

Approaches and Applications of Inductive Programming

Edited by

José Hernández-Orallo¹, Stephen H. Muggleton², Ute Schmid³, and Benjamin Zorn⁴

1 Technical University of Valencia, ES, jorallo@dsic.upv.es

2 Imperial College London, GB, s.muggleton@imperial.ac.uk

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

4 Microsoft Research – Redmond, US, zorn@microsoft.com

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 15442 “Approaches and Applications of Inductive Programming”. After a short introduction to the state of the art to inductive programming research, an overview of the talks and the outcomes of discussion groups is given.

Seminar October 25–30, 2015 – <http://www.dagstuhl.de/15442>

1998 ACM Subject Classification I.2.2 Automatic Programming, Program Synthesis

Keywords and phrases inductive program synthesis, end-user programming, probabilistic programming, constraint programming, universal artificial intelligence, cognitive modeling,

Digital Object Identifier 10.4230/DagRep.5.10.89

Edited in cooperation with Fernando Martínez-Plumed


1 Executive Summary

José Hernández-Orallo

Stephen H. Muggleton

Ute Schmid

Benjamin Zorn

License  Creative Commons BY 3.0 Unported license

© José Hernández-Orallo, Stephen H. Muggleton, Ute Schmid, and Benjamin Zorn

Inductive programming research addresses the problem of learning (mostly declarative) programs from incomplete specifications, such as input/output examples, observed traces, or constraints. Beginning in the 1960s, this area of research was initiated in artificial intelligence (AI) exploring the complex intellectual cognitive processes involved in producing program code which satisfies some specification. Furthermore, applications of AI for software engineering are investigated resulting in methodologies and techniques for automating parts of the program development process. Inductive programming can be seen as a very special subdomain of machine learning where the hypothesis space consists of classes of computer programs.

Nowadays, researchers working on inductive programming are distributed over different communities, especially inductive logic programming, evolutionary programming, grammar inference, functional programming, and programming languages and verification. Furthermore, similar approaches are of interest in programming by demonstration applications for end-user programming as well as in cognitive models of inductive learning.



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Approaches and Applications of Inductive Programming, *Dagstuhl Reports*, Vol. 5, Issue 10, pp. 89–111

Editors: José Hernández-Orallo, Stephen H. Muggleton, Ute Schmid, and Benjamin Zorn



Dagstuhl Reports

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

The recent release of FlashFill as a plug-in inductive programming tool for Microsoft Excel is an impressive demonstration that inductive programming research has matured in such a way that commercial applications become feasible. Similarly, the field has attracted widespread interest in computer science as a whole, as illustrated by the recent review article published by the Communications of the ACM [1].

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

We addressed many aspects which partially were identified as relevant topics during the previous Dagstuhl Seminar 13502 (<http://www.dagstuhl.de/13502>). In particular, we had the following sessions for presentations:

- Session: General techniques, languages and systems for inductive logic and inductive functional programming.
- Session: End-user programming, programming by example and applications
- Session: Program synthesis and transformation
- Cognitive Aspects of Induction

In addition, we had several systems demos and tutorials (some of them ‘hands-on’):

- System demo on Metagol.
- Hands-on-Tutorial: The MagicHaskeller Library and Server System
- Tutorial: FlashMeta SDK for creating programming-by-example tools
- Tutorial: Sketch synthesis infrastructure

The seminar also included a DemoFest, where several systems were demonstrated in small groups in a relaxed atmosphere.

The first and second days the following topics were identified and further discussed in working groups during the rest of the seminar:

- Benchmarks, Evaluation, and Applications
- General-Purpose IP Infrastructures and Applicability and Evaluation Criteria for IP Approaches in the Context of AI
- Probabilities in IP

Concluding remarks and future plans

In the final panel discussion the results of the seminar were summarised as well as future plans.

Regarding the seminar, there were several suggestions that topics should be more mixed, instead of grouping them too much into “silo” sessions. About the format of the seminar, there was a general agreement that the change from half a week to one week had been beneficial, and that the DemoFest had been a real success. Indeed, the possibility of having several independent demos earlier in the week and more demo sessions (or DemoFests) was a possible suggestion for subsequent meetings.

The following topics were elaborated about future actions:

- Make the community and the area more visible through tutorials and workshops at major conferences, or summer schools.

- Integrate tools, demos, videos, tutorials and other kinds of material at www.inductive-programming.org/resources.html
- Focus on benchmarks and a common representation language for problems, and use the inductive-programming website to publish the benchmarks. Organise competitions (or hackathons building apps that use the underlying IP engines).
- Revitalise the mailing list (at the moment of writing this report the list is fully operative again, see <http://www.inductive-programming.org/>)
- Attract people from other areas (e.g., cognitive robotics).
- Change the frequency of the meetings to a 1-year cadence, with perhaps Dagstuhl every other year and a competition or summer school in between.

Overall, the main conclusion can be summarised as the realisation of a very significant progress in techniques and its exploitation in new applications, so it is now time to strengthen the visibility of the IP research and its community, for which this Dagstuhl seminar has served as a lever.

References

- 1 S. Gulwani, J. Hernández-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, and B. Zorn. *Inductive programming meets the real world*. Communications of the ACM, 58, 11, 2015.

2 Table of Contents

Executive Summary

José Hernández-Orallo, Stephen H. Muggleton, Ute Schmid, and Benjamin Zorn . . . 89

Overview of Talks on: Inductive Logic and Inductive Functional Programming

Meta-Interpretive Learning: achievements and challenges

Stephen H. Muggleton 94

Towards Probabilistic Logic Program Synthesis

Luc De Raedt 95

Learning Complex Sequential Patterns with Progol

Michael Siebers 95

RuleML as a Declarative Language for Inputs, Outputs, and Background Knowledge
in Inductive Programming

Harold Boley 96

Overview of Talks on: End-user Programming, Programming by Example and Applications

Applications to Data Wrangling

Sumit Gulwani 96

Generic vs. domain specific approaches to IP – How good does Igor perform on
FlashFill problems?

