A Critical Review of Inductive Logic Programming Techniques for Explainable AI

Zheng Zhang*, Levent Yilmaz*, Member, IEEE, and Bo Liu*, Senior Member, IEEE

Abstract—Despite recent advances in modern machine learning algorithms, the opaqueness of their underlying mechanisms continues to be an obstacle in adoption. To instill confidence and trust in artificial intelligence (AI) systems, explainable AI (XAI) has emerged as a response to improve modern machine learning algorithms’ explainability. Inductive logic programming (ILP), a subfield of symbolic AI, plays a promising role in generating interpretable explanations because of its intuitive logic-driven framework. ILP effectively leverages abductive reasoning to generate explainable first-order clausal theories from examples and background knowledge. However, several challenges in developing methods inspired by ILP need to be addressed for their successful application in practice. For example, the existing ILP systems often have a vast solution space, and the induced solutions are very sensitive to noises and disturbances. This survey paper summarizes the recent advances in ILP and a discussion of statistical relational learning (SRL) and neural-symbolic algorithms, which offer synergistic views to ILP. Following a critical review of the recent advances, we delineate observed challenges and highlight potential avenues of further ILP-motivated research toward developing self-explanatory AI systems.

Index Terms—Differentiable inductive logic programming (dILP), explainable artificial intelligence (XAI), ILP, machine learning, meta-interpretive learning (MIL), neuro-symbolic AI, probabilistic ILP (PILP), statistical relational learning (SRL).

I. INTRODUCTION

EXPLAINABILITY has become an important research area to overcome the challenges in addressing the complexity and understandability of artificial intelligence (AI) systems [1]. Explainable AI (XAI) is an emerging field [2] in machine learning that refers to methods and techniques that enable experts to understand the decisions made by AI algorithms. The U.S. Defense Advanced Research Project Agency describes three AI explainability criteria: prediction accuracy, decision understanding or trust, and traceability [3]. Prediction accuracy refers to explaining how specific conclusions are derived, while decision understanding involves fostering trust in the underlying mechanisms and processes. Traceability involves inspection of the actions and their cause-effect relations to improve causal and evidential reasoning and to immerse into the decision loops to influence tasks as needed. According to these criteria, an AI system should perform a specific task or recommend decisions and produce an explainable characterization of why it renders specific decisions along with the supporting rationale.

Recently, with the success of precise but largely inscrutable deep learning models, explainability received significant attention [4]. The opaqueness of deep learning algorithms motivates AI researchers to open the black box for bringing transparency and avoiding reliance only on model accuracy [1]. Explanations are essential for the explainability of a machine-learned model. Symbolic AI is the overarching framework for AI methods that are based on high-level “symbolic” (human-readable) representations of problems, plans, and solutions. Symbolic AI requires a modest amount of training and can achieve reasonable performance with limited data. Symbolic methods also grant the benefits of explainability and generalization at the concept level through inferential and inductive reasoning while offering connections to established planning algorithms [5]. Fig. 1 shows a flowchart of the symbolic AI pipeline: by learning from symbolic background knowledge and examples (positive and negative), the symbolic model can elaborate and revise logic frames in an explainable manner while enabling the provision of valid and provable answers for symbolic queries. The user could interact with the trained system as follows: the system can take action for the current task and provide an explanation to the user that justifies its action; the user can then make a decision based on the explanation and improve the knowledge base (KB) as well as the training process.

As symbolic AI achieves unprecedented impact on problem-solving via search, planning, and decision-making, one of the hallmarks of symbolic AI is inductive logic programming (ILP) [6]. Besides explainability, ILP systems are also data-efficient, while most machine learning models require large quantities of data to reach acceptable accuracy during learning. ILP, as a knowledge-based strategy, can provide a symbolic, goal-driven, and causal interpretation of data.

