Inductive Programming Meets the Real World

Sumit Gulwani, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, USA.
sumitg@microsoft.com

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

Emanuel Kitzelmann, Adam-Josef-Cüppers Commercial College, Ratingen, Germany.
ekitzelmann@gmail.com

Stephen H. Muggleton, Department of Computing, Imperial College London, UK.
s.muggleton@imperial.ac.uk

Ute Schmid, University of Bamberg, Germany.
ute.schmid@uni-bamberg.de

Ben Zorn, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, USA.
Ben.Zorn@microsoft.com

Since most end-users lack programming skills they often spend considerable time and effort performing tedious and repetitive tasks such as capitalising a column of names manually. Inductive Programming has a long research tradition and recent developments demonstrate it can liberate users from many tasks of this kind.

Key insights

1. Real-world applications emerge with spreadsheet tools, intelligent program tutors, and robotics.

2. Learning from few examples is possible because users and systems share the same background knowledge.

3. Search is guided by domain-specific languages and the use of higher-order knowledge.

Much of the world’s population use computers for everyday tasks, but most fail to benefit from the power of computation due to their inability to program. Most crucially, users often have to perform repetitive actions manually because they are not able to use the macro languages which are available for many application programs. Recently, a first mass market product was presented in the form of the FlashFill feature in Microsoft Excel 2013. FlashFill allows end-users to automatically generate string processing programs for spreadsheets from one or more user-provided examples. The fact FlashFill is able to learn a large variety of quite complex programs from only a few examples is due to the incorporation of inductive programming methods.

Inductive Programming (IP) is an inter-disciplinary domain of research in computer science, artificial intelligence, and cognitive science that studies the automatic synthesis of computer programs from examples and background knowledge. IP developed from research on inductive program synthesis, nowadays termed inductive functional programming (IFP), and from inductive inference techniques using logic.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2008 ACM 0001-0782/08/0X00 ...$5.00.

1. REAL-WORLD APPLICATIONS

The application of IP to general purpose tasks originally focused on synthesizing functional or logic programs for tasks such as manipulating data structures (e.g., sorting or reversing a list). Early investigations showed small programs could be synthesized from a few input/output examples.

1.1 Synthesizing small programs
As programming languages and systems have evolved, many high-level domain-specific features have been added to them, enabling powerful tasks to be expressed in a small number of operations using abstractions. For example: spreadsheet programs (leveraging the calc engine abstraction), perl/python programs (leveraging the Unix runtime libraries, regular expression sub-language, and pipe-based commands on text), and web programs (leveraging JavaScript and the DOM abstraction). These developments have enabled IP solutions to real-world problems for two key reasons: (a) While the number of people able to program with these frameworks is much larger than C/C++/Java programmers, the plethora of platforms make it difficult for them to be familiar with the details of each platform. (b) Synthesis techniques work more effectively (in real-time) when the programs to be synthesized are relatively small.

One domain of application is bit-vector manipulating programs [14]. In the 2013 ICFP programming contest (http://research.microsoft.com/en-us/events/icfpcontest2013/), contestants were asked to synthesize programs for variable length bit-vector manipulation. The winners shocked everyone, including the contest organizers, when they were able to synthesize correct programs of up to 51 instructions from examples in less than 5 minutes.

1.2 Data manipulation

Figure 1: FlashFill: An Excel 2013 feature that lets users automate repetitive string transformations using examples. Once the user performs one instance of the desired transformation (row 2, column B) and proceeds to transforming another instance (row 3, column B), FlashFill learns a program Concatenate(ToLower(SubString(v,WordToken,1)," ",ToLower(SubString(v,WordToken,2)))) that extracts the first two words in the input string v (in column A), converts them to lowercase, and concatenates them together separated by a space character.

