Deep Knowledge: Inductive Programming as an Answer

José Hernández-Orallo
DSIC, Universitat Politècnica de València, Spain
jorallo@dsic.upv.es

December 5, 2013

Abstract

Inductive programming has focussed on problems where data are not necessarily big, but representation and patterns may be deep (including recursion and complex structures). In this context, we will discuss what really makes some problems hard and whether this difficulty is related to what humans consider hard. We will highlight the relevance of background knowledge in this difficulty and how this has influence on a preference of inferring small hypotheses that are added incrementally. When dealing with the techniques to acquire, maintain, revise and use this knowledge, we argue that symbolic approaches (featuring powerful construction, abstraction and/or higher-order features) have several advantages over non-symbolic approaches, especially when knowledge becomes complex. Also, inductive programming hypotheses (in contrast to many other machine learning paradigms) are usually related to the solutions that humans would find for the same problem, as the constructs that are given as background knowledge are explicit and shared by users and the inductive programming system. This makes inductive programming a very appropriate paradigm for addressing and better understanding many challenging problems humans can solve but machines are still struggling with. Some important issues for the discussion will be the relevance of pattern intelligibility, and the concept of scalability in terms of incrementality, learning to learn, constructive induction, bias, etc.
Keywords: Inductive programming, deep knowledge, big data, incremental learning, constructive induction, knowledge bases.

1 Introduction

Much attention has recently been cast on Big Data [58, 48] and Deep Learning [2]. While this attention is deserved, we see that some important challenges in computer science and artificial intelligence lie in the construction of systems that are able to acquire and use both previous knowledge and context information [6] to deploy solutions for some unexpected situations. Learning to learn, meta-learning, transfer learning, incremental learning, context-aware computing, are terms of possible approaches for this. These approaches, however, are limited by a still inappropriate handling of knowledge: as knowledge becomes more complex and abstract, it is no longer processed in a completely automatic way. In fact, while the capability of automatically storing and handling factual, textual, numerical and statistical information has increased exponentially over the years, dealing automatically with knowledge bases of arbitrary depth and sophistication has increased at a much lower pace. Expert systems and knowledge-based systems have clearly improved, and the use of ontologies has provided more depth, but most knowledge bases are still based on sets of rules over some predefined features and concepts, where an abstract and constructive learning is not fully integrated (apart from some kind of incrementally learning more rules). Basically, the constructs and elements the system deals with after a time are the same it had initially. All of these uses are basically the knowledge acquisition, handling and application problem [52, 97], but where depth (and not necessary size or diversity) is the change.

On many occasions, the above problems have nothing to do with big data. In fact, much learning in humans and many of the prospective applications that machines cannot solve today work with *small* data. More precisely, each particular inference is not performed from a large number of examples, but just a few. Once a small piece of knowledge is integrated, other (small) inductive inference processes can take place,

incrementally. In other words, each new example is added to the knowledge base, which may force an increment or revision of the existing knowledge with generalisation or abstraction mechanisms.

The above problems are related to the deep learning approach, as architectures, concepts and features are said to be hierarchical. However, most deep learning approaches are based on artificial neural networks and other statistical approaches, and it is not clear how knowledge can be accessed, revised and integrated with other sources of knowledge.

In this short note we discuss on the appropriateness of inductive programming as a learning paradigm that may facilitate the acquisition, integration, modification, maintainance and application of *deep* knowledge, where new constructs and concepts can be developed, and being examined and evaluated because of the intelligibility of symbolic languages. We include many pointers for further reference. But this large number of references does not mean that we aim at being comprehensive nor exhaustive; this is just a position paper where we suggest that inductive programming can be the right approach for a series of fundamental problems to make machines learn from experience in a more incremental and constructive way. Henceforth, I recommend to read some of the surveys about inductive programming [30, 54, 29, 30] before reading this note. The site `http://inductive-programming.org` is also an excellent source as an overview of inductive programming, its community, events and systems.

2 Deep data, deep knowledge

Data can be small or big, but it can also be shallow or deep. Shallow or flat data is the common data representation in machine learning, where data is in a form of a table, where each column is a scalar feature. Deep data is usually characterised by existing relations between features, between examples, with the existence of complex structures such as lists, sets, trees, graphs, etc. Some deep data (toy) examples have been used in the literature of inductive logic programming and inductive programming, such as the East-West trains dataset [69], mutagenesis [95], block towers [87], the rectangle problems [80], Bongard problems [16], to name a few. Of course, natural language is also a source of complex data, and grammar inference [14] is one of the areas where this deep structure is unravelled.