However, there are also some drawbacks to these symbolic approaches: they are inherently brittle and do not deal well with noise, it is not easy to express complex nonlinear decision surfaces in logic [7], and the guiding method for hypothesis search is questionable [8]. To improve the efficacy
of ILP methods and mitigate the challenges, several types of ILPs have been developed: meta-interpretive learning (MIL) framework [9] has made progress in predicate invention and recursive generalizations using abduction in terms of a meta-interpreter; probabilistic ILP (PILP) that introduced by De Raedt and Kersting [10] is more powerful than ILP and, in turn, traditional attribute-value approaches due to its ability to deal explicitly with the uncertainty; differentiable ILP ($\partial$ILP) [11] augments ILP with neural networks, making the system robust to noise and error in the training data that ILP cannot cope with alone.

The rest of this article is organized based on the flowchart of the symbolic AI pipeline. In Section II, related works in the extant literature regarding ILP are reviewed. Section III discusses three types of ILP, including PILP, MIL, and $\partial$ILP. Section IV introduces symbolic-based frameworks, including statistical relational learning (SRL) and neural-symbolic AI (NeSy), and their relation. In Section V, we present a user-centered explanation and a measurement of trust appropriate for ILP. The experimental evaluations of ILP systems are depicted in Section VI. Section VII delineates the challenges and highlights future research directions for academics and domain experts, and Section VIII concludes this article.

II. BACKGROUND

ILP, a classical rule-based system, is a subfield of symbolic AI that uses logic programming as a uniform representation, for example, background knowledge and hypotheses. An ILP system will derive a hypothesized logic program, which entails all the positive and none of the negative examples given an encoding of the general background knowledge and a set of examples representing a logical database of facts.

ILP was first introduced in [12]. The successes of ILP have been in the area of inductive construction of expert systems: MYCIN [13] and XCON [14] are built using hand coding of rules; GASOIL [15] and BMT [16] are built using software derived from Quinlan’s [17] inductive decision tree building algorithm ID3. Along with the successes of this technology, the following limitations have become apparent: 1) propositional-level systems cannot be used in areas requiring essentially relational knowledge representations; 2) inability to make use of background knowledge when learning; and 3) the systems construct hypotheses within the limits of a fixed vocabulary of propositional attributes [12].

Later, Plotkin [18] and Shapiro [19] introduce computer-based inductive systems within the framework to solve these problems of full first-order logic (FOL). Plotkin’s significant contributions are as follows: 1) the introduction of relative subsumption, a relationship of generality between clauses and 2) the inductive mechanism of relative least general generalization (RLGG). Shapiro [19] investigates an approach to Horn clause induction in which the search for hypotheses is from general to specific, rather than Plotkin’s specific to the general approach. These all become the foundation of ILP. Later, Sammut and Banerji [20] describe a system called MARVIN, which generalizes a single example at a time regarding a set of background clauses. Quinlan [21] has described a highly efficient program called Foil, which relies on a general to specific heuristic search, which is guided by an information criterion related to entropy. Muggleton et al. [22] apply a “determinate” restriction to hypotheses, and their learning program, Golem, has been demonstrated [22] to have a level of efficiency similar to Quinlan’s Foil but without the accompanying loss of scope. In the late 1980s, right before ILP appears, Muggleton and Buntine [23] present inverse resolution (IR) and implement it in Cigol [23]. They also introduce a mechanism for automatically inventing and generalizing first-order Horn clause predicates.

ILP produces a widely used technology with a firm theoretical foundation based on principles from both logic and statistics. On the side of statistical justification of hypotheses, Muggleton et al. [24] discuss the possible relationship between algorithmic complexity theory and probably approximately correct (PAC) learning; in terms of logic, Muggleton provides a unifying framework for IR and RLGG by rederiving RLGG in terms of IR. Great progress has been made since the birth of ILP. Muggleton introduces more comprehensive approaches to the FOL inductive theory: inverting implication and inverse entailment. Kietz [25] proves that ILP generally is not PAC-learnable. Nienhuys-Cheng and De Wolf [26] show the theoretical basis of ILP. Besides theoretical works, some ILP algorithms and implementations have also been developed: Progol is Stephen Muggleton’s implementation of ILP that
combines “inverse entailment” with the “general-to-specific search” through a refinement graph; Golem, developed by Muggleton et al. [22], is based on RLGG; Lavrac et al. present an ILP system called LINUS based on propositional logic; CLINT uses queries to eliminate irrelevant literals and raises the generality of hypotheses by proposing more complex hypothesis clauses. Along with the era of machine learning and data-driven AI, ILP naturally becomes an important part of the ML model, because the learned hypotheses of ILP are represented in a symbolic form. They are inspectable by humans and, therefore, provide transparency and comprehensibility of the machine-learned classifiers.