Existing programmatic solutions to data manipulation in documents of various types such as text/log files, spreadsheets and webpages have three key limitations. First, the solutions are domain-specific and require knowledge/expertise in different technologies for different document types. Second, they require understanding of the entire underlying document structure including the data fields the end-user is not interested in (some of which may not even be visible in the presentation layer of the document). Third, and most significantly, they require knowledge of programming. As a result, users have to resort to manual copy-paste, which is both time-consuming and error prone.

Inductive synthesis can help out with a variety of data manipulation tasks. These include: (a) Extracting data from semi-structured documents including text files, web pages, and spreadsheets [21] (as in Fig. 2). (b) Transformation of atomic data types such as strings [7] (as in Fig. 1) or numbers. Transformation of composite data types such as tables [9] and XML [32]. (c) Formatting data [33]. Combining these technologies in a pipeline of extraction, transformation, and formatting can allow end-users to perform sophisticated data manipulation tasks.

1.3 Computer-aided Education

Human learning and communication is often structured around examples—be it a student trying to understand or master a certain concept using examples, or be it a teacher trying to understand a student’s misconceptions or provide feedback using example behaviors. Example-based reasoning techniques developed in the inductive synthesis community can help automate several repetitive and structured tasks in education including problem generation, solution generation, and feedback generation [8]. These tasks can be automated...
for a wide variety of STEM subject domains including logic, automata theory, programming, arithmetic, algebra, and geometry. For instance, Fig. 3 shows the output of an inductive synthesis technique for generating algebraic proof problems similar to a given example.

<table>
<thead>
<tr>
<th>Example Problem</th>
<th>[ \sin A + \frac{1 + \cos A}{\sin A} = 2 \sec A ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalized Problem Template</td>
<td>[ \frac{T_1 A}{1 + T_2 A} + \frac{1 + T_3 A}{T_4 A} = 2 T_5 A ]</td>
</tr>
<tr>
<td>where ( T_i \in { \cos, \sin, \tan, \cot, \sec, \csc } )</td>
<td></td>
</tr>
</tbody>
</table>
| New Similar Problems | \[ \begin{align*}
\cos A + \frac{1 - \sin A}{\cos A} &= 2 \tan A \\
1 - \sin A + \frac{1 + \sin A}{\cos A} &= 2 \sec A \\
1 + \sin A + \frac{1 + \sec A}{\cos A} &= 2 \sec A \\
1 + \sec A + \frac{1 + \tan A}{\cot A} &= 2 \sec A \\
\tan A + \frac{1 + \sec A}{\tan A} &= 2 \sec A \\
\sin A + \frac{1 - \cos A}{\sin A} &= 2 \cot A \\
1 - \cos A + \frac{1 - \sec A}{\sin A} &= 2 \cot A \\
\end{align*} \] |

Figure 3: Problem generation for algebraic proof problems involving identities over analytic functions such as trigonometry. A given problem is generalized into a template and valid instantiations are found by testing on random values for free variables.

1.4 Future opportunities

We have described important real world applications of IP. We believe there are many other domains to which IP can and will be applied in the near future. Any domain in which a set of high-level abstractions already exists is a strong candidate for IP. For example, the If This Then That (IFTTT) service (http://ifttt.com/), which allows end-users to express small rule-based programs using triggers and actions, is an excellent candidate for application of IP. Similarly, smartphones have many powerful built-in operators that can be combined in simple ways to perform useful tasks and as such are an excellent domain for IP. Looking farther out, programming robots, which are increasingly prevalent in our lives, is likely to benefit from IP.

2. INDUCTIVE PROGRAMMING VS. OTHER MACHINE LEARNING PARADIGMS

IP is concerned about making machines learn programs automatically and can hence be considered another machine learning paradigm. So, what is distinctive about inductive programming? Table 1 outlines a series of differences, some of which we discuss below.

2.1 Small data

As collecting and storing data is becoming cheaper, it is easy to gain the impression that the only interesting datasets nowadays involve big data. However, datasets from a single user’s interaction with whatever kind of device are usually quite small, such as the amount of data gathered about a person’s agenda, as shown at the top of Fig. 4.

