Machine learning manuscript No.
(will be inserted by the editor)

Inductive logic programming at 30

Andrew Cropper · Sebastijan Dumančić · Richard Evans · Stephen H. Muggleton

the date of receipt and acceptance should be inserted later

Abstract Inductive logic programming (ILP) is a form of logic-based machine learning. The goal of ILP is to induce a hypothesis (a logic program) that generalises given training examples and background knowledge. As ILP turns 30, we survey recent work in the field. In this survey, we focus on (i) new meta-level search methods, (ii) techniques for learning recursive programs that generalise from few examples, (iii) new approaches for predicate invention, and (iv) the use of different technologies, notably answer set programming and neural networks. We conclude by discussing some of the current limitations of ILP and discuss directions for future research.

1 Introduction

Inductive logic programming (ILP) [76, 79] is a form of machine learning (ML). As with other forms of ML, the goal of ILP is to induce a hypothesis that generalises training examples. However, whereas most forms of ML use vectors/tensors to represent data (examples and hypotheses), ILP uses logic programs (sets of logical rules). Moreover, whereas most forms of ML learn functions, ILP learns relations.

To illustrate ILP[1] suppose you want to learn a string transformation program from the following examples.

A. Cropper
University of Oxford
E-mail: andrew.cropper@cs.ox.ac.uk

S. Dumančić
KU Leuven
E-mail: sebastijan.dumancic@cs.kuleuven.be

R. Evans
Imperial College London
E-mail: richardevans@google.com

S. H. Muggleton
Imperial College London
E-mail: s.muggleton@imperial.ac.uk

[1] We do not introduce ILP in detail and refer the reader to the introductory paper of Cropper and Dumancic [17] or the textbooks of Nienhuys-Cheng and Wolf [89] and De Raedt [29].

Input	Output
inductive	e
logic	c
programming	g

Most forms of ML would represent these examples as a table, where each row would be an example and each column would be a feature, such as a one-hot-encoding representation of the string. By contrast, in ILP, we would represent these examples as logical atoms, such as `f([i,n,d,u,c,t,i,v,e], e)`, where `f` is the target predicate that we want to learn (the relation to generalise). We would also provide auxiliary information (features) in the form of background knowledge (BK), also represented as a logical theory (a logic program). For instance, for the string transformation problem, we could provide BK that contains logical definitions for string operations, such as `empty(A)`, which holds when the list `A` is empty; `head(A,B)`, which holds when `B` is the head of the list `A`; and `tail(A,B)`, which holds when `B` is the tail of the list `A`. Given the aforementioned examples and BK, an ILP system could induce the hypothesis (a logic program):

```
f(A,B):- tail(A,C),empty(C),head(A,B).
f(A,B):- tail(A,C),f(C,B).
```

Each line of the program is a rule. The first rule says that the relation `f(A,B)` holds when the three literals `tail(A,C)`, `empty(C)`, and `head(A,B)` hold. In other words, the first rule says that `B` is the last element of `A` when the tail of `A` is empty and `B` is the head of `A`. The second rule is recursive and says that the relation `f(A,B)` holds when the two literals `tail(A,C)` and `f(C,B)` hold. In other words, the second rule says that `f(A,B)` holds when the same relation holds for the tail of `A`.

1.1 Why ILP?

Compared to most ML approaches, ILP has several attractive features [25, 17]:

Data efficiency. Many forms of ML are notorious for their inability to generalise from small numbers of training examples, notably deep learning [71, 13]. As Evans and Grefenstette [40] point out, if we train a neural system to add numbers with 10 digits, it might generalise to numbers with 20 digits, but when tested on numbers with 100 digits, the predictive accuracy drastically decreases [95, 54]. By contrast, ILP can induce hypotheses from small numbers of examples, often from a single example [70, 84].

Background knowledge. ILP learns using BK represented as a logic program. Using logic programs to represent data allows ILP to learn with complex relational information, such as constraints about causal networks [51], the axioms of the event calculus when learning to recognise events [56, 57], and using a theory of light to understand images [84]. Moreover, because hypotheses are symbolic, hypotheses can be added the to BK, and thus ILP systems naturally support lifelong and transfer learning [70, 15, 16].

Expressivity. Because of the expressivity of logic programs, ILP can learn complex relational theories, such as cellular automata [52, 41], event calculus theories [56, 57], Petri nets [5], and general algorithms [19]. Because of the symbolic nature of logic programs, ILP can reason about hypotheses, which allows it to learn *optimal* programs, such as minimal time-complexity programs [22] and secure access control policies [66].

Expainability. Because of logic's similarity to natural language, logic programs can be easily read by humans, which is crucial for explainable AI. For instance, Muggleton et al [88] provide the first demonstration of *ultra-strong ML* [74], where a learned hypothesis is expected to not only be accurate but to also demonstrably improve the performance of a human when provided with the learned hypothesis.

1.2 Recent advances

Some of the aforementioned advantages come from recent developments, which we survey in this paper[2]. To aid the reader, we coarsely compare old and new ILP systems, where new represents systems from the past decade. We use FOIL [93], Progol [77], TILDE [9], and HYPER [12] as representative old systems and ILASP [63], Metagol [21], ∂ILP [40], and Popper [19] as representative new systems. This comparison, shown in Table 1, is, of course, vastly oversimplified, and there are many exceptions. In the rest of this paper, we survey these developments (each row in the table) in turn. After discussing these new ideas, we discuss recent application areas (Section 8) before concluding by proposing directions for future research.

	Old ILP	**New ILP**
Search method	Top-down and Bottom-up	Meta-level
Recursion	Limited	Yes
Predicate invention	No	Limited
Hypotheses	First-order	Higher-order, ASP
Optimality	No	Yes
Technology	Prolog	Prolog, ASP, NNs

Table 1 A simplified comparison of old and new ILP systems.

2 Search methods

The fundamental ILP problem is to efficiently search a large hypothesis space. Most older ILP approaches search in either a *top-down* or *bottom-up* fashion. These methods rely on notions of generality (typically using theta-subsumption [92]), where one program is more *general* or more *specific* than another. A third new search approach has recently emerged called *meta-level* ILP [51, 87, 50, 67, 19]. We discuss these approaches in turn.

2.1 Top-down and bottom-up

Top-down approaches [93, 9, 12] start with a general hypothesis and then specialise it. HYPER, for instance, searches a tree in which the nodes correspond to hypotheses and each child of a hypothesis in the tree is more specific than or equal to its predecessor

[2] This paper extends the paper of Cropper et al [25].

in terms of theta-subsumption. An advantage of top-down approaches is that they can often learn recursive programs (although not all do). A disadvantage is that they can be prohibitively inefficient because they can generate many hypotheses that do not cover the examples.

Bottom-up approaches, by contrast, start with the examples and generalise them [75, 78, 80, 52]. For instance, Golem [80] generalises pairs of examples based on relative least-general generalisation [89]. Bottom-up approaches can be seen as being *data-* or *example-driven*. An advantage of these approaches is that they are typically fast. As Bratko [12] points out, disadvantages include (i) they typically use unnecessarily long hypotheses with many clauses, (ii) it is difficult for them to learn recursive hypotheses and multiple predicates simultaneously, and (iii) they do not easily support predicate invention.

Progol [77], which inspired many other ILP approaches [106, 94, 1, 104], combines both top-down and bottom-up approaches. Starting with an empty program, Progol picks an uncovered positive example to generalise. To generalise an example, Progol uses mode declarations to build the *bottom clause* [77], the logically most-specific clause that explains the example. The bottom clause bounds the search from below (the bottom clause) and above (the empty set). Progol then uses an A* algorithm to generalise the bottom clause in a top-down (general-to-specific) manner and uses the other examples to guide the search.