We conclude our introduction with Fig. 2 that depicts the overlay visualization of keywords related to ILP based on the analysis of over 300 articles. The term ILP system is associated with articles across multiple clusters. The articles that use ILP to develop systems have strong connections with the terms FOL, Aleph [27], Foil [28], and Prolog [29]. The high-order logic abduction method connects the term ILP system with the terms MIL [9], Metagol [30], and predicate invention. Common terms that combine probability with the term ILP system are SRL, Markov logic network (MLN), and Bayesian network. Keywords, including learnability, inductive inference, and classification, also have close relations to the term ILP system.

A. Concepts of ILP

We use the following example to introduce the basic concepts of the ILP system:

\[
\text{parent}(i, a) \quad \text{parent}(a, b)
\]

\[
\text{grandparent}(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y).
\]

ILP is defined using FOL [31]. In FOL, a formula, \(\text{parent}(i, a)\) or \(\text{parent}(a, b)\), that contains no logical connectives is called the atom. An atom or its negation, i.e., \(\neg\text{parent}(i, a)\) and \(\neg\text{parent}(a, b)\), is called literal. A definite clause is a many-way OR (disjunction) of literals (formula 3). \(a, b,\) and \(i\) in the example are constants, and \(X, Y,\) and \(Z\) are variables, and all constants and variables are terms. A term that does not contain any free variables is called a ground term [\(\text{parent}(i, a)\)]. A Boolean-valued function \(P : X \rightarrow \{\text{true, false}\}\) is called the predicate (parent and grandparent) on \(X\); \(\text{grandparent}/2\) and \(\text{parent}/2\) denote predicates with their arity, i.e., number of arguments. A function can be any value unlike a predicate, and it will never appear except as arguments to predicates. FOL is a structure of logic consisting of constants, variables, predicates, functions, and sentences. It uses quantified variables over nonlogical objects and allows the use of sentences that contain variables.

B. Semantics of ILP

There are two different semantics for ILP: standard and nonmonotonic semantics. For the normal semantic, given background (prior) knowledge \(B\) and example \(E\) \((E = E^+ \land E^-)\) consists of positive example \(E^+\) and negative example \(E^-\)), find a hypothesis \(H\), such that the following conditions hold.

\[
\begin{align*}
\text{Prior Satisfiability:} & \quad B \land E^- \not\vdash \Box \, . \\
\text{Posterior Satisfiability:} & \quad B \land H \land E^- \not\vdash \Box \, . \\
\text{Prior Necessity:} & \quad B \not\vdash E^+ \\
\text{Posterior Sufficiency:} & \quad B \land H \vdash E^+.
\end{align*}
\]

A general setting is used for the normal semantics. In most ILP systems, the definite setting will be used as a simple version of the normal setting, as the background theory and hypotheses are restricted to being definite. The example setting is a special case of definite semantics, where the example is restricted to true and false ground facts. Notice that the example setting is equivalent to the normal semantics, where \(B\) and \(H\) are definite clauses and \(E\) is a set of ground unit
clauses. The example setting is the main setting of ILP. It is employed by the large majority of ILP systems. Table I shows the grandparent dataset for the ILP system. The task is to learn the grandparent relation from various facts involving the father-of and mother-of relations.

In the nonmonotonic setting of ILP, the background theory is a set of definite clauses, the evidence is empty, and the hypotheses are sets of general clauses expressible using the same alphabet as the background theory. The reason that the evidence is empty is that the positive evidence is considered part of the background theory, and the negative evidence is derived implicitly by making a kind of closed world assumption (realized by taking the least Herbrand model [32]). The nonmonotonic semantics realizes induction by deduction. The induction principle of the nonmonotonic setting states that the hypothesis $H$, which is, in a sense, deduced from the set of observed examples $E$ and the background theory $B$ (using a kind of closed world and closed domain assumption), holds for all possible sets of examples. This produces generalizations beyond the observations. As a consequence, properties derived in the nonmonotonic setting are more conservative than those derived in the normal setting.