In the same way, knowledge can either be small or big, but it can also be shallow or deep. Shallow (flat) knowledge can be represented by series of propositional rules, e.g., as those extracted from many association rule algorithms. Also, many expert systems are composed of hundreds or thousands of rules, but the features or concepts they handle have a predetermined flat structure. In contrast, some problems require theories that are deep, including different datatypes, structures, variables and recursion. The typical example of a deep theory or model is a program or algorithm. Handling knowledge bases that are composed of programs and algorithms is not an easy task, such as software repositories (to be precise, knowledge usually has some truth connotations, while software repositories are operational). We are interested in knowledge that can be learned semi-automatically. Examples of learning deep knowledge are any model that is able to capture the underlying patterns behind any of the deep data examples above, but many other diverse examples have been mentioned in the literature: learning algorithm or function implementations such as quicksort [75], learning the product from the addition [27], learning properties of lists [31], learning relations with probabilities [17], learning modal concepts [57], to name a few.

Complex structured (in the form of structured data types or structured prediction) do not entail an inductive programming approach. Kernels, distances or other notions [32, 25] can be used to convert a structured problem into a scalar feature representation. In fact, structured machine learning can include *or not* inductive (logic) programming techniques. Even the field of Statistical Relational Learning [33] goes beyond what inductive (logic) programming has been.

3 What's distinctive about inductive programming?

Inductive programming has emerged as a general term to refer to inductive inference with the use of programming languages. The term appeared in the late 1980s [68, 72] as a generalisation of both inductive functional programming (learning with functional programming languages such as Lisp) and inductive logic programming (learning with logic programming languages such as Prolog), which was steadily growing in the 1960s, 1970s and 1980s [82, 83, 37, 92, 96, 4, 3, 55, 91]. An inductive programming special interest group

was also created [38]. Some of the early works were produced in the context of artificial intelligence, while others took place in the context of program synthesis from examples. This wide range of applications and origins has been a distinctive trait of inductive programming, even from the first uses of the term, from software [68] to robotics [72].

Also, inductive programming is naturally associated with machine learning, although the term 'symbolic learning' was used in contrast to the so-called connectionist approach [71, 84]. While this opposition seems to be no longer a big issue in artificial intelligence (as many hybrid approaches exist [36] and many ways of bridging non-comprehensible models into comprehensible patterns have been studied [11, 20, 78, 22, 5, 24, 23]), it is still meaningful, at least in the context of machine learning. However, inductive programming has usually been associated with the notion of algorithmic models, possibly using variables and recursion, while symbolic learning is more usually associated to the use of rules, which can be just propositional. Nowadays, although the term inductive programming has been occasionally used to cover some non-declarative approaches to machine learning [81] there seems to be some consensus in its use for symbolic approaches [30, 54, 29, 30].

The term 'symbolic' may mean many things, including Turing machines. In praactice, the languages used in inductive programming for the representation of examples, hypotheses and background knowledge have been varied in terms of several features:

- Semantics: logical, functional, functional logic, hybrid...
- Conditions: unconditional, conditional, constraints...
- Expressive power: propositional, first-order, higher-order...
- Uncertainty and probability: close-world, open-world, abductive, probabilistic, stochastic, Bayesian...
- Beliefs and dynamics: modal, action...

Clearly, inductive logic programming [75, 76] has covered many of the previous paradigms, and it is the area that has been more active in the past twenty years. Nonetheless, other areas in artificial intelligence, machine learning, evolutionary computation, inductive inference, formal methods, functional programming and cognitive science have also contributed with techniques and systems that can be considered part of inductive programming. And many categories are no longer strict. For instance, the hybridisation of functional and logic programming led to the idea of inductive functional logic programming, and some of the associated papers were published in ILP conferences, functional programming conferences or artificial intelligence indistinctly [45, 46, 26, 27]

So, what is distinctive about inductive programming? Several characteristics have been suggested:

- Examples include relations between objects.
- Features are non-scalar.
- Patterns are constructive, rather than flat.
- Use of variables or constructor terms (or both).
- Use of recursion.
- Models can be comprehensible.

However, not all of these features are found in every particular inductive programming approach.