It is well known that learning from small numbers of examples is more difficult and unreliable than learning from lots of data. The fewer examples we have, the more prone we are to overfitting, especially with expressive languages. IP is particularly useful when the number of examples is small but the hypothesis space is large (Turing-complete). How can inductive programming deal with this?

2.2 Declarative representation

Most (statistical) machine learning techniques are based on probabilities, distances, weights, kernels, matrices, etc. None of these approaches, except for techniques based on (propositional) decision trees and rules, are declarative, i.e., expressed as a set of potentially comprehensible rules. Hence, another distinctive feature of IP is that it uses a rich symbolic representation, as hypotheses are usually declarative programs.

The declarative approach permits the use of a single language to represent background knowledge, examples and hypotheses, as shown in Fig. 4. Apart from the accessibility of one single language for the (end-)user, knowledge can be inspected, revised and integrated with other sources of knowledge much more easily. As a result, incremental, cumulative or life-long learning becomes easier [11]. For instance, NELL (Never-Ending Language Learner) [2] uses an ILP algorithm that learns probabilistic Horn clauses.

Nowadays, many languages in IP are hybrid such as functional and non-symbolic approaches to deep learning [1], a new approach in machine learning where more complex models and features are built in a hierarchical way.

2.3 Refinement and abstraction

Another particular issue about IP is the way the hypothesis space is arranged by properly combining several inference mechanisms such as deduction, abduction and induction. Many early IP operators were inversions of deduction operators, leading to bottom-up and top-down approaches, where generalization and specialization operators, respectively, are used [30]. More generally, refinement and abstraction operators, including the use of higher-order functions, predicates and function invention, can be defined according to the operational semantics of the language.

This configures, possibly infinitely, many levels between merely extensional facts and more intensional knowledge, leading to a hierarchical structure. This is very different to other non-symbolic approaches to deep learning [1], a new approach in machine learning where more complex models and features are built in a hierarchical way.

2.4 Deep knowledge

Because of the abstraction mechanisms and the use of background knowledge, IP considers learning as a knowledge acquisition process. In Fig. 4, for instance, inductive programming has access to some information about contact groups as well as relationships between the contacts (such as family bonds or work hierarchies). Such knowledge is known as background knowledge and works as a powerful explicit bias to reduce the search space and to find the right level of generalization.

Knowledge can be considered to be deep if it references lower level definitions, including recursively referencing itself. Representation of such structured and deep knowledge is achieved by programming languages that feature variables, rich operational semantics and, most especially, recursion.
Recursion is a key issue in inductive programming [38, 27, 35, 18]. Note that both the background knowledge and the new hypothesis in the example of Fig. 4 are recursive.

This is in contrast to other machine learning approaches where background knowledge has only the form of prior distributions, probabilities or features.

2.5 Purpose and evaluation

In other machine learning approaches, hypotheses are measured by different metrics accounting for a degree of error. The purpose of IP is not just to maximise some particular error metric, but to find meaningful programs that are operational, according to the purpose of the IP application. This usually implies that they have to be consistent with most (if not all) the data but also with the background knowledge and other possible constraints. Also, as hypotheses are declarative (and possibly recursive), the evaluation is more diverse, including criteria such as simplicity, comprehensibility, coherence and time/space complexity.

3. INDUCTIVE PROGRAMMING BECOMES MORE COGNITIVE

Among the factors which helped inductive programming to come out of the box is the newly established relation with human cognition. On the one hand, IP takes into account interaction with (end-) users, having realised that intuitive human-computer interaction is fundamental for successful applications as discussed in Section 1. On the other hand, cognitive science research became aware of IP. Its declarative, constructive character and its ability of performing high-level abstractions are making IP algorithms good candidates for exploring how a cognitive system is able to learn inductively from experience. For human-computer interaction as well as for cognitive modeling, the crucial advantage of IP approaches is they learn in a way similar to humans’ beings. The inferred hypotheses are usually closer (or equal) to those humans would get for the same problem.