Ute Schmid 97

Resolving Ambiguity in Programming By Example

Rishabh Singh 98

Cooperative Programming: Integrating Software Synthesis with the Live Paradigm

Ruzica Piskac 98

Deductive Techniques for Synthesis from Inductive Specifications

Sumit Gulwani 99

Supervision by observation using inductive programming

José Hernández-Orallo 99

Software is not fragile

William B. Langdon 100

Overview of Talks on: Program synthesis and transformation

Synthesis of service specifications: Applying evolutionary algorithms to On-the-fly
computing

Lorijn van Rooijen 101

Inductive Program Synthesis for Bidirectional Transformations

Janis Voigtländer 102

D3: Data-Driven Disjunctive Abstraction

Hila Peleg 102

Program transformation using examples

Gustavo Soares 103

Overview of Talks on: Cognitive Aspects of Induction

Learning new Domains Without the Help of Engineers
Claes Strannegård 103

Knowledge acquisition with forgetting: an incremental and developmental view
Fernando Martínez-Plumed 104

A Visual Language for Solving Bongard Problems
Frank Jäkel 105

The Artificial Jack of All Trades: The Importance of Generality in Approaches to AI
Tarek R. Besold 105

System Demonstrations

Hands-on Tutorial for Hacking MagicHaskeller
Susumu Katayama 106

Working groups

Benchmarks, Evaluation, and Applications
Umair Zafrulla Ahmed, Harold Boley, José Hernández-Orallo, Petra Hofstedt, Frank Jäkel, William B. Langdon, Fernando Martínez-Plumed, Martin Möhrmann, Hila Peleg, Maria José Ramírez Quintana, Gustavo Soares, Armando Solar-Lezama, Lorijn van Rooijen, Janis Voigtländer, and Benjamin Zorn 106

General-Purpose IP Infrastructures and Applicability and Evaluation Criteria for IP Approaches in the Context of AI
Ute Schmid, Tarek R. Besold, Andrew Cropper, Sumit Gulwani, Petra Hofstedt, Susumu Katayama, William B. Langdon, and Stephen H. Muggleton 108

Probabilities in Inductive Programming
Rishabh Singh, Luc De Raedt, Cesar Ferri Ramirez, Frank Jäkel, Maria José Ramírez Quintana, Michael Siebers, Armando Solar-Lezama, and Christina Zeller 110

Participants 111

3 Overview of Talks on: Inductive Logic and Inductive Functional Programming

3.1 Meta-Interpretive Learning: achievements and challenges

Stephen H. Muggleton (*Imperial College London, GB*)

License © Creative Commons BY 3.0 Unported license

© Stephen H. Muggleton

Joint work of Muggleton, Stephen; Cropper, Andrew; Lin, Dianhuan; Tamaddoni-Nezhad, Alireza; Tenenbaum, Joshua; Dechter, Eyal; Ellis, Kevin; Dai, Wang-Zhou

Main reference S. H. Muggleton, D. Lin, A. Tamaddoni-Nezhad, “Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited,” *Machine Learning*, 100(1):49–73, 2015; pre-print available from author’s webpage.

URL <http://dx.doi.org/10.1007/s10994-014-5471-y>

URL http://www.doc.ic.ac.uk/~shm/Papers/metagolD_MLJ.pdf

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 reusable 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 computable 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 representations 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.

References

- 1 A. Cropper and S. H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *Proceedings of the 24th International Joint Conference Artificial Intelligence (IJCAI 2015)*, pp. 3423–3429, 2015.
- 2 W.-Z. Dai, S. H. Muggleton, and Z.-H. Zhou. Logical vision: Meta-interpretive learning for simple geometrical concepts. In *Short Paper Proceedings of the 25th International Conference on Inductive Logic Programming*. National Institute of Informatics, Tokyo, 2015.
- 3 S. H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.
- 4 D. Lin, E. Dechter, K. Ellis, J. B. Tenenbaum, and S. H. Muggleton. Bias reformulation for one-shot function induction. In *ECAI*, 2014.

3.2 Towards Probabilistic Logic Program Synthesis

Luc De Raedt (KU Leuven, BE)

License © Creative Commons BY 3.0 Unported license
© Luc De Raedt

Probabilistic logic programs combine the power of a programming language with a possible world semantics; they are typically based on Sato's distribution semantics [5] and they have been studied for over twenty years now. They have recently been extended towards defining continuous distributions and dynamics, which enables their use in robotics and perception [1]. The talk shall briefly introduce these formalisms and then present some progress on synthesising such probabilistic programs from examples, both in the discrete and the continuous case. For the discrete case, I shall report on our results in applying ProbFOIL [5] to the problem of machine reading in CMU's Never Ending Language Learning system. ProbFOIL is an extension of the traditional rule-learning system FOIL for use with the distribution semantics. For the continuous case, I shall present our ongoing work in learning affordances in robotics, where the goal is to learn the conditions under which actions can be applied on particular objects [2, 3].

References

- 1 B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 2011(11), pp. 663–680.
- 2 D. Nitti, T. De Laet, L. De Raedt, A particle filter for hybrid relational domains. in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2013*, pp. 2764–2771.
- 3 B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, L. De Raedt, Learning relational affordance models for robots in multi-object manipulation tasks. In *Proc. IEEE International Conference on Robotics and Automation, ICRA 2012*, pp. 4373–4378.
- 4 L. De Raedt, A. Dries, I. Thon, G. Van den Broeck, M. Verbeke, Inducing Probabilistic Relational Rules from Probabilistic Examples. In *Proc. International Joint Conference on AI, IJCAI 2015*, in press.
- 5 T. Sato, A Statistical Learning Method for Logic Programs with Distribution Semantics. In *Proc. 12th International Conference on Logic Programming, ICLP 1995*, pp. 715–729

3.3 Learning Complex Sequential Patterns with Progol

Michael Siebers (Universität Bamberg, DE)


License © Creative Commons BY 3.0 Unported license
© Michael Siebers

Joint work of Schmid, Ute; Siebers, Michael

The analysis of facial expressions is a new application domain for ILP. Facial expressions may be described by symbols, called action units, which represent movements in the face. Then, a facial expression is a sequence of action units possibly occurring in parallel. I will present first results on learning a generalized representation of facial expressions of pain using Progol. Additionally, I will provide ideas for additional background knowledge to improve results.

3.4 RuleML as a Declarative Language for Inputs, Outputs, and Background Knowledge in Inductive Programming

Harold Boley (University of New Brunswick at Fredericton, CA)

License  Creative Commons BY 3.0 Unported license
© Harold Boley

RuleML is a system of families of languages for Web rules connecting related efforts such as: SWRL, SWSL, and RIF (W3C); SBVR, PRR, API4KP, and OntoIOp (OMG); Common Logic (ISO); and LegalRuleML (OASIS).

