

Approaches and Applications of Inductive Programming

Luc De Raedt^{*1}, Ute Schmid^{*2}, and Johannes Langer^{†3}

1 KU Leuven, BE. luc.deraedt@cs.kuleuven.be

2 Universität Bamberg, DE. ute.schmid@uni-bamberg.de

3 Universität Bamberg, DE. johannes.langer@uni-bamberg.de

Abstract

The Dagstuhl Seminar “Approaches and Applications of Inductive Programming” (AAIP) has taken place for the sixth time. The Dagstuhl Seminar series brings together researchers concerned with learning programs from input/output examples from different areas, mostly from machine learning and other branches of artificial intelligence research, cognitive scientists interested in human learning in complex domains, and researchers with a background in formal methods and programming languages. Main topics addressed in the AAIP 2023 seminar have been neurosymbolic approaches to IP bringing together learning and reasoning, IP as a post-hoc approach to explaining decision-making of deep learning blackbox models, and exploring the potential of deep learning approaches, especially large language models such as OpenAI Codex for IP. Topics discussed in working groups were Large Language Models and inductive programming in cognitive architectures, avoiding too much search in inductive programming, finding suitable benchmark problems, and evaluation criteria for interpretability and explainability of inductive programming.

Seminar October 29 – November 3, 2023 – <https://www.dagstuhl.de/23442>

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Human-centered computing; Computing methodologies → Machine learning


Keywords and phrases explainable ai, human-like machine learning, inductive logic programming, interpretable machine learning, neuro-symbolic ai

Digital Object Identifier 10.4230/DagRep.13.10.182

1 Executive Summary

Ute Schmid (Universität Bamberg, DE)

Luc De Raedt (KU Leuven, BE)

License  Creative Commons BY 4.0 International license
© Ute Schmid and Luc De Raedt

Inductive programming (IP) is a special perspective on program synthesis, addressing learning programs from incomplete specifications such as input/output examples. The seminar “Approaches and Applications of Inductive Programming” (AAIP) took place in Dagstuhl for the sixth time. This Dagstuhl Seminar brings together researchers from different areas of artificial intelligence research, machine learning, formal methods, programming languages, cognitive science, and human-computer-interaction interested in methods and applications of IP. Focus topics of AAIP’23 have been neurosymbolic approaches to IP bringing together learning and reasoning, IP as a post-hoc approach to explaining decision-making of deep learning blackbox models, and exploring the potential of deep learning approaches, especially large language models such as OpenAI Codex for IP.

* Editor / Organizer

† Editorial Assistant / Collector



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 4.0 International license

Approaches and Applications of Inductive Programming, *Dagstuhl Reports*, Vol. 13, Issue 10, pp. 182–211

Editors: Luc De Raedt and Ute Schmid



DAGSTUHL
REPORTS

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The focus topics have been introduced and discussed in a series of talks addressing neuro-symbolic IP, IP for learning in planning, explainable AI and IP, and IP and generative AI. Furthermore, a series of talks were dedicated to the relation of cognitive science to IP: Human-like few-shot learning via Bayesian reasoning over natural language, the child as hacker, using program synthesis to model strategy diversity in human visual reasoning, a neurodiversity-inspired solver for the Abstraction and Reasoning Corpus (ARC) using visual imagery and program synthesis, and using natural language for self-programming in cognitive architectures. The relation between IP and explainability has been highlighted with talks about explainable models via compression of relational ensembles, and effects of explaining machine-learned logic programs for human comprehension and discovery. Relations between IP and knowledge based methods have been addressed in a talk about learning disjointness axioms for knowledge graph refinement and for making knowledge graph embedding methods more robust. Methods of IP as an approach to learning interpretable rules have been presented with a focus on inductive logic programming (ILP), deep-rule learning, relational program synthesis with numerical reasoning, improving rule classifiers learned from quantitative data by recovering information lost by discretisation, meta-interpretive learning for generalised planning, probabilistic inductive logic programming, abstraction for answer set programs, anti-unification and generalization, programmatic reinforcement learning, and making program synthesis fast on a GPU. These talks have been complemented by several system demos presenting the ILP systems Popper and Louise, an RDF rules learner, and learning rules to sort e-mails into folders (EmFORE).

We identified four relevant research problems for current and future research in IP which were addressed in in-depth discussions in working groups and afterwards discussed in plenary sessions: (1) Large Language Models and Inductive Programming in Cognitive Architectures: one main outcome has been that combining learning and reasoning by integrating LLMs and reasoners in a cognitive architecture could be an enabler for validating programs that get executed by the overall architecture and to possibly get nearer to human performance. (2) Avoiding too much search in Inductive Programming: It was noted that for IP in general we do need to learn structure as well as probabilities. Classic IP approaches focus on structure learning and – in contrast to neural network architectures – can learn recursion explicitly. The main result has been that suitable problem domains should be identified for systematic evaluation, such as string transformation which combine syntactic (e.g. return first letter) and semantic (e.g. give the capital of a country) transformations. (3) Finding Suitable Benchmark Problems for Inductive Programming: Here, the discussion from the second topic has been extended and systematised with the formulation of several relevant criteria for benchmark problems to evaluate IP approaches, among them problem domains which are not solvable by LLMs and solvable efficiently by humans. (4) Evaluation Criteria for Interpretability and Explainability of Inductive Programming: The main insight has been that the degree of interpretability and the quality of explanations is strongly context-dependent, being influenced by the recipient (who), the content (what), the information need and reason for an explanation (why), and the form of the explanation (how). Different candidates for metrics were identified, such as complexity measures, semantic coherence, and reliability of generated code.

In a final discussion round, several outcomes have been summarized and action points have been discussed. A crucial problem which might impact scientific progress as well as visibility could be that there is no core general approach to IP (such as gradient descent for neural networks). Relevant use cases might not have a focus on learning recursion/loops but on relations (e.g. in medicine and biology). The focus on learning programs (including

recursion) might profit from using Python as the target language instead of more specific languages such as Prolog. Furthermore, current IP systems are mostly not easy to find and to use. Providing a toolbox which can be easily used (such as Weka for standard ML) might be helpful. There was a general agreement among the participants that the format of Dagstuhl Seminars is especially fruitful for bringing together the different perspectives on IP from machine learning, cognitive science, and program language research.

2 Table of Contents

Executive Summary

<i>Ute Schmid and Luc De Raedt</i>	182
--	-----

Overview of Talks

Effects of explaining machine-learned logic programs for human comprehension and discovery <i>Lun Ai</i>	187
Making program synthesis fast on a GPU <i>Martin Berger</i>	188
Anti-unification and Generalization: What's next? <i>David Cerna</i>	189
On the Need of Learning Disjointness Axioms for Knowledge Graph Refinement and for Making Knowledge Graph Embedding Methods more Robust <i>Claudia d'Amato</i>	189
How to make logics neurosymbolic <i>Luc De Raedt</i>	190
What should we do next in ILP? <i>Sebastijan Dumančić</i>	191
Human-like Few-Shot Learning via Bayesian Reasoning over Natural Language <i>Kevin Ellis</i>	191
Towards Programmatic Reinforcement Learning <i>Nathanaël Fijalkow</i>	192
Inductive Programming for Explainable Artificial Intelligence (IP for XAI) <i>Bettina Finzel</i>	192
On Deep Rule Learning <i>Johannes Fürnkranz</i>	193
Three Learning Problems in Planning <i>Hector Geffner</i>	194
A tutorial on Popper <i>Céline Hocquette</i>	194
Relational program synthesis with numerical reasoning <i>Céline Hocquette</i>	195
On the role of natural language for self-programming in cognitive architectures <i>Frank Jäkel</i>	196
QCBA: improving rule classifiers learned from quantitative data by recovering information lost by discretisation <i>Tomáš Kliegr</i>	196
RDFrules: A Swiss knife for relational association rule learning, classification and knowledge graph completion <i>Tomáš Kliegr</i>	197

The Child as Hacker <i>Josh Rule</i>	198
Abstraction for Answer Set Programs <i>Zeynep G. Saribatur</i>	199
Explanatory Inductive Programming (XAI for IP) <i>Ute Schmid</i>	200
Explainable models via compression of tree ensembles <i>Sriraam Natarajan</i>	201
Inductive Programming meets Large Language Models <i>Gust Verbruggen</i>	202
Inductive Programming meets Real User Problems <i>Gust Verbruggen</i>	202
Probabilistic Logic Programming: Quo Vadis? <i>Felix Weitzkämper</i>	203
Working groups	
Large Language Models and Inductive Programming in Cognitive Architectures <i>Bettina Finkel and Frank Jäkel</i>	204
Avoiding too much search in Inductive Programming <i>Ute Schmid, David Cerna, and Hector Geffner</i>	204
Evaluation Criteria for Interpretability and Explainability of Inductive Programming <i>Ute Schmid, Lun Ai, Claudia d’Amato, and Johannes Fürnkranz</i>	205
Finding Suitable Benchmark Problems for Inductive Programming <i>Ute Schmid, Martin Berger, Sebastijan Dumancic, Nathanaël Fijalkow, and Gust Verbruggen</i>	207
Panel discussions	
Inductive Programming – How to Go On? <i>Ute Schmid, Claudia d’Amato, Hector Geffner, Sriraam Natarajan, and Josh Rule</i>	209
Participants	211

3 Overview of Talks

3.1 Effects of explaining machine-learned logic programs for human comprehension and discovery

Lun Ai (Imperial College London, GB)

License © Creative Commons BY 4.0 International license
© Lun Ai

Joint work of Lun Ai, Johannes Langer, Stephen H. Muggleton, Ute Schmid

Main reference Lun Ai, Johannes Langer, Stephen H. Muggleton, Ute Schmid: “Explanatory machine learning for sequential human teaching”, *Mach. Learn.*, Vol. 112(10), pp. 3591–3632, 2023.

