Report from Dagstuhl Seminar 17382

Approaches and Applications of Inductive Programming

Edited by

Ute Schmid[1], Stephen H. Muggleton[2], and Rishabh Singh[3]

1 **Universität Bamberg, DE,** ute.schmid@uni-bamberg.de
2 **Imperial College London, GB,** s.muggleton@imperial.ac.uk
3 **Microsoft Research – Redmond, US,** rishabh@csail.mit.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 17382 "Approaches and Applications of Inductive Programming". 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 September 17–20, 2017 – http://www.dagstuhl.de/17382
1998 ACM Subject Classification I.2.2 Automatic Programming, Program Synthesis
Keywords and phrases inductive program synthesis, inductive logic programming, probabilistic programming, end-user programming, human-like computing
Digital Object Identifier 10.4230/DagRep.7.9.86
Edited in cooperation with Sebastian Seufert

1 Executive summary

Ute Schmid

Inductive programming (IP) 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 [5]. 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 [17, 8, 18]. On the other hand, inductive programming is a special approach to program synthesis. It complements deductive and transformational approaches [20, 14, 2]. 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 [16].

Inductive program synthesis is of interest for researchers in artificial intelligence since the late sixties [1]. 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 [3, 12, 4]. For example, the Microsoft Excel plug-in Flashfill synthesizes programs from a small set of observations of user behavior [8, 7, 6]. Related

applications are in process mining and in data wrangling [11]. Inductive programming in general offers powerful approaches to learning from relational data [15, 13] and to learning from observations in the context of autonomous intelligent agents [10, 17]. Furthermore, inductive programming can be applied in the context of teaching programming [19, 21].

Recent Trends and Applications

When the first Dagstuhl Seminar on Approaches and Applications of Inductive Programming took place in 2013, the following trends could be identified:

- Combining different approaches to inductive programming to leverage their complementary strengths.
- New inductive programming approaches based on adapting and using well-developed techniques such as SAT-solving.
- Putting inductive programming to application, for example in the areas of automated string manipulations in spreadsheets or web programming.
- Applying concepts of inductive programming to cognitive models of learning structural concepts.

One of the major outcomes of the first Dagstuhl Seminar was a joint publication in the Communications of the ACM [8] where these trends and first applications and results were described. In the seminar 2015, the following additional trends were identified:

- Evaluation of inductive programming approaches – in relation to general intelligence and in relation to standard machine learning.
- Application of inductive programming to teaching programming.
- Inductive programming as a model of human inductive learning.

The main outcomes of the second seminar were (1) a joint publication in the Artificial Intelligence Journal with respect to the evaluation of computational models solving intelligence test problems – among them inductive programming systems [9], (2) a joint publication addressing comprehensibility as a second criterium to evaluate machine learning approaches besides accuracy [18], and (3) a NIPS'2016 workshop on Neural Nets and Program Induction[1].

Based on the results of the second seminar, the focus of the third seminar has been on 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 more suited than standard machine learning approaches, including deep learning.
- 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 enduser programming and programming education and identifying further relevant areas of application.
- Establishing stronger relations between cognitive science research on inductive learning and inductive programming.

In the seminar, we brought together researchers from different areas of computer science – especially from machine learning, AI, declarative programming, and software engineering

[1] https://uclmr.github.io/nampi/

– and researchers from cognitive psychology interested in inductive learning as well as in teaching and learning computer programming. Furthermore, participants from industry presented current as well as visionary applications for inductive programming.

The seminar was opened with lecture style talks introducing the four major approaches of inductive programming: Inductive functional programming, inductive logic programming, inductive probabilistic logical programming, and programming by example.

Talks covered current developments of IP algorithms, challenging applications –especially in data wrangling and in education –, and relations of IP to cognition.

In addition, several system demons and tutorials were given: Igor and EasyIgor (by Sebastian Seufert and Ute Schmid), MagicHaskeller (by Susumu Katayama), Sketch (by Armando Solar-Lezama), PROSE (by Oleksandr Polozov), Slipcover (by Fabrizio Riguzzi), and TaCLe (by Luc De Raedt).

The following topics were identified and further discussed in working groups during the seminar:

- How to determine relevancy of background knowledge to reduce search?
- Integrating IP with other types of machine learning, especially Deep Learning?
- Data wrangling as exiting area of application.

Additional topics identified as relevant have been anomaly detection, noise, robustnes, as well as non-example based interaction (e.g., natural language).

Concluding remarks and future plans

In the wrapping-up section, we collected answers to the question

"What would constitute progress at Dagstuhl 2019?"

The most prominent answers were

- make available systems, data sets (via IP webpage[2])
- compare systems
- common vocabulary, work on applications attempted by others: drawing problems, string transformation, general ai challenge, benchmarks starexec, learn robot strategy, grammar learning what is inductive programming
- open problems

As the grand IP challenge we came up with: **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 A. W. Biermann, G. Guiho, and Y. Kodratoff, editors. *Automatic Program Construction Techniques*. Macmillan, New York, 1984.
2 Rastislav Bodik and Emina Torlak. Synthesizing programs with constraint solvers. In *CAV*, page 3, 2012.
3 A. Cypher, editor. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, 1993.
4 Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols, editors. *No Code Required: Giving Users Tools to Transform the Web*. Elsevier, 2010.

[2] www.inductive-programming.org

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

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

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

8 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.