### C. Searching Method

An enumeration algorithm will be used to solve the ILP problem. Generalization and specialization form the basis for pruning the search space. Generalization corresponds to induction, and specialization to deduction, implying that induction is viewed here as the inverse of deduction. A generic ILP system can now be defined.

**Algorithm 1 Searching Algorithm [33]**

```
QH := Initialize
repeat
    Delete H from QH
    Choose the inference rules $r_1, \ldots, r_k \in R$ to be applied to H
    Apply the rules $r_1, \ldots, r_k$ to H to yield $H_1, \ldots, H_n$
    Add $H_1, \ldots, H_n$ to QH
    Prune QH
until stop-criterion(QH) satisfied
```

$QH$ denotes a queue of candidate hypotheses. Algorithm 1 works as follows: it keeps track of $QH$, and then repeatedly deletes a hypothesis $H$ from the queue and expands the hypotheses using inference rules. Then, the expanded hypotheses are added to the queue of hypotheses $QH$, which may be pruned to discard the unpromising hypothesis from further consideration. This process continues until the stop criterion is satisfied.

There are two kinds of search methods for ILP systems. First, “specific-to-general” systems, a.k.a. bottom-up systems, start from the examples and background knowledge and repeatedly generalize their hypothesis by applying inductive inference rules. During the search, they take care that the hypothesis remains satisfiable (i.e., does not imply negative examples). Second, “general-to-specific” systems, a.k.a. top-down systems, start with the most general hypothesis (i.e., the inconsistent clause $\Box$) and repeatedly specialize the hypothesis by applying deductive inference rules to remove inconsistencies with the negative examples. During the search, care is taken that the hypotheses remain sufficient with regard to the positive evidence. Table II shows some related systems for both two types.

#### D. Inductive Inference Rules

Induction can be considered as the inverse of deduction. Given the formulas $B \land H \models E^+$, deriving $E^+$ from $B \land H$ is deduction, and deriving $H$ from $B \land E^+$ is induction. Therefore, inductive inference rules can be obtained by inverting deductive ones. Table III summarizes commonly used rules for both deduction and induction.

Since this “inverting deduction” paradigm can be studied under various assumptions, corresponding to different assumptions about the deductive rule for $\models$ and the format of background theory $B$ and evidence $E^+$, different models of inductive inference are obtained. Four frameworks of inference rules, $\theta$-subsumption, inverse resolution, inverse implication, and inverse entailment, will be described in this section.

1) $\theta$-Subsumption: The $\theta$-subsumption inductive inference rule is

$$
\theta - \text{subsumption: } \frac{c^2}{c^1}, \text{ where } c^1 \theta \subseteq c^2.
$$

For example, $\text{grandparent}(X, Z) \leftarrow \text{parent}(X, Y), \text{parent}(Y, Z) \theta$-subsumes $\text{grandparent}(i, b) \leftarrow \text{parent}(i, a), \text{parent}(a, b)$ with $\theta = \{ X = i, Y = a, Z = b \}$. In the simplest $\theta$-subsumption, the background knowledge is supposed to be empty, and the deductive inference
rule corresponds to  \( \theta \)-subsumption among single clauses. One extension of  \( \theta \)-subsumption that takes into account background knowledge is called relative subsumption. Similar to  \( \theta \)-subsumption, it is straightforward to define relatively reduced clauses using a straightforward definition of relative clause equivalence. Relative subsumption forms a lattice over relatively reduced clauses.

2) Inverse Resolution: Inductive inference rules can be viewed as the inverse of deductive rules of inference. Since the deductive rule of resolution is complete for the deduction, an inverse of resolution should be complete for induction. IR takes into account background knowledge and aims at inverting the resolution principle. Four main rules of IR are widely used.

Absorption: \[ q \leftarrow A \quad p \leftarrow A, B \]

Identification: \[ q \leftarrow A \quad p \leftarrow q, B \]

Inter-Construction: \[ q \leftarrow B \quad p \leftarrow A, q \quad q \leftarrow C \]

In these rules, lower-case letters are atoms, and upper-case letters are conjunctions of atoms. Both absorption and identification invert a single resolution step. The rules of inter- and intra-construction introduce “predicate invention,” which leads to reducing the hypothesis space and the length of clauses.