URL <https://doi.org/10.1007/S10994-023-06351-8>

The talk focused on the assumption in the Logic Programming community: logic programs are human-comprehensible. This had resulted in very few empirical assessments on the effects of explaining machine-learned logic programs. Empirical results by the authors showed explaining logic programs do not always lead to improved human performance. In addition, the authors stressed the need for objective and operational measurements of explainability. Their results provided novel insights on the explanatory effects of curriculum order and the presence of machine-learned explanations for sequential problem-solving.

The topic of comprehensibility of machine-learned theories has recently drawn increasing attention. Inductive logic programming uses logic programming to derive logic theories from small data based on abduction and induction techniques. Learned theories are represented in the form of rules as declarative descriptions of obtained knowledge. In earlier work, the authors provided the first evidence of a measurable increase in human comprehension based on machine-learned logic rules for simple classification tasks. In a later study, it was found that the presentation of machine-learned explanations to humans can produce both beneficial and harmful effects in the context of game learning.

The talk concentrated on a most recent investigation on the effects of the ordering of concept presentations and logic program explanations. The authors proposed a framework for the effects of sequential teaching based on an existing definition of comprehensibility. This empirical study involved curricula that teach novices the merge sort algorithm. They provided performance-based and trace-based evidence for support. Results show that sequential teaching of concepts with increasing complexity (a) has a beneficial effect on human comprehension and (b) leads to human re-discovery of divide-and-conquer problem-solving strategies, and (c) allows adaptations of human problem-solving strategy with better performance when machine-learned explanations are also presented.

Several open questions were discussed during and after the talk. For instance, the audience suggested an investigation on “learning how to learn” and comparisons between the human traces and the machine learner (ILP) trace. In the context of increasing the popularity of logic programs, some challenges in higher-education curricula were discussed showing the significance of how to best design Logic Programming teaching interactions. Importantly, this talk highlighted the limitations to performance-based evaluations. This led to an extended discussion on computable and objective assessments for various perspectives of explainability.

3.2 Making program synthesis fast on a GPU

Martin Berger (University of Sussex – Brighton, GB)

License © Creative Commons BY 4.0 International license
© Martin Berger

Joint work of Mojtaba Valizadeh, Martin Berger

Main reference Mojtaba Valizadeh, Martin Berger: “Search-Based Regular Expression Inference on a GPU”, Proc. ACM Program. Lang., Vol. 7(PLDI), pp. 1317–1339, 2023.

URL <https://doi.org/10.1145/3591274>

Inductive programming is stuck!

GPUs are the work-horses of computing. Applications that fit the GPU style of programming typically run orders of magnitude faster on GPUs than on CPUs. This gives opportunities for scaling not achievable with CPUs. The recent success of deep learning amply demonstrates this. Unfortunately, large classes of applications are not known to benefit from GPU acceleration. That includes most tools in program synthesis, inductive programming, theorem proving, ... (from now on: automated reasoning) such as SAT and SMT solvers. How can we change this? Simplifying a bit, a GPU can only accelerate applications if they are “GPU-friendly”, meaning they are

- highly parallel,
- have little to no data-dependent branching, and have
- predictable data-movement, and high temporal and spatial data locality.

Algorithms in automated reasoning, as implemented today, mostly lack those properties. Many are extremely branching heavy, for example because they branch on syntactic structure. Some are seemingly sequential (e.g. unit propagation, a core step modern SAT solvers for simplifying formulae). This might be because an algorithmic problem is intrinsically sequential, or because a way of making an algorithmic problem GPU-friendly has not yet been found.

Research question: Can we identify workloads arising in industrial automatic reasoning practise, and scale them up on GPUs by developing suitable, GPU-friendly algorithms? The GPU-based algorithms should give at least 100x speedup (for comparable problem instances), and be able to handle at least 1000x bigger problem instances, both in comparison with state-of-the-art open (= non-proprietary) software for the same problem domain.

Preliminary answer, based on [1]: all program synthesis that uses the generate-and-test approach can see orders of magnitude speedup on GPUs.

Recommendation to the ILP community: stop what your are doing and implement your ideas on a GPU.

References

- 1 Mojtaba Valizadeh, Martin Berger: *Search-Based Regular Expression Inference on a GPU*. Proc. ACM Program. Lang. 7(PLDI): 1317-1339 (2023), <https://doi.org/10.1145/3591274>

3.3 Anti-unification and Generalization: What's next?

David Cerna (*The Czech Academy of Sciences – Prague, CZ*)

License © Creative Commons BY 4.0 International license
© David Cerna

Joint work of David M. Cerna, Temur Kutsia

Main reference David M. Cerna, Temur Kutsia: “Anti-unification and Generalization: A Survey”, in Proc. of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, pp. 6563–6573, International Joint Conferences on Artificial Intelligence Organization, 2023.

URL <https://doi.org/10.24963/ijcai.2023/736>

Anti-unification (AU) is a fundamental operation for the computation of symbolic generalizations useful for inductive inferencing [1]. It is the dual operation to *unification*, an operation at the foundation of automated theorem proving. In contrast to unification, where one is interested in constructing *most general unifiers (mgus)*, anti-unification is concerned with the construction of *least general generalizations (lggs)*; that is, expressions capturing the commonalities shared between members of a set of symbolic expressions.

The operation was introduced by Plotkin and Reynolds and found many applications within the area of *Inductive synthesis* and, in particular, early inductive logic programming (ILP) systems. However, since their seminal work, the number of applications has grown tremendously with uses in program analysis, program repair, automated reasoning, and beyond. With the growing number of applications, several investigations have developed anti-unification methods over various symbolic objects, such as the simply-typed lambda calculus, term graphs, and hedge expression, to name a few. In particular, there has been significant progress in understanding equational anti-unification and the cardinality of the set of solutions (set of lggs). In many cases, the solution sets are either infinitely large or do not exist (every generalization allows a more specific generalization).

We ask, is *least general generalization* the right characterization of a solution to an anti-unification problem? In particular, is there a characterization of a solution more amenable to modern approaches to inductive synthesis? Secondly, what does the inductive synthesis community need from symbolic generalization techniques, which is currently missing?

References

- 1 David M. Cerna, Temur Kutsia: *Anti-unification and Generalization: A Survey*. IJCAI 2023: 6563–6573, <https://doi.org/10.24963/IJCAI.2023/736>

3.4 On the Need of Learning Disjointness Axioms for Knowledge Graph Refinement and for Making Knowledge Graph Embedding Methods more Robust

Claudia d'Amato (*University of Bari, IT*)

License © Creative Commons BY 4.0 International license
© Claudia d'Amato

Joint work of Giuseppe Rizzo, Claudia d'Amato, Nicola Fanizza

Main reference Giuseppe Rizzo, Claudia d'Amato, Nicola Fanizza: “An unsupervised approach to disjointness learning based on terminological cluster trees”, *Semantic Web*, Vol. 12(3), pp. 423–447, 2021.

URL <https://doi.org/10.3233/SW-200391>

Knowledge Graphs (KGs) are multi-relational graphs designed to organize and share real-world knowledge where nodes represent entities of interest and edges represent different types of relationships between such entities [1]. Despite the large usage, it is well known that KGs suffer from incompleteness and noise. For tackling these problems, solutions to the link

prediction task, that amount at predicting an unknown component of a triple, have been investigated. Mostly, Knowledge Graph Embedding methods (KGE) have been devised since they have been shown to scale even to very large KGs. KGE convert the data graph into an optimal low dimensional space where structural graph information is preserved as much as possible. Embeddings are learned based on the constraint that a valid (positive) triple score has to be lower than the invalid (negative) triple score. As KGs mainly encode positive triples, negative triples are obtained by randomly corrupting true/observed triples [2], thus possibly injecting false negatives during the learning process.

In this talk we present a solution for an informed generation of negative examples that, by exploiting the semantics of the KGs and reasoning capabilities, is able to limit false negatives. A key element is represented by disjointness axioms, that are essential for making explicit the negative knowledge about a domain. Yet, disjointness axioms are often overlooked during the modeling process [3]. For the purpose, a symbolic method for discovering disjointness axioms from the data distribution is illustrated. Moving from the assumption that two or more concepts may be mutually disjoint when the sets of their (known) instances do not overlap, the problem is cast as a conceptual clustering problem, where the goal is both to find the best possible partitioning of the individuals in (a subset of) the KG and also to induce intensional definitions of the corresponding classes expressed in the standard representation languages.

The talk will conclude with the analysis of some open challenges related to the presented solutions.

