we sample the programs, we can use a distribution that is inspired by the domain (e.g., a stochastic grammar, Kevin's paper [2]), or we can use a different distribution, but still the system can learn to learn (or we can use a language for generating programs that is different from the language we are using for learning them). In both cases if we adjust the bias to be more efficient for the generated data, we have a system that gets better incrementally. But would this converge?
- How can we do this sampling better than random? This is motivated because even playing has a cost, and we don't want to play (or mind play) with millions of examples.

The distribution may generate very similar cases again and again: this is the same as in sampling theory where we want to cover the space with very few instances. One idea is a diversity of tasks. How can diversity be measured?

- What are good examples for trying this? There are potential problem domains suitable for playing, and building new Lego designs seemed especially promising. Rebrickable (https://rebrickable.com/) is a web site with lots of existing Lego build instructions. A similar (simpler) setting is Build Battle in Malmo, where different configurations of bricks have to be reproduced by the agent (see, e.g., http://microsoft.github.io/malmo/blog/BuildBattle/Introduction/)

Then we explored the actual ways in which we can improve the (use of) the background knowledge (or the library)? We discussed whether forgetting (dropping least useful primitives or predicates) could be useful. Where in humans and other previous studies (e.g., gErl, [3]) it works. In other systems (e.g., Metagol, [1]) it doesn't help much. There may be some reasons for this, such as noise, or perhaps the fact that some of these systems are simply robust to overfitting. Also, we remember that breadth is on the base and not the exponent of the search cost.

So perhaps this is why ranking or selecting primitives by relevance is more useful and general than forgetting ([2], [4]). One interesting question in this approach is whether we need two "modules": one for solving the learning problem and another one for calculating the relevance? Could we think of an integrated system instead? Or something behind, such as causal model of the world that could exclude many primitives or knowledge rules?

In any case, the choice of relevance for this process is key. Do we agree on the metrics for the second module, i.e., for relevance? There are dependencies (e.g., primitive A is useless if you introduce primitive B) so a single score (as the one used for the rankings) is a simple option, but some other structures could be more powerful (a relevance graph with dependencies). Apart from the relevance metrics, we want metrics to analyse if the knowledge we extract gets more coherent or abstract, or more explainable.

As an action from this group we suggested to explore metrics for relevance (e.g., a primitive is relevant if removing it makes search deeper, a primitive is relevant if adding it makes average depth shorter) and generality (e.g., a primitive is generally useful = aggregation of relevance for a wide range of problems).

So solving this optimisation problem depends on finding the size (and composition) of this set of primitives in terms of the time (or effort, measured in bits of the solution) for solving a range of problems (see Figure 2).

Acknowledgments. This research was supported by the EU (FEDER) and the Spanish MINECO (RTI2018-094403-B-C32), Universitat Politècnica de València (PAID-06-18), and the Generalitat Valenciana (PROMETEO/2019/098 and BEST/2018/027). L. Contreras-Ochando was also supported by the Spanish MECD (FPU15/03219). J. Hernández-Orallo is also funded by FLI (RFP2-152).

References

1 Cropper, Andrew. "Playgol: learning programs through play." arXiv preprint arXiv:1904.08993 (2019).

2 Ellis, Kevin, et al. "Dreamcoder: Bootstrapping domain-specific languages for neurally-guided bayesian program learning." Proceedings of the 2nd Workshop on Neural Abstract Machines and Program Induction (2018).

3 Martínez-Plumed, Fernando, et al. "Knowledge acquisition with forgetting: an incremental and developmental setting." Adaptive Behavior 23.5 (2015): 283-299.