Inductive Programming can use RuleML to represent inputs, outputs, and background knowledge, since:

- it combines relational/logical and equational/functional representations, e.g. Prolog-like relations and functions defined as oriented (conditional) equations;
- its canonical XML format allows modular (Relax NG-schema) validation of (challenge/benchmark/...)library entries, each w.r.t. the most precise language;
- RuleML/XML also allows (XSLT) transformation, e.g. to favorite presentation syntaxes of library users and to other XML-based standards;
- its engines permit the execution/querying of induced programs/rulebases;
- its canonical syntax, validation methods, interoperation techniques, as well as, e.g., operational and model-theoretic semantics can support the evaluation of Inductive Programming systems.

4 Overview of Talks on: End-user Programming, Programming by Example and Applications

4.1 Applications to Data Wrangling

Sumit Gulwani (Microsoft Corporation – Redmond, US)

License  Creative Commons BY 3.0 Unported license
© Sumit Gulwani

Main reference S. Gulwani, “Programming by Examples (and its applications in Data Wrangling),” In Verification and Synthesis of Correct and Secure Systems 2016, Marktobendorf Lecture Notes, IOS Press, to appear; pre-print available from author’s webpage.

URL <http://research.microsoft.com/en-us/um/people/sumitg/pubs/pbe16.pdf>

99% of computer end users do not know programming and struggle with repetitive tasks. Inductive synthesis can revolutionize this landscape by enabling end users to automate repetitive tasks using examples. In order to realize this potential, we need to apply inductive synthesis to the right set of application domains. Data wrangling turns out to be a killer application area for inductive synthesis.

Data is the new oil. Evolution of digital revolution, social media, cloud computing, IoT has led to production of massive amounts of digital data. This data is the new currency of the digital world since it can help drive business decisions, advertising, recommendation systems, etc. Data wrangling refers to the tedious process of transforming data from its raw format to a more structured form that is amenable for drawing insights. It is estimated that data scientists spend 80% of their time wrangling with data. Inductive synthesis can enable easier and faster data wrangling. We have developed inductive synthesis tools for assisting with various data wrangling activities including string/number/date transformations (FlashFill), extraction of structured data from semi-structured log files or webpages (FlashExtract),

and formatting or table layout transformations (FlashRelate). FlashFill has been released as an Excel 2013 feature, while FlashExtract has been released as the ConvertFrom-string Powershell cmdlet and the custom field extraction capability in Azure OMS.

Practical deployment of inductive synthesis tools require addressing an important challenge associated with inductive synthesis systems, namely resolving ambiguity in the example based specification. We address this challenge using two key ideas: (i) machine learning based ranking techniques to predict an intended program from within the set of programs that are consistent with the examples provided by the user, (ii) user interaction models (including program navigation and active-learning based conversational clarification) that communicate actionable information to the user to help resolve ambiguity in the example based specification.

4.2 Generic vs. domain specific approaches to IP – How good does Igor perform on FlashFill problems?

Ute Schmid (Universität Bamberg, DE)

License © Creative Commons BY 3.0 Unported license
© Ute Schmid

Joint work of Kitzelmann, E.; Hofmann, M.; Hofmann, J., Schmid, U.

Main reference U. Schmid, E. Kitzelmann, “Inductive rule learning on the knowledge level,” *Cognitive Systems Research*, 12(3–4):237–248, 2011.

URL <http://dx.doi.org/10.1016/j.cogsys.2010.12.002>

Approaches to inductive programming (IP) learn (declarative) (recursive) programs from a small number of (positive) input/output examples. These approaches are generic, that is, if the given problem can be solved by operations produced by a program, the program can be induced (within some restricted time and space and if the program can be expressed by the underlying language restriction given by an implicit or explicit program scheme). On the other hand, applications of IP in the domain of enduser programming are typically based on a domain specific language which allows to extract suitable examples from observing interactions of users with the given software. If IP is generic, it should be able to be applied ‘from the shelf’ to different domains and consequently also to the domains investigated in enduser programming.

In my talk I demonstrated how the inductive functional programming system Igor performs on programming problems, planning problems, number series problems, XSLT problems, and string transformation problems. The last domain is where the Excell plug-in Flashfill performs in a very impressive way. Looking at the problem specifications given to Igor, we can see that for some domains these are quite simple and natural, for other domains (such as string transformations in Excell) they are quite clumsy. Furthermore, to present the examples in a form suitable for inductive generalization, the selection of examples as well as their representation is crucial. So I claim that extracting suitable representations from raw data such as observations of user interactions or images is the bottleneck of IP.

References

- 1 Flener, P., & Schmid, U. (2010). Inductive programming. In C. Sammut & G. Webb (Eds.), *Encyclopedia of machine learning*, pp. 537–544. Springer.
- 2 Hofmann, J., Kitzelmann, E., & 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.

- 3 Hofmann, M., Kitzelmann, E., & Schmid, U. (2009). A unifying framework for analysis and evaluation of inductive programming systems. *Proceedings of the Second Conference on Artificial General Intelligence (AGI-09, Arlington, Virginia, March 6-9 2009)*, pp. 55–60. Amsterdam: Atlantis Press.
- 4 Kitzelmann, E., & Schmid, U. (2006). Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 7, pp. 429–454.
- 5 Schmid, U., & Kitzelmann, E. (2011). Inductive rule learning on the knowledge level. *Cognitive Systems Research*, 12, pp. 237–248.

4.3 Resolving Ambiguity in Programming By Example

Rishabh Singh (Microsoft Research – Redmond, US)

License  Creative Commons BY 3.0 Unported license
© Rishabh Singh

Since Programming By Examples is inherently ambiguous, there are typically a large number of programs that conform to the given set of examples especially if the hypothesis space is large. In this talk, we present two way to tackle this ambiguity problem. The first approach uses machine learning techniques to learn a ranking function from training data to efficiently rank the set of learnt programs and select the top ranked program. We have implemented this technique in the FlashFill PBE system, which reduces the average number of examples needed to learn the desired program from 4.17 to 1.48 on 175 benchmarks. The second approach adds probabilistic semantics to the underlying DSL for learning robust transformations on semantic data type strings to handle ambiguity.

4.4 Cooperative Programming: Integrating Software Synthesis with the Live Paradigm

Ruzica Piskac (Yale University, US)

License  Creative Commons BY 3.0 Unported license
© Ruzica Piskac

Live programming is an emerging paradigm that is promising a vast change in the techniques used to develop modern software. A live programming environment allows a programmer to immediately see the effects of changes to a program on its outputs and effectively eliminates the edit-run-debug cycle that dominates programming workflows today.