One possible inclusive characterisation of inductive programming can be the *inductive inference using symbolic languages that are (nearly) Turing-complete*. We use the term "symbolic languages" instead of a more specific "declarative (programming) languages", as imperative languages such as *C* or *Java* are not excluded, although it is much more usual to use declarative languages (functional, logical, etc.), because inference is much easier. Also, we do not use "programming languages" or "learning of algorithms" because we can consider languages that are not used for programming as a useful representation mechanism for examples, hypotheses and background knowledge, such as description logic. Note that this characterisation does not exclude the possibility of inductive inference from noisy or malicious data, while it is true that many applications of inductive programming (and some systems) only deal with perfect examples.

The previous characterisation does not fix any particular technique: refinement, analytical approaches, schema-driven, top-down approaches, bottom-up approaches, genetic programming approaches, explanation-based, generate-and-test, example-driven, Levin search, Monte Carlo, SAT solvers, etc. All of them are possible approaches for inductive programming.

The previous characterisation is also compatible with a wide variety of applications, as learning with complex representations is useful in many domains. For instance, the first workshop on Approaches and Applications of Inductive Programming (AAIP) held in conjunction with ICML 2005 (http://www.cogsys.wiai.uni-bamberg.de/aaip05/objectives.html) identified all applications where "learning of programs or recursive rules are called for, are first in the domain of software engineering where structural learning, software assistants and software agents can help to relieve programmers from routine tasks, give programming support for endusers, or support of novice programmers and programming tutor systems. Further areas of application are language learning, learning recursive control rules for AI-planning, learning recursive concepts in web-mining or for data-format transformations". Nonetheless, it is in program synthesis [31], programming by example [35, 56] or by demonstration [13] and automatic data manipulation [34] where inductive programming has found the most relevant and successful applications. Other areas where inductive inference has been recently applied or may be applied are knowledge acquisition, artificial general intelligence [12], cognitive systems [88], intelligent agents, games, robotics, personalisation, ambient intelligence and human interfaces.

4 What makes an inductive problem hard

If we are not particularly interested in big data (at least not learning from huge amounts of data *at a time*), where does the difficulty come from? Clearly, in inductive programming, the difficulty comes from the expressive power of using (nearly) Turing-complete languages, with the use of things such as constructor terms and recursion. But let us analyse where the difficulty lies exactly. First of all, we will limit our analysis to the difficulty of problem *instances* instead of problem *classes*. This distinction is important, as many problems in the same class are easy while many others are very difficult. In fact, considering problem instance is consistent with our use of Turing-complete languages, where we know that induction is not only intractable but incomputable, *in general*.

There are four main ingredients to determine the difficulty of a learning instance: the data D, the target model(s) or hypothesis h, the hypotheses space H and background knowledge B (which has influence on H). The direct question can be: are each of them big and deep? Actually, we need to consider all of them together to make a meaningful question from this. First, it is insightful to think about B and H together (as the bias). Second, it is important to realise that B has a dual effect, namely:

- If the background knowledge B does not contain key auxiliary concepts, the problem becomes very difficult as the concepts need to be invented, but
- If the background knowledge B contains too many auxiliary concepts, the problem is now how the appropriate auxiliary concepts are chosen.

In fact, the order of finding a hypothesis h with background knowledge B is in $O((|h|\cdot|B|)^{|h|})$. From here, we can be tempted to reduce B. However, as fewer auxiliary concepts are provided, they have to be created, and h becomes much larger, which is base and exponent here. So, the problem of induction becomes tractable when h is syntactically very small (the exponent), but a selection on B can be made.

Note that the number of examples is not included in the above considerations. We are assuming them to be a small number. But note that we also reach the conclusion that the hypothesis has to be small. In other words, at each small induction step, tractability for Turing-complete languages only seems to be possible if learning infers a small hypothesis from small data. This is precisely what inductive programming has been specialised in, learning small chunks of code or algorithms from a few examples. It is important to highlight that *complex (deep) concepts can only be expressed shortly if powerful abstract constructs have been introduced previously and are available in the background knowledge.*

If we look at how humans learn, we see some similarities. For instance, the constraints about hypothesis size is well known, as humans have important limitations on working memory, so complex hypotheses can only be constructed over previously derived or existing concepts [70]. Humans are not good at big data (except for perception mechanisms: e.g., vision, speech, music, etc.). In contrast, humans are good at appropriately handling B, both in the way it develops and is refined, and in the way it is applied to new problems. In fact, in humans, difficulty depends on how unrelated or non-contextual the solution is w.r.t. previous knowledge.