3) Inverse Implication: Since the deductive inference rule is incomplete regarding implication among clauses, extensions of inductive inference under  \( \theta \)-subsumption have been studied under the header “inverting implication.” The inability to invert implication between clauses limits the completeness of IR and RLGGs, since  \( \theta \)-subsumption is used in place of clause implication in both. The difference between  \( \theta \)-subsumption and implication between clauses  \( C \) and  \( D \) is only pertinent when  \( C \) can self-resolve. Attempts were made to do the following: 1) extend IR and 2) use a mixture of IR and LGG [37] to solve the problem. The extended IR method suffers from problems of nondeterminacy. Due to the nondeterminacy problem, the development of algorithms regarding inverse implication in ILP is limited. Idestam-Almquist’s use of LGG suffers from the standard problem of intractably large clauses. Both approaches are incomplete for inverting implication, though Idestam-Almquist’s technique is complete for a restricted form of entailment called T-implication [38].

4) Inverse Entailment: The general problem specification of PILP is, given background knowledge  \( B \) and examples  \( E \) find the simplest consistent hypothesis  \( H \), such that  \( B \land H \models E \). In general,  \( B \),  \( H \), and  \( E \) could be arbitrary logic programs. Each clause in the simplest  \( H \) should explain at least one example, since there is a simpler  \( H' \) that will do otherwise. Then, consider the case of  \( H \) and  \( E \), each being single Horn clauses. By rearranging entailment relationship  \( B \land H \models E \), Muggleton [35] proposed the “inverse entailment”

\[ B \land \neg E \models \neg H. \]

In particular,  \( \neg \bot \) is the (potentially infinite) conjunction of ground literals, which are true in all models of  \( B \land \neg E \). Since  \( H \) must be true in every model of  \( B \land \neg E \), it must contain a subset of the ground literals in  \( \neg \bot \). Therefore

\[ B \land \neg E \models \neg \bot \models \neg H \]

\[ H \models \bot. \]

A subset of the solutions for  \( H \) can be found by considering the clauses which  \( \theta \)-subsume  \( \bot \). The complete set of candidates for  \( H \) can be found by considering all clauses which  \( \theta \)-subsume sub-saturants of  \( \bot \).

E. ILP Systems

A large amount of logic learning systems have been developed based on the inference rules we mentioned before. Fig. 3 shows the timeline of the development of logic learning systems, including Prolog [29], MIS [19], CLINT [34], Foil [28], LINUS [36], Golem [22], Aleph [27], Progol [35], Cigol [23], Metagol [30], ProbLog [39], DeepProbLog [40], \( \partial \)ILP [11], and Popper [41]. Note that there are no common-used inverse implication systems, because it suffers from problems of nondeterminacy.

III. VARIANTS OF ILP

A. Probabilistic Inductive Logic Programming

PILP is a machine learning technique based on probabilistic logic programming. It addresses one of the central questions of AI by integrating probabilistic reasoning with machine learning, and first-order relational logic representations [10]. Dealing explicitly with uncertainty, MIL [9] holds merits on predicate invention and recursive generalizations, and  \( \partial \)ILP [11] can speed up the learning process and also robust to noise and error. We will introduce each of them in this section.

B. Inter- and Intra-Construction

The general problem specification of PILP is, given background knowledge  \( B \) and examples  \( E \), find the simplest consistent hypothesis  \( H \), such that  \( B \land H \models E \). In general,  \( B \),  \( H \), and  \( E \) could be arbitrary logic programs. Each clause in the simplest  \( H \) should explain at least one example, since there is a simpler  \( H' \) that will do otherwise. Then, consider the case of  \( H \) and  \( E \), each being single Horn clauses. By rearranging entailment relationship  \( B \land H \models E \), Muggleton [35] proposed the “inverse entailment”

\[ B \land \neg E \models \neg H. \]
cov(\epsilon, H, B) = P(\epsilon|H, B), a probabilistic logical language \(L_H\) for hypotheses, and a background theory \(B\), find a hypothesis \(H^*\) in \(L_H\), such that

\[ H^* = \arg\max_{H \in L} \text{score}(E, H, B) \]

and the following constraints hold:

\[ \exists \epsilon_p \in E_p : \text{cov} (\epsilon_p, H^*, B) > 0 \]
\[ \exists \epsilon_i \in E_i : \text{cov} (\epsilon_i, H^*, B) = 0. \]