We propose an approach that seeks to develop new synthesis techniques that make use of the real-time feedback loop provided by a live programming environment. We propose a system that will allow a user to track a set of examples and synthesize subroutines to fit that set. When combined with fault localization techniques, programmers will be able to quickly find incorrect sections of code and initiate repairs that leverage information gleaned from the provided examples.

4.5 Deductive Techniques for Synthesis from Inductive Specifications

Sumit Gulwani (Microsoft Corporation – Redmond, US)

License © Creative Commons BY 3.0 Unported license
© Sumit Gulwani

Main reference S. Gulwani, “Programming by Examples (and its applications in Data Wrangling),” In Verification and Synthesis of Correct and Secure Systems 2016, Marktobendorf Lecture Notes, IOS Press, to appear; pre-print available from author’s webpage.

URL <http://research.microsoft.com/en-us/um/people/sumitg/pubs/pbe16.pdf>

We propose two key ideas for designing efficient algorithms for inductive synthesis problems. (i) restrict the search space to an appropriate domain-specific language that provides the right set of abstractions that enable readability and balanced expressivity (i.e., expressive enough to capture a wide range of tasks, but restricted enough to allow efficient search). (ii) design a domain-specific search algorithm using the divide-and-conquer paradigm that reduces the problem of synthesizing an expression of a given kind that satisfies a given specification into sub-problems that refer to sub-expressions or sub-specifications.

The problem reduction logics can be refactored into domain-independent parts and operator-specific parts. We leverage this observation to develop a general framework that allows construction of efficient inductive synthesizers from a mere description of the domain-specific language and inverse properties of operators that are used in the underlying DSL.

4.6 Supervision by observation using inductive programming

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

License © Creative Commons BY 3.0 Unported license
© José Hernández-Orallo

Joint work of Monserrat, Carlos; Hernández-Orallo, José

Despite the fact that many professional or personal tasks we do every day are still very repetitive, the technologies to automate them are still incipient. A less idealistic, but still challenging short and mid-term goal is a scenario where the tasks are still performed by humans but supervised by machines. In other words, we aim at automatically *supervising* these tasks, an area of potentially high impact.

For instance, the training of new skills and the human errors during their execution entail a high cost that could be reduced by automated supervision systems indicating when, where and how a protocol error takes place, in a comprehensible way for the user or the professional. These systems should assimilate the procedures from expert knowledge and the observation of a few correct examples for the procedure without the need of being coded by a programmer.

I present the embryo project “SuPERVaSION”, which aims at exploring the pervasive use of inductive programming to address this problem domain. The potentiality of the project is planned to be demonstrated by their application to automatic skill supervision in minimally invasive surgery.

Acknowledgements. This work has been partially supported by the EU (FEDER) and the Spanish MINECO under grant TIN 2013-45732-C4-1-P and Generalitat Valenciana PROMETEOII/2015/013.

References

- 1 S. Gulwani, J. Hernández-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, and B. Zorn. *Inductive programming meets the real world*. Communications of the ACM, 58, 11, 2015.
- 2 J. Hernández-Orallo. *Deep knowledge: Inductive programming as an answer*. Technical Report, Seminar 13502, Dagstuhl, 2013.
- 3 C. Monserrat, A. Lucas, J. Hernández-Orallo, and M. José Rupérez. *Automatic supervision of gestures to guide novice surgeons during training*. Surgical endoscopy, 28(4):1360–1370, 2014.
- 4 C. Monserrat, M. J. Rupérez, M. Alcañiz, J. Mataix. *Markerless monocular tracking system for guided external surgery*. Computerized Medical Imaging and Graphics, Elsevier, pp. 1–8, 2014.

4.7 Software is not fragile

William B. Langdon (University College London, GB)

License © Creative Commons BY 3.0 Unported license
© William B. Langdon

Joint work of Mark Harman, Justyna Petke, Estibaliz Aldecoa-Otalor, Phil Cunningham, Matthew J. Arno, Westley Weimer

Main reference W. B. Langdon, J. Petke, “Software is Not Fragile,” in Proc. of the 2015 World e-Conference Complex Systems Digital Campus (CS-CD’15), to appear; pre-print available from the author’s webpage.

URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon_2015_csdcd.pdf

URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/grammar_test_ilp.tar.gz

The talk, as requested, concentrated upon insights gained, in my case from working on genetic programming. These are summarised on slide 3 of my talk: work on industrial strength languages, focus search, evolve patches (change to C program source), evolve source code v. machine code, ensure many patches/mutants compile, software resilient to mutation, choose receptive domain, separate fitness from validation, evolution exploits fitness, present results on a slide, e.g. source code, . . .

I then concentrated upon recent work which counters the “folk wisdom” with results from experiments from a number of real programs which show software is robust to many (albeit not all) random changes. I also included recent results that show more than a 70x speed up on a real program when customised to the task in hand with the use of genetic improvement using mutations of the C++ source code.

During the seminar I was also asked about the in silico spread of bacterial genes in the Human reference genome, genetic programming, human competitive GI and learning context free language benchmarks (benchmarks are available as tar file from http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/grammar_test_ilp.tar.gz)

Trying all simple changes (first order mutations) to executed C, C++ and CUDA source code shows software engineering artefacts are more robust than is often assumed. Of those that compile, up to 89percent run without error. Indeed a few mutants are improvements. Program fitness landscapes are smoother. Analysis of these programs, a parallel nVidia GPGPU kernel, all CUDA samples and the GNU C library shows many lines of code and integer values are repeated and may follow Zipf’s law.

We show genetic improvement of programs (GIP) can scale by evolving increased performance in a widely-used and highly complex 50 000 line system. GISMOE found code that is 70 times faster (on average) and yet is at least as good functionally. Indeed it even gives a small semantic gain.

References

- 1 William B. Langdon and Justyna Petke. Software is not fragile. In Paul Bourguine and Pierre Collet, editors, *Complex Systems Digital Campus E-conference, CS-DC'15*, Proceedings in Complexity, page Paper ID: 356. Springer, September 30 – October 1 2015. Invited talk, Forthcoming.
- 2 William B. Langdon and Mark Harman. Optimising existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(1):118–135, February 2015.
- 3 William B. Langdon. Mycoplasma contamination in the 1000 genomes project. *BioData Mining*, 7(3), 29 April 2014. Highly accessed.
- 4 Estibaliz Aldecoa-Otalora, William B. Langdon, Phil Cunningham, and Matthew J. Arno. Unexpected presence of mycoplasma probes on human microarrays. *BioTechniques*, 47(6):1013–1016, December 2009.
- 5 Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. Using genetic improvement and code transplants to specialise a C++ program to a problem class. In Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo Garcia-Sanchez, Juan J. Merelo, Victor M. Rivas Santos, and Kevin Sim, editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pp. 137–149, Granada, Spain, 23–25 April 2014. Springer.
- 6 Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008. (With contributions by J. R. Koza).
- 7 William B. Langdon. Genetically improved software. In Amir H. Gandomi, Amir H. Alavi, and Conor Ryan, editors, *Handbook of Genetic Programming Applications*, chapter 8. Springer. Forthcoming.