This sets a different view of scalability for inductive inference, where the key issue is the size of knowledge. In humans, e.g., fluid vs. crystallised intelligence are distinguished, where fluid intelligence is affected by scalability on D, H and h, and crystallised intelligence is also affected by scalability on B.

As a result, we need new ways of comparing inductive systems such that the role of background knowledge is taken into account. First, many different kinds of problems have been used in ILP, IP, AI, ML and program synthesis, but we only have informal assessments of their difficulty. Second, some systems are able to solve them from scratch, others with background knowledge. Third, some systems are able to solve just one type of problems, others are more general. Consequently, we need libraries of problems (featuring deep data and deep background knowledge), to really know what the challenges are, when there is real progress, etc. The construction of new benchmarks can originate from real problems: e.g,. program synthesis, ILP problems, AI problems and IQ tests provide some starting collection, as done in [47] with a systematic assessment of their properties la [60, 59] or with artificial problems (possibly enriching existing approaches for flat artificial dataset generators [86, 65]). Also, the difficulty of the problems can be set subjectively (e.g., relative to humans) or can be derived theoretically (e.g., using algorithmic information theory, [42, 41, 44].

5 Big deep knowledge

Naturally, the idea of learning incrementally, as well as the construction and contextual application of (large repositories of) background knowledge or function libraries for inductive problems has been a key and recurrent issue in the past:

- Incrementality (data [53, 27] and knowledge [80, 94, 89, 39]), repeat learning [51].
- Learning to learn, meta-learning, incremental self-improvement [90], policy reuse [64, 63, 62].
- Function and predicate invention [74, 79, 40]
- Constructive induction [73] and constructive reinforcement learning [43].
- Meta-knowledge [10, 67], declarative bias [7], transfer learning, relational reinforcement learning , beliefs and modality [57].
- (Interactive) Theory Revision [85, 15], Theory completion [77].
- Integration with abduction [28] and deduction.
- Representation, knowledge level change and learning [19],
- Use of large repositories of functions or predicates [49],
- Knowledge dependency and redundancy, what to keep explicitly and implicitly (inductive and deductive gains) [42],
- Learning and evolution in expert systems and knowledge-based systems [61, 1, 97]

Some of these areas are oldies (but goldies) in Inductive Logic Programming (ILP), Inductive Programming (IP), program synthesis, AI, cognitive science and other areas. Many also have recently had a new revival.

While some of the above approaches are not in the scope of inductive programming, now we will argue that inductive programming (in any of its forms, including, of course ILP) is appropriate for this problem. First, we must acknowledge that in machine learning and even artificial intelligence, two paradigms are now predominant: the connectionist approach and the statistical-probabilistic approach (if they can now be considered different approaches, as the mathematical interpretation of neural networks as non-linear discriminants sets a continuum between the two). Even in cognitive science, and robotics, the use of neural networks seems to be preferred, as the human and animal brains are based on (natural) neural networks. Actually, the deep learning paradigm is largely influenced by both the statistical-probabilistic approach and the connectionist approach.

Inductive programming, while clearly different to the way humans brains work, has several advantages. First, as knowledge grows but systems do not develop natural language, it becomes more and more difficult to analyse, supervise, fix and understand the knowledge of a system if it is not represented *internally* in an intelligible way. Using inductive programming does not ensure this introspection, but can make it possible. In contrast, connectionist systems are much more difficult to understand once they begin to have emergent property. For instance, when a system integrates millions of neurons in several subsystems, and

mentally develop for days or even weeks, how can we supervise and understand this knowledge? For instance, Spaun[21], an impressive brain model based on millions of neurons that is able to do several tasks, including some list processing, is difficult to understand and really see how some problems are solved. For instance, it is difficult to check whether Spaun solves list problems by capturing the recursive concept of a list structure or because it uses some shortcuts (such as recognising the first and last element).

Secondly, many tools for handling knowledge have been developed with symbolic or rule-based approaches, such as theory revision, consistency check, adding or removing constraints, etc. In fact, we can understand many pieces (or all) of H, h, D and B. This can hold in the short, mid and long terms for B. We can provide start-up knowledge B_0. We can revise and fix their knowledge. We have tools to combine different sources of knowledge. Agents can exchange knowledge before they can develop any (natural) language. In the end, many applications require model understanding (scientific discovery, multi-agent systems, software engineering, engineering modelling...). In particular, cognitive science, development robotics and robot programming by demonstration are also areas where this application can be more productive because of additional reasons: inductive programming hypotheses are usually related to the solutions that humans would find for the same problem, we can understand the solutions reached by the system and we can see explicitly the mental constructs the system has been given. Several examples are proliferating: common cognitive tasks [87], cognitive tutors [66], analysis of number or symbol series tasks [9, 93], etc.