9 José Hernández-Orallo, Fernando Martínez-Plumed, Ute Schmid, Michael Siebers, and David L Dowe. Computer models solving intelligence test problems: Progress and implications. *Artificial Intelligence*, 230:74–107, 2016.

10 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.

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

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

13 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.

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

15 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.

16 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.

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

18 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.

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

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

21 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 A Short Introduction to Inductive Functional Programming

Ute Schmid (Universität Bamberg, DE)

Joint work of Ute Schmid, Emanuel Kitzelmann
Main reference Pierre Flener, Ute Schmid: "An introduction to inductive programming", Artif. Intell. Rev., Vol. 29(1), pp. 45–62, 2008.
URL http://dx.doi.org/10.1007/s10462-009-9108-7

In this talk, a short introduction to inductive functional programming is given. Specifically, a brief outline of the history of inductive functional programming is presented. The milestone system Thesys (Summers, 1977) is introduced. Current developments are presented with a focus on our own system Igor.

3.2 Inductive Logic Programming

Stephen H. Muggleton (Imperial College London, GB)

Main reference Stephen H. Muggleton: "Meta-Interpretive Learning: Achievements and Challenges (Invited Paper)", in Proc. of the Rules and Reasoning – International Joint Conference, RuleML+RR 2017, London, UK, July 12-15, 2017, Proceedings, Lecture Notes in Computer Science, Vol. 10364, pp. 1–6, Springer, 2017.
URL http://dx.doi.org/10.1007/978-3-319-61252-2_1

Meta-Interpretive Learning (MIL) is a recent Inductive Logic Programming technique aimed at supporting learning of recursive definitions. A powerful and novel aspect of MIL is that when learning a predicate definition it automatically introduces sub-definitions, allowing decomposition into a hierarchy of reuseable parts. MIL is based on an adapted version of a Prolog meta-interpreter. Normally such a meta-interpreter derives a proof by repeatedly fetching first-order Prolog clauses whose heads unify with a given goal. By contrast, a meta-interpretive learner additionally fetches higher-order meta-rules whose heads unify with the goal, and saves the resulting meta-substitutions to form a program. This talk will overview theoretical and implementational advances in this new area including the ability to learn Turing computabale functions within a constrained subset of logic programs, the use of probabilistic representations within Bayesian meta-interpretive and techniques for minimising the number of meta-rules employed. The talk will also summarise applications of MIL including the learning of regular and context-free grammars, learning from visual represention with repeated patterns, learning string transformations for spreadsheet applications, learning and optimising recursive robot strategies and learning tactics for proving correctness of programs. The talk will conclude by pointing to the many challenges which remain to be addressed within this new area.

3.3 A Short Introduction to Probabilistic Logic Programming

Luc De Raedt (KU Leuven, BE)

Joint work of Luc De Raedt, Angelika Kimmig
Main reference Luc De Raedt, Angelika Kimmig: "Probabilistic (logic) programming concepts", Machine Learning, Vol. 100(1), pp. 5–47, 2015.
URL http://dx.doi.org/10.1007/s10994-015-5494-z

In this introductory talk, a short introduction to probabilistic programming principles was given. It was centered around the distribution semantics of Sato and the probabilistic programming language Problog.

3.4 Programming By Examples for Data Transformation and Integration

Rishabh Singh (Microsoft Research – Redmond, US)

In this talk, I will briefly summarize some of the past work in building efficient programming by example (PBE) systems for data wrangling tasks using Version-space algebras [3]. There are four key parts of designing such PBE systems: 1) designing an expressive and concise domain-specific language (DSL) for constructing the hypothesis space, 2) designing efficient data structures to succinctly represent an exponential number of programs in polynomial space, 3) a learning algorithm that learns a set of consistent programs in the DSL that conform to a set of user-provided input-output examples, and finally 4) a ranking function for selecting the most likely programs. I will then present some of the more recent systems we have built with new advances on top of a similar methodology in the domains of semantic data transformations [2], input-driven data mainpulation [1], and data integration from web sources [4].

References

1 Rishabh Singh. BlinkFill: Semi-supervised Programming By Example for Syntactic String Transformations. PVLDB 9(10): 816-827 (2016)
2 Rishabh Singh, Sumit Gulwani. Transforming spreadsheet data types using examples. POPL 2016: 343-356
3 Sumit Gulwani, William R. Harris, Rishabh Singh. Spreadsheet data manipulation using examples. Commun. ACM 55(8): 97-105 (2012)
4 Jeevana Inala and Rishabh Singh. WebRelate: Integrating Web Data with Spreadsheets using Examples. POPL 2018

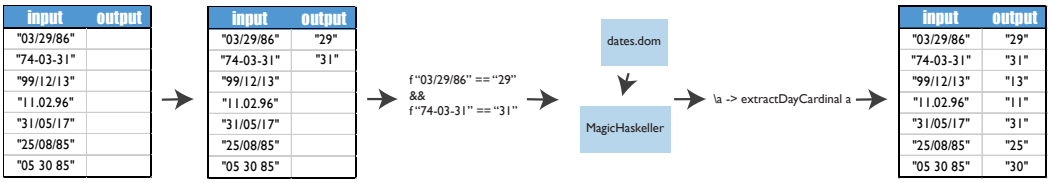

Figure 1 System functionality. Once the user provides input/output examples and the domain (DBBK), the system tries to induce the possible transformation to be applied.