References

- 1 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, Antoine Zimmermann: *Knowledge Graphs*. Synthesis Lectures on Data, Semantics, and Knowledge, Morgan & Claypool Publishers 2021, ISBN 978-3-031-00790-3, pp. 1-257. <https://doi.org/10.2200/S01125ED1V01Y202109DSK022>
- 2 Hongyun Cai, Vincent W. Zheng, Kevin Chen-Chuan Chang: *A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications*. IEEE Trans. Knowl. Data Eng. 30(9): 1616-1637 (2018). <https://doi.org/10.1109/TKDE.2018.2807452>
- 3 Taowei David Wang, Bijan Parsia, James A. Hendler: *A Survey of the Web Ontology Landscape*. ISWC 2006: 682-694. https://doi.org/10.1007/11926078_49

3.5 How to make logics neurosymbolic

Luc De Raedt (KU Leuven, BE)

License © Creative Commons BY 4.0 International license
© Luc De Raedt

Joint work of Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, Luc De Raedt
Main reference Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, Luc De Raedt: “From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey”, CoRR, Vol. abs/2108.11451, 2021.
URL <https://arxiv.org/abs/2108.11451>

Neurosymbolic AI (NeSy) is regarded as the third wave in AI. It aims at combining knowledge representation and reasoning with neural networks. Numerous approaches to NeSy are being developed and there exists an ‘alphabet-soup’ of different systems, whose relationships are often unclear. I will discuss the state-of-the art in NeSy and argue that there are many similarities with statistical relational AI (StarAI).

Taking inspiration from StarAI, and exploiting these similarities, I will argue that Neurosymbolic AI = Logic + Probability + Neural Networks. I will also provide a recipe for developing NeSy approaches: start from a logic, add a probabilistic interpretation, and then turn neural networks into “neural predicates”. Probability is interpreted broadly here, and is necessary to provide a quantitative and differentiable component to the logic. At the semantic and the computation level, one can then combine logical circuits (ako proof structures) labeled with probability, and neural networks in computation graphs.

I will illustrate the recipe with NeSy systems such as DeepProbLog, a deep probabilistic extension of Prolog, and DeepStochLog, a neural network extension of stochastic definite clause grammars (or stochastic logic programs).

3.6 What should we do next in ILP?

Sebastija Dumančić (TU Delft, NL)

License © Creative Commons BY 4.0 International license
© Sebastijan Dumančić

This talk consists of two parts. In the first part, I provide a brief introduction to Inductive Logic Programming: what is it, why is it interesting, and what interesting has recently happened. In the second part, I will explore what I think we should do next in ILP and program synthesis to further advance the field, all centered around the idea of avoiding search.

3.7 Human-like Few-Shot Learning via Bayesian Reasoning over Natural Language

Kevin Ellis (Cornell University – Ithaca, US)

License © Creative Commons BY 4.0 International license
© Kevin Ellis

Main reference Kevin Ellis: “Modeling Human-like Concept Learning with Bayesian Inference over Natural Language”, CoRR, Vol. abs/2306.02797, 2023.

URL <https://doi.org/10.48550/ARXIV.2306.02797>

A core tension in models of concept learning is that the model must carefully balance the tractability of inference against the expressivity of the hypothesis class. Humans, however, can efficiently learn a broad range of concepts. We introduce a model of inductive learning that seeks to be human-like in that sense. It implements a Bayesian reasoning process where a language model first proposes candidate hypotheses expressed in natural language, which are then re-weighted by a prior and a likelihood. By estimating the prior from human data, we can predict human judgments on learning problems involving numbers and sets, spanning concepts that are generative, discriminative, propositional, and higher-order.

3.8 Towards Programmatic Reinforcement Learning


Nathanaël Fijalkow (CNRS – Talence, FR)

License  Creative Commons BY 4.0 International license
© Nathanaël Fijalkow

This short talk was a pitch for a new problem, called Programmatic Reinforcement Learning: assuming that the environment is given as a program, the goal is to construct an optimal policy in the form of a program. Some motivations, basic examples, and preliminary experimental results were presented and discussed.

3.9 Inductive Programming for Explainable Artificial Intelligence (IP for XAI)

Bettina Finzel (Universität Bamberg, DE)

License  Creative Commons BY 4.0 International license
© Bettina Finzel

Methods of explainable artificial intelligence (XAI) and of inductive programming (IP) can profit from each other in two ways: (1) Inductive programming results in symbolic models (programs) which are inherently interpretable. These programs can provide expressive, relational explanations for learned black box models, for instance Convolutional Neural Networks for image classification. This perspective (IP for XAI) is addressed in this summary. (2) On the other hand, there might be a need for explainability of IP programs to humans. This perspective (XAI for IP) is addressed in the contribution of U. Schmid in this report.

End-to-end and data-driven approaches to learning, like deep convolutional neural networks in image classification, have become prevalent and the center of attention in many research and application areas. However, some research objectives and real world problems may not be solvable by just processing large amounts of data. In some cases, like medical diagnostics, “big data” simply may not be available [2]. At the same time, deep learning models are not inherently transparent opposed to those generated by interpretable machine learning algorithms, such as Inductive Logic Programming (ILP) [6]. This may be a crucial deficiency and a barrier to high stakes applicability of deep learning. At the same time, ILP frameworks provide symbolic representations in the form of predicates in First-Order-Logic, tracing capabilities and the integration of relational background knowledge by design, e.g., from human expertise and domain knowledge [3]. Moreover, their learning process is data-efficient in comparison to deep learning. In addition, being a relational learning approach qualifies ILP for explainability [1], e.g., in complex knowledge domains like medicine [2] and AI evaluation in general [5]. Deep learning may therefore profit from being combined with ILP for explanation, validation and a bi-directional interaction between a human and an AI system [3]. A crucial part of this avenue is the design of interfaces between internal representations of what a deep learning model has learned and the relational background knowledge of IP systems, like ILP, to provide human-understandable surrogate models, explanations and interactions. First attempts to bridge this gap have already been proposed [4]. However, several open questions remain to date: How can we find and extract relevant internal representations from deep learning models and present them in a human-understandable manner? How can we disambiguate representations? Which relations should be included in the IP module and satisfied by the deep learning model? How can we implement a knowledge exchange between IP and deep learning models to support the interplay of learning and reasoning in


knowledge discovery and AI evaluation? In my opinion, to build such systems is the way toward approximating the strengths of the human inductive bias and adaptability of AI systems to the real world.

References

- 1 Gesina Schwalbe, Bettina Finzel: *A comprehensive taxonomy for explainable artificial intelligence: a systematic survey of surveys on methods and concepts*. Data Mining and Knowledge Discovery: 1-59 (2023). <https://doi.org/10.1007/s10618-022-00867-8>
- 2 Sebastian Bruckert, Bettina Finzel, Ute Schmid: The Next Generation of Medical Decision Support: A Roadmap Toward Transparent Expert Companions. *Frontiers Artif. Intell.* 3: 507973 (2020). <https://doi.org/10.3389/FRAI.2020.507973>
- 3 Ute Schmid, Bettina Finzel: *Mutual Explanations for Cooperative Decision Making in Medicine*. *Künstliche Intell.* 34(2): 227-233 (2020). <https://doi.org/10.1007/S13218-020-00633-2>
- 4 Johannes Rabold, Michael Siebers, Ute Schmid: *Explaining Black-Box Classifiers with ILP – Empowering LIME with Aleph to Approximate Non-linear Decisions with Relational Rules*. *ILP 2018*: 105-117. https://doi.org/10.1007/978-3-319-99960-9_7
- 5 José Hernández-Orallo: *The Measure of All Minds: Evaluating Natural and Artificial Intelligence*. Cambridge University Press 2017, ISBN 9781316594179. <https://doi.org/10.1017/9781316594179>
- 6 Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, Benjamin G. Zorn: Inductive programming meets the real world. *Commun. ACM* 58(11): 90-99 (2015). <https://doi.org/10.1145/2736282>

3.10 On Deep Rule Learning

Johannes Fürnkranz (Johannes Kepler Universität Linz, AT)

License  Creative Commons BY 4.0 International license
© Johannes Fürnkranz

Joint work of Florian Beck, Johannes Fürnkranz

Main reference Florian Beck, Johannes Fürnkranz: “An Empirical Investigation Into Deep and Shallow Rule Learning”, *Frontiers in Artificial Intelligence*, Vol. 4, 2021.

URL <https://doi.org/10.3389/frai.2021.689398>

Rule learning algorithms form the basis of classic inductive logic programming algorithms such as FOIL or PROGOL. Studying them in a propositional logic setting allows to focus on the algorithmic aspects. A key limitation of the current state-of-the-art such as the LORD algorithm recently developed in our group [1], is that they are all limited to learning rule sets that directly connect the input features to the target feature. In a logical setting, this corresponds to learning a DNF expression. While every logical function can be expressed as a DNF formula, we argue in this talk that learning deeply structured theories may be beneficial, by drawing an analogy to (deep) neural networks [3], and recapitulating some recent empirical results [2].


References

- 1 Phuong Huynh Van Quoc, Johannes Fürnkranz, Florian Beck: *Efficient learning of large sets of locally optimal classification rules*. *Machine Learning* 112(2): 571-610 (2023) <https://doi.org/10.1007/s10994-022-06290-w>
- 2 Florian Beck, Johannes Fürnkranz, Phuong Huynh Van Quoc: *Layerwise Learning of Mixed Conjunctive and Disjunctive Rule Sets*. *Proceedings of the 7th International Joint Conference on Rules and Reasoning (RuleML+RR)*, 2023, 2023:95-109. https://doi.org/10.1007/978-3-031-45072-3_7

- 3 Florian Beck, Johannes Fürnkranz: *An Empirical Investigation Into Deep and Shallow Rule Learning*. *Frontiers in Artificial Intelligence* 4: 689398 (2021). <https://doi.org/10.3389/frai.2021.689398>

3.11 Three Learning Problems in Planning

Hector Geffner (RWTH Aachen, DE)

License  Creative Commons BY 4.0 International license
© Hector Geffner

Joint work of Hector Geffner, Simone Ståhlberg, Blai Bonet, Dominik Drexler, RLeap team

I'll talk about three learning problems in planning: learning lifted action models, learning generalized policies, and learning general problem decomposition or sketches. We have been approaching these problems in a top-down fashion, making a clear distinction between what is to be learned and how it is to be learned. Indeed, we have been pursuing two types of approaches in parallel: formulations that rely on combinatorial optimization solvers on the one hand, and deep (reinforcement) learning approaches on the other. I'll also discuss the relation between the two approaches which in the common form are limited by the expressive power of C2 logic; first-order logic with two variables and counting, and challenges to get beyond C2.