All this is basically a vindication of symbolic AI and symbolic learning, but some considerations have to be made. Deep learning, and other paradigms, illustrate that new concepts, feature transformations and abstractions need to be learned. This cannot be done with rule-based symbolic systems that are not able to construct new concepts. This was one of the reasons why symbolic representation was criticised in the 1980s and 1990s [8]. In other words, if deep learning suggests that multi-level architectures of several conceptual levels are needed for emergence or more complex constructs, we say that knowledge representation languages must be able to use notions such as higher-order functions, predicate/function invention and other kinds of abstractions, so that emergence can take place. The field that has worked on these ideas in the past decades is precisely inductive programming.

6 Discussion

One of the motivations of this note was to put emphasis on the idea that learning from small data is interesting and useful, and also that it can be difficult if the data is deep, especially if we have an expressive language and a rich deep knowledge. In fact, some phenomena have been observed when learning takes place in the context of large knowledge bases, such as decreased performance when the knowledge base gets larger [50, 18, 49]. Many pointers have been included, but it is more evident that much more needs to be done on knowledge acquisition and reuse. We have (briefly) argued that inductive programming is a good way to address the knowledge acquisition problem.

As a final remark, I would like to criticise a research bias towards *disposable* learning and *disposable* research. We have all created many systems that learn from data again and again, and start completely void for each application, without any knowledge acquisition and reuse. Creating incremental systems that need to acquire knowledge and keep this knowledge to improve their learning abilities is a challenge that requires a more continuous effort and larger teams, as results can be discouraging initially: newborn baby inductive programming systems are not expected to be very useful until they can be trained and can fully develop.

Acknowledgements

This work was supported by the MEC/MINECO projects CONSOLIDER-INGENIO CSD2007-00022 and TIN 2010-21062-C02-02, GVA project Prometeo/2008/051, the COST - European Cooperation in the field of Scientific and Technical Research IC0801 AT, and the REFRAME project granted by the European Coordinated Research on Long-term Challenges in Information and Communication Sciences & Technologies ERA-Net (CHIST-ERA), and funded by the respective national research councils and ministries (Ministerio de Economía y Competitividad, with code PCIN-2013-037, in Spain).

References

[1] Rajendra Akerkar and Priti Sajja. *Knowledge-based systems*. Jones & Bartlett Publishers, 2010. 5

[2] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009. 1

[3] Alan W Biermann. The inference of regular lisp programs from examples. *Systems, Man and Cybernetics, IEEE Transactions on*, 8(8):585–600, 1978. 2

[4] Alan W. Biermann and Douglas R. Smith. The hierarchical synthesis of LISP scanning programs. In B. Gilchrist, editor, *Information Processing 77*, pages 41–45, Amsterdam, 1977. North-Holland Publishing. 2

[5] R Blanco-Vega, José Hernández-Orallo, and M José Ramírez-Quintana. Knowledge acquisition through machine learning: minimising expert's effort. In *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, pages 6–pp. IEEE, 2005. 3

[6] Patrick Brézillon. Context in problem solving: a survey. *The Knowledge Engineering Review*, 14(1):47–80, 1999. 1

[7] Will Bridewell and Ljupčo Todorovski. Learning declarative bias. In *Inductive Logic Programming*, pages 63–77. Springer, 2008. 5

[8] Rodney A Brooks. Intelligence without representation. *Artificial intelligence*, 47(1):139–159, 1991. 6

[9] Jochen Burghardt. E-generalization using grammars. *Artificial intelligence*, 165(1):1–35, 2005. 6

[10] John Cabral, Robert C Kahlert, Cynthia Matuszek, Michael Witbrock, and Brett Summers. Converting semantic meta-knowledge into inductive bias. In *Inductive Logic Programming*, pages 38–50. Springer, 2005. 5

[11] Gail A Carpenter and Ah-Hwee Tan. Rule extraction: From neural architecture to symbolic representation. *Connection Science*, 7(1):3–27, 1995. 3