2.1.1 Meta-level

Top-down and bottom-up approaches refine and revise a single hypothesis. A third approach has recently emerged called *meta-level* ILP [51, 87, 50, 67, 19, 90]. There is no standard definition for *meta-level ILP*. Most approaches encode the ILP problem as a meta-level logic program, i.e. a program that reasons about programs. Meta-level approaches then often delegate the search for a hypothesis to an off-the-shelf solver [14, 21, 63, 55, 104, 41, 19] after which the meta-level solution is translated back to a standard solution for the ILP task. In other words, instead of writing a procedure to search in a top-down or bottom-up manner, most meta-level approaches formulate the learning problem as a declarative search problem. For instance, ASPAL [14] translates an ILP task into a meta-level ASP program which describes every example and every possible rule in the hypothesis space. ASPAL then delegates the search to an ASP system to find a subset of the rules that covers all the positive but none of the negative examples.

The main advantage of meta-level approaches is that they can more easily learn recursive programs and optimal programs [14, 63, 21, 55, 41, 19], which we discuss in Sections 3 and 6 respectively. Moreover, whereas classical ILP systems were almost entirely based on Prolog, meta-level approaches use diverse techniques and technologies, such as ASP solvers [14, 63, 55, 19, 41], which we expand on in Section 7. The development of meta-level ILP approaches has, therefore, diversified ILP from the standard clause refinement approach of earlier ILP systems.

Most meta-level approaches encode the ILP learning task as a single static meta-level program [14, 63, 55, 41]. A major issue with this approach is that the meta-level program can be very large so these approaches can struggle to scale to problems with non-trivial domains and to programs with large clauses. Two related approaches try to overcome this limitation by continually revising the meta-level program.

ILASP3 [62] employs a counter-example-driven select-and-constrain loop. ILASP3 first pre-computes every clause in the hypothesis space defined by a set of given mode

declarations [77]. ILASP3 then starts its select-and-constrain loop. With each iteration, ILASP3 uses an ASP solver to find the best hypothesis (a subset of the rules) it can. If the hypothesis does not cover one of the examples, ILASP3 finds a reason why and then generates constraints (boolean formulas over the rules) which it adds to the meta-level program to guide subsequent search. Another way of viewing ILASP3 is that it uses a counter-example-guided approach and translates an uncovered example e into a constraint that is satisfied if and only if e is covered.

Popper [19] adopts a similar approach but differs in that it (i) does not precompute every possible rule in the hypothesis space, and (ii) translates a *hypothesis* into a set of constraints, rather than an uncovered example. Popper works in three repeating stages: *generate*, *test*, and *constrain*. Popper first constructs a meta-level logic program where its models correspond to hypotheses. In the generate stage, Popper asks an ASP solver to find a model (a hypothesis). In the test stage, Popper tests the hypothesis against the examples. A hypothesis *fails* when it is incomplete (does not entail all the positive examples) or inconsistent (entails a negative example). If a hypothesis fails, Popper learns constraints from the failure, which it then uses to restrict subsequent generate stages. For instance, if a hypothesis is inconsistent, then Popper generates a generalisation constraint to prune all generalisations of the hypothesis and adds the constraint to the meta-level program, which eliminates models and thus prunes the hypothesis space. This process repeats until Popper finds a complete and consistent program.

For more information about meta-level learning, we suggest the work of Inoue [50] and Law et al [67].

3 Recursion

Learning recursive programs has long been considered a difficult problem for ILP [83, 17]. The power of recursion is that an infinite number of computations can be described by a finite recursive program [111]. To illustrate the importance of recursion, reconsider the string transformation problem from the introduction. Without recursion, an ILP system would need to learn a separate clause to find the last element for each list of length n, such as this program for when $n = 3$:

```
f(A,B):- tail(A,C),empty(C),head(A,B).
f(A,B):- tail(A,C),tail(C,D),empty(D),head(C,B).
f(A,B):- tail(A,C),tail(C,D),tail(D,E),empty(E),head(D,B).
```

This program does not generalise to lists of arbitrary lengths. Moreover, most ILP systems would need examples of lists of each length to learn such a program. By contrast, an ILP system that supports recursion can learn the compact program:

```
f(A,B):- tail(A,C),empty(C),head(A,B).
f(A,B):- tail(A,C),f(C,B).
```

Because of the symbolic representation and the recursive nature, this program generalises to lists of arbitrary length and which contain arbitrary elements (e.g. integers and characters). In general, without recursion, it can be difficult for an ILP system to generalise from small numbers of examples [24].

Older ILP systems struggle to learn recursive programs, especially from small numbers of training examples. A common limitation with existing approaches is that they

rely on *bottom clause* construction [77]. In this approach, for each example, an ILP system creates the most specific clause that entails the example, and then tries to generalise the clause to entail other examples. However, this sequential covering approach requires examples of both the base and inductive cases.

Interest in recursion has resurged with the introduction of meta-interpretive learning (MIL) [86, 87, 27] and the MIL system Metagol [21]. The key idea of MIL is to use *metarules* [23], or program templates, to restrict the form of inducible programs, and thus the hypothesis space[3]. A metarule is a higher-order clause. For instance, the *chain* metarule is $P(A,B) \leftarrow Q(A,C),R(C,B)$, where the letters P, Q, and R denote higher-order variables and A, B and C denote first-order variables. The goal of a MIL system, such as Metagol, is to find substitutions for the higher-order variables. For instance, the *chain* metarule allows Metagol to induce programs such as `f(A,B):- tail(A,C),head(C,B)`[4]. Metagol induces recursive programs using recursive metarules, such as the *tailrec* metarule *P(A,B)* ← *Q(A,C), P(C,B)*.

Following MIL, many meta-level ILP systems can learn recursive programs [63, 40, 55, 19]. With recursion, ILP systems can now generalise from small numbers of examples, often a single example [70]. Moreover, the ability to learn recursive programs has opened up ILP to new application areas, including learning string transformations programs [70], answer set grammars [65], and general algorithms [19].

4 Predicate invention

A key characteristic of ILP is the use of BK. BK is similar to features used in most forms of ML. However, whereas features are tables, BK contains facts and rules (extensional and intensional definitions) in the form of a logic program. For instance, when learning string transformation programs, we may provide helper background relations, such as `head/2` and `tail/2`. For other domains, we may supply more complex BK, such as a theory of light to understand images [84] or higher-order operations, such as `map/3`, `filter/3`, and `fold/4`, to solve programming puzzles [27].

As with choosing appropriate features, choosing appropriate BK is crucial for good learning performance. ILP has traditionally relied on hand-crafted BK, often designed by domain experts. This approach is limited because obtaining suitable BK can be difficult and expensive. Indeed, the over-reliance on hand-crafted BK is a common criticism of ILP [40].

Rather than expecting a user to provide all the necessary BK, the goal of *predicate invention* (PI) [78, 108] is for an ILP system to automatically invent new auxiliary predicate symbols. This idea is similar to when humans create new functions when manually writing programs, as to reduce code duplication or to improve readability. Whilst PI has attracted interest since the beginnings of ILP [78], and has subsequently been repeatedly stated as a major challenge [59, 83, 61], most ILP systems do not support it.

A key challenge faced by early ILP systems was deciding when and how to invent a new symbol. As Kramer [60] points out, PI is difficult because it is unclear how many

[3] The idea of using metarules to restrict the hypothesis space has been widely adopted by many approaches [110, 3, 100, 40, 5, 55]. However, despite their now widespread use, there is little work determining which metarules to use for a given learning task ([23] is an exception), which future work must address.

[4] Metagol can induce longer clauses though predicate invention, which is described in Section 4.

arguments an invented predicate should have, how the arguments should be ordered, etc. Several PI approaches try to address this challenge, which we discuss in turn.

4.1 Placeholders

A classical approach to PI is to predefine invented symbols through mode declarations, which Leban et al [68] call *placeholders*. However, this placeholder approach is limited because it requires that a user manually specify the arity and argument types of a symbol [63], which rather defeats the point, or requires generating all possible invented predicates [40, 41], which is computationally expensive.