References

- 1 Blai Bonet, Hector Geffner: *General Policies, Subgoal Structure, and Planning Width*. *CoRR* abs/2311.05490 (2023). <https://doi.org/10.48550/ARXIV.2311.05490>
- 2 Simon Ståhlberg, Blai Bonet, Hector Geffner: *Learning General Policies with Policy Gradient Methods*. *KR 2023*: 647-657. <https://doi.org/10.24963/KR.2023/63>
- 3 Dominik Drexler, Jendrik Seipp, Hector Geffner: *Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width*. *ICAPS 2022*: 62-70 <https://doi.org/10.1609/icaps.v32i1.19786>
- 4 Ivan D. Rodriguez, Blai Bonet, Javier Romero, Hector Geffner: *Learning First-Order Representations for Planning from Black Box States: New Results*. *KR 2021*: 539-548. <https://doi.org/10.24963/KR.2021/51>

3.12 A tutorial on Popper

Céline Hocquette (University of Oxford, GB)

License  Creative Commons BY 4.0 International license
© Céline Hocquette

Joint work of Andrew Cropper, Céline Hocquette

Main reference Andrew Cropper, Céline Hocquette: "Learning Logic Programs by Combining Programs", in Proc. of the ECAI 2023 – 26th European Conference on Artificial Intelligence, September 30 – October 4, 2023, Kraków, Poland – Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023), *Frontiers in Artificial Intelligence and Applications*, Vol. 372, pp. 501–508, IOS Press, 2023.

URL <https://doi.org/10.3233/FAIA230309>

Inductive logic programming (ILP) is a form of program synthesis. The goal is to induce a logic program that generalises training examples. Popper is a recent ILP system which frames the ILP problem as a constraint satisfaction problem [1, 2]. Popper continually generates hypotheses and tests them on the training examples. If a hypothesis is not a solution, Popper

builds constraints to prune hypotheses which are also provably no solutions. Popper supports learning of recursive programs, predicate invention and learning moderately large programs. We present a recent extension of Popper which supports learning minimal description length programs from noisy data [3]. Our approach leverages recent progress in MaxSAT solvers to efficiently find an optimal program.

References

- 1 Andrew Cropper, Rolf Morel: *Learning programs by learning from failures*. Mach. Learn. 110(4): 801-856 (2021). <https://doi.org/10.1007/S10994-020-05934-Z>
- 2 Andrew Cropper, Céline Hocquette: *Learning Logic Programs by Combining Programs*. ECAI 2023: 501-508 <https://doi.org/10.3233/FAIA230309>
- 3 Céline Hocquette, Andreas Niskanen, Matti Järvisalo, Andrew Cropper: *Learning MDL logic programs from noisy data*. CoRR abs/2308.09393 (2023). <https://doi.org/10.48550/ARXIV.2308.09393>

3.13 Relational program synthesis with numerical reasoning

Céline Hocquette (University of Oxford, GB)

License © Creative Commons BY 4.0 International license
© Céline Hocquette

Joint work of Céline Hocquette, Andrew Cropper



Main reference Céline Hocquette, Andrew Cropper: “Relational Program Synthesis with Numerical Reasoning”, in Proc. of the Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023, pp. 6425–6433, AAAI Press, 2023.

URL <https://doi.org/10.1609/AAAI.V37I5.25790>

Learning programs with numerical values is fundamental to many AI applications, including bio-informatics and drug design. However, current program synthesis approaches struggle to learn programs with numerical values. Program synthesis approaches based on enumeration of candidate numerical symbols cannot handle infinite domains. Recent program synthesis approaches also have difficulties reasoning from multiple examples, which is required for instance to identify numerical thresholds or intervals. To overcome these limitations, we introduce an inductive logic programming approach which combines relational learning with numerical reasoning [1]. Our approach uses satisfiability modulo theories solvers to efficiently identify numerical values. Our approach can identify numerical values in linear arithmetic fragments, such as real difference logic, and from infinite domains, such as real numbers or integers. Our results show our approach can outperform existing program synthesis approaches. However, our approach has limited scalability with respect to the complexity of the numerical reasoning stage.

3.14 On the role of natural language for self-programming in cognitive architectures



Frank Jäkel (TU Darmstadt, DE)

License  Creative Commons BY 4.0 International license
 Frank Jäkel

Human problem solvers are able to adapt their problem solving strategies to new situations. They program their own behavior. In order to do so, they introspect, test, debug, and optimize their problem solving algorithms. These metacognitive activities can be implemented in standard cognitive architectures that can store code in working memory and execute it with an interpreter that is implemented as a set of rules in a production system. Additional rules can then modify the code at runtime. Unfortunately, the programming language in which such mental code is written has remained elusive. Here, I will argue that it is time to revive the old idea that program code is directly given in natural language. Traditionally, research on cognitive architectures has mostly avoided natural language even though language is obviously an important aspect of human cognition. With the advent of large language models it seems more plausible than ever that natural language interpreters might become an essential part of a new generation of cognitive architectures. In particular, the metacognitive activity of modifying your own programs might simply consist of transforming one natural language expression into another – the task that transformers were developed for and have turned out to be quite successful at.

3.15 QCBA: improving rule classifiers learned from quantitative data by recovering information lost by discretisation

Tomáš Kliegr (University of Economics – Prague, CZ)

License  Creative Commons BY 4.0 International license
 Tomáš Kliegr

Main reference Tomáš Kliegr, Ebrou Izquierdo: “QCBA: improving rule classifiers learned from quantitative data by recovering information lost by discretisation”, *Appl. Intell.*, Vol. 53(18), pp. 20797–20827, 2023.

URL <https://doi.org/10.1007/S10489-022-04370-X>

Many rule-learning algorithms require prior discretization before they can effectively process datasets with numerical data. For example, consider a dataset with attributes such as temperature and humidity. Discretization (also called quantization) means binning their values into intervals. A simple equidistant algorithm would produce intervals such as (0;10], (10;20], and (20; 30]. If we consider rule learning algorithms based on association rule learning, such as Classification based on Associations [3], discretization is necessary to ensure fast pruning of the state space and also learning of sufficiently generalized rules. Only after the discretization is it possible to learn the rules of the type IF temperature=(20;30] and humidity=(50;60] THEN worker_comfort= good.

While some rule learning algorithms can directly work with numerical attributes, such as the recently proposed extension of the POPPER ILP system [4], for those based on association rule learning, integrating quantization may not be efficient as it could excessively slow down the candidate generation phase. A common approach is thus to apply prediscretization, e.g., following the Minimum Description Length Principle (MDLP)-based method proposed by [2]. However, as the determination of interval lengths is done globally (i.e., same intervals for all instances) and outside of the learning algorithm (e.g. CBA), information is lost, resulting in inefficiencies in the final classifier.

Following this problem, this talk introduced the Quantitative CBA (QCBA) algorithm for the subsequent processing of rule models learned on arbitrarily pre-discretized data (e.g., with equidistant binning, MDLP or other method). Extensive experiments have shown that the proposed algorithm consistently reduces the models' size and thus makes them more understandable. Additionally, in many cases, the predictive performance is also improved. The algorithm can be used to process the results of many rule learning algorithms, including CBA, Interpretable Decision Sets [1] and Scalable Bayesian Rule Lists [5]. The results are available in the R package qCBA available in CRAN. The method is described in detail in [6].

References

- 1 Himabindu Lakkaraju, Stephen H. Bach, Jure Leskovec: *Interpretable Decision Sets: A Joint Framework for Description and Prediction*. KDD 2016: 1675-1684 <https://doi.org/10.1145/2939672.2939874>
- 2 Usama M. Fayyad, Keki B. Irani: *Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning*. IJCAI 1993: 1022-1029
- 3 Bing Liu, Wynne Hsu, Yiming Ma: *Integrating Classification and Association Rule Mining*. KDD 1998: 80-86
- 4 Céline Hocquette, Andrew Cropper: *Relational Program Synthesis with Numerical Reasoning*. AAAI 2023: 6425-6433 <https://doi.org/10.1609/AAAI.V37I5.25790>
- 5 Hongyu Yang, Cynthia Rudin, Margo I. Seltzer: *Scalable Bayesian Rule Lists*. ICML 2017: 3921-3930
- 6 Tomás Kliegr, Ebroul Izquierdo: *QCBA: improving rule classifiers learned from quantitative data by recovering information lost by discretisation*. Appl. Intell. 53(18): 20797-20827 (2023) <https://doi.org/10.1007/S10489-022-04370-X>

3.16 RDFrules: A Swiss knife for relational association rule learning, classification and knowledge graph completion

Tomáš Kliegr (*University of Economics – Prague, CZ*)

License  Creative Commons BY 4.0 International license
© Tomáš Kliegr

Joint work of Václav Zeman, Tomáš Kliegr, Vojtech Svátek

Main reference Václav Zeman, Tomáš Kliegr, Vojtech Svátek: “RDFrules: Making RDF rule mining easier and even more efficient”, *Semantic Web*, Vol. 12(4), pp. 569–602, 2021.