5 Overview of Talks on: Program synthesis and transformation

5.1 Synthesis of service specifications: Applying evolutionary algorithms to On-the-fly computing

Lorijn van Rooijen (Universität Paderborn, DE)

License © Creative Commons BY 3.0 Unported license
© Lorijn van Rooijen

The aim of our research project on On-the-fly (OTF) Computing is to develop concepts and techniques for automatic on-the-fly composition of individualized IT services out of base services that are available on world-wide markets. One part of this project deals with the problem of obtaining, in a user-friendly way, requirements specifications for the services that users would want to acquire. To facilitate the user, our objective is to automatically synthesize such requirements specifications from sample specifications that are provided by the user. These sample specifications may be incomplete and imprecise and might even contain contradictions. We plan to employ evolutionary algorithms to synthesize comprehensive requirements specifications that do justice to the sample specifications by extending them in a natural way.

5.2 Inductive Program Synthesis for Bidirectional Transformations

Janis Voigtländer (Universität Bonn, DE)

License © Creative Commons BY 3.0 Unported license
© Janis Voigtländer

Joint work of Tobias Gödderz, Helmut Grohne, Janis Voigtländer

Main reference J. Voigtländer, “Ideas for connecting inductive program synthesis and bidirectionalization,” in Proc. of ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation (PEPM’12), pp. 39–42, ACM, 2012.

URL <http://dx.doi.org/10.1145/2103746.2103757>

Bidirectional transformations are a mechanism for preserving the consistency of (at least) two related data structures. A classical incarnation is the view-update problem, which has received considerable attention from programming language research. Specifically, bidirectionalization techniques attempt to automatically produce a well-behaving backwards transformation from a given forwards transformation. Well-behavedness is expressed by certain roundtripping laws. We explore the use of inductive program synthesis for bidirectionalization. The main idea is to use the roundtripping laws, and possibly additional constraints derived from supposed programmer intentions, as a source of input/output examples for the backwards transformation. We report on recent experiments using the Igor-II system.

5.3 D3: Data-Driven Disjunctive Abstraction

Hila Peleg (Technion – Haifa, IL)

License © Creative Commons BY 3.0 Unported license
© Hila Peleg

Joint work of Hila Peleg, Sharon Shoham, Eran Yahav]

We address the problem of computing an abstraction for a set of examples, which is precise enough to separate them from a set of counterexamples. The challenge is to find an over-approximation of the positive examples that does not represent any negative example. Conjunctive abstractions (e.g., convex numerical domains) and limited disjunctive abstractions, are often insufficient, as even the best such abstraction might include negative examples. One way to improve precision is to consider a general disjunctive abstraction. We present D3, a new algorithm for learning general disjunctive abstractions. Our algorithm is inspired by widely used machine-learning algorithms for obtaining a classifier from positive and negative examples. In contrast to these algorithms which cannot generalize from disjunctions, D3 obtains a disjunctive abstraction that minimizes the number of disjunctions. The result generalizes the positive examples as much as possible without representing any of the negative examples. We demonstrate the value of our algorithm by applying it to the problem of data-driven differential analysis, computing the abstract semantic difference between two programs. Our evaluation shows that D3 can be used to effectively learn precise differences between programs even when the difference requires a disjunctive representation.

5.4 Program transformation using examples

Gustavo Soares (Universidade Federal – Campina Grande, BR)

License © Creative Commons BY 3.0 Unported license
© Gustavo Soares

Joint work of Reudismam Rolim, Rohit Gheyi, Sumit Gulwani

During software evolution, developers may have to apply similar but not identical changes to different locations in the program. Manually performing these changes is time-consuming and error-prone. We then propose an inductive programming technique for automating repetitive changes using examples. First, the developer provides examples of locations to be changed. The technique learns a program that finds similar code fragments throughout the developers' code. The developer then applies the transformation to one or more locations. Based on the examples of the desired transformation, the technique learns a program that applies the transformation and transforms all locations identified in step 1. We identified 68 scenarios where developers performed systematic changes and our approach was able to automate these tasks.

6 Overview of Talks on: Cognitive Aspects of Induction

6.1 Learning new Domains Without the Help of Engineers

Claes Strannegård (Chalmers University of Technology – Göteborg, SE)

License © Creative Commons BY 3.0 Unported license
© Claes Strannegård

It has been postulated that the goal of artificial general intelligence is to create general intelligence at the human level and beyond [4]. To get anywhere near that goal one needs to construct versatile agents that can adapt to a wide range of environments without any human intervention. In natural nervous systems, reinforcement learning is a powerful mechanism that enables organisms to adapt to different environments and survive there [3]. In artificial systems, reinforcement learning has been used as the basis of relatively versatile agents, e.g. for robotic locomotion across different anatomies and for gaming across different arcade games [1].

In this talk I will discuss how more versatile systems might be constructed, e.g. systems that can both learn to move like a snake and do simple mathematics. I will sketch a system whose main parts are (i) a long-term memory in the form of a Markov Decision Process based on transparent networks that develops dynamically [2], (ii) a learner consisting of rules for evolving the long-term memory by adding and removing memory structures, and (iii) a decision-maker for planning and acting.

References

- 1 Volodymyr Mnih and others. *Human-level control through deep reinforcement learning*. *Nature*, 518(7540):529–533, 2015
- 2 Claes Strannegård, Simone Cirillo, and Johan Wessberg. *Emotional Concept Formation*. *Proc. of the 8th Conf. on Artificial General Intelligence*. pp. 166–176, Springer, 2015
- 3 Yael Niv. *Reinforcement learning in the brain*. *Journal of Mathematical Psychology*. 53(3):139–154, 2009
- 4 Ben Goertzel and Cassio Pennachin. *Artificial General Intelligence*. Vol. 2, Springer, 2007.

6.2 Knowledge acquisition with forgetting: an incremental and developmental view

Fernando Martínez-Plumed (Technical University of Valencia, ES)

License © Creative Commons BY 3.0 Unported license

© Fernando Martínez-Plumed

Joint work of Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, María José Ramírez-Quintana:

Main reference F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, M. J. Ramírez-Quintana, “Knowledge acquisition with forgetting: an incremental and developmental setting,” *Adaptive Behaviour*, 23(5):283–299, 2015.