4.2 Metarules

Interest in *automatic* PI (where a user does not need to predefine an invented symbol) has resurged with the introduction of MIL. MIL avoids the issues of older ILP systems by using metarules to define the hypothesis space and in turn reduce the complexity of inventing a new predicate symbol. For instance, the *chain* metarule ($P(A,B) \leftarrow Q(A,C),R(C,B)$) allows Metagol to induce programs such as `f(A,B):- tail(A,C),tail(C,D)`, which would drop the first two elements from a list. To induce longer clauses, such as to drop first three elements from a list, Metagol uses the same metarule but invents a predicate symbol to chain their application, such as to induce the program:

```
f(A,B):- tail(A,C),inv1(C,B).
inv1(A,B):- tail(A,C),tail(C,B).
```

To learn this program, Metagol invents the predicate symbol `inv1` and induces a definition for it using the *chain* metarule. Metagol uses this new predicate symbol in the definition for the target predicate `f`.

A side-effect of this metarule-driven approach is that problems are forced to be decomposed into reusable solutions. For instance, to learn a program that drops the first four elements of a list, Metagol learns the following program, where the invented predicate symbol `inv1` is used twice:

```
f(A,B):- inv1(A,C),inv1(C,B).
inv1(A,B):- tail(A,C),tail(C,B).
```

PI has been shown to help reduce the size of target programs, which in turns reduces sample complexity and improves predictive accuracy [15]. Several new ILP systems support PI using a metarule-guided approach [40, 55, 48].

4.3 Pre/post-processing

Metarule-driven PI approaches perform PI during the learning task. A recent trend is to perform PI as a pre- or post-processing step to improve knowledge representation [37, 38, 15, 48].

CUR^{2}LED [37] performs PI by clustering constants and relations in the provided BK, turning each identified cluster into a new BK predicate. The key insight of CUR^{2}LED is

not to use a single similarity measure, but rather a set of various similarities. This choice is motivated by the fact that different similarities are useful for different tasks, but in the unsupervised setting the task itself is not known in advance. CUR^{2}LED performs PI by producing different clusterings according to the features of the objects, community structure, and so on.

ALPs [38] perform PI using an auto-encoding principle: they learn an *encoding* logic program that maps the provided data to a new, compressive latent representation (defined in terms of the invented predicates), and a *decoding* logic program that can reconstruct the provided data from its latent representation. This approach shows improved performance on supervised tasks, even though the PI step is task-agnostic.

Knorf [36] pushes the idea of ALPs even further. Knorf compresses a program by removing redundancies in it. If the learnt program contains invented predicates, Knorf revises them and introduces new ones that would lead to a smaller program. The refactored program is smaller in size and contains less redundancy in clauses, both of which lead to improved performance. The authors experimentally demonstrate that refactoring improves learning performance in lifelong learning and that Knorf substantially reduces the size of the BK program, reducing the number of literals in a program by 50% or more.

4.4 Lifelong Learning

An approach to acquiring BK is to learn it in a lifelong learning setting. The general idea is to reuse knowledge gained from solving one problem to help solve a different problem.

Metagol$_{DF}$ is an ILP system [70] which given a set of tasks, uses Metagol to try to learn a solution for each task using at most one clause. If Metagol finds a solution for a task, it adds the solution to the BK and removes the task from the set. Metagol$_{DF}$ then asks Metagol to find solutions for the rest of the tasks but can now (i) use an additional clause, and (ii) reuse solutions from previously solved tasks. This process repeats until Metagol$_{DF}$ solves all the tasks or reaches a maximum program size. In this approach, Metagol$_{DF}$ automatically identifies easier problems, learn programs for them, and then reuses the solutions to help learn programs for more difficult problems. The authors experimentally show that their multi-task approach performs substantially better than a single-task approach because learned programs are frequently reused and leads to a hierarchy of induced programs.

Metagol$_{DF}$ saves all learned programs (including invented predicates) to the BK, which can be problematic because too much irrelevant BK is detrimental to learning performance [107]. To address this problem, Forgetgol [16] introduces the idea of *forgetting*. In this approach, Forgetgol continually grows and shrinks its hypothesis space by adding and removing learned programs to and from its BK. The authors show that forgetting can reduce both the size of the hypothesis space and the sample complexity of an ILP learner when learning from many tasks.

4.5 Limitations

The aforementioned techniques have improved the ability of ILP to invent high-level concepts. However, PI is still difficult and there are many challenges to overcome. The challenges are that (i) many systems struggle to perform PI at all, and (ii) those that do

support PI mostly need much user-guidance, metarules to restrict the space of invented symbols or that a user specifies the arity and argument types of invented symbols.

By developing better approaches for PI, we can make progress on existing challenging problems. For instance, in *inductive general game playing* [26], the task is to learn the symbolic rules of games from observations of gameplay, such as learning the rules of *connect four*. The target solutions, which come from the general game playing competition [45], often contain auxiliary predicates. For instance, the rules for *connect four* are defined in terms of definitions for lines which are themselves defined in terms of columns, rows, and diagonals. Although these auxiliary predicates are not strictly necessary to learn the target solution, inventing such predicates significantly reduces the size of the solution, which in turns makes them easier to learn. Although new methods for PI can invent high-level concepts, they are not yet sufficiently powerful enough to perform well on the IGGP dataset. Making progress in this area would constitute a major advancement in ILP.

5 Hypotheses

ILP systems have traditionally induced definite and normal logic programs, typically represented as Prolog programs. A recent development has been to use different hypothesis representations.

5.1 Datalog

Datalog is a syntactical subset of Prolog which disallows complex terms as arguments of predicates and imposes restrictions on the use of negation. Datalog is a truly declarative language, whereas in Prolog reordering clauses can change the program. Moreover, Datalog query is guaranteed to terminate, though this guarantee is at the expense of not being a Turing-complete language, which Prolog is. Several works [3, 40, 55] induce Datalog programs. The general motivation for reducing the expressivity of the representation language from Prolog to Datalog is to allow the problem to be encoded as a satisfiability problem, particularly to leverage recent developments in SAT and SMT. We discuss the advantages of this approach more in Section 7.1.

5.2 Answer Set Programming

ASP [43] is a logic programming paradigm based on the stable model semantics of normal logic programs that can be implemented using the latest advances in SAT solving technology. Law et al [64] discuss some of the advantages of learning ASP programs, rather than Prolog programs, which we reiterate. When learning Prolog programs, the procedural aspect of SLD-resolution must be taken into account. For instance, when learning Prolog programs with negation, programs must be stratified; otherwise program may loop under certain conditions. By contrast, as ASP is a truly declarative language, no such consideration need be taken into account when learning ASP programs. Compared to Datalog and Prolog, ASP supports addition language constructs, such as disjunction in the head of a clause, choice rules, and hard and weak constraints. A key difference between ASP and Prolog is semantics. A definite logic program has only one model (the

least Herbrand model). By contrast, an ASP program can have one, many, or even no stable models (answer sets). Due to its non-monotonicity, ASP is particularly useful for expressing common-sense reasoning [62].

To illustrate the benefits of learning ASP programs, we reuse an example from Law et al [67]. Given a sufficient examples of Hamiltonian graphs, ILASP [63] can learn a program to definite them:

```
0  in(V0, V1)  1 :- edge(V0, V1).
reach(V0) :- in(1, V0).
reach(V1) :- reach(V0), in(V0, V1).
:- not reach(V0), node(V0).
:- V1 != V2, in(V0, V2), in(V0, V1).
```

This program illustrates useful language features of ASP. The first rule is a *choice* rule and the last two rules are *hard constraints*.

Approaches to learning ASP programs can mostly be divided into two categories: *brave learners*, which aim to learn a program such that at least one answer set covers the examples, and *cautious learners*, which aim to find a program which covers the examples in all answer sets. ILASP is notable because it supports both brave and cautious learning, which are both needed to learn some ASP programs [64]. Moreover, ILASP differs from most Prolog-based ILP systems because it learns unstratified ASP programs, including programs with normal rules, choice rules, and both hard and weak constraints, which classical ILP systems cannot. Learning ASP programs allows for ILP to be used for new problems, such as inducing answer set grammars [65].