URL <https://doi.org/10.3233/SW-200413>

Many commonly used machine learning algorithms are limited to tabular data sets, but real-world data is often stored in relational databases and increasingly in knowledge graphs. Processing of such data with standard „tabular“ machine learning usually requires extensive data transformation and aggregations, resulting in a loss of information. As an alternative, relational Horn rules can be used to model complex relational structures naturally and use these in a range of machine learning tasks, including exploratory analysis, classification, and imputation of missing information.

The RDFrules system for learning rules from knowledge graphs is based on the high-performance AMIE+ algorithm [1] and includes a number of improvements based on more than 10 years of experience with the development of its sister tabular EasyMiner [2] rule learning system. While the AMIE+ algorithm was initially designed for a narrower exploratory task of discovery of rules with the potential to perform knowledge graph (KG) completion,

the current version of the RDRules system goes significantly beyond the original capabilities of the AMIE+ algorithm [1] as it now makes possible to perform the following tasks:

- load not only graph data in RDF but also relational databases described as SQL scripts,
- specify fine-grained patterns to limit the search space,
- preprocess numerical literals,
- cluster discovered rules,
- perform classification tasks,
- evaluate results using standard metrics adapted to graph data and open world assumption,
- support the KG completion task,

The new features make it possible to graph-based rule learning directly on complex real-world data.

The system is described in [3] and available at <https://github.com/propi/rdrules>.

References

- 1 Luis Galárraga, Christina Teflioudi, Katja Hose, Fabian M. Suchanek: *Fast rule mining in ontological knowledge bases with AMIE+*. VLDB J. 24(6): 707-730 (2015) <https://doi.org/10.1007/S00778-015-0394-1>
- 2 Stanislav Vojír, Vaclav Zeman, Jaroslav Kuchar, Tomáš Kliegr: *EasyMiner.eu: Web framework for interpretable machine learning based on rules and frequent itemsets*. Knowl. Based Syst. 150: 111-115 (2018) <https://doi.org/10.1016/J.KNOSYS.2018.03.006>
- 3 Václav Zeman, Tomáš Kliegr, Vojtech Svátek: *RDRules: Making RDF rule mining easier and even more efficient*. Semantic Web 12(4): 569-602 (2021) <https://doi.org/10.3233/SW-200413>

3.17 The Child as Hacker

Josh Rule (University of California – Berkeley, US)

License © Creative Commons BY 4.0 International license
© Josh Rule

Main reference Joshua S. Rule: “The child as hacker: building more human-like models of learning”, Doctoral dissertation, Massachusetts Institute of Technology (2020)

URL <https://hdl.handle.net/1721.1/129232>

I describe the *child as hacker* hypothesis, which relates program induction with aspects of human cognition, particularly learning [1]. By the deep relationship proposed to exist between knowledge and program-like structures, the child as hacker treats the activities and values of human programmers as hypotheses for the activities and values of many forms of human learning. After introducing this idea, I then look briefly at a project where we’ve begun to implement it in a system called HL (Hacker-Like) [2]. HL explains human behaviour better than some recent alternative program induction systems by representing a concept not only in terms of its object-level content but also in terms of the inferences required to produce that content. By searching over both kinds of representations, HL learns orders of magnitude faster than competing systems. I close by discussing three major areas ripe for future research: i) developing a better empirical understanding of how people solve hard search problems; ii) understanding the neural and psychological basis for human computational abilities; and iii) better understanding the goals and values of human programmers. All three areas have the potential to significantly improve both our understanding of human intelligence and our ability to use program induction systems to solve complex problems.

References

- 1 Joshua S. Rule, Joshua B. Tenenbaum, Steven T. Piantadosi: *The child as hacker*. Trends in cognitive sciences 24(11): 900-915 (2020) <https://doi.org/10.1016/j.tics.2020.07.005>
- 2 Joshua S. Rule: *The child as hacker: building more human-like models of learning*. Doctoral dissertation, Massachusetts Institute of Technology (2020) <https://hdl.handle.net/1721.1/129232>

3.18 Abstraction for Answer Set Programs

Zeynep G. Saribatur (TU Wien, AT)

License © Creative Commons BY 4.0 International license
© Zeynep G. Saribatur

Joint work of Zeynep G. Saribatur, Thomas Eiter, Peter Schüller

Main reference Zeynep G. Saribatur, Thomas Eiter, Peter Schüller: “Abstraction for non-ground answer set programs”, Artif. Intell., Vol. 300, p. 103563, 2021.

URL <https://doi.org/10.1016/J.ARTINT.2021.103563>

In this talk, I present our notion of abstraction for answer set programming, a prominent rule-based language for knowledge representation and reasoning with roots in logic programming and non-monotonic reasoning. With the aim to abstract over the irrelevant details of answer set programs, we focus on two approaches of abstraction: (1) abstraction by omission [2], and (2) domain abstraction [1], and introduce a method to construct an abstract program with a smaller vocabulary, by ensuring that the original program is over-approximated. We provide an abstraction & refinement methodology that makes it possible to start with an initial abstraction and upon encountering spurious solutions automatically refining the abstraction until an abstract program with a non-spurious solution is reached. Experiments based on the prototypical implementations reveal the potential of the approach for problem analysis by focusing on the parts of the program that cause the unsatisfiability, some even matching a human-like focus shown by a user study, and by achieving generalization of the answer sets that reflect relevant details only. This makes abstraction an interesting topic of research whose further use in human-understandability of logic programs remains to be explored.

References

- 1 Zeynep G. Saribatur, Thomas Eiter, Peter Schüller: *Abstraction for non-ground answer set programs*. Artif. Intell. 300: 103563 (2021) <https://doi.org/10.1016/J.ARTINT.2021.103563>
- 2 Zeynep G. Saribatur, Thomas Eiter: *Omission-Based Abstraction for Answer Set Programs*. Theory Pract. Log. Program. 21(2): 145-195 (2021) <https://doi.org/10.1017/S1471068420000095>

3.19 Explanatory Inductive Programming (XAI for IP)

Ute Schmid (Universität Bamberg, DE)

License © Creative Commons BY 4.0 International license
© Ute Schmid

Joint work of Johannes Rabold, Michael Siebers, Ute Schmid

Main reference Johannes Rabold, Michael Siebers, Ute Schmid: “Generating contrastive explanations for inductive logic programming based on a near miss approach”, *Mach. Learn.*, Vol. 111(5), pp. 1799–1820, 2022.

URL <https://doi.org/10.1007/S10994-021-06048-W>

Methods of explainable artificial intelligence (XAI) and of inductive programming (IP) can profit from each other in two ways: (1) Inductive programming results in symbolic models (programs) which are inherently interpretable. Nevertheless, there might be a need for explainability to humans – end-users or domain experts from other areas than computer science. This perspective (XAI for IP) is addressed in this summary. (2) On the other hand, expressive, relational explanations for learned black box models, for instance Convolutional Neural Networks for image classification, can be provided by IP. This perspective (IP for XAI) is addressed in the contribution of B. Finzel in this report.

The power of IP approaches lies in their ability to learn highly expressive models from small sets of examples [2]. Learned programs can support humans to get insights into complex relational or recursive patterns underlying a set of observed data. That is, IP might be an ultra-strong learning approach as defined by Donald Michie (see [3]) under the condition that the learning system can teach the learned model to a human, whose performance is consequently increased to a level beyond that of the human studying the training data alone. For programs which consist of several rules or for programs involving complex relations or recursion, different approaches to construct explanations might support human understanding. One possibility to reduce complexity is to introduce new predicates. For instance, the introduction of a predicate `parent/2` as generalization for `father/2` and `mother/2`, reduces four rules for the `grandparent/2` relation to one (see [3]). Another possibility is, to translate the rule which covers the current instant to a verbal explanation for humans without background in computer science. This can be realized by simple template-based methods [5]. Alternatively, Large Language Models could be used. For effective teaching a concept to humans, near miss explanations have been proposed by [4]. Winston showed in his early work on learning rules for relational perceptual concepts such as arcs, that providing near misses rather than arbitrary negative examples results in faster convergence of the learned model. In cognitive science it has been shown that teaching concepts by their difference to similar concepts is much more efficient than contrasting them with more distant concepts (for a discussion of these aspects and references, see [4]). In [4] an algorithm for constructing near miss explanations is presented and applied to different domains. Furthermore, an empirical study is presented where it could be shown that in pairwise comparisons, participants preferred near miss explanations over other types of explanations as more helpful.

Augmenting IP models with explanations can also be helpful to support medical decision making [1]. Here, it might be helpful to go beyond ultrastrong machine learning and bring the human expert in the loop for incremental model correction and adaptation. In contrast to standard interactive machine learning, human feedback might go beyond label correction and allow human domain experts to also correct explanations which might be right for the wrong reasons. Correcting explanations can be seen as a special case of knowledge injection in human-in-the-loop IP which exploits such corrections for efficient model adaptation.