4 Overview of Talks

4.1 The BigLambda Project

Aws Albarghouthi (University of Wisconsin, Madison, US)

In this talk, I cover various pieces of the BigLambda project, which aims to synthesize data analysis programs from examples. First, I discuss the BigLambda system and how it synthesizes MapReduce-style programs that can execute in parallel. Second, I describe how one can extract domain-knowledge of a synthesis domain by observing test runs of an API. Third, I will describe future problems involving synthesizing programs under privacy constraints.

4.2 Domain Specific Induction for Data Wrangling Automation

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

Joint work of Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Susumu Katayama, Fernando Martínez-Plumed, María José Ramírez-Quintana
Main reference Lidia Contreras-Ochando, Cèsar Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, Susumu Katayama: "Domain specific induction for data wrangling automation (Demo)", Automatic Machine Learning Workshop (AutomML) @ ICML 2017, Sidney, Australia, 2017.
URL http://users.dsic.upv.es/ flip/papers/AutoML_ICML2017.pdf

Inside the data science process, data wrangling is the step that involves transforming data, cleaning datasets and combining them to create new ones, and this step can consume up to 80% of the project time [1]. Automating data wrangling process is essential to reduce time and cost in our projects. Our proposal to solve this problem includes the use of general-purpose inductive programming learning systems with general-purpose declarative languages, using an appropriate library that defines a domain-specific background knowledge [2]. The overall idea is to automate the process of transforming data from one format to another, depending on the data domain and using MagicHaskeller as the inductive programming system, with only one or few examples (Figure 1). Our approach is able to solve several problems by using the correct domain independently of the data format.

Acknowledgments. This work has been partially supported by the EU (FEDER) and the Spanish MINECO project TIN 2015-69175-C4-1-R (LOBASS) and by Generalitat Valenciana under ref. PROMETEOII/2015/013 (SmartLogic). Lidia Contreras was supported by FPU-ME grant FPU15/03219.

References

1 Steinberg, D. *How Much Time Needs to be Spent Preparing Data for Analysis?* https://info.salford-systems.com/blog/bid/299181/How-Much-Time-Needs-to-be-Spent-Preparing-Data-for-Analysis

2 Contreras-Ochando, L. and Martínez-Plumed, F. and Ferri, C. and Hernández-Orallo, J. and Ramírez-Quintana, M. J. *General-Purpose Inductive Programming for Data Wrangling Automation*. AI4DataSci @ NIPS 2016, 2016.

4.3 Learning Higher-Order Logic Programs through Abstraction and Invention

Andrew Cropper (Imperial College London, GB) and Stephen H. Muggleton (Imperial College London, GB)

Main reference Andrew Cropper, Stephen H. Muggleton: "Learning Higher-Order Logic Programs through Abstraction and Invention", in Proc. of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, pp. 1418–1424, IJCAI/AAAI Press, 2016.
URL http://www.ijcai.org/Abstract/16/204

Many tasks in AI require the design of complex programs and representations, whether for programming robots, designing game-playing programs, or conducting textual or visual transformations. This paper [1] explores a novel inductive logic programming approach to learn such programs from examples. To reduce the complexity of the learned programs, and thus the search for such a program, we introduce higher-order operations involving an alternation of Abstraction and Invention. Abstractions are described using logic program definitions containing higher-order predicate variables. Inventions involve the construction of definitions for the predicate variables used in the Abstractions. The use of Abstractions extends the Meta-Interpretive Learning framework and is supported by the use of a user-extendable set of higher-order operators, such as *map*, *until*, and *ifthenelse*. Using these operators reduces the textual complexity required to express target classes of programs. We provide sample complexity results which indicate that the approach leads to reductions in the numbers of examples required to reach high predictive accuracy, as well as significant reductions in overall learning time. Our experiments demonstrate increased accuracy and reduced learning times in all cases. We believe that this paper is the first in the literature to demonstrate the efficiency and accuracy advantages involved in the use of higher-order abstractions.

References

1 Andrew Cropper and Stephen H. Muggleton. Learning higher-order logic programs through abstraction and invention. In Subbarao Kambhampati, editor, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, pages 1418–1424. IJCAI/AAAI Press, 2016.

4.4 Learning Constraints in Spreadsheets and Tabular Data

Luc De Raedt (KU Leuven, BE)

Joint work of Samuel Kolb, Sergey Paramonov, Tias Guns, Luc De Raedt
Main reference Samuel Kolb, Sergey Paramonov, Tias Guns, Luc De Raedt: "Learning constraints in spreadsheets and tabular data", Machine Learning, Vol. 106(9-10), pp. 1441–1468, 2017.
URL http://dx.doi.org/10.1007/s10994-017-5640-x

Spreadsheets, comma separated value files and other tabular data representations are in wide use today. However, writing, maintaining and identifying good formulas for tabular data and spreadsheets can be time-consuming and error-prone. We investigate the automatic learning of constraints (formulas and relations) in raw tabular data in an unsupervised way. We represent common spreadsheet formulas and relations through predicates and expressions whose arguments must satisfy the inherent properties of the constraint. The challenge is to automatically infer the set of constraints present in the data, without labeled examples or user feedback. This approach is based on inductive programming.

4.5 Applying ILP to Sequence Induction Tasks

Richard Evans (Google DeepMind – London, GB)

We describe a system for solving Hofstadter's "Seek Whence" tasks. This is an inductive logic program system, that uses (broadly) Kantian constraints to group its percepts into persistent objects, changing over time, according to causal laws. This system is able to achieve human-level performance in the "Seek Whence" domain.