5.3 Higher-order programs

Imagine learning a *droplasts* program, which removes the last element of each sublist in a list, e.g. *[alice,bob,carol]* $\mapsto$ *[alic,bo,caro]*. Given suitable input data, Metagol can learn this first-order recursive program:

```
f(A,B):- empty(A),empty(B).
f(A,B):- head(A,C),tail(A,D),head(B,E),tail(B,F),f1(C,E),f(D,F).
f1(A,B):- reverse(A,C),tail(C,D),reverse(D,B).
```

Although semantically correct, the program is verbose. To learn smaller programs, Metagol$_{ho}$ [27] extends Metagol to support learning higher-order programs, where predicate symbols can be used as terms. For instance, for the same *droplasts* problem, Metagol$_{ho}$ learns the higher-order program:

```
f(A,B):- map(A,B,f1).
f1(A,B):- reverse(A,C),tail(C,D),reverse(D,B).
```

To learn this program, Metagol$_{ho}$ invents the predicate symbol `f1`, which is used twice in the program: as term in the `map(A,B,f1)` literal and as a predicate symbol in the `f1(A,B)` literal. Compared to the first-order program, this higher-order program is smaller because it uses `map/3` (predefined in the BK) to abstract away the manipulation of the list and to avoid the need to learn an explicitly recursive program (recursion is implicit in `map/3`). Metagol$_{ho}$ has been shown to reduce sample complexity and learning times and improve predictive accuracies [27].

5.4 Probabilistic logic programs

A major limitation of logical representations, such as Prolog and its derivatives, is the implicit assumption that the BK is perfect. That is, most ILP systems assume that atoms are true or false, leaving no room for uncertainty. This assumption is problematic if data is noisy, which is often the case.

Integrating probabilistic reasoning into logical representations is a principled way to handle such uncertainty in data. This integration is the focus of statistical relational artificial intelligence (StarAI) [30, 33]. In essence, StarAI hypothesis representations extend BK with probabilities or weights indicating the degree of confidence in the correctness of parts of BK. Generally, StarAI techniques can be divided in two groups: *distribution representations* and *maximum entropy* approaches.

Distribution semantics approaches [102], including Problog [31] and PRISM [103], explicitly annotate uncertainties in BK. To allow such annotation, they extend Prolog with two primitives for stochastic execution: probabilistic facts and annotated disjunctions. Probabilistic facts are the most basic stochastic primitive and they take the form of logical facts labelled with a probability p. Each probabilistic fact represents a Boolean random variable that is `true` with probability p and `false` with probability $1-p$. For instance, the following probabilistic fact states that there is 1% chance of an earthquake in Naples.

```
0.01::earthquake(naples).
```

An alternative interpretation of this statement is that 1% of executions of the probabilistic program would observe an earthquake. The second type of stochastic primitive is an annotated disjunction. Whereas probabilistic facts introduce non-deterministic behaviour on the level of facts, annotated disjunctions introduce non-determinism on the level of clauses. Annotated disjunctions allow for multiple literals in the head, where only one of the head literals can be `true` at a time. For instance, the following annotated disjunction states that a ball can be either green, red, or blue, but not a combination of colours:

```
1/3::colour(B,green); 1/3::colour(B,red); 1/3::colour(B,blue) :- ball(B).
```

By contrast, maximum entropy approaches annotate uncertainties only at the level of a logical theory. That is, they assume that the predicates in the BK are labelled as either true or false, but the label may be incorrect. These approaches are not based on logic programming, but rather on first-order logic. Consequently, the underlying semantics are different: rather than consider proofs, these approaches consider models or groundings of a theory. This difference primarily changes what uncertainties represent. For instance, Markov Logic Networks (MLN) [99] represent programs as a set of weighted clauses. The weights in MLN do not correspond to probabilities of a formula being true but, intuitively, to a log odds between a possible world (an interpretation) where the clause is true and a world where the clause is false. For instance, a clause that is true in 80% of the worlds would have a weight of 1.386 ($\log \frac{0.8}{0.2}$)

The techniques from learning such probabilistic programs are typically direct extensions of ILP techniques. For instance, ProbFOIL [32] extends FOIL [93] with probabilistic clauses. Similarly, SLIPCOVER [8] is a bottom-up approach, similar to Aleph [106] and Progol [77]. Huynh and Mooney [49] use Aleph to find interesting clauses and then learn the corresponding weights. Kok and Domingos [58] use relational pathfinding over BK to identify useful clauses. That is, they interpret the BK as a hypergraph in which constants

form vertices and atoms form hyper-edges and perform random walks. Frequently occurring walks, or their subparts, are then turned into clauses. Such random walks could be seen as an approximate way to construct bottom clauses.

6 Optimality

There are often multiple (sometimes infinitely many) hypotheses that explain the data. Deciding which hypothesis to choose has long been a difficult problem. Older ILP systems were not guaranteed to induce optimal programs, where optimal typically means with respect to the size of the induced program or the coverage of examples. A key reason for this limitation was that most search techniques learned a single clause at a time, leading to the construction of sub-programs which were sub-optimal in terms of program size and coverage. For instance, programs induced by Aleph offer no guarantee of optimality with respect to the program size and coverage.

Newer ILP systems try to address this limitation. As with the ability to learn recursive programs, the main development is to take a global view of the induction task by using meta-level search techniques. In other words, rather than induce a single clause at a time from a single example, the idea is to induce multiple clauses from multiple examples. For instance, ILASP uses ASP's optimisation abilities to provably learn the program with the fewest literals.

The ability to learn optimal programs opens up ILP to new problems. For instance, learning efficient logic programs has long been considered a difficult problem in ILP [79, 83], mainly because there is no declarative difference between an efficient program, such as mergesort, and an inefficient program, such as bubble sort. To address this issue, Metaopt [22] extends Metagol to support learning efficient programs. Metaopt maintains a cost during the hypothesis search and uses this cost to prune the hypothesis space. To learn minimal time complexity logic programs, Metaopt minimises the number of resolution steps. For instance, imagine trying to learn a *find duplicate* program, which finds any duplicate element in a list e.g. *[p,r,o,g,r,a,m]* $\mapsto$ *r*, and *[i,n,d,u,c,t,i,o,n]* $\mapsto$ *i*. Given suitable input data, Metagol can induce the program:

```
f(A,B):- head(A,B),tail(A,C),element(C,B).
f(A,B):- tail(A,C),f(C,B).
```

This program goes through the elements of the list checking whether the same element exists in the rest of the list. Given the same input, Metaopt induces the program:

```
f(A,B):- mergesort(A,C),f1(C,B).
f1(A,B):- head(A,B),tail(A,C),head(C,B).
f1(A,B):- tail(A,C),f1(C,B).
```

This program first sorts the input list and then goes through the list to check whether for duplicate adjacent elements. Although larger, both in terms of clauses and literals, the program learned by Metaopt is more efficient $O(n \log n)$ than the program learned by Metagol $O(n^2)$. Metaopt has been shown to learn efficient robot strategies, efficient time complexity logic programs, and even efficient string transformation programs.

FastLAS [66] is an ASP-based ILP system that takes as input a custom scoring function and computes an optimal solution with respect to the given scoring function. The authors show that this approach allows a user to optimise domain-specific performance metrics on real-world datasets, such as access control policies.

7 Technologies

Older ILP systems mostly use Prolog for reasoning. Recent work considers using different technologies.