3.1 Human-computer vs. human-human interactions

IP offers a very natural mechanism to replace programming by imitation learning. A natural strategy for humans to exchange knowledge and procedures with other people. Knowledge is usually exchanged declaratively in natural language and procedures are more generally explained by doing. These are exactly the two inputs an inductive programming system has: background knowledge and factual examples, as we see in the top of Fig. 4. However, when humans learn by demonstration and the procedure is more complex, some intermediate steps can be shown as well. This type of interaction is also supported by several IP techniques with the use of program traces.

In fact, interfaces that learn (or learning systems that interface well with humans) go beyond the notion of ‘invisible’, ‘intuitive’ or ‘natural’ user interfaces, to embrace the so-called Cognitive User Interfaces (CUI) [40] with the ultimate goal that machines interact like humans, thereby becoming more intuitive, trustable, familiar and predictable — including predicting when the system is going to fail.

In order to achieve this through IP we need to settle the interaction model. For instance, the supervision from the user can be limited to some rewards (“OK” buttons) or penalties (“Cancel” buttons) about what the system is doing, as illustrated in Figure 4. Alternatively, the user can give a few examples, the IP systems makes guesses for other examples and the users corrects them [9, 3]. In this interactive (or query) learning process the user can choose among a set of candidate hypotheses by showing where they differ, using a distinguishing input generated by the user — or more effectively — by the IP system itself.

3.2 Inductive programming vs. human learning

Cognitive science and psychology have shown that humans have limited short-memory, learn from a short number of (usually positive) examples and are relatively intolerant to

### Table 1: A simplified comparison between Inductive Programming and other machine learning paradigms

<table>
<thead>
<tr>
<th></th>
<th>Inductive Programming</th>
<th>Other Machine Learning Paradigms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of examples</strong></td>
<td>Small.</td>
<td>Large, e.g. big data.</td>
</tr>
<tr>
<td><strong>Kind of data</strong></td>
<td>Relational, constructor-based datatypes.</td>
<td>Flat tables, sequential data, textual data, etc.</td>
</tr>
<tr>
<td><strong>Data source</strong></td>
<td>Human experts, software applications, HCI, etc.</td>
<td>Transactional databases, Internet, sensors (IoT), etc.</td>
</tr>
<tr>
<td><strong>Hypothesis language</strong></td>
<td>Declarative: general programming languages or domain-specific languages.</td>
<td>Linear, non-linear, distance-based, kernel-based, rule-based, probabilistic, etc.</td>
</tr>
<tr>
<td><strong>Search space arrangement</strong></td>
<td>Refinement, abstraction operators, brute-force.</td>
<td>Gradient-descent, divide-and-conquer, covering, instance-based, etc.</td>
</tr>
<tr>
<td><strong>Pattern comprehensibility</strong></td>
<td>Common.</td>
<td>Uncommon.</td>
</tr>
<tr>
<td><strong>Pattern expressiveness</strong></td>
<td>Usually recursive, even Turing-complete.</td>
<td>Feature-value, non-Turing complete.</td>
</tr>
<tr>
<td><strong>Learning bias</strong></td>
<td>Using background knowledge and constraints.</td>
<td>Using prior distributions, parameters and features.</td>
</tr>
<tr>
<td><strong>Evaluation</strong></td>
<td>Diverse criteria, including simplicity, comprehensibility and time/space complexity.</td>
<td>Oriented to error (or loss) minimisation.</td>
</tr>
</tbody>
</table>
Coherence, simplicity and explanatory power are guiding rules in human inductive inference. The role of background knowledge and the necessary constructs that need to be developed in order to acquire more abstract concepts has also been a matter of study in cognitive science [39]. This progressive acquisition of deep knowledge in humans is prominent in language learning. From Chomsky's early proposal of a language acquisition device to the present linguists and cognitive scientists have been aware of the way in which a specific bias (innate or acquired) is able to generalize “the right things” “on the right level of abstraction” from streams of experience.