Similar to Hofmann, Kitzelmann, and Schmid [1], we avoid domain-specific heuristics, and focus on a domain-independent solution to this task. We claim that the prior constraints we use, inspired by Kant's Principles in the Critique of Pure Reason, are domain-independent prior knowledge.

References

1 Hofmann, J., Kitzelmann, E. and Schmid, U. (2014). Applying Inductive Program Synthesis to Induction of Number Series A Case Study with IGOR2. *KI 2014: Advances in Artificial Intelligence*, pp. 25-36. Springer.

4.6 What's behind this model?

Cesar Ferri Ramirez (Technical University of Valencia, ES)

Joint work of Raül Fabra Boluda, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana

Machine learning is being widely used in applications related to security or applications that deal with confidential information. Examples of these applications are spam detection, malware classification, detection of network intrusion ... In many of these cases, data

can be actively manipulated by an intelligent adversary. Those adversarial agents can also try to extract and mimic (possibly confidential) machine learning models aiming at taking advantage of them and, in this way, evade detections and alarms. Many of the developed methods to manipulate/attack models are technique-based. This implies that these methods have been defined considering that they know the family of the machine learning technique (decision trees, neural networks,...) used to learn the target model that they want to manipulate. In this work we propose some methods to capture information about models (seen as black boxes). The information we plan to obtain is: family of the learning technique, significance of the different members of the feature space and the possible existence of attribute transformations. Our first approach is based on mimeting the target models, and then from the mimetic models. we extract meta-features by the application of meta-learning techniques. We also discuss about the feasibility of extracting knowledge in declarative models. Examples of details we want to discover from declarative models are: the existence of relational patterns between features, recursive patterns in the model, or the existence of attribute transformations, such as the use of propositionalisation for complex features. We also discuss about the feasibility of extracting knowledge from declarative models such as the existence of relational patterns between features, recursive patterns in the model, or the existence of attribute transformations (for instance, propositionalisation for complex features).

Acknowledgments. This work has been partially supported by the EU (FEDER) and the Spanish MINECO project TIN 2015-69175-C4-1-R (LOBASS) and by Generalitat Valenciana under ref. PROMETEOII/2015/013 (SmartLogic).

4.7 Interacting with Program Synthesis by Example: Designing Around Human Cognition

Elena Glassman (University of California – Berkeley, US)

Joint work of Elena Glassman, Gustavo Soares, Andrew Head, Ryo Suzuki, Lucas Figueredo, Bjoern Hartmann, Loris D'Antoni
Main reference Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, Björn Hartmann: "Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis", in Proc. of the Fourth ACM Conference on Learning @ Scale, L@S 2017, Cambridge, MA, USA, April 20-21, 2017, pp. 89–98, ACM, 2017.
URL http://dx.doi.org/10.1145/3051457.3051467

This talk is motivated by recent successes and challenges in designing interfaces for interacting with example-driven synthesis backends, i.e., MistakeBrowser and FixPropagator [Head et al. 2017]. I then explain Variation Theory, a well-tested theory of how humans form hypotheses, i.e., learn concepts and rules, from examples. Finally, I discuss how interfaces in the future can help (1) users accurately teach programs to example-driven synthesis backends and (2) communicate those learned programs back to the user.

4.8 The Draughtsman's Assistant

Stephen H. Muggleton (Imperial College London, GB)

Abstraction is an essential and defining property of human learning and thought. Human programmers use abstraction to define meaningful variables, data types, procedures, parameters, conditions and hierarchical problem decomposition. This presentation provides a demonstration of the idea of programming by drawing using the Metagol_AI system. We assume a human teacher draws diagrams on a 2D array and a Meta-Interpretretive Learner uses primitives from the Postscript drawing language to build programs which imitate the drawings. It is assumed programs for drawing symbols, such as an L, from a single example. Abstraction and invention mechanisms can then be used to learn numbers, such as two three or four which can be applied to L to produce multiple instances of L. Incremental learning is then used to build larger programs by building on previously learned programs.

4.9 Towards Ultra-Strong Machine Learning Comprehensibility of Programs Learned with ILP

Stephen H. Muggleton (Imperial College London, GB)

Joint work of Ute Schmid, Christina Zeller, Tarek Besold, Alireza Tamaddoni-Nezhad, Stephen Muggleton
Main reference Ute Schmid, Christina Zeller, Tarek Besold, Alireza Tamaddoni-Nezhad, Stephen Muggleton: "How Does Predicate Invention Affect Human Comprehensibility?", in Proc. of the Inductive Logic Programming – 26th International Conference, ILP 2016, London, UK, September 4-6, 2016, Revised Selected Papers, Lecture Notes in Computer Science, Vol. 10326, pp. 52–67, Springer, 2016.
URL http://dx.doi.org/10.1007/978-3-319-63342-8_5

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, tended to use a one-dimensional approach to Machine Learning based solely on predictive accuracy, ultimately favouring statistical over symbolic Machine Learning approaches. In this presentation we provide a definition of comprehensibility of hypotheses which can be estimated using human participant trials. We present the results of experiments testing human comprehensibility of logic programs learned with and without predicate invention. Results indicate that comprehensibility is affected not only by the complexity of the presented program but also by the existence of anonymous predicate symbols.

4.10 Learning from Observation

Katsumi Inoue (National Institute of Informatics – Tokyo, JP)

Two approaches to Learning from Observation are given in this talk. One is Meta-level abduction (MLA), and the other is Learning from interpretation transition (LFIT). In MLA, abduction is performed at the meta-level, enabling us to abduce rules and predicate/object invention. In LFIT, relational dynamics is learned from transition of states of a system in the form of state transition rules.