References

- 1 Sebastian Bruckert, Bettina Finzel, Ute Schmid: *The Next Generation of Medical Decision Support: A Roadmap Toward Transparent Expert Companions*. *Frontiers Artif. Intell.* 3: 507973 (2020) <https://doi.org/10.3389/FRAI.2020.507973>
- 2 Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, Benjamin G. Zorn: *Inductive programming meets the real world*. *Commun. ACM* 58(11): 90-99 (2015) <https://doi.org/10.1145/2736282>
- 3 Stephen H. Muggleton, Ute Schmid, Christina Zeller, Alireza Tamaddoni-Nezhad, Tarek R. Besold: *Ultra-Strong Machine Learning: comprehensibility of programs learned with ILP*. *Mach. Learn.* 107(7): 1119-1140 (2018) <https://doi.org/10.1007/S10994-018-5707-3>
- 4 Johannes Rabold, Michael Siebers, Ute Schmid: *Generating contrastive explanations for inductive logic programming based on a near miss approach*. *Mach. Learn.* 111(5): 1799-1820 (2022) <https://doi.org/10.1007/S10994-021-06048-W>
- 5 Ute Schmid: *Interactive Learning with Mutual Explanations in Relational Domains*. *Human-Like Machine Intelligence 2022*: 338-354 <https://doi.org/10.1093/OSO/9780198862536.003.0017>

3.20 Explainable models via compression of tree ensembles

Sriraam Natarajan (University of Texas at Dallas – Richardson, US)

License © Creative Commons BY 4.0 International license
© Sriraam Natarajan

Joint work of Siwen Yan, Sriraam Natarajan, Saket Joshi, Roni Khardon, Prasad Tadepalli

Main reference Siwen Yan, Sriraam Natarajan, Saket Joshi, Roni Khardon, Prasad Tadepalli: “Explainable Models via Compression of Tree Ensembles” *Mach. Learn.*: 1-26 (2023)

URL <https://doi.org/10.1007/s10994-023-06463-1>

We consider the problem of explaining learned (relational) ensemble models. Ensemble models (bagging and gradient-boosting) of relational decision trees have proved to be one of the most effective learning methods in the area of probabilistic logic models (PLMs). While effective, they lose one of the most important aspect of PLMs – interpretability.

Our key hypothesis in this work is that combining large number of logical decision trees would yield in a more compressed model compared to that of combining standard decision trees. This is due to the fact that unification of variables in logic would allow for effective and efficient compression.

To this effect, we propose CoTE – Compression of Tree Ensembles – that produces a single small decision list as a compressed representation. CoTE first converts the trees to decision lists and then performs the combination and compression with the aid of the original training set. Experiments on standard benchmarks demonstrate the value of this approach and justifies the hypotheses that compression is more effective in logical decision trees.

3.21 Inductive Programming meets Large Language Models

Gust Verbruggen (Microsoft – Keerbergen, BE)

License  Creative Commons BY 4.0 International license
© Gust Verbruggen

Joint work of Gust Verbruggen, Vu Le, Sumit Gulwani

Main reference Gust Verbruggen, Vu Le, Sumit Gulwani: “Semantic programming by example with pre-trained models”, Proc. ACM Program. Lang., Vol. 5(OOPSLA), pp. 1–25, 2021.

URL <https://doi.org/10.1145/3485477>

Both inductive programming (IP) and large language models (LLMs) are able to complete a task from a few examples. Instead of pitting them against each other, together they can achieve a lot more. One example of such integration is FlashGPT, which iteratively uses witness functions to break an inductive programming problem into smaller subproblems until all are solved (*FlashFill*) and leverages an LLM to solve the subproblems that cannot be solved symbolically (*GPT-3*). Instead of reiterating what has been discovered, this talk focused on (a non-exhaustive list of) next steps for combining IP and LLMs.

First, we discuss how the LLM can be used to improve learning in a fully symbolic IP system. Two approaches are (1) using the LLM to generate additional input-output examples for the IP system, or (2) using the LLM to generate candidate solutions to serve as seeds for initiating a search. The latter is a combination of component-based synthesis [1] and sketching, both of which rely on generating useful substructures over the grammar of the target language.

Second, we show how the LLM can be used to improve the experience of working with an IP system, by providing natural language descriptions of the learned programs.

Third, we show how the scope of IP can be improved with LLMS in systems that do not leverage witness functions. One potential method is masking semantic components, performing IP as usual and learning a program that emits masks, and then resolving the masks using an LLM.

Fourth, we show how operators that only use the embeddings from LLMs strike a balance between the inference speed of symbolic operations and the number of examples and capabilities of semantic operations. When the domain of a semantic relation is finite, or when the task is extraction of relevant parts of the input, we can use embeddings of tokens from the input to capture semantic relations between input and output.

References

- 1 Yoad Lustig, Moshe Y. Vardi: *Synthesis from component libraries*. Int. J. Softw. Tools Technol. Transf. 15(5-6): 603-618 (2013) <https://doi.org/10.1007/S10009-012-0236-Z>
- 2 Armando Solar-Lezama: *The Sketching Approach to Program Synthesis*. APLAS 2009: 4-13 https://doi.org/10.1007/978-3-642-10672-9_3

3.22 Inductive Programming meets Real User Problems

Gust Verbruggen (Microsoft – Keerbergen, BE)

License  Creative Commons BY 4.0 International license
© Gust Verbruggen

We show two novel applications of inductive programming that bring some unique challenges with respect to parsing user input. Both problems share some challenges: the need for speed, noisy input and labels, inferring constant values that adhere to a semantic bias, underspecification of the problem, and suppression of programs with low confidence.

First, we consider the problem of predicting the folder to which an email should be moved. Popular email clients offer to automate this functionality by setting rules, and the expected output of our learner is thus such a rule. An additional challenge with this problem is concept drift. Our approach [1] learns simple propositional rules in conjunctive normal form by generalizing (if an email is mistakenly not covered) or specializing (if an email is mistakenly covered) the rule corresponding to a folder. Because we guarantee that all historical emails are correctly classified, we easily adapt to concept drift. This classic inductive programming approach performs better than many neural and hybrid baselines.

Second, we consider the problem of learning conditional formatting rules in spreadsheets. An additional challenge is the scope of functions that can be used. Our approach [2, 3] uses semi-supervised clustering of input values to tackle underspecification, then learns different rules as decision trees, and ranks them with a learned ranker. Our corpus of 102K rules from real spreadsheets allows this ranker to encode the semantic bias, which allows us to outperform many neural and symbolic approaches, even if they have access to the same set of base predicates.

References

- 1 Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Gust Verbruggen: *EmFore: Online Learning of Email Folder Classification Rules*. CIKM 2023: 2280-2290 <https://doi.org/10.1145/3583780.3614863>
- 2 Mukul Singh, José Pablo Cambronero Sánchez, Sumit Gulwani, Vu Le, Carina Negreanu, Mohammad Raza, Gust Verbruggen: *CORNET: Learning Table Formatting Rules By Example*. Proc. VLDB Endow. 16(10): 2632-2644 (2023) <https://doi.org/10.14778/3603581.3603600>
- 3 Mukul Singh, José Pablo Cambronero Sánchez, Sumit Gulwani, Vu Le, Carina Negreanu, Gust Verbruggen: *CORNET: Learning Spreadsheet Formatting Rules By Example*. Proc. VLDB Endow. 16(12): 4058-4061 (2023) <https://doi.org/10.14778/3611540.3611620>

3.23 Probabilistic Logic Programming: Quo Vadis?

Felix Weitekämper (LMU München, DE)

License © Creative Commons BY 4.0 International license
© Felix Weitekämper


Probabilistic Inductive Logic Programming refers to learning probabilistic relational programs from “examples”. These could be probabilistic logic programs, but many considerations also apply to learning other statistical relational models. From the perspective of statistical relational artificial intelligence, this is usually referred to as *structure learning*. Probabilistic Inductive Logic Programming is key in several areas of artificial intelligence, including knowledge discovery in stochastic, relational domains and causal structure discovery in a Boolean relational setting. Probabilistic inductive logic programming is traditionally considered difficult, since it adds another dimension to the classical ILP problem. Current approaches are still based on traditional ILP approaches developed in the 1990s, while the field of ILP has since made huge progress: Metainterpretive learning provides a new conceptual framework for rethinking Inductive Logic Programming, Constraints and learning from failures can help prune the search space, and Powerful ASP encodings can be leveraged to achieve more consistent outcomes.

This raises the question: Can we leverage these modern techniques for PILP?

4 Working groups

4.1 Large Language Models and Inductive Programming in Cognitive Architectures

Bettina Finzel (Universität Bamberg, DE) and Frank Jäkel (TU Darmstadt, DE)

License  Creative Commons BY 4.0 International license
© Bettina Finzel and Frank Jäkel

Cognitive architectures provide frameworks to simulate and test principles of cognition [1]. There are different components in cognitive architectures that qualify for being enhanced by large language models (LLMs) [3] and inductive programming (IP) [2]. LLMs could be used in the production module to generate rules for execution, in the memory module as a compressor of information for more efficient access and possibly as the stimuli of a general cognitive architecture as a model that produces outputs on which further reasoning could be applied for decision making and learning. An open challenge remains in mimicking the abilities of humans to switch between modalities in the sense that they are able to dynamically choose between the representations they need. With respect to this, we were discussing about some form of reward or reinforcer to increase the response for certain signals or items in the process of inference and problem solving. Combining learning and reasoning by integrating LLMs and IP in a cognitive architecture could be an enabler for validating programs that get executed by the overall architecture and to possibly get nearer to human performance.