URL <http://dx.doi.org/10.1177/1059712315608675>

The development of a knowledge discovery system that is meant to be incremental and cumulative is not an easy task: the use of background knowledge and the consolidation of new knowledge is one of the conspicuous problems in the understanding and creation of cognitive systems, and the management of more lifelong knowledge discovery systems. Knowledge acquisition, as the process of abstracting knowledge from facts and other knowledge, cannot be understood as a naive accumulation of what is being learnt. Identifying the balance between remembering and forgetting is the key to abstraction in the human brain (creation of memories and knowledge) [1] and, therefore, this could be also applied when developing knowledge acquisition AI systems.

In this talk I present the work in [4], an incremental, lifelong view of knowledge acquisition which tries to improve task after task by determining what to keep, what to consolidate and what to forget, overcoming *The Stability-Plasticity* dilemma [2]. This framework is designed to combine any rule-based inductive engine (which learns new rules) with a deductive engine (which derives a coverage graph for all rules) and integrates them into a lifelong learner through the use of a hierarchical knowledge assessment structure (*coverage graphs*) and by introducing several MML-based [3] metrics. The metrics are not only used to forget some of the worst rules, but also to set a consolidation process to promote those selected rules to the knowledge base, which is also mirrored by a demotion system.

Acknowledgements. This work has been partially supported by the EU (FEDER) and the Spanish MINECO under grant TIN 2013-45732-C4-1-P and Generalitat Valenciana PROMETEOII/2015/013.

References

- 1 Quiroga, R. Q. *Concept cells: the building blocks of declarative memory functions*. *Nature Reviews Neuroscience*, vol. 13, pp. 587–597, 2012.
- 2 Carpenter, G. A. and Grossberg, S. *The art of adaptive pattern recognition by a selforganizing neural network*. *Computer*, vol. 21, no. 3, pp. 77–88, 1988.
- 3 Wallace, C. S. and Boulton, D. M. *An information measure for classification*. *The Computer Journal*, vol. 11, no. 2, pp. 185–194, 1968.
- 4 F. Martínez-Plumed and C. Ferri and J. Hernández-Orallo and M. J. Ramírez-Quintana *Knowledge acquisition with forgetting: an incremental and developmental setting*. *Adaptive Behavior*, 23(5):283–299, 2015.

6.3 A Visual Language for Solving Bongard Problems

Frank Jäkel (*Universität Osnabrück, DE*)

License © Creative Commons BY 3.0 Unported license
© Frank Jäkel

Joint work of Depeweg, S.; Rothkopf, C.; Jäkel, F.

More than 50 years ago [1] introduced a set of 100 vision problems, now known as Bongard problems, as a test-bed for visual pattern recognition. Although these problems are well known in the cognitive science and AI communities only moderate progress has been made towards solving a substantial subset of them. The approach we present here extracts standard visual features as a basic visual vocabulary. We introduce a formal language that allows representing complex visual concepts and relations using this vocabulary. Finally, using Bayesian inference on the space of concepts formulated in this visual language, we compare the concepts with high posterior probability to the solutions formulated by Bongard himself when designing his problems. We find good agreement for a sizeable fraction of the problems.

References

- 1 M. Bongard. *Pattern Recognition*. Spartan Books, New York, 1970.

6.4 The Artificial Jack of All Trades: The Importance of Generality in Approaches to AI

Tarek R. Besold (*Free University of Bozen-Bolzano, IT*)

License © Creative Commons BY 3.0 Unported license
© Tarek R. Besold

Joint work of Besold, Tarek R.; Schmid, Ute

Main reference T. R. Besold, U. Schmid, “The Artificial Jack of All Trades: The Importance of Generality in Approaches to Human-Level Artificial Intelligence”, in Proc. of the 3rd Annual Conf. on Advances in Cognitive Systems (ACS’15), Article No. 18, 18 pages, Cognitive Systems Foundation, 2015.


URL <http://www.cogsys.org/papers/ACS2015/article18.pdf>

We advocate the position that research efforts working towards solving human-level AI necessarily have to rely on general mechanisms and (models of) cognitive capacities, with domain-specific systems or task-dependent approaches only being of minor help towards the final goal. We revisit psychological research on intelligence and the application of psychometric methods in AI, before discussing IGOR2 (implementing program learning) and HDTP (implementing generalisation-based analogy-making) under the light of the previous considerations as examples of systems respectively implementing a general mechanism or modeling a general capacity. In closing, we summarise our considerations, point out three characteristics we consider suitable candidates for serving as generally recommendable properties of HLAI systems, and motivate why AI as a field could greatly profit from closer interaction with Inductive Programming (and vice versa).

7 System Demonstrations

7.1 Hands-on Tutorial for Hacking MagicHaskeller

Susumu Katayama (University of Miyazaki, JP)


License  Creative Commons BY 3.0 Unported license
 © Susumu Katayama

This presentation was a hands-on tutorial of MagicHaskeller, that is a powerful inductive functional programming system for Haskell focusing on practical use. Although the presentation material was prepared for the users who are interested in using its API, the actual presentation focused on its web interface and was mainly for those who were not familiar with it.

8 Working groups

8.1 Benchmarks, Evaluation, and Applications

Umair Zafrulla Ahmed (Indian Institute of Technology – Kanpur, IN), Harold Boley (University of New Brunswick at Fredericton, CA), José Hernández-Orallo (Technical University of Valencia, ES), Petra Hofstedt (TU Cottbus, DE), Frank Jäkel (Universität Osnabrück, DE), William B. Langdon (University College London, GB), Fernando Martinez-Plumed (Technical University of Valencia, ES), Martin Möhrmann (Universität Osnabrück, DE), Hila Peleg (Technion – Haifa, IL), Maria José Ramírez Quintana (Technical University of Valencia, ES), Gustavo Soares (Universidade Federal – Campina Grande, BR), Armando Solar-Lezama (MIT – Cambridge, US), Lorijn van Rooijen (Universität Paderborn, DE), Janis Voigtländer (Universität Bonn, DE), and Benjamin Zorn (Microsoft Research – Redmond, US)

License  Creative Commons BY 3.0 Unported license
 © Umair Zafrulla Ahmed, Harold Boley, José Hernández-Orallo, Petra Hofstedt, Frank Jäkel, William B. Langdon, Fernando Martinez-Plumed, Martin Möhrmann, Hila Peleg, Maria José Ramírez Quintana, Gustavo Soares, Armando Solar-Lezama, Lorijn van Rooijen, Janis Voigtländer, and Benjamin Zorn

The goal of this working group is to increase the visibility and impact of research in the areas of inductive programming as applied to important practical problems. We want to facilitate a broad effort to coordinate and increase communication among researchers, identify important research directions, and show the utility of IP in practice across important application domains. To achieve this goal, we identify two important sub-goals: creating a repository of problem description and links to open-source systems (which has been already initiated at <http://inductive-programming.org/>) and to host competitions focused on specific tasks and domains to highlight the significant progress being made on practical applications of IP.