References

1 Katsumi Inoue, Koichi Furukawa, Ikuo Kobayashi, Hidetomo Nabeshima: Discovering Rules by Meta-level Abduction, in: Post-Proceedings of ILP 2009, LNAI, Vol.5989, pp.49-64, Springer (2010).
2 Katsumi Inoue, Andrei Doncescu, Hidetomo Nabeshima: Completing causal networks by meta-level abduction, Machine Learning, 91(2): 239-277 (2013).
3 Katsumi Inoue: Meta-Level Abduction, IfCoLog Journal of Logics and their Applications, 3(1): 7-35 (2016).
4 Katsumi Inoue, Tony Ribeiro, Chiaki Sakama: Learning from Interpretation Transition, Machine Learning, 94(1):51-79 (2014).
5 Tony Ribeiro, Morgan Magnin, Katsumi Inoue, Chiaki Sakama: Learning delayed influences of biological systems, Frontiers in Bioengineering and Biotechnology, 2(81) (2015).
6 Sophie Tourret, Enguerrand Gentet, Katsumi Inoue: Learning Human-Understandable Description of Dynamical Systems from Feed-Forward Neural Networks, in: Proceedings of ISNN'17(1), LNCS, 10261: 483-492 (2017).
7 David Martínez, Guillem Alenya, Tony Ribeiro, Katsumi Inoue, Carme Torras: Relational reinforcement learning for planning with exogenous effects, Journal of Machine Learning Research, 18(78) (2017).

4.11 MagicHaskeller-based Incrementally Learning Agent

Susumu Katayama (University of Miyazaki, JP)

This presentation introduced a general AI agent with incremental learning that uses MagicHaskeller. In addition, experiences of applying the agent to Round 1 of General AI Challenge was presented.

4.12 SUPERVASION project – Supervision by Observation using Inductive Programming

David Nieves Cordones (Technical University of Valencia, ES)

Joint work of David Nieves Cordones, Carlos Monserrat, José Hernández-Orallo
Main reference C. Monserrat, J. Hernandez-Orallo, J.-F. Dolz, M.-J. Ruperez, P. Flach: "Knowledge acquisition by abduction for skills monitoring: Application to surgical skills" In Inductive Logic Programming, 2016.
URL http://ilp16.doc.ic.ac.uk/program/papers/24

We present some progress in the project SUPERVASION, part of which was presented at ILP16 [1]. This project proposes a system for automated monitoring of apprentices using information from one high-level explanation given by an expert (a narrative) and one (or very few) video-recorded executions of the procedure. This process of supervision is divided in two phases: knowledge acquisition, where system learns from expert examples; and online supervision, in which the automated supervisor assists as a virtual expert to the trainee during the training. Also, we use Event Calculus [2] for logical reasoning about observable properties and abstract concepts in time. The newly system has been used to learn and detect the starting and ending points of high-level fluents in different trainings in minimally invasive surgery. The experimental results show the potential of the developed tool.

Acknowledgments. This work has been partially supported by the EU (FEDER) and the Spanish MINECO under grant TIN2014-61716-EXP (SUPERVASION) and TIN 2015-69175-C4-1-R (LOBASS), and by Generalitat Valenciana under grant PROMETEOII/2015/013. David Nieves was supported by FPI-MINECO grant BES-2016-078863.

J. H-O was visiting the Leverhulme Centre for the Future of Intelligence, generously funded by the Leverhulme Trust. He thanks UPV for the sabbatical leave and the funding from the Spanish MECD programme "Salvador de Madariaga" (PRX17/00467) and the GVA grants for research stays.

References

1 Monserrat, C., Hernández-Orallo, J., Dolz, J.F., Rupérez, M.J. and Flach, P. *Knowledge Acquisition by Abduction for Skills Monitoring: Application to Surgical Skills*. 26th International Conference on Inductive Logic Programming, ILP2016, 2016.
2 Artikis, A., Sergot, M., and Paliouras, G. *An event calculus for event recognition*. IEEE Transactions on Knowledge and Data Engineering Computer, vol. 27, no. 4, pp. 895–908, 2015.

4.13 Programming Not Only by Examples

Hila Peleg (Technion – Haifa, IL)

Joint work of Hila Peleg, Shaorn Shoham, Eran Yahav

In recent years, there has been tremendous progress in automated synthesis techniques that are able to automatically generate code based on some intent expressed by the programmer. A major challenge for the adoption of synthesis remains in having the programmer communicate their intent. When the expressed intent is coarse-grained (for example, restriction on the

expected type of an expression), the synthesizer often produces a long list of results for the programmer to choose from, shifting the heavy-lifting to the user. An alternative approach, successfully used in end-user synthesis is programming by example (PBE), where the user leverages examples to interactively and iteratively refine the intent. However, using only examples is not expressive enough for programmers, who can observe the generated program and refine the intent by directly relating to parts of the generated program.

We present a novel approach to interacting with a synthesizer using a granular interaction model. Our approach employs a rich interaction model where (i) the synthesizer decorates a candidate program with debug information that assists in understanding the program and identifying good or bad parts, and (ii) the user is allowed to provide feedback not only on the expected output of a program, but also on the underlying program itself. That is, when the user identifies a program as (partially) correct or incorrect, they can also explicitly indicate the good or bad parts, to allow the synthesizer to accept or discard parts of the program instead of discarding the program as a whole.