[12] Neil Crossley, Emanuel Kitzelmann, Martin Hofmann, and Ute Schmid. Combining analytical and evolutionary inductive programming. In *Second Conference on Artificial General Intelligence*, pages 19–24, 2009. 4

[13] Ellen Cypher and Daniel Conrad Halbert. *Watch what I do: programming by demonstration*. The MIT Press, 1993. 4

[14] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010. 2

[15] Luc De Raedt. Interactive theory revision: an inductive logic programming approach. *status: published*, 1992. 5

[16] Luc De Raedt. Inductive logic programming. In *Sammut, Claude and Webb, Geoffrey I "Encyclopedia of machine learning"*, pages 529–534. Springer-Verlag, 2011. 2

[17] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467, 2007. 2

[18] Nathaniel Leonard Derbinsky. *Effective and Efficient Memory for Generally Intelligent Agents*. PhD thesis, The University of Michigan, 2012. 6

[19] Thomas G Dietterich. Learning at the knowledge level. *Machine Learning*, 1(3):287–315, 1986. 5

[20] Pedro Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(1):187–202, 1998. 3

[21] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Charlie Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012. 6

[22] Vicent Estruch, César Ferri, José Hernández-Orallo, and M José Ramírez-Quintana. Simple mimetic classifiers. In *Machine Learning and Data Mining in Pattern Recognition*, pages 156–171. Springer, 2003. 3

[23] Vicent Estruch, César Ferri, José Hernández-Orallo, and M José Ramírez-Quintana. Similarity functions for structured data. an application to decision trees. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 10(29):109–121, 2006. 3

[24] Vicent Estruch, César Ferri, José Hernández-Orallo, and M José Ramírez-Quintana. Bridging the gap between distance and generalization. *Computational Intelligence*, pages no–no, 2014. 3

[25] Vicent Estruch, Cèsar Ferri, José Hernández-Orallo, and María-José Ramírez-Quintana. A survey of (pseudo-distance) functions for structured-data. In *TAMIDA, CEDI*, pages 233–242, 2005. 2

[26] Cèsar Ferri, José Hernández-Orallo, and María-José Ramírez-Quintana. The FLIP system homepage, 2000. 3

[27] Cesar Ferri-Ramírez, José Hernández-Orallo, and M José Ramírez-Quintana. Incremental learning of functional logic programs. In *Functional and Logic Programming*, pages 233–247. Springer, 2001. 2, 3, 5

[28] Peter A Flach and Antonis C Kakas. *Abduction and Induction: Essays on their relation and integration*, volume 18. Springer, 2000. 5

[29] Pierre Flener and Derek Partridge. Inductive programming. *Automated Software Engineering*, 8(2):131–137, 2001. 2, 3

[30] Pierre Flener and Ute Schmid. An introduction to inductive programming. *Artificial Intelligence Review*, 29(1):45–62, 2008. 2, 3

[31] Pierre Flener and Serap Yıilmaz. Inductive synthesis of recursive logic programs: Achievements and prospects. *The Journal of Logic Programming*, 41(2):141–195, 1999. 2, 4

[32] Thomas Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003. 2

[33] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. The MIT press, 2007. 2

[34] Sumit Gulwani, William R Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105, 2012. 4

[35] Daniel Conrad Halbert. *Programming by example*. PhD thesis, University of California, Berkeley, 1984. 4

[36] Barbara Hammer and Pascal Hitzler. *Perspectives of neural-symbolic integration*. Springer Publishing Company, Incorporated, 2007. 3

[37] S. Hardy. Synthesis of LISP functions from examples. In *IJCAI'75: Proceedings of the 4th International Joint Conference on Artificial Intelligence (Tbilisi, Georgia, USSR, Sept. 3–8, 1975)*, pages 240–245, 1975. 2

[38] JE Hayes-Michie. Pragmatica: Bulletin of the inductive programming special interest group. *Turing Institute P ress, Glasgow, UK*, 1990. 3

[39] R. Henderson. Incremental learning in inductive programming. In *Approaches and Applications of Inductive Programming*, volume 5812 of *LNCS*, pages 74–92. Springer, 2010. 5

[40] R. J. Henderson and S. H. Muggleton. Automatic invention of functional abstractions. In *Latest Advances in Inductive Logic Programming*. Imperial College Press, 2012. To appear. 5

[41] José Hernández-Orallo. Beyond the Turing test. *Journal of Logic, Language and Information*, 9(4):447–466, 2000. 5