7.1 Constraint satisfaction and satisfiability

There have been tremendous recent advances in SAT [47]. To leverage these advances, much recent work in ILP uses related techniques, notably ASP [14, 86, 63, 56, 57, 104, 55, 41, 19]. The main motivations for using ASP are to leverage (i) the language benefits of ASP (Section 5.2), and (ii) the efficiency and optimisation techniques of modern ASP solvers, such as CLASP [44], which supports conflict propagation and learning. With similar motivations, other approaches encode the ILP problem as SAT [1] or SMT [3] problems. These approaches have been shown able to reduce learning times compared to standard Prolog-based approaches. However, some unresolved issues remain. A key issue is that most approaches encode an ILP problem as a single (often very large) satisfiability problem. These approaches therefore often struggle to scale to very large problems [27], although preliminary work attempts to tackle this issue [19].

7.2 Neural networks

With the rise of deep learning, several approaches have explored using gradient-based methods to learn logic programs. These approaches all replace discrete logical reasoning with a relaxed version that yields continuous values reflecting the confidence of the conclusion.

The various neural approaches can be characterised along four orthogonal dimensions. The first dimension is whether the neural network implements forward or backward inference. While some [100] use backward (goal-directed) chaining with a neural implementation of unification, most approaches [40, 112, 34] use forward chaining. The second dimension is whether the network is designed for big data problems [112, 100] or for data-efficient learning from a handful of data items [40]. Few neural systems to date are capable of handling both big data and small data, with the notable exception of [34]. The third dimension is whether the neural system jointly learns embeddings (mapping symbolic constants to continuous vectors) along with the logical rules [100]. The advantage of jointly learning embeddings is that it enables fuzzy unification between constants that are similar but not identical. The challenge for these approaches that jointly learn embeddings is how to generalize appropriately to constants that have not been seen at training time. The fourth dimension is whether or not the neural system is designed to allow explicit human-readable logical rules to be extracted from the weights of the network. While most neural ILP systems [112, 100, 40] do produce explicit logic programs, some [34] do not. It is perhaps moot whether implicit systems that do not produce explicit programs count as ILP systems at all – but note that even in the implicit neural systems, the weight sharing of the neural net is designed to achieve strong generalisation by performing the same computation on all tuples of objects.

Currently, most neural approaches to ILP require the use of metarules or templates to make the search space tractable, and fail to support predicate invention, recursion and abduction. This severely limits the applicability of these approaches, as the user

cannot always be expected to provide suitable an complete background knowledge and metarules for a new problem. The only approach that avoids the use of metarules or templates is Neural Logic Machines [34], and the only one to fully integrate neural net learning with predicate invention, recursion and abduction is Abductive Meta-Interpretive Learning [28].

8 Applications

We now survey recent application areas for ILP.

Scientific discovery. Perhaps the most prominent application of ILP is in scienfitic discovery. ILP has, for instance, been used to identify and predict ligands (substructures responsible for medical activity) [53] and infer missing pathways in protein signalling networks [51]. There has been much recent work on applying ILP in ecology [10, 109, 11]. For instance, Bohan et al [10] use ILP to generate plausible and testable hypotheses for trophic relations ('who eats whom') from ecological data.

Program analysis. Due to the expressivity of logic programs as a representation language, ILP systems have found successful applications in software design. ILP systems have proven effective in learning SQL queries [3, 105], programming language semantics [7], and code search [105].

Robotics. Robotics applications often require incorporating domain knowledge or imposing certain requirements on the learnt programs. For instance, The Robot Engineer [101] uses ILP to design tools for robot and even complete robots, which are tests in simulations and real-world environments. Metagol$_o$ [20] learns robot strategies considering their resource efficiency and Antanas et al [4] recognise graspable points on objects through relational representations of objects.

Vision. Background knowledge is also valuable in Computer Vision. In recent work Muggleton et al [85] demonstrated that *Logical Vision*, which employs MIL, can outperform state-of-the-art statistical machine learning in particular image recognition tasks, given general Newtonian physics background knowledge concerning reflection of light.

Games. Inducing game rules has a long history in ILP, where chess has often been the focus [82]. Legras et al [69] show that Aleph and TILDE can outperform an SVM learner in the game of Bridge. Law et al [63] use ILASP to induce the rules for Sudoku and show that this more expressive formalism allows for game rules to be expressed more compactly. Cropper et al [26] introduce the ILP problem of *inductive general game playing*: the problem of inducing game rules from observations, such as *Checkers*, *Sokoban*, and *Connect Four*. Muggleton and Hocquette [81] show the MIL system MIGO consistently outperforms deep reinforcement learning for both Noughts-and-Crosses and Hexapawn.

Data curation and transformation. Another successful application of ILP is in data curation and transformation, which is again largely because ILP can learn executable programs. The most prominent example of such tasks are string transformations, such as the example given in the introduction. There is much interest in this topic, largely due to success in synthesising programs for end-user problems, such as string transformations in Microsoft Excel [46]. String transformation have become a standard benchmark for recent ILP papers [70, 27, 15, 18]. Other transformation tasks include extracting values from semi-structured data (e.g. XML files or medical records), extracting relations from ecological papers, and spreadsheet manipulation [24].

Learning from trajectories. Learning from interpretation transitions (LFIT) [52] automatically constructs a model of the dynamics of a system from the observation of its state transitions. Given time-series data of discrete gene expression, it can learn gene interactions, thus allowing to explain and predict states changes over time [98]. LFIT has been applied to learn biological models, like Boolean Networks, under several semantics: memory-less deterministic systems [52, 96], and their multi-valued extensions [97, 72]. Martínez et al [72] combine LFIT with a reinforcement learning algorithm to learn probabilistic models with exogenous effects (effects not related to any action) from scratch. The learner was notably integrated in a robot to perform the task of clearing the tableware on a table. In this task external agents interacted, people brought new tableware continuously and the manipulator robot had to cooperate with mobile robots to take the tableware to the kitchen. The learner was able to learn a usable model in just five episodes of 30 action executions. Evans et al [41] apply the *Apperception Engine* to explain sequential data, such as cellular automata traces, rhythms and simple nursery tunes, image occlusion tasks, game dynamics, and sequence induction intelligence tests. Surprisingly, they show that their system can achieve human-level performance on the sequence induction intelligence tests in the zero-shot setting (without having been trained on lots of other examples of such tests, and without hand-engineered knowledge of the particular setting). At a high level, these systems take the unique selling point of ILP systems (the ability to strongly generalise from a handful of data), and apply it to the self-supervised setting, producing an explicit human-readable theory that explains the observed state transitions.

9 Summary and future work

In a survey paper from a decade ago, Muggleton et al [83] proposed directions for future research. In the decade since, there have been major advances on many of the topics, notably in predicate invention (Section 4), using higher-order logic as a representation language (Section 4.2) and to represent hypotheses (Section 5.3), and applications in learning actions and strategies (Section 8). Despite the advances, there are still many limitations in ILP that future work should address.

9.1 Limitations and future research

Better systems. Muggleton et al [83] argue that a problem with ILP is the lack of well-engineered tools. They state that whilst over 100 ILP systems have been built, less than a handful of systems can be meaningfully used by ILP researchers. In the decade since the

authors highlighted this problem, little progress has been made: most ILP systems are not easy to use. In other words, ILP systems are still notoriously difficult to use and you often need a PhD in ILP to use any of the tools. Even then, it is still often only the developers of a system that know how to properly use it. By contrast, driven by industry, other forms of ML now have reliable and well-maintained implementations, such as PyTorch and TensorFlow, which has helped drive research. A frustrating issue with ILP systems is that they use many different language biases or even different syntax for the same biases. For instance, the way of specifying a learning task in Progol, Aleph, TILDE, and ILASP varies considerably despite them all using mode declarations, If it is difficult for ILP researchers to use ILP tools, then what hope do non-ILP researchers have? For ILP to be more widely adopted both inside and outside of academia, we must develop more standardised, user-friendly, and better-engineered tools.