We show the value of our approach in a controlled user study. Our study shows that participants have strong preference to using granular feedback instead of examples, and are able to provide granular feedback much faster.

4.14 Probabilistic Inductive Logic Programming with SLIPCOVER

Fabrizio Riguzzi (University of Ferrara, IT)

Joint work of Fabrizio Riguzzi, Elena Belliodi
Main reference Elena Bellodi, Fabrizio Riguzzi: "Structure learning of probabilistic logic programs by searching the clause space", TPLP, Vol. 15(2), pp. 169–212, 2015.
URL http://dx.doi.org/10.1017/S1471068413000689

The combination of logic and probability is very useful for modeling domains with complex and uncertain relationships among entities. Machine learning approaches based on such combinations have recently achieved important results, originating the fields of Statistical Relational Learning, Probabilistic Inductive Logic Programming and, more generally, Statistical Relational Artificial Intelligence.

Probabilistic languages based on Logic Programming are particularly promising because of the large body of techniques for inference and learning developed in Logic Programming. Sato's distribution semantics [9] is a possible worlds semantics that emerged as one of the more prominent approach for giving a meaning to Probabilistic Logic Programs. It is adopted by many languages such as the Independent Choice Logic, PRISM, Logic Programs with Annotated Disjunctions, CP-logic and ProbLog.

The talk will illustrates the basics of semantics and inference for these languages and will present the SLIPCOVER system [2] for Probabilistic Inductive Logic Programming. SLIPCOVER learns both the structure and the parameters of Logic Programs with Annotated Disjunctions and CP-logic by performing clause revision followed by greedy theory search.

The talk will also present the cplint on SWISH [6, 1] (http://cplint.ml.unife.it) web application for experimenting with SLIPCOVER. The application can also be used to

- perform probabilistic inference with the PITA [7, 8] and MCITYRE algorithms [5]
- perform Inductive Logic Programming with Aleph [10]
- draw ROC and Precision-Recall curves [3, 4].

These features will be briefly discusses in the talk.

References

1 Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., Zese, R.: cplint on SWISH: Probabilistic Logical Inference with a Web Browser. Intell. Artif. 11(1), 47–64 (2017)
2 Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theor. Pract. Log. Prog. 15(2), 169–212 (2015)
3 Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: European Conference on Machine Learning (ECML 2006). pp. 233–240. ACM (2006)
4 Fawcett, T.: An introduction to ROC analysis. Pattern Recognit. Lett. 27, 861–874 (2006)
5 Riguzzi, F.: MCINTYRE: A Monte Carlo system for probabilistic logic programming. Fund. Inform. 124(4), 521–541 (2013)
6 Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Probabilistic logic programming on the web. Softw.-Pract. Exper. 46(10), 1381–1396 (October 2016)
7 Riguzzi, F., Swift, T.: The PITA system: Tabling and answer subsumption for reasoning under uncertainty. Theor. Pract. Log. Prog. 11(4–5), 433–449 (2011)
8 Riguzzi, F., Swift, T.: Well-Definedness and Efficient Inference for Probabilistic Logic Programming under the Distribution Semantics. Theor. Pract. Log. Prog. 13(Special Issue 02 – 25th Annual GULP Conference), 279–302 (2013)
9 Sato, T.: A Statistical Learning Method for Logic Programs with Distribution Semantics. In: Sterling, L. (ed.) 12th International Conference on Logic Programming, Tokyo, Japan. pp. 715–729. MIT Press, Cambridge, Massachusetts (1995)
10 Srinivasan, A.: Aleph (03-04-2012), http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html

4.15 Human Learning in the Michalski Train Domain

Ute Schmid (Universität Bamberg, DE)

Joint work of Ute Schmid, Jonas Troles, Johannes Birk, Christina Zeller

Human-like computing (HLC) is relevant for designing AI systems which allow for a comprehensible interaction between humans and machines. One aspect of HLC is to get a better understanding of human cognitive processes. We are interested in comparing human relational learning with ILP and conducted a first experiment to explore how good humans are to learn rules off different structural complexity in the Michalski train domain. We could show, that a simple linear recursive rule can be generalized nearly as efficiently as a conjunction of features while a rule characterizing a relation between objects which can occur at an arbitrary position is nearly as difficult to learn as a disjunction.

4.16 Learning and Decision-making in Artificial Animals

Claes Strannegård (Chalmers University of Technology Göteborg, SE)

I will discuss artificial ecosystems with animals and plants that live in blocks worlds, e.g. in the Minecraft environment. The artificial animals (or animats) are capable of perception, learning, decision-making, and action. Moreover, they have fixed sets of needs for certain

resources, e.g. water and energy. If an animat runs out of some resource, then it dies by definition. The learning and decision-making mechanisms are the same for all animats. The sole goal for all animats is to avoid death. The animats perceive their environment by means of extended boolean circuits. These circuits evolve over time according to a fixed set of rules for learning and forgetting. The animats are capable of sexual or asexual reproduction. In the sexual case, two animats with similar enough genome are able to reproduce under certain circumstances. Together these mechanisms give rise to autonomous ecosystems of animats that interact with each other and continuously adapt to changing environments by learning and evolution.