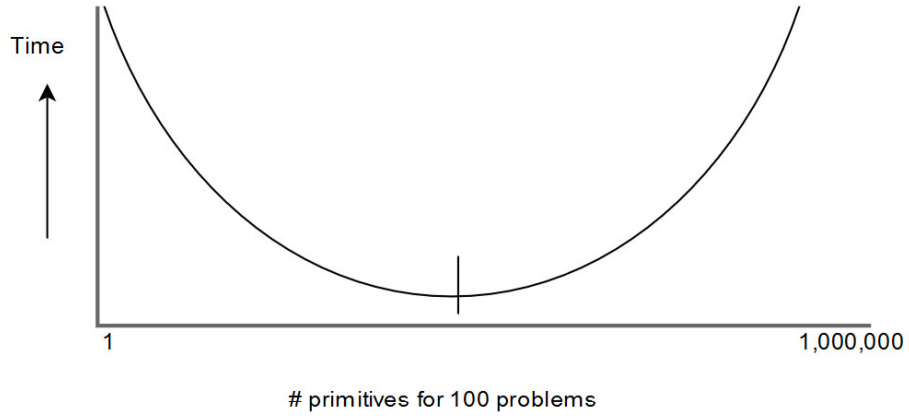

Figure 2 The plot shows that the optimal curve has a minimum, but we may only discover/approximate a curve above (through sampling or assuming a strong prior) that may have a different minimum.

4 Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., Ramírez-Quintana, M. J., Katayama, S.: Automated data transformation with inductive programming and dynamic background knowledge. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2019 (to appear). ECML-PKDD '19 (2019).

5.3 Game Strategy Discussion Group Report

Stephen H. Muggleton (Imperial College London, GB), Ivan Bratko (University of Ljubljana, SI), Johannes Fürnkranz (TU Darmstadt, DE), Céline Hocquette (Imperial College London, GB), Mark Law (Imperial College London, GB), and Stassa Patsantzis (Imperial College London, GB)

Main reference Stephen H. Muggleton, Céline Hocquette: "Machine Discovery of Comprehensible Strategies for Simple Games Using Meta-interpretive Learning", New Generation Comput., Vol. 37(2), pp. 203–217, 2019.
URL http://dx.doi.org/10.1007/s00354-019-00054-2

We are interested into learning strategies for a variety of games, that can be played with one or several players. Chess endgames are rich and challenging enough which allow to demonstrate convincing results. Another advantage is that minimax databases can be computed and provide a basis for learning, and a baseline for comparison. Examples of such endgames are KRK or KPRKR. Games are equivalent to fully quantified boolean formula.

Determining a strategy is proving the truth of such sentences, for instance with SAT-solving approaches.Incomplete information games such as card games can be solved following an extension of the minimax algorithm. A minimax game tree can be build for each possible world. The strategy is then to minimize the expectation of the regret over several worlds. Another categories of games interesting for the AI community is video games, for which a standing example is the Atari games.

Participants

- Eli Bingham
Uber AI Labs – San Francisco, US
- Ivan Bratko
University of Ljubljana, SI
- Maurice Chandoo
Leibniz Universität Hannover, DE
- Lidia Contreras-Ochando
Technical University of Valencia, ES
- Andrew Cropper
University of Oxford, GB
- Luc De Raedt
KU Leuven, BE
- Amit Dhurandhar
IBM TJ Watson Research Center – Yorktown Heights, US
- Kevin Ellis
MIT – Cambridge, US
- Richard Evans
Google DeepMind – London, GB
- Cesar Ferri Ramirez
Technical University of Valencia, ES
- Johannes Fürnkranz
TU Darmstadt, DE
- Elena Leah Glassman
Harvard University – Cambridge, US
- José Hernández-Orallo
Technical University of Valencia, ES
- Céline Hocquette
Imperial College London, GB
- Tomáš Kliegr
University of Economics – Prague, CZ
- Mark Law
Imperial College London, GB
- Eneldo Loza Mencía
TU Darmstadt, DE
- Stephen H. Muggleton
Imperial College London, GB
- Stassa Patsantzis
Imperial College London, GB
- Ruzica Piskac
Yale University – New Haven, US
- Johannes Rabold
Universität Bamberg, DE
- Harald Ruess
fortiss GmbH – München, DE
- Ute Schmid
Universität Bamberg, DE
- Michael Siebers
Universität Bamberg, DE
- Armando Solar-Lezama
MIT – Cambridge, US
- Gust Verbruggen
KU Leuven, BE