In cognitive science, models are proposed to investigate possible mechanisms which capture the characteristics of human inductive learning. Recently, IP has also been used to construct these models or analyse these issues [34], with the assumption that many solutions that IP systems generate are equivalent to those solutions found by humans. For instance, Fig. 5 shows the result of learning the Tower of Hanoi problem induced by the IP system Igor2. This result is equivalent to the generalization from three disc to n disc problems (some) humans would infer from these three examples [19].

Besides puzzles like the Tower of Hanoi, IP can be applied to model how abstract problems, such as number series continuation (found in many IQ tests) or other IQ test problems [25] are solved. For instance, Igor2 was recently applied to number series [12] such as 3, 7, 15, 31, 63, leading to solutions such as $f(n) = f(n-1) + 1$. IP solutions can be used to evaluate the difficulty of each problem and really check whether there are competing hypotheses of similar complexity. Overall, many possibilities present themselves for the use of IP as a cognitive tool, especially at the knowledge level [34].

4. RECENT TECHNIQUES

IP is essentially a search problem, and can benefit from techniques developed in various communities. We present below certain classes of techniques frequently used in recent IP work.

4.1 DSL synthesizers

Domain-specific languages (DSL) have been introduced in the IP scenario under the following methodology:

1. Problem Definition: Identify a vertical domain of tasks and collect common scenarios by studying help forums and conducting user studies.
2. Domain-specific language (DSL): Design a DSL which is expressive enough to capture several real-world tasks in the domain, but also restricted enough to enable efficient learning from examples.
3. Synthesis Algorithm: Most of these algorithms work by systematically reducing the problem to the synthesis of sub-expressions of the original expression (by translating the examples for the expression to the examples for the sub-expressions). These algorithms typically end up computing a set of DSL programs.
4. Ranking: Rank the various programs returned by the synthesizer perhaps using machine learning techniques.

The above methodology has been applied to various domains including syntactic string transformations [7], semantic string transformations, number transformations, and table transformations [9].

4.2 Meta-synthesis frameworks

Developing synthesizers which are specific to a DSL offers several advantages, but developing a synthesizer requires thinking and implementation, and the underlying DSL is not easily extensible (since any changes to the DSL might require making non-trivial changes to the synthesizer algorithm).

A meta-synthesis framework allows easy development of synthesizers for a related family of DSLs which supports a common user interaction model, with the following steps:

1. Identify a family of vertical task domains which allow a common user interaction model.
2. Design an algebra for DSLs. A DSL is an ordered set of grammar rules (to model ranking).
3. Design a search algorithm for each algebra operator such that it is compositional and inductive.

Meta-synthesis frameworks can allow synthesizer writers to easily develop domain-specific synthesizers, similar to how declarative parsing frameworks allow a compiler writer to
4.3 Higher-order functions

Higher-order functions are a possibility to give a bias when searching for hypotheses. One of the first systems which made use of higher-order functions in IP was MAGIC-HASKELL [15], which generates Haskell functions from a small set of positive inputs realized as instantiation of the generic higher-order function fold. An extension of the analytical IP system Igor2, also implemented in Haskell, allows to identify catamorphic relations in the input/output examples and use this information to instantiate a higher function. The argument function to instantiate the higher-order function is either picked from background knowledge or, if not existing, is invented as an auxiliary function. The use of this technique results not only in a speed-up of synthesis but also enlarges the scope of synthesizable programs [13]. An example for induction with higher-order functions in given in Fig. 6. Finally, Henderson [10] proposed to use higher-order for cumulative learning where functions induced from examples are abstracted and can then be used to induce more complex programs.

4.4 Meta-interpretive learning

Meta-Interpretive Learning (MIL) is a recent ILP technique [29, 28] aimed at supporting learning of recursion and invention of auxiliary predicates. The approach 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. To illustrate the idea consider the meta-rule below.