References

- 1 Paul Thagard: *Cognitive Architectures*. The Cambridge Handbook of Cognitive Science 2012: 50-70 <https://doi.org/10.1017/CB09781139033916.005>
- 2 Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, Benjamin G. Zorn: Inductive programming meets the real world. *Commun. ACM* 58(11): 90-99 (2015) <https://doi.org/10.1145/2736282>
- 3 Naman Jain, Skanda Vaidyanath, Arun Shankar Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram K. Rajamani, Rahul Sharma: Jigsaw: Large Language Models meet Program Synthesis. *ICSE 2022*: 1219-1231 <https://doi.org/10.1145/3510003.3510203>

4.2 Avoiding too much search in Inductive Programming

Ute Schmid (Universität Bamberg, DE), David Cerna (The Czech Academy of Sciences – Prague, CZ), and Hector Geffner (RWTH Aachen, DE)

License  Creative Commons BY 4.0 International license
© Ute Schmid, David Cerna, and Hector Geffner

A crucial part of inductive programming (IP) is search. Since search is costly, an important question is how we can avoid to search so much or too much.

What we search for can be very different things: logic or functional programs, but also decision lists, policies, classifications, language representations or deep learned models. How search is performed can be also realized with many different approaches: enumeration, anti-unification, genetic programming, greedy strategies, combinatorial optimization, stochastic gradient descent, deep reinforcement learning, or Monte Carlo Tree Search. In general, we do need to learn structure as well as probabilities.

To evaluate the quality of the learned program, different aspects might be focused on alone or in combination which is a challenge for search. Obvious criteria are sample complexity and scalability. But one might also be interested in novelty of the learned program, how similar is the inductive strategy to human learning (humans do not enumerate first and then select but typically generalise over few examples).

To push research on becoming more search efficient, a set of benchmark problems and a competition should be introduced. Promising challenge data sets might come from the FlashFill domain (learning more complex Excel functions and string transformations), the abstract reasoning challenge (ARC) and the modified ILP version might be interesting, furthermore, we could look at problems from the International Math Olympiad Challenge.

We should critically evaluate for which problems deep learning/generative approaches are more successful and hopefully identify a class of problems where symbolic IP is superior. For instance, the IP system FlashFill performs better than the transformer-based SmartFill. The core difference between neural network approaches and symbolic IP is that IP returns explicit programs which give the intensional characterisation of the input/output-examples while neural networks are extensional representations. Therefore, one might postulate that neural networks cannot learn recursion.

Currently, string transformation problems are often either syntactic (return the first letter of a string) – which works very well for symbolic IP – or semantic (give the capital for a country) – where generative AI is very good at. Maybe we should look for transformation problems which combine syntactic and semantic transformations (return the first letter for every string which names a capital). As it is so often the case, it might be a good idea to combine symbolic IP and deep learning/generative approaches. A paper we should look at is [1].

References

- 1 Stéphane d’Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, François Charton: *Deep symbolic regression for recurrence prediction*. ICML 2022: 4520-453

4.3 Evaluation Criteria for Interpretability and Explainability of Inductive Programming

Ute Schmid (Universität Bamberg, DE), Lun Ai (Imperial College London, GB), Claudia d’Amato (University of Bari, IT), and Johannes Fürnkranz (Johannes Kepler Universität Linz, AT)

License © Creative Commons BY 4.0 International license
© Ute Schmid, Lun Ai, Claudia d’Amato, and Johannes Fürnkranz

Inductive programming results in symbolic models (programs) which are inherently interpretable. Nevertheless, there might be a need for explainability to humans – end-users or domain experts from other areas than computer science. In the discussion group we focused on the question of how to measure the quality of interpretable representations (programs) and of post-hoc generated explanations. The main challenge is to provide for assessment metrics which are not dependent on studies with humans but which can be evaluated directly for the interpretation/explanations. A core difficulty is that the quality of an explanation is context-dependent: it depends on what is explained to whom in what way (how) and for what reason (why). The way to explain something can be a set of symbolic rules (learned with IP or a rule learning system or extracted from a neural net), highlighting important

features (which is done by many XAI approaches such as LIME, SHAP or LRP), a natural language explanation, prototypical or near miss examples. Furthermore, explanations can be more abstract or give more details. Explanations can either be constructed to explain for what reason a learned (black box) model gave a specific output (mechanistic explanation) or to explain the learned content to a human (functional explanation, ultra-strong machine learning).

As candidates for assessment metrics we discussed (1) complexity measures (proposals for cognitive complexity measures, structural information theory, Kolmogorov complexity), (2) semantic coherence, (3) reliability of a component (of a program) which can result in abstracting this part away if a human has sufficient/justified trust. A further aspect for evaluation might be a suitable trade-off between the size of the explanation (memory) and the effort to interpret it (run time) as proposed, for instance by Donald Michie [1] or Lun Ai [2].

A program itself can be a good explanation, depending of its complexity. Abstraction might be a useful method to make explanations more comprehensible. Here approaches like predicate/function invention, anti-unification, introducing higher-orderness (such as map/fold), or compression might be helpful. A hierarchy of abstractions can be helpful for providing the ‘right’ level of detail for a given explanatory context. There are first approaches of explanations as a dialogue where more detailed or different forms of explanations can be presented to a human [3]. Learned (Prolog) programs are also suitable in this context: The highest level of abstraction refers to a single (left-hand/target/head) predicate, the next level of detail can be achieved by presenting the instantiated right-hand side of a rule (or a verbal description of it), continued by expanding predicates in the body until ground facts are reached.

Recently, an explainable version FlashFill has been developed. It showed that users sometimes reject a FlashFill rule which correctly covers the examples because they do not understand it. Here approximate symbolic regression has been applied to provide simpler explanations [4]. In the group of Josh Tenenbaum, the system LILO [5] has been developed which provides explanations by abstraction.

A final idea on assessing the quality of an interpretation/explanation has been to input explanations of a learned programs to a LLM, let the LLM generate a program from that and then compare the originally synthesized program with the one generated by the LLM (a kind of loss function). Comparison can be done by behavioral comparison for test cases or by comparing the code.

References

- 1 Donald Michie: *Experiments on the Mechanization of Game-Learning. 2-Rule-Based Learning and the Human Window*. Comput. J. 25(1): 105-113 (1982) <https://doi.org/10.1093/COMJNL/25.1.105>
- 2 Lun Ai, Stephen H. Muggleton, Céline Hocquette, Mark Gromowski, Ute Schmid: *Beneficial and harmful explanatory machine learning*. Mach. Learn. 110(4): 695-721 (2021) <https://doi.org/10.1007/S10994-020-05941-0>
- 3 Bettina Finzel, David Elias Tafler, Anna Magdalena Thaler, Ute Schmid: *Multimodal Explanations for User-centric Medical Decision Support Systems*. HUMAN@AAAI Fall Symposium 2021
- 4 Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, Gust Verbruggen: *CodeFusion: A Pre-trained Diffusion Model for Code Generation*. EMNLP 2023: 11697-11708 <https://doi.org/10.48550/arXiv.2310.17680>

- 5 Gabriel Grand, Lionel Wong, Matthew Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, Jacob Andreas: *LILLO: Learning Interpretable Libraries by Compressing and Documenting Code*. CoRR abs/2310.19791 (2023) <https://doi.org/10.48550/ARXIV.2310.19791>

4.4 Finding Suitable Benchmark Problems for Inductive Programming

Ute Schmid (Universität Bamberg, DE), Martin Berger (University of Sussex – Brighton, GB), Sebastijan Dumancic (TU Delft, NL), Nathanaël Fijalkow (CNRS – Talence, FR), and Gust Verbruggen (Microsoft – Keerbergen, BE)

License © Creative Commons BY 4.0 International license

© Ute Schmid, Martin Berger, Sebastijan Dumancic, Nathanaël Fijalkow, and Gust Verbruggen

To advance progress as well as visibility of IP, a collection of suitable benchmarks, convincing use cases, and joint formats to represent problems, as well as starting an IP challenge have been identified as helpful. In the discussion group, we focussed on benchmark sets.

First we collected problems currently used in different groups: List problems, regular expressions (RegEx), boolean language inference, competitive programming, Math Olympiad Challenge, Reasoning/Theorem Proving, Planning, Knowledge Graphs, Zendo, Games, Navigation, Biology, standard ML benchmarks (UCML Repository), natural language to programs (NL2P), abstract reasoning challenge (ARC).

Then we discussed what characteristics benchmark problems should have: tunable, clear performance metrics, standard format, correct annotations, noise, social recognition/PR, breadth, not solvable by LLMs (alone), conceptual jumps, linkable to external resources, curriculum, dramatic finish line, doable by humans. Several of these characteristics were discarded. For instance, clear metrics (beyond just right or wrong) did not seem to be a good fit (but see discussion results about explainability). Format has also been seen as not relevant compared to having good environments to execute and evaluate learned programs and tools/environments which are easily usable.

For a selected set of characteristics, we identified that problems from the list above which fulfill the respective characteristic:

- Tunable: List problems, RegEx, Boolean languages, Knowledge Graphs, Games, Navigation, (competitive programming)
- Breadth: List problems, competitive programming, Games, NLP2P, Math Olymp, ARC
- Not solvable by LLM: List problems, Knowledge Graphs, Math Olymp, ARC
- Dramatic finish: Math Olymp, Biology, NLP2P, ARC
- Curriculum: List problems, RegEx, Math Olymp, Navigation
- Doable by (average) humans: List Problems, RegEx, Games, Navigation, ARC

Given the number of criteria which are met, the following problem domains have been identified as the most promising ones: List problems (including string transformations and other domain-specific language based approaches), RegEx, Math Olymp, ARC.

We then had a further critical look at the selected problem classes and evaluated the following aspects: Not suitable for application, lack of format, producible, lack of probabilistic benchmarks, tension between standard and generation, domain specific problems, not perceived as difficult/relevant, need/miss to have a relational core, loss of propositional benchmarks, lack of diversity of evolution, novelty (invent a new sorting algorithm, automated computer scientist), plugable.