Language biases. As Cropper et al [25] state, one major issue with ILP is choosing an appropriate language bias. For instance, Metagol uses metarules (Section 4.2) to restrict the syntax of hypotheses and thus the hypothesis space. If a user can provide suitable metarules, then Metagol is extremely efficient. However, if a user cannot provide suitable metarules (which is often the case), then Metagol is almost useless. This same brittleness applies to ILP systems that employ mode declarations [77]. In theory, a user can provide very general mode declarations, such as only using a single type and allowing unlimited recall. In practice, however, weak mode declarations often lead to very poor performance. For good performance, users of mode-based systems often need to manually analyse a given learning task to tweak the mode declarations, often through a process of trial and error. Moreover, if a user makes a small mistake with a mode declaration, such as giving the wrong argument type, then the ILP system is unlikely to find a good solution. Even for ILP experts, determining a suitable language bias is often a frustrating and time-consuming process. We think the need for an almost perfect language bias is severely holding back ILP from being widely adopted. We think that an important direction for future work in ILP is to develop techniques for automatically identifying suitable language biases. Although there is some work on mode learning [73, 42, 91] and work on identifying suitable metarules [23], this area of research is largely under-researched.

Better datasets. Interesting problems, alongside usable systems, drive research and attract interest in a research field. This relationship is most evident in the deep learning community which has, over a decade, grown into the largest AI community. This community growth has been supported by the constant introduction of new problems, datasets, and well-engineered tools. Challenging problems that push the state-of-the-art to its limits are essential to sustain progress in the field; otherwise, the field risks stagnation through only small incremental progress. ILP has, unfortunately, failed to deliver on this front: most research is still evaluated on 20-year old datasets. Most new datasets that have been introduced often come from toy domains and are designed to test specific properties of the introduced technique. To an outsider, this sends a message that ILP is not applicable to real-world problems. We think that the ILP community should learn from the experiences of other AI communities and put significant efforts into developing datasets that identify limitations of existing methods as well as showcase potential applications of ILP.

Relevance. New methods for predicate invention (Section 4) have improved the abilities of ILP systems to learn large programs. Moreover, these techniques raise the potential

for ILP to be used in lifelong learning settings. However, inventing and acquiring new BK could lead to a problem of too much BK, which can overwhelm an ILP system [107, 16]. On this issue, a key under-explored topic is that of *relevancy*. Given a new induction problem with large amounts of BK, how does an ILP system decide which BK is relevant? One emerging technique is to train a neural network to score how relevant programs are in the BK and to then only use BK with the highest score to learn programs [6, 39]. However, the empirical efficacy of this approach has yet to be demonstrated. Moreover, these approaches have only been demonstrated on small amounts of BK and it is unclear how they scale to BK with thousands of relations. Without efficient methods of relevance identification, it is unclear how efficient lifelong learning can be achieved.

Handling mislabelled and ambiguous data. A major open question in ILP is how best to handle noisy and ambiguous data. Neural ILP systems [100, 40] are designed from the start to robustly handle mislabelled data. Although there has been work in recent years on designing ILP systems that can handle noisy mislabelled data, there is much less work on the even harder and more fundamental problem of designing ILP systems that can handle *raw ambiguous data*. ILP systems typically assume that the input has already been preprocessed into symbolic declarative form (typically, a set of ground atoms representing positive and negative examples). But real-world input does not arrive in symbolic form. Consider e.g. a robot with a video camera, where the raw input is a sequence of pixel images. Converting each pixel image into a set of ground atoms is a challenging non-trivial achievement that should not be taken for granted. For ILP systems to be widely applicable in the real world, they need to be redesigned so they can handle raw ambiguous input from the outset [40, 35].

Probabilistic ILP. Real-world data is often noisy and uncertain. Extending ILP to deal with such uncertainty substantially broadens its applicability. While StarAI is receiving growing attention, learning probabilistic programs from data is still largely under-investigated due to the complexity of joint probabilistic and logical inference. When working with probabilistic programs, we are interested in the probability that a program covers an example, not only whether the program covers the example. Consequently, probabilistic programs need to compute all possible derivations of an example, not just a single one. Despite added complexity, probabilistic ILP opens many new challenges. Most of the existing work on probabilistic ILP considers the minimal extension of ILP to the probabilistic setting, by assuming that either (i) BK facts are uncertain, or (ii) that learned clauses need to model uncertainty. These assumptions make it possible to separate structure from uncertainty and simply reuse existing ILP techniques. Following this minimal extension, the existing work focuses on discriminative learning in which the goal is to learn a program for a single target relation. However, a grand challenge in probabilistic programming is generative learning. That is, learning a program describing a generative process behind the data, not a single target relation. Learning generative programs is a significantly more challenging problem, which has received very little attention in probabilistic ILP.

Explainability. Explainability is one of the claimed advantages of a symbolic representation. Recent work [88, 2] evaluates the comprehensibility of ILP hypotheses using Michie's [74] framework of *ultra-strong machine learning*, where a learned hypothesis is expected to not only be accurate but to also demonstrably improve the performance of

a human being provided with the learned hypothesis. [88] empirically demonstrate improved human understanding directly through learned hypotheses. However, given the demonstration of both beneficial and harmful effects of explainability [2] more work is required to better understand the conditions under which this can be achieved, especially given the rise of PI.

9.2 Summary

As ILP approaches 30, we think that the recent advances surveyed in this paper have opened up new areas of research for ILP to explore. Moreover, we hope that the next decade sees developments on the numerous limitations we have discussed so that ILP can have a significant impact on AI.

References

1. Ahlgren J, Yuen SY (2013) Efficient program synthesis using constraint satisfaction in inductive logic programming. J Machine Learning Res 14(1):3649–3682
2. Ai L, Muggleton S, Hocquette C, Gromowski M, Schmid U (2020) Beneficial and harmful explanatory machine learning. Machine Learning In Press, available http://arxiv.org/abs/2009.06410
3. Albarghouthi A, Koutris P, Naik M, Smith C (2017) Constraint-based synthesis of datalog programs. In: Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Springer, Lecture Notes in Computer Science, vol 10416, pp 689–706
4. Antanas L, Moreno P, De Raedt L (2015) Relational kernel-based grasping with numerical features. In: Inductive Logic Programming - 25th International Conference, ILP 2015, Springer, Lecture Notes in Computer Science, vol 9575, pp 1–14
5. Bain M, Srinivasan A (2018) Identification of biological transition systems using meta-interpreted logic programs. Machine Learning 107(7):1171–1206
6. Balog M, Gaunt AL, Brockschmidt M, Nowozin S, Tarlow D (2017) Deepcoder: Learning to write programs. In: 5th International Conference on Learning Representations, ICLR 2017, OpenReview.net
7. Bartha S, Cheney J (2019) Towards meta-interpretive learning of programming language semantics. In: Inductive Logic Programming - 29th International Conference, ILP 2019, Springer, Lecture Notes in Computer Science, vol 11770, pp 16–25
8. Bellodi E, Riguzzi F (2015) Structure learning of probabilistic logic programs by searching the clause space. Theory Pract Log Program 15(2):169–212
9. Blockeel H, De Raedt L (1998) Top-down induction of first-order logical decision trees. Artif Intell 101(1-2):285–297
10. Bohan DA, Caron-Lormier G, Muggleton S, Raybould A, Tamaddoni-Nezhad A (2011) Automated discovery of food webs from ecological data using logic-based machine learning. PLoS One 6(12):e29,028
11. Bohan DA, Vacher C, Tamaddoni-Nezhad A, Raybould A, Dumbrell AJ, Woodward G (2017) Next-generation global biomonitoring: large-scale, automated reconstruction of ecological networks. Trends in Ecology & Evolution 32(7):477–487