4.17 Learning to Forget – First Explorations

Michael Siebers (Universität Bamberg, DE)

Joint work of Michael Siebers, Ute Schmid, Kyra Göbel, Cornelia Niessen

I present recent advances in the Dare2Del-Project. The goal of the project is to assess the irrelevance of digital objects. In this talk I will focus on files in an artificial file system. A first representation is shown together with preliminary results on the learning of the derived target predicate. I conclude with challenges posed by the learning task.

4.18 Neural Program Synthesis

Rishabh Singh (Microsoft Research – Redmond, US)

Main reference Rishabh Singh, Pushmeet Kohli: "AP: Artificial Programming", in Proc. of the 2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA, LIPIcs, Vol. 71, pp. 16:1–16:12, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
URL http://dx.doi.org/10.4230/LIPIcs.SNAPL.2017.16

The key to attaining general artificial intelligence is to develop architectures that are capable of learning complex algorithmic behaviors modeled as programs[1, 2, 3, 4]. The ability to learn programs allows these architectures to learn to compose high-level abstractions with complex control-flow, which can lead to many potential benefits: i) enable neural architectures to perform more complex tasks, ii) learn interpretable representations (programs which can be analyzed, debugged, or modified), and iii) better generalization to new inputs (like algorithms). In this talk, I will present some of our recent work in developing neural architectures for learning complex regular-expression based data transformation programs from input-output examples, and will also briefly discuss some other applications such as program repair[6] and fuzzing[5] that can benefit from learning neural program representations.

References

1 Rishabh Singh, Pushmeet Kohli. *AP: Artificial Programming*. SNAPL 2017: 16:1-16:12
2 Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, Pushmeet Kohli. *RobustFill: Neural Program Learning under Noisy I/O*. ICML 2017: 990-998

3 Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, Pushmeet Kohli. *Neuro-Symbolic Program Synthesis*. ICLR 2017

4 Jacob Devlin, Rudy R. Bunel, Rishabh Singh, Matthew J. Hausknecht, Pushmeet Kohli. *Neural Program Meta-Induction*. NIPS 2017: 2077-2085

5 Patrice Godefroid, Hila Peleg, Rishabh Singh. *Learn & Fuzz: machine learning for input fuzzing*. ASE 2017: 50-59

6 Sahil Bhatia, Rishabh Singh. *Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks*. CoRR abs/1603.06129 (2016)

5 Working groups

5.1 Meta-Knowledge and Relevance of Background Knowledge

Stephen H. Muggleton, Ute Schmid, Fabrizio Riguzzi, Susumu Katayama, Andrew Cropper, Hila Peleg, Richard Evans

Motivation

Imagine a life-long learning system, which learns new functions (or predicates) from examples during a series of teaching episodes. At the end of each episode the learned functions are added to the library of those available for definitions in subsequent episodes. As the system's library of functions expands we need to decide how to prevent the progressive build-up of available functions swamping the search. We refer to this as the **Relevance Problem**. In particular, we need a method for efficiently identifying a small subset of the available functions which should be used in each new episode. This is a hard and, as yet, unsolved problem for Inductive Programming.

Identifying Relevant Background Knowledge

The Relevance Problem has some similarities to the Frame Problem, which involves finding an adequate collections of axioms for a viable description of a robot environment. In the case of building a learning assistant the Relevance Problem would be critical for applications such as developing a "personal background knowledge manager".

Concrete Proposals

Below are a list of the concrete proposals for relevance detection.

1. Order the functions according to how frequently they are used.
2. Annotating functions with type information, and then consider only functions whose types match the input/output pairs in the provided examples.
3. Simply do not bothering remembering learned functions from previous episodes but instead invent (or reinvent) new ones as needed.
4. Look at analogous functions: eg this function has the right form, but it operates on lists, not trees.
5. Generalise metarules from background definitions to form a template. This could be done by abstracting out the particular variables and predicates used.
6. Store a set of input/output pairs for each learned function, and use this to filter suitable functions.

Further Ideas

The group also discussed the related issue of IP systems which invent new data-types as needed to solve a problem. The idea of creating new data-types is a core ingredient of real programming, and the group agreed it should be studied within IP. Also we discussed the fact that knowing what is relevant requires knowing what we know and what we do not know. For this purpose it was suggested we need some way of modelling our ignorance, such as auto-epistemic logic. Andrew Cropper and Stephen Muggleton have shown that all programs over dyadic predicates can be subsumed by a single pair of metarules. Ute Schmid asked whether it is possible that there is one individual metarule that subsumes all of them? (A Scheffer's stroke for metarules). We also discussed whether it would be possible to mine Prolog programs from github to extract a library of background predicates?

Conclusion

Program Induction presently faces a bottleneck in only ever being able to construct small programs. The group agreed that solving the Relevance Problem was key in overcoming this bottleneck and suggested a variety of approaches which might be implemented and tested to see how effective they are.

5.2 Combining Inductive Programming with Machine Learning

Rishabh Singh, Oleksandr Polozov, Cesar Ferri Ramirez, Aws Albarghouthi, Katsumi Inoue, Michael Siebers, Christina Zeller

There has been a lot of recent interest in using machine learning techniques, especially deep learning techniques, for learning programs from inductive specifications such as input-output examples. There are three main reasons for this surge in recent interest: 1) Learning programs allows neural architectures to perform complex algorithmic tasks compared to simple classification tasks, 2) the prior coming from the underlying hypothesis space in terms of the Domain-specific language of programs allows the learnt models to generalize better on unseen data, and finally 3) the learnt models (in the form of programs) are more interpretable, which can be inspected, verified, and even modified.