[42] José Hernández-Orallo. Computational measures of information gain and reinforcement in inference processes. *AI Communications*, 13(1):49–50, 2000. 5

[43] José Hernández-Orallo. Constructive reinforcement learning. *International Journal of Intelligent Systems*, 15(3):241–264, 2000. 5

[44] José Hernández-Orallo. On the computational measurement of intelligence factors. *NIST special publication*, pages 72–79, 2001. 5

[45] José Hernández-Orallo and M José Ramírez-Quintana. Inverse narrowing for the induction of functional logic programs. In *APPIA-GULP-PRODE*, pages 379–392. Citeseer, 1998. 3

[46] José Hernández-Orallo and M José Ramírez-Quintana. A strong complete schema for inductive functional logic programming. In *Inductive Logic Programming*, pages 116–127. Springer, 1999. 3

[47] Martin Hofmann, Emanuel Kitzelmann, and Ute Schmid. A unifying framework for analysis and evaluation of inductive programming systems. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 55–60, 2009. 5

[48] Adam Jacobs. The pathologies of big data. *Communications of the ACM*, 52(8):36–44, 2009. 1

[49] Susumu Katayama. Efficient exhaustive generation of functional programs using monte-carlo search with iterative deepening. In *PRICAI 2008: Trends in Artificial Intelligence*, pages 199–210. Springer, 2008. 5, 6

[50] William G Kennedy and J Gregory Trafton. Long-term symbolic learning. *Cognitive Systems Research*, 8(3):237–247, 2007. 6

[51] Khalid Khan, Stephen Muggleton, and Rupert Parson. Repeat learning using predicate invention. In *Inductive Logic Programming*, pages 165–174. Springer, 1998. 5

[52] Alison L Kidd. *Knowledge acquisition for expert systems: a practical handbook*. Plenum Press, 1987. 1

[53] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In *Inductive logic programming*, 1992. 5

[54] Emanuel Kitzelmann. Inductive programming: A survey of program synthesis techniques. In *Approaches and Applications of Inductive Programming*, pages 50–73. Springer, 2010. 2, 3

[55] Yves Kodratoff and J. Fargues. A sane algorithm for the synthesis of LISP functions from example problems: The Boyer and Moore algorithm. In Derek H. Sleeman, editor, *Proceedings of AISB/GI Conference (Hamburg, Germany, July 18–20, 1978*, pages 169–175. Leeds University, 1978. Now ECAI: Proceedings of the 4th European Conference on Artificial Intelligence. 2

[56] Henry Lieberman. *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001. 4

[57] John W Lloyd and Kee Siong Ng. Learning modal theories. In *Inductive Logic Programming*, pages 320–334. Springer, 2007. 2, 5

[58] Clifford Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008. 1

[59] Núria Macià and Ester Bernadó-Mansilla. Towards uci+: A mindful repository design. *Information Sciences*, 2013. 5

[60] Núria Macià, Ester Bernadó-Mansilla, Albert Orriols-Puig, and Tin Kam Ho. Learner excellence biased by data set selection: A case for data characterisation and artificial data sets. *Pattern Recognition*, 2012. 5

[61] Sandra Marcus. *Automating knowledge acquisition for expert systems*, volume 57. Kluwer Academic Publishers Dordrecht,, Holland, 1988. 5

[62] Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, and María José Ramírez-Quintana. Learning with configurable operators and rl-based heuristics. In *New Frontiers in Mining Complex Patterns*, pages 1–16. Springer, 2013. 5

[63] Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, and María-José Ramírez-Quintana. On the definition of a general learning system with user-defined operators. *arXiv preprint arXiv:1311.4235*, 2013. 5

[64] Fernando Martınez-Plumed, Cèsar Ferri, José Hernández-Orallo, and María-José Ramírez-Quintana. Policy reuse in a general learning framework. In *CAEPIA*. "", 2013. 5

[65] Syahaneim Marzukhi, Will N Browne, and Mengjie Zhang. Adaptive artificial datasets through learning classifier systems for classification tasks. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 1243–1250. ACM, 2013. 5

[66] Noboru Matsuda, William W Cohen, Jonathan Sewall, and Kenneth R Koedinger. Applying machine learning to cognitive modeling for cognitive tutors. *Human-Computer Interaction Institute. Paper 248.*, 2006. 6

[67] Eric McCreath and Arun Sharma. Extraction of meta-knowledge to restrict the hypothesis space for ilp systems. In *AI-CONFERENCE*, pages 75–82. Citeseer, 1995. 5