When attempting to prove a particular example, the meta-interpretive learner builds a set of higher-order substitutions for meta-rules, such as the Chain rule above, which allows the proof to complete. Given the higher-order substitutions, instantiated program clauses can be reconstructed and re-used in later proofs, allowing a form of IP which supports the automatic construction of a hierarchically defined program.

In [23] the authors applied MIL to a task involving string transformations tasks previously studied by Gulwani [7]. Fig. 7 shows the outcome of applying MIL to learning a set of such tasks, using two approaches, dependent and independent learning, where in the former new definitions are allowed to call already learned definitions at lower levels. Also, in dependent learning a size bound restriction for the programs is progressively relaxed. The approach is analogous to a human programmer’s use of bottom-up programming.

4.5 New kinds of brute-force search

The general idea here is to systematically explore the entire state space of artifacts and check the correctness of each candidate against the given examples. This approach works relatively well when the specification consists of examples (as opposed to a formal relational specification) since checking the correctness of a candidate solution against examples can be done much faster than validating the correctness against a formal relational specification. However, this is easier said than done and often requires innovative nontrivial op-
Applying IP to new domains efficiently will also require new approaches, including the creation of meta-synthesizers as mentioned above. Because the application of IP techniques in real-world applications is relatively new, there is insufficient experience in exploring the space of applications to clearly identify common patterns that might arise across domains. It is likely that in the short term, domain-specific IP systems will be developed in an ad hoc way, and which over time, as experience with such systems grows, new approaches will systematize and formalize the ad hoc practices, so systems become more general and reusable across different domains.

5.3 Validation
The act of programming requires the programmer to have confidence in the correctness of the resulting program, and the same holds for end-user programming. It is important that the artifacts produced by IP give the end-user confidence that what they have created is correct and makes sense. For instance, the plethora of automatically named hierarchy of invented sub-tasks generated by approaches such as Meta-Interpretive Learning (Section 4.4) can lead to confusion if the new names do not bear a clear correspondence to the semantics of the sub-tasks being defined. To address these kinds of challenge, we must find new approaches to explain the resulting program to the user in intuitive terms and find ways for them to guide the solution if it is incorrect.

5.4 Noise tolerance
Real data inevitably contains missing and/or incorrect values, occurs in multiple formats, such as representations for dates and numbers, or even the background knowledge can be incorrect if the user accidently makes mistakes in providing them.

Addressing the issue of robustness to noise may be best done in a domain-specific manner. For example, if a table contains mostly correct data with a few outliers, existing techniques to detect and report outliers (or even just missing values) will help the IP process. Fortunately, there is a body of work in the existing ML literature which can be applied to this problem.

6. CONCLUSION
In the second half of the last century basic research in IFP and ILP resulted in the development of fundamental algorithms tackling the problem of inducing programs from input/output examples. However, these approaches remained within the context of artificial intelligence research and did not trigger a successful transfer into technologies applicable in a wider context. In 2009 Tessa Lau proclaimed programming by demonstration has failed [20]. In this paper we have presented recent work in IP which shows this claim might no longer be sustainable: learning from very few positive examples becomes possible when users and systems share background knowledge which can be represented in a declarative and thereby human-understandable way; merging algorithmic techniques developed in IFP and IFP as well as the use of higher-order functions and meta-interpretative learn-
ing resulted in more powerful IP algorithms; joint research on general principles of inductive learning in cognitive psychology and IP results which are similar to those humans would find and therefore are easier to understand and revise; the adoption by the general IP algorithms of techniques based on domain-specific languages has allowed the realization of technologies which are ready to use in mass market products as demonstrated by FlashFill. Hopefully, the recent achievements will attract more researchers from the different areas in which IP originated – AI, machine learning, functional programming, ILP, automated software engineering, and cognitive science – to tackle the challenge of bringing IP from the lab into the real world.

7. REFERENCES