There are many ways in which machine learning techniques can aid in the synthesis process to learn programs. One idea can be to automatically learn a policy to perform efficient search over a large space of programs. There has been some recent work (e.g. Neural Turing Machines, Neural Random Access Machines) in embedding program semantics in a differentiable manner such that the programs can be learned using gradient descent based optimization techniques in an end-to-end manner. Some other approaches such as RobustFill learn a generative model of programs in a language conditioned on the examples by learning a function that predicts expansions in a context-free grammar (DSL). Finally, there can be some approaches to combine symbolic search approaches (such as VSA, constraint-solving, integer programming etc.) with machine learning techniques to guide and reduce the search space of programs.

One of the key tasks in inductive programming is to define a good hypothesis space that is expressive enough to express a large class of desired programs but at the same concise enough for efficient search. Machine learning techniques can be used to automatically construct such hypothesis spaces given a large amount of data in a given domain. For example, they can be used to learn idioms/sub-routines that are commonly used over a large number of tasks, which can be used to add new operators in the DSL. One longer term goal would be to see if the learning techniques may help in constructing the complete DSL from scratch only from data.

Some specification mechanisms are inherently noisy and ambiguous. For example, consider specifications in the form of hand drawings, natural language, or pictures. Converting these ambiguous specifications into a formal representation requires machine learning techniques such that the specifications can become usable. Even for deterministic specifications such as FlashFill input-output example strings, there can sometimes be noise in the example strings. Since the traditional synthesis techniques are sound, they will either generate a very complex

program for such noisy examples or return no program if there is no DSL program that is consistent with all noisy examples. Using machine learning techniques can aid these systems to tolerate some noise in the specification and make the synthesis process more robust.

Inductive Programming techniques can also be used for helping machine learning techniques. For example, inductive programming can be used to automate the search of neural network architectures that work best for a given dataset. We can design a DSL consisting of neural network primitives, which can then be searched to compute new networks that compose the primitives with an objective to achieve the best performance on a dataset. Inductive programming can also be used for preprocessing background knowledge and generate new features for machine learning systems. Finally, inductive programming techniques can also be useful for learning hierarchical symbolic structures on top of low-level black box modules that are learnt using machine learning. For example, consider the task of summing up all the digits in a vehicle number plate, where the blackbox modules would be digit recognition functions, whereas the learnt symbolic structure will iterate over the digits of the recognized number.

To summarize, Inductive programming and machine learning techniques can greatly benefit from each other, and it is exciting to see a lot of progress happening currently in both fronts. We expect even more synergy between the two research areas in coming years.

5.3 IP for Data Wrangling

Frank Jäkel, Lidia Contreras-Ochando, Luc De Raedt, Martin Möhrmann

In data science the usual work flow starts with data collection and ends with an analysis script that distills the insights from the data. In between, there is a lot of data wrangling: The data need to be imported from sometimes esoteric file formats and need to be transformed into appropriate data structures. Data scientists spend most of their time on writing custom code for data wrangling, checking and rechecking the correctness of data transformations and the integrity of the data.

IP systems can help automate this process by learning data transformations from examples. However, while for small data sets a data scientist can check by hand that data wrangling code does what it is supposed to do and, e.g., also covers edge cases, exceptions or coding errors, for large data sets this is not possible. Hence, IP support tools for data scientists have to be interactive and help them discover potential problems by, e.g., grouping similar cases for bulk inspection or flagging suspicious or uncertain cases in an active learning mode. Otherwise it is unlikely that IP systems will be trusted, especially by power users.

Luckily, small training sets of hand-checked examples can not only be used for learning data transformation but also to automatically generate tests by learning constraints. For example, if all input strings in a hand-checked subset are of length four, then this constraint can be learned (e.g. with Sketch) and input data that violate this constraint can be discovered without the user having to write an explicit test. Such interactive IP system that also uncover implicit assumptions about the data have the potential to provide substantial increases in productivity for data scientists.

Participants

- Aws Albarghouthi
University of Wisconsin – Madison, US
- Peter Buhler
IBM Research Zurich, CH
- Lidia Contreras-Ochando
Technical University of Valencia, ES
- Andrew Cropper
Imperial College London, GB
- Luc De Raedt
KU Leuven, BE
- Richard Evans
Google DeepMind – London, GB
- Cesar Ferri Ramirez
Technical University of Valencia, ES
- Elena Glassman
University of California – Berkeley, US
- Katsumi Inoue
National Institute of Informatics – Tokyo, JP
- Frank Jäkel
PSIORI – Freiburg, DE
- Susumu Katayama
University of Miyazaki, JP
- Martin Möhrmann
Universität Osnabrück, DE
- Stephen H. Muggleton
Imperial College London, GB
- David Nieves Cordones
Technical University of Valencia, ES
- Hila Peleg
Technion – Haifa, IL
- Oleksandr Polozov
Microsoft Corporation – Redmond, US
- Fabrizio Riguzzi
University of Ferrara, IT
- Ute Schmid
Universität Bamberg, DE
- Sebastian Seufert
Universität Bamberg, DE
- Michael Siebers
Universität Bamberg, DE
- Rishabh Singh
Microsoft Research – Redmond, US
- Armando Solar-Lezama
MIT – Cambridge, US
- Claes Strannegård
Chalmers University of Technology – Göteborg, SE
- Janis Voigtländer
Universität Duisburg-Essen, DE
- Christina Zeller
Universität Bamberg, DE