Application areas: In our discussions we identified that there are many emerging areas where practical applications of IP are both significant and commercially valuable. These include general tests of intelligence, data manipulation, education, programming (code transformations, test generation, etc.), grammar discovery, and visual reasoning. Our hope is that we can add problem sets in all of these areas in the coming year that will significantly broaden the set of problems that are already present in our repository.

Common representations: One of the challenges of defining a set of shared problem instances is the challenge of defining the problems in such a way that they can be used across systems. Ideally, we would have a way to encode the problem (capturing the input/output pairs, etc.). We also have to encode characteristics of the resulting artifact (e.g., it might be a program in a domain specific language). Finally, encoded in the solution and also the search used in finding the solution is the embedded background knowledge that allows the solution to closely match that a person might construct in solving the problem. It is especially difficult to tease out this background knowledge in some systems because it can be encoded in a domain-specific language or even as a set of constraints that exist in the solver system that is used.

Competitions: One of the key areas we wanted to increase activity in during the coming years is competitions that highlight important application areas for IP and allow the systems that are emerging to demonstrate how effective they are. Armando shared some of his experiences running the SyGuS synthesis competition for 2 consecutive years. He pointed out a number of lessons that include: avoid narrowing the community, allow multiple winners and different categories, have in mind for the first competition a list of participants that will definitely enter, think carefully about the problems ahead of time.

We considered specific opportunities for connecting a competition to a particular meeting and KDD2016 was discussed but the deadline is too soon. We decided that focusing the first competition on data wrangling would limit the scope to an important area and allow a number of different approaches to be used (string manipulation is common to a number of the existing systems). The plan would be to change the topic of the competition each year to diversify the applications explored. ICDM 2016 in Barcelona is a possible alternative venue we are considering. <http://www.cs.uvm.edu/~icdm/>

Evaluation: Another important dimension of benchmarks and evaluation is what metrics are applied to determine if the solution is “good”. In many cases, the “naturalness” of a solution to a human is an important requirement. We explored increasing the use of crowdsourcing as a way to evaluate aspects of a solution that are more difficult to measure using objective techniques. One starting point for this is to create a catalog of the existing metrics that different researchers are using and understand if those metrics are domain specific or general.

Community: One of the important challenges in building collections of benchmarks, maintaining repositories and keeping them fresh, and organizing competitions is that none of these activities directly leads to new results or publications. As a result, a community is needed that gets benefit from shared infrastructure and sees synergy in the related efforts. One possible goal in this area might be to publish a paper that captures a set of benchmarks in a way that makes it much easier for subsequent research to build on that result. For example, the Dicap benchmarks for Java are highly used and cited in the Java research community. One of the things we plan to do going forward is maintaining regular interactions among the attendees of the seminar interested in this topic via regular Skype meetings. We also discussed other forms of social media and how to share and communicate important results in this research area.

8.2 General-Purpose IP Infrastructures and Applicability and Evaluation Criteria for IP Approaches in the Context of AI

Ute Schmid (Universität Bamberg, DE), Tarek R. Besold (Free University of Bozen-Bolzano, IT), Andrew Cropper (Imperial College London, GB), Sumit Gulwani (Microsoft Corporation – Redmond, US), Petra Hofstedt (TU Cottbus, DE), Susumu Katayama (University of Miyazaki, JP), William B. Langdon (University College London, GB), and Stephen H. Muggleton (Imperial College London, GB)

License © Creative Commons BY 3.0 Unported license
 © Ute Schmid, Tarek R. Besold, Andrew Cropper, Sumit Gulwani, Petra Hofstedt, Susumu Katayama, William B. Langdon, and Stephen H. Muggleton

The topic of inductive programming (IP) was discussed in the context of the general goals and quality criteria of “classical” artificial intelligence (AI) research (i.e., research aiming at developing human-level AI systems). The initial proposition was that general purpose approaches to IP – such as inductive logic programming (ILP) and inductive functional programming (IFP) – are more compatible with the goals of AI research than specialized, domain-specific approaches. However, the general scientific goal of AI research to formalize human-level intelligence by means of general algorithms is in practice complemented by the application oriented engineering goal to provide efficient and reliable algorithmic solutions for problems of interest for industry and end-users. Nevertheless, taking into account insights about human cognition is also of relevance for application-oriented AI in order to provide for adequate and comprehensible human-computer interaction and human-understandable and -interpretable processing approaches.

Generality of mechanisms (for reasoning, induction, problem solving, etc.) is hypothesized as hallmark of human cognition. That is, it is assumed that the same processing principles guide, for example, solving geometric induction problems such as Raven progressive matrices, induction of number series, induction of replacement patterns for string transformations, or induction of a general routine to solve Tower of Hanoi or sorting of lists. While ILP and IFP are such general mechanism, the domain specific approach of the Excel plug-in Flashfill for string transformations demonstrates the enormous power of specialized approaches within their restricted domains. An engineering approach to reach both the performance of specialized systems as well as the coverage of a broader set of domains could be to provide a generator mechanism which allows to construct domain-specific solutions.

Within AI research machine learning is the area of research most closely related to IP: In IP as well as in standard ML, the main goal is to provide algorithms for inductive generalization over incomplete information. In standard ML, training examples are typically presented in the form of feature vectors, learning algorithms are robust to noise, and in order to ensure suitable generalization and avoid overfitting it is not desirable that the learned hypothesis covers all training examples. In contrast, in IP, training examples are presented in the form of input/output relations and in addition background knowledge can be provided. Furthermore, since the learned hypothesis is a (declarative) program, it is necessary that all input/output examples are treated correctly. While most of standard ML approaches are blackbox learners, IP approaches are whitebox learners. That is, the generated hypothesis is a symbolic and structured representation which is comprehensible by humans. The big difference between a standard ML approach such as deep learning neural networks and IP is, that IP addresses deep comprehension! In consequence, we propose that comprehensibility might be a suitable performance criterium for the output of ML/IP approaches.