After discussing these additional aspects, we came up with the following set of potentially interesting benchmark problems:

- The Automated Computer Scientist (from Andrew Cropper): learning novel (e.g. sorting) algorithms or novel data structures [1]
- Joint IP and KG (Knowledge Graph) problems, especially for combining syntactic and semantic transformations (e.g. give the capital for a country and then take the first letter of it), this can be list problems or Excel tables [5]
- Strategy learning (explicit compared to implicit policy learning in reinforcement learning): for human problem solving, planning (look at problems from the planning competition) [2, 3, 4]
- Online encyclopedia of integer sequences OEIS (not for all of them exists a closed formula)
- Expert domains: learning strategies/patterns for SAT-solvers, theorem provers
- Constructing ML pipelines (AutoML)

Links to benchmark data sets:

- Popper's (includes Zendo, many others)
<https://github.com/logic-and-learning-lab/Popper/tree/main/examples>
- SyGuS (includes list problems, FlashFill, phone numbers)
<https://github.com/SyGuS-Org/benchmarks>
- IP Repository (programming benchmarks, including problem solving like Tower of Hanoi)
<https://www.inductive-programming.org/repository.html>
- Regular expressions
<https://codalab.lisn.upsaclay.fr/competitions/15096>
- Boolean language inference
<https://www.iwls.org/contest/>
- Competitive programming
<https://github.com/openai/human-eval>
- Math Olympiad Challenge
<https://github.com/lupantech/dl4math#-mathematical-reasoning-benchmarks>
<https://github.com/openai/miniF2F/tree/v1>
- Planning
<https://github.com/AI-Planning/pddl-generators>
- Program synthesis benchmarks from genetic programming community
<https://cs.hamilton.edu/~thelmuth/PSB2/PSB2.html>
<https://zenodo.org/records/5084812>
- Standard ML benchmarks: UC Irvine ML Repository
<https://archive.ics.uci.edu/>
- Abstract Reasoning Challenge (ARC) [7, 8]
<https://github.com/fchollet/ARC>
<https://lab42.global/arc/>
- E-Mail Folder Classification
<http://www-2.cs.cmu.edu/~enron/>
<https://catalog.ldc.upenn.edu/LDC2015T03>
- Rule learning
<https://github.com/kliegr/arcbench>

References

- 1 Andrew Cropper: *The Automatic Computer Scientist*. AAAI 2023: 15434 <https://doi.org/10.1609/AAAI.V37I13.26801>
- 2 Mario Martín, Hector Geffner: *Learning Generalized Policies from Planning Examples Using Concept Languages*. Appl. Intell. 20(1): 9-19 (2004) <https://doi.org/10.1023/B:APIN.0000011138.20292.DD>
- 3 Ute Schmid, Emanuel Kitzelmann: *Inductive rule learning on the knowledge level*. Cogn. Syst. Res. 12(3-4): 237-248 (2011) <https://doi.org/10.1016/J.COGSYS.2010.12.002>
- 4 Jude W. Shavlik: *Acquiring Recursive and Iterative Concepts with Explanation-Based Learning*. Mach. Learn. 5: 39-40 (1990) <https://doi.org/10.1007/BF00115894>
- 5 Mukul Singh, José Pablo Cambronero Sánchez, Sumit Gulwani, Vu Le, Carina Negreanu, Mohammad Raza, Gust Verbruggen: *CORNET: A neurosymbolic approach to learning conditional table formatting rules by example*. CoRR abs/2208.06032 (2022) <https://doi.org/10.48550/ARXIV.2208.06032>
- 6 Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Gust Verbruggen: *EmFore: Online Learning of Email Folder Classification Rules*. CIKM 2023: 2280-2290 <https://doi.org/10.1145/3583780.3614863>
- 7 James Ainooson, Deepayan Sanyal, Joel P. Michelson, Yuan Yang, Maithilee Kunda: *An Approach for Solving Tasks on the Abstract Reasoning Corpus*. CoRR abs/2302.09425 (2023) <https://doi.org/10.48550/ARXIV.2302.09425>
- 8 Jonas Witt, Stef Rasing, Sebastijan Dumancic, Tias Guns, Claus-Christian Carbon: *A Divide-Align-Conquer Strategy for Program Synthesis*. CoRR abs/2301.03094 (2023) <https://doi.org/10.48550/ARXIV.2301.03094>

5 Panel discussions

5.1 Inductive Programming – How to Go On?

Ute Schmid (Universität Bamberg, DE), Claudia d'Amato (University of Bari, IT), Hector Geffner (RWTH Aachen, DE), Sriraam Natarajan (University of Texas at Dallas – Richardson, US), and Josh Rule (University of California – Berkeley, US)

License © Creative Commons BY 4.0 International license
© Ute Schmid, Claudia d'Amato, Hector Geffner, Sriraam Natarajan, and Josh Rule

In a final discussion we addressed topics and activities to make scientific progress and make the topic more visible. A crucial problem might be that there is no core general approach to IP (such as gradient descent for neural networks). The most prominent IP task is to learn programs from input/output examples. Other approaches address learning programs from traces or constraints. Methods range from classic inductive generalization and folding for induction of functional programs over genetic and evolutionary programming to a collection of ILP methods (sequential covering, theta-subsumption, combining with tools from answer set programming). Relevant use cases might not have a focus on learning recursion/loops but on relations (e.g. in medicine and biology). The focus on learning programs (including recursion) might profit from using Python as target language.

Furthermore, current IP systems are mostly not easy to find and to use. Therefore, a toolbox which can be easily used (such as Weka for standard ML) might be helpful. Currently Sebastijan Dumancic is working on such a tool box (Herb.jl). A collection of data sets and benchmark problems (see summary of the discussion group about benchmarks) would also be very helpful, especially when they are given in a standardized, easy to parse format.

To make the field of IP less distributed, it might be helpful to write a primer to IP including the classic approaches of inductive functional programming and relate also to deductive and transformational program synthesis methods and genetic/evolutionary programming. Papers falling in this category are:

- Andrew Cropper, Sebastijan Dumancic:
Inductive Logic Programming At 30: A New Introduction. J. Artif. Intell. Res. 74: 765-850 (2022) <https://doi.org/10.1613/JAIR.1.13507>
- Pierre Flener, Ute Schmid:
Inductive Programming. Encyclopedia of Machine Learning and Data Mining 2017: 658-666 https://doi.org/10.1007/978-1-4899-7687-1_137
(updated in 2020, not online yet)
- Pierre Flener, Ute Schmid:
An introduction to inductive programming. Artif. Intell. Rev. 29(1): 45-62 (2008)
<https://doi.org/10.1007/S10462-009-9108-7>
- Sumit Gulwani, Oleksandr Polozov, Rishabh Singh: *Program Synthesis*. Found. Trends Program. Lang. 4(1-2): 1-119 (2017) <https://doi.org/10.1561/2500000010>
- Emanuel Kitzelmann:
Inductive Programming: A Survey of Program Synthesis Techniques. AAIP 2009: 50-73
https://doi.org/10.1007/978-3-642-11931-6_3

As outcome of the AAIP 2023 seminar we plan to publish a book “Inductive Programming” which contains systematic introductions into the core topics as well as a collection of recent work (from participants of the seminar plus an open call for contributions) addressing topics such as “New Approaches to IP”, “Cognitive Aspects of IP”, “Applications of IP”.

Furthermore, it has been proposed to apply for an IP workshop at IJCAI and to try to include IP as a topic at the next European Summer School on AI (ESSAI). It might also be helpful for visibility and community building to propose a COST Network on IP.

The website <https://www.inductive-programming.org/> should be kept but made more general and link from there to a github for IP. The Wikipedia Entry on Inductive Programming https://en.wikipedia.org/wiki/Inductive_programming should be updated by the community. We also might make at least a tag `inductiveprogramming` at linkedin which the IP community should include in posts. More persons of the community should give Inductive Programming as Keyword in their google scholar profile. We could sample all videos related to IP in a YouTube channel, and we could produce a 3 minute introductory video.

Participants

- Lun Ai
Imperial College London, GB
- Martin Berger
University of Sussex –
Brighton, GB
- David Cerna
The Czech Academy of Sciences –
Prague, CZ
- David J. Crandall
Indiana University –
Bloomington, US
- Claudia d’Amato
University of Bari, IT
- Luc De Raedt
KU Leuven, BE
- Sebastijan Dumančić
TU Delft, NL
- Kevin Ellis
Cornell University – Ithaca, US
- Nathanaël Fijalkow
CNRS – Talence, FR
- Bettina Finzel
Universität Bamberg, DE
- Johannes Fürnkranz
Johannes Kepler Universität
Linz, AT
- Hector Geffner
RWTH Aachen, DE
- Céline Hocquette
University of Oxford, GB
- Frank Jäkel
TU Darmstadt, DE
- Emanuel Kitzelmann
Technische Hochschule
Brandenburg, DE
- Tomáš Kliegr
University of Economics –
Prague, CZ
- Maithilee Kunda
Vanderbilt University –
Nashville, US
- Johannes Langer
Universität Bamberg, DE
- Sriraam Natarajan
University of Texas at Dallas –
Richardson, US
- Stassa Patsantzis
University of Surrey –
Guildford, GB
- Josh Rule
University of California –
Berkeley, US
- Zeynep G. Saribatur
TU Wien, AT
- Ute Schmid
Universität Bamberg, DE
- Gust Verbruggen
Microsoft – Keerbergen, BE
- Felix Weitkämper
LMU München, DE