[68] Donald Michie. Technology lecture. the superarticulacy phenomenon in the context of software manufacture. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 405(1829):185–212, 1986. 2, 3

[69] Donald Michie, Stephen Muggleton, David Page, and Ashwin Srinivasan. To the international computing community: A new east-west challenge. Technical report, Technical report, Oxford University Computing laboratory, Oxford, UK, 1994. 2

[70] George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956. 4

[71] Raymond Mooney, Jude Shavlik, Geoffrey Towell, and Alan Gove. An experimental comparison of symbolic and connectionist learning algorithms. *Readings in machine learning*, pages 171–176, 1990. 3

[72] Peter Mowforth and Ivan Bratko. Ai and robotics; flexibility and integration. *Robotica*, 5(2):93–98, 1987. 2, 3

[73] Stephen Muggleton. Duce, an oracle-based approach to constructive induction. In *IJCAI*, pages 287–292. Citeseer, 1987. 5

[74] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. *Inductive logic programming*, pages 261–280, 1992. 5

[75] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994. 2, 3

[76] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. Ilp turns 20. *Machine Learning*, 86(1):3–23, 2012. 3

[77] Stephen H Muggleton and Christopher H Bryant. Theory completion using inverse entailment. In *Inductive Logic Programming*, pages 130–146. Springer, 2000. 5

[78] Haydemar Núñez, Cecilio Angulo, and Andreu Català. Rule extraction from support vector machines. In *ESANN*, pages 107–112, 2002. 3

[79] J Roland Olsson. How to invent functions. In *Genetic Programming*, pages 232–243. Springer, 1999. 5

[80] Roland Olsson. Inductive functional programming using incremental program transformation. *Artificial intelligence*, 74(1):55–81, 1995. 2, 5

[81] Derek Partridge. The case for inductive programming. *Computer*, 30(1):36–41, 1997. 3

[82] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, Edinburgh, 1969. 2

[83] G. D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, 1971. 2

[84] John Ross Quinlan. Comparing connectionist and symbolic learning methods. In *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*. Citeseer, 1994. 3

[85] Bradley L Richards and Raymond J Mooney. First-order theory revision. In *ML*, pages 447–451, 1991. 5

[86] Joaquin Rios-Boutin, Albert Orriols-Puig, and J-M Garrell-Guiu. Artificial data sets based on knowledge generators: Analysis of learning algorithms efficiency. In *Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on*, pages 873–878. IEEE, 2008. 5

[87] Ute Schmid, Martin Hofmann, and Emanuel Kitzelmann. Analytical inductive programming as a cognitive rule acquisition devise. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 162–167, 2008. 2, 6

[88] Ute Schmid and Emanuel Kitzelmann. Inductive rule learning on the knowledge level. *Cognitive Systems Research*, 12(3):237–248, 2011. 4

[89] Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004. 5

[90] Jürgen Schmidhuber, Jieyu Zhao, and Marco Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997. 5

[91] Ehud Y Shapiro. An algorithm that infers theories from facts. In *IJCAI*, pages 446–451, 1981. 2

[92] D. Shaw, W. Swartout, and C. Green. Inferring LISP programs from examples. In *IJCAI'75: Advance Papers of the 4th International Joint Conference on Artificial Intelligence (Tbilisi, Georgia, USSR,Sept. 3–8, 1975)*, pages 260–267, 1975. 2

[93] Michael Siebers and Ute Schmid. Semi-analytic natural number series induction. In *KI 2012: Advances in Artificial Intelligence*, pages 249–252. Springer, 2012. 6

[94] Ray J Solomonoff. Progress in incremental machine learning. In *NIPS Workshop on Universal Learning Algorithms and Optimal Search*, pages 211–256, 2002. 5

[95] Ashwin Srinivasan, Stephen Muggleton, Ross D King, and Micheal JE Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In *Proceedings of the 4th international workshop on inductive logic programming*, volume 237, pages 217–232. Citeseer, 1994. 2

[96] Phillip D. Summers. *Program Construction from Examples*. PhD thesis, Department of Computer Science, Yale University, New Haven, US-CT, 1975. 2

[97] Liang-Jun Zang, Cong Cao, Ya-Nan Cao, Yu-Ming Wu, and CAO Cun-Gen. A survey of commonsense knowledge acquisition. *Journal of Computer Science and Technology*, 28(4):689–719, 2013. 1, 5