A promising line of research in IP might be to make systems which generate output which

is (even more) compatible with human-to-human communication. That is, the output of an IP system should have high comprehensibility. Comprehensibility implies natural interaction, safety and trust. As a first step in this direction, we need to characterize what defines the comprehensibility of system output (such as names, concepts, action sequences, programs, texts, theories, designs, pictures, emotional responses and so on). Consequently, a new IP system could be designed to use comprehensibility as inductive bias. To determine the comprehensibility of a learned program, the characteristics of comprehensibility could be learned in form of a mapping from characteristics of the candidate hypotheses to the degree or probability of understanding. Understanding could be empirically measured as probability that the subject can provide the correct classification (or action) for arbitrary input based on the provided program. A first step could be attempting to machine learn definitions of comprehensibility from labelled examples provided by humans. This could be done in the form of setting comprehension tests, similar in spirit to those set for texts in schools, in which the human subjects were asked to classify the implied consequences of various machine generated hypotheses. The expected accuracy of such tests would then be taken as a proxy measure for the degree of comprehensibility of individual hypotheses with respect to the given group of humans tested.

One characteristic of making a program more comprehensible than another one could be the use of predicate (or function) invention as used by ILP systems such as Golem, Progol or Metagol and by the IFP system Igor. Invented predicates/functions introduce a new level of abstraction and allow to structure a program by using higher-level concepts in the program body which are defined separately. A simple example is a Prolog definition of the concept grandfather/2 with and without predicate invention:

```

; without predicate invention
p(X,Y) ← father(X,Z), father(Z,Y)
p(X,Y) ← father(X,Z), mother(Z,Y)
p(X,Y) ← mother(X,Z), mother(Z,Y)
p(X,Y) ← mother(X,Z), father(Z,Y)

;with predicate invention
p(X,Y) ← p1(X,Z), p1(Z,Y)
p1(X,Y)← father(X,Y)
p1(X,Y)← mother(X,Y)

```

As outcome of the discussion group, it was planned to design and conduct an empirical study about comprehensibility of Prolog programs with versus without invented predicates.

8.3 Probabilities in Inductive Programming

Rishabh Singh (Microsoft Research – Redmond, US), Luc De Raedt (KU Leuven, BE), Cesar Ferri Ramirez (Technical University of Valencia, ES), Frank Jäkel (Universität Osnabrück, DE), Maria José Ramirez Quintana (Technical University of Valencia, ES), Michael Siebers (Universität Bamberg, DE), Armando Solar-Lezama (MIT – Cambridge, US), and Christina Zeller (Universität Bamberg, DE)

License © Creative Commons BY 3.0 Unported license
© Rishabh Singh, Luc De Raedt, Cesar Ferri Ramirez, Frank Jäkel, Maria José Ramirez Quintana, Michael Siebers, Armando Solar-Lezama, and Christina Zeller

Our discussion group focused on the role of probabilistic analysis in Inductive programming. We identified three main ways in which probabilities can be used: 1) Modelling priors of the programs in the hypothesis space to specify a program P1 is more likely than another program P2, 2) Using probabilistic programs for inductive reasoning so that a program can generate a distribution of outputs for modeling noise, and finally 3) a unified setting where both noise and prior can be modeled together using a joint distribution. We then identified some applications in inductive programming that can be enabled using such probabilistic analysis. One of the key new applications was synthesizing programs in settings where there is some inherent ambiguity in inputs and outputs, e.g. human-generated data, weather/GPS and sensor data. The probabilistic analysis can also increase the robustness of previous deterministic inductive programming approaches by using better priors for the hypothesis space and handling noisy data, e.g. a robust FlashFill learning algorithm. Finally, another interesting application was in formalizing the heuristics used in synthesis approaches.

We then contrasted probabilistic analysis in Inductive Programming with Machine Learning and Probabilistic Programming. As compared to Machine Learning, the probabilistic IP approaches can handle both hard and soft constraints, can learn human-understandable programs instead of complex high-dimensional models, and provide more control over the hypothesis space. Probabilistic Programming, on the other hand, can be used to formulate our problem, but they are not yet scalable enough to handle real-world IP problems. There is also a strong bias towards sampling based methods in Probabilistic Programming and lesser emphasis on combinatorial approaches that are more common in IP. Finally, we discussed some ways in which to perform a good evaluation of Probabilistic IP systems. The common measures of number of examples needed to learn a program and the scalability of the learning algorithm are similar to the traditional IP systems, but are harder to evaluate in our setting. Another interesting measure to evaluate in probabilistic IP systems is to verify how faithfully the learning algorithm is conforming to the probability distributions. Finally, we discussed some potential ways to evaluate the prior models for the hypothesis space and input-output distributions.

Participants

- Umair Zafrulla Ahmed
Indian Institute of Technology –
Kanpur, IN
- Tarek R. Besold
Free Univ. of Bozen-Bolzano, IT
- Harold Boley
University of New Brunswick at
Fredericton, CA
- Andrew Cropper
Imperial College London, GB
- Luc De Raedt
KU Leuven, BE
- Cesar Ferri Ramirez
Technical Univ. of Valencia, ES
- Sumit Gulwani
Microsoft Corporation –
Redmond, US
- José Hernández-Orallo
Technical Univ. of Valencia, ES
- Petra Hofstedt
TU Cottbus, DE
- Frank Jäkel
Universität Osnabrück, DE
- Susumu Katayama
University of Miyazaki, JP
- William B. Langdon
University College London, GB
- Fernando Martinez-Plumed
Technical Univ. of Valencia, ES
- Martin Möhrmann
Universität Osnabrück, DE
- Stephen H. Muggleton
Imperial College London, GB
- Hila Peleg
Technion – Haifa, IL
- Ruzica Piskac
Yale University, US
- Maria José Ramirez Quintana
Technical Univ. of Valencia, ES
- Ute Schmid
Universität Bamberg, DE
- Michael Siebers
Universität Bamberg, DE
- Rishabh Singh
Microsoft Res. – Redmond, US
- Gustavo Soares
Universidade Federal – Campina
Grande, BR
- Armando Solar-Lezama
MIT – Cambridge, US
- Claes Strannegård
Chalmers University of
Technology – Göteborg, SE
- Lorijn van Rooijen
Universität Paderborn, DE
- Janis Voigtländer
Universität Bonn, DE
- Christina Zeller
Universität Bamberg, DE
- Benjamin Zorn
Microsoft Res. – Redmond, US