12. Bratko I (1999) Refining complete hypotheses in ILP. In: Inductive Logic Programming, 9th International Workshop, ILP-99, Springer, Lecture Notes in Computer Science, vol 1634, pp 44–55
13. Chollet F (2019) On the measure of intelligence. CoRR abs/1911.01547
14. Corapi D, Russo A, Lupu E (2011) Inductive logic programming in answer set programming. In: Inductive Logic Programming - 21st International Conference, ILP 2011, Springer, Lecture Notes in Computer Science, vol 7207, pp 91–97
15. Cropper A (2019) Playgol: Learning programs through play. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, ijcai.org, pp 6074–6080
16. Cropper A (2020) Forgetting to learn logic programs. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, AAAI Press, pp 3676–3683
17. Cropper A, Dumancic S (2020) Inductive logic programming at 30: a new introduction. CoRR abs/2008.07912, URL https://arxiv.org/abs/2008.07912, 2008.07912
18. Cropper A, Dumančić S (2020) Learning large logic programs by going beyond entailment. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, pp 2073–2079
19. Cropper A, Morel R (2021) Learning programs by learning from failures. Machine Learning
20. Cropper A, Muggleton SH (2015) Learning efficient logical robot strategies involving composable objects. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, AAAI Press, pp 3423–3429
21. Cropper A, Muggleton SH (2016) Metagol system. URL https://github.com/metagol/metagol
22. Cropper A, Muggleton SH (2019) Learning efficient logic programs. Machine Learning 108(7):1063–1083
23. Cropper A, Tourret S (2020) Logical reduction of metarules. Machine Learning 109(7):1323–1369
24. Cropper A, Tamaddoni-Nezhad A, Muggleton SH (2015) Meta-interpretive learning of data transformation programs. In: Inductive Logic Programming - 25th International Conference, ILP 2015, Springer, Lecture Notes in Computer Science, vol 9575, pp 46–59
25. Cropper A, Dumančić S, Muggleton SH (2020) Turning 30: New ideas in inductive logic programming. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, pp 4833–4839
26. Cropper A, Evans R, Law M (2020) Inductive general game playing. Machine Learning 109(7):1393–1434
27. Cropper A, Morel R, Muggleton S (2020) Learning higher-order logic programs. Machine Learning 109(7):1289–1322
28. Dai WZ, Muggleton SH (2021) Abductive knowledge induction from raw data. In: Proceedings of the 35th Conference on Artificial Intelligence (IJCAI 2021), IJCAI, in Press
29. De Raedt L (2008) Logical and relational learning. Cognitive Technologies, Springer
30. De Raedt L, Kersting K (2008) Probabilistic Inductive Logic Programming, Springer-Verlag, Berlin, Heidelberg, p 1–27
31. De Raedt L, Kimmig A, Toivonen H (2007) Problog: A probabilistic prolog and its application in link discovery. In: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007,

pp 2462–2467
32. De Raedt L, Dries A, Thon I, den Broeck GV, Verbeke M (2015) Inducing probabilistic relational rules from probabilistic examples. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, AAAI Press, pp 1835–1843
33. De Raedt L, Kersting K, Natarajan S, Poole D (2016) Statistical Relational Artificial Intelligence: Logic, Probability, and Computation. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers
34. Dong H, Mao J, Lin T, Wang C, Li L, Zhou D (2019) Neural logic machines. In: ICLR
35. Dong H, Mao J, Lin T, Wang C, Li L, Zhou D (2019) Neural logic machines. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, URL https://openreview.net/forum?id=B1xY-hRctX
36. Dumancic S, Guns T, Cropper A (2020) Knowledge refactoring for inductive program synthesis. AAAI
37. Dumančić S, Blockeel H (2017) Clustering-based relational unsupervised representation learning with an explicit distributed representation. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, ijcai.org, pp 1631–1637
38. Dumančić S, Guns T, Meert W, Blockeel H (2019) Learning relational representations with auto-encoding logic programs. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, ijcai.org, pp 6081–6087
39. Ellis K, Morales L, Sablé-Meyer M, Solar-Lezama A, Tenenbaum J (2018) Learning libraries of subroutines for neurally-guided bayesian program induction. In: NeurIPS 2018, pp 7816–7826
40. Evans R, Grefenstette E (2018) Learning explanatory rules from noisy data. J Artif Intell Res 61:1–64
41. Evans R, Hernández-Orallo J, Welbl J, Kohli P, Sergot M (2021) Making sense of sensory input. Artificial Intelligence p 103438
42. Ferilli S, Esposito F, Basile TMA, Mauro ND (2004) Automatic induction of first-order logic descriptors type domains from observations. In: Inductive Logic Programming, 14th International Conference, ILP 2004, Springer, Lecture Notes in Computer Science, vol 3194, pp 116–131
43. Gebser M, Kaminski R, Kaufmann B, Schaub T (2012) Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers
44. Gebser M, Kaufmann B, Schaub T (2012) Conflict-driven answer set solving: From theory to practice. Artif Intell 187:52–89
45. Genesereth MR, Björnsson Y (2013) The international general game playing competition. AI Magazine 34(2):107–111
46. Gulwani S (2011) Automating string processing in spreadsheets using input-output examples. In: Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, ACM, pp 317–330
47. Heule MJH, Kullmann O, Marek VW (2016) Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In: Creignou N, Berre DL (eds) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings, Springer, Lecture Notes in Computer Science, vol 9710, pp 228–245, DOI 10.1007/978-3-319-

40970-2_15, URL https://doi.org/10.1007/978-3-319-40970-2_15
48. Hocquette C, Muggleton SH (2020) Complete bottom-up predicate invention in meta-interpretive learning. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, pp 2312–2318
49. Huynh TN, Mooney RJ (2008) Discriminative structure and parameter learning for markov logic networks. In: Proceedings of the 25th International Conference on Machine Learning, Association for Computing Machinery, New York, NY, USA, p 416?423, DOI 10.1145/1390156.1390209
50. Inoue K (2016) Meta-level abduction. FLAP 3(1):7–36
51. Inoue K, Doncescu A, Nabeshima H (2013) Completing causal networks by meta-level abduction. Machine Learning 91(2):239–277
52. Inoue K, Ribeiro T, Sakama C (2014) Learning from interpretation transition. Machine Learning 94(1):51–79
53. Kaalia R, Srinivasan A, Kumar A, Ghosh I (2016) ILP-assisted de novo drug design. Machine Learning 103(3):309–341
54. Kaiser L, Sutskever I (2016) Neural gpus learn algorithms. In: 4th International Conference on Learning Representations, ICLR 2016
55. Kaminski T, Eiter T, Inoue K (2018) Exploiting answer set programming with external sources for meta-interpretive learning. Theory Pract Log Program 18(3-4):571–588
56. Katzouris N, Artikis A, Paliouras G (2015) Incremental learning of event definitions with inductive logic programming. Machine Learning 100(2-3):555–585
57. Katzouris N, Artikis A, Paliouras G (2016) Online learning of event definitions. Theory Pract Log Program 16(5-6):817–833
58. Kok S, Domingos P (2009) Learning markov logic network structure via hypergraph lifting. In: Proceedings of the 26th International Conference on Machine Learning, Association for Computing Machinery, New York, NY, USA, p 505?512, DOI 10.1145/1553374.1553440
59. Kok S, Domingos PM (2007) Statistical predicate invention. In: Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), ACM, ACM International Conference Proceeding Series, vol 227, pp 433–440
60. Kramer S (1995) Predicate invention: A comprehensive view. Rapport technique OFAI-TR-95-32, Austrian Research Institute for Artificial Intelligence, Vienna
61. Kramer S (2020) A brief history of learning symbolic higher-level representations from data (and a curious look forward). In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, pp 4868–4876
62. Law M (2018) Inductive learning of answer set programs. PhD thesis, Imperial College London, UK
63. Law M, Russo A, Broda K (2014) Inductive learning of answer set programs. In: Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Springer, Lecture Notes in Computer Science, vol 8761, pp 311–325
64. Law M, Russo A, Broda K (2018) The complexity and generality of learning answer set programs. Artif Intell 259:110–146
65. Law M, Russo A, Bertino E, Broda K, Lobo J (2019) Representing and learning grammars in answer set programming. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, AAAI Press, pp 2919–2928
66. Law M, Russo A, Bertino E, Broda K, Lobo J (2020) Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria. In: The

Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, AAAI Press, pp 2877–2885
67. Law M, Russo A, Broda K (2020) The ilasp system for inductive learning of answer set programs. The Association for Logic Programming Newsletter
68. Leban G, Zabkar J, Bratko I (2008) An experiment in robot discovery with ILP. In: Inductive Logic Programming, 18th International Conference, ILP 2008, Springer, Lecture Notes in Computer Science, vol 5194, pp 77–90
69. Legras S, Rouveirol C, Ventos V (2018) The game of bridge: A challenge for ILP. In: Inductive Logic Programming - 28th International Conference, ILP 2018, Springer, Lecture Notes in Computer Science, vol 11105, pp 72–87
70. Lin D, Dechter E, Ellis K, Tenenbaum JB, Muggleton S (2014) Bias reformulation for one-shot function induction. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, IOS Press, Frontiers in Artificial Intelligence and Applications, vol 263, pp 525–530
71. Marcus G (2018) Deep learning: A critical appraisal. CoRR abs/1801.00631
72. Martínez D, Alenyà G, Torras C, Ribeiro T, Inoue K (2016) Learning relational dynamics of stochastic domains for planning. In: Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, AAAI Press, pp 235–243
73. McCreath E, Sharma A (1995) Extraction of meta-knowledge to restrict the hypothesis space for ilp systems. In: Eighth Australian Joint Conference on Artificial Intelligence, pp 75–82
74. Michie D (1988) Machine learning in the next five years. In: Sleeman DH (ed) Proceedings of the Third European Working Session on Learning, EWSL 1988, Turing Institute, Pitman Publishing, pp 107–122
75. Muggleton S (1987) Duce, an oracle-based approach to constructive induction. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence., Morgan Kaufmann, pp 287–292
76. Muggleton S (1991) Inductive logic programming. New Generation Computing 8(4):295–318
77. Muggleton S (1995) Inverse entailment and progol. New Generation Comput 13(3&4):245–286
78. Muggleton S, Buntine WL (1988) Machine invention of first order predicates by inverting resolution. In: Machine Learning, Proceedings of the Fifth International Conference on Machine Learning, Morgan Kaufmann, pp 339–352
79. Muggleton S, De Raedt L (1994) Inductive logic programming: Theory and methods. J Log Program 19/20:629–679
80. Muggleton S, Feng C (1990) Efficient induction of logic programs. In: Algorithmic Learning Theory, First International Workshop, ALT '90, pp 368–381
81. Muggleton S, Hocquette C (2019) Machine discovery of comprehensible strategies for simple games using meta-interpretive learning. New Generation Computing 37:203–217
82. Muggleton S, Paes A, Costa VS, Zaverucha G (2009) Chess revision: Acquiring the rules of chess variants through FOL theory revision from examples. In: Inductive Logic Programming, 19th International Conference, ILP 2009, Springer, Lecture Notes in Computer Science, vol 5989, pp 123–130
83. Muggleton S, De Raedt L, Poole D, Bratko I, Flach PA, Inoue K, Srinivasan A (2012) ILP turns 20 - biography and future challenges. Machine Learning 86(1):3–23

84. Muggleton S, Dai W, Sammut C, Tamaddoni-Nezhad A, Wen J, Zhou Z (2018) Meta-interpretive learning from noisy images. Machine Learning 107(7):1097–1118
85. Muggleton S, Dai WZ, Sammut C, Tamaddoni-Nezhad A, Wen J, Zhou ZH (2018) Meta-interpretive learning from noisy images. Machine Learning 107:1097–1118
86. Muggleton SH, Lin D, Pahlavi N, Tamaddoni-Nezhad A (2014) Meta-interpretive learning: application to grammatical inference. Machine Learning 94(1):25–49
87. Muggleton SH, Lin D, Tamaddoni-Nezhad A (2015) Meta-interpretive learning of higher-order dyadic Datalog: predicate invention revisited. Machine Learning 100(1):49–73
88. Muggleton SH, Schmid U, Zeller C, Tamaddoni-Nezhad A, Besold TR (2018) Ultra-strong machine learning: comprehensibility of programs learned with ILP. Machine Learning 107(7):1119–1140
89. Nienhuys-Cheng SH, Wolf Rd (1997) Foundations of Inductive Logic Programming. Springer-Verlag New York, Inc., Secaucus, NJ, USA
90. Patsantzis S, Muggleton S (2021) Top program construction and reduction for polynomial time meta-interpretive learning. Machine Learning 110:755–778
91. Picado J, Termehchy A, Fern A, Pathak S (2017) Towards automatically setting language bias in relational learning. In: Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning, DEEM@SIGMOD 2017, ACM, pp 3:1–3:4
92. Plotkin G (1971) Automatic methods of inductive inference. PhD thesis, Edinburgh University
93. Quinlan JR (1990) Learning logical definitions from relations. Machine Learning 5:239–266
94. Ray O (2009) Nonmonotonic abductive inductive learning. J Applied Logic 7(3):329–340
95. Reed SE, de Freitas N (2016) Neural programmer-interpreters. In: 4th International Conference on Learning Representations, ICLR 2016
96. Ribeiro T, Inoue K (2014) Learning prime implicant conditions from interpretation transition. In: Inductive Logic Programming - 24th International Conference, ILP 2014, Springer, Lecture Notes in Computer Science, vol 9046, pp 108–125
97. Ribeiro T, Magnin M, Inoue K, Sakama C (2015) Learning multi-valued biological models with delayed influence from time-series observations. In: 14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, IEEE, pp 25–31
98. Ribeiro T, Folschette M, Magnin M, Inoue K (2020) Learning any semantics for dynamical systems represented by logic programs, working paper or preprint
99. Richardson M, Domingos PM (2006) Markov logic networks. Machine Learning 62(1-2):107–136, DOI 10.1007/s10994-006-5833-1, URL https://doi.org/10.1007/s10994-006-5833-1
100. Rocktäschel T, Riedel S (2017) End-to-end differentiable proving. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, pp 3788–3800
101. Sammut C, Sheh R, Haber A, Wicaksono H (2015) The robot engineer. In: Late Breaking Papers of the 25th International Conference on Inductive Logic Programming, CEUR-WS.org, CEUR Workshop Proceedings, vol 1636, pp 101–106
102. Sato T (1995) A statistical learning method for logic programs with distribution semantics. In: Sterling L (ed) Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995,

MIT Press, pp 715–729
103. Sato T, Kameya Y (2001) Parameter learning of logic programs for symbolic-statistical modeling. J Artif Intell Res 15:391–454, DOI 10.1613/jair.912, URL https://doi.org/10.1613/jair.912
104. Schüller P, Benz M (2018) Best-effort inductive logic programming via fine-grained cost-based hypothesis generation - the inspire system at the inductive logic programming competition. Machine Learning 107(7):1141–1169
105. Sivaraman A, Zhang T, den Broeck GV, Kim M (2019) Active inductive logic programming for code search. In: Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, IEEE / ACM, pp 292–303
106. Srinivasan A (2001) The ALEPH manual. Machine Learning at the Computing Laboratory, Oxford University
107. Srinivasan A, King RD, Bain M (2003) An empirical study of the use of relevance information in inductive logic programming. J Machine Learning Res 4:369–383
108. Stahl I (1995) The appropriateness of predicate invention as bias shift operation in ILP. Machine Learning 20(1-2):95–117
109. Tamaddoni-Nezhad A, Bohan D, Raybould A, Muggleton S (2014) Towards machine learning of predictive models from ecological data. In: Inductive Logic Programming - 24th International Conference, ILP 2014, Springer, Lecture Notes in Computer Science, vol 9046, pp 154–167
110. Wang WY, Mazaitis K, Cohen WW (2014) Structure learning via parameter learning. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, ACM, pp 1199–1208
111. Wirth N (1985) Algorithms and data structures. Prentice Hall
112. Yang F, Yang Z, Cohen WW (2017) Differentiable learning of logical rules for knowledge base reasoning. In: NIPS 2017