The score is some objective function, usually involving the probabilistic covers relation of the observed examples, such as the observed likelihood \(\prod_{E \in E} \text{cov} (\epsilon_p, H^*, B)\) [10]. We denote \(H = (L, \lambda)\), where \(L\) represents all FOL program rules in \(H\), and \(\lambda\) indicates probabilistic parameters. Two subtasks should be considered when solving PILP learning problems: 1) parameter estimation, where it is assumed that the underlying logic program \(L\) is fixed, and the learning task consists of estimating the parameters \(\lambda\) that maximize the likelihood and 2) structure learning where both \(L\) and \(\lambda\) have to be learned from the data.

Similar to the ILP learning problem, the language \(L_E\) is selected for representing the examples, and the probabilistic covers relation determines different learning settings. In ILP, this leads to learning from interpretations [42], from proofs [10], and from entailment [43]. Therefore, it should be no surprise that this very same distinction also applies to probabilistic knowledge representation formalisms. There are three PILP settings as well: probabilistic learning from interpretations, from entailment, and proofs [10]. The main idea is to lift ILP settings by associating probabilistic information with clauses and interpretations and by replacing ILP’s deterministic covers relation with a probabilistic one. The large majority of PILP techniques proposed so far fall into the learning from interpretations setting, including parameter estimation of probabilistic logic programs [44], learning of probabilistic relational models (PRMs) [45], parameter estimation of relational Markov models [46], learning of object-oriented Bayesian networks [47], learning relational dependency networks (RDNs) [48], and learning logic programs with annotated disjunctions (LPAD) [49]. To define probabilities on proofs, ICL [50], Prism [51], and stochastic logic programs (SLPs) [52] attach probabilities to facts (respectively, clauses) and treat them as stochastic choices within resolution. PILP techniques that learn from proofs have been developed, including hidden Markov model induction by Bayesian model merging [53], relational Markov models [54], and logical hidden Markov models [55]. Learning from interpretations setting has been investigated for learning SLPs [56], [57] and for parameter estimation of Prism programs [58], [59] from observed examples.

Many algorithms have been developed since the PILP framework: Muggleton describes a method, which based on an approximate Bayes “maximum a posterior probability” (MAP) algorithm, for learning SLPs from examples and background knowledge [57], which is considered one of the earliest structure learning methods for PILP: De Raedt and Thon [60] upgrade rule learning to a probabilistic setting, in which both the examples themselves as well as their classification can be probabilistic. To solve “large groundings” problem, Wang et al. [61] present a first-order probabilistic language, which is well suited to approximate “local” grounding. The algorithm “Structure Learning of Probabilistic logic progrAmS with Em over bdds” (SLIPCASE) performs a beam search in the space of the language of LPAD using the log-likelihood of the data as the guiding heuristics [62]; Fierens et al. [63] investigate how classical inference and learning tasks known from the graphical model community can be tackled for probabilistic logic programs; and the algorithm “Structure Learning of Probabilistic logic programs by searching OVER the clause space” (SLIPCOVER) performs a beam search in the space of probabilistic clauses and a greedy search in the space of theories, using the log-likelihood of the data as the guiding heuristics [64].

B. Meta-Interpretive Learning

MIL is a framework developed by Cropper and Muggleton [65], which uses higher-order metarules to support predicate invention and learning of recursive definitions. Metarules, second-order Horn clauses, are widely discussed [65], [66], [67], [68] as a form of declarative bias. Metarules define the structure of learnable programs, which, in turn, defines the hypothesis space. For instance, to learn the grandparent/2 relation given the parent/2 relation, the chain metarule\(^1\) would be suitable

\[ P(A, B) \leftrightarrow Q(A, C), R(C, B). \]

In this metarule, the letters \(P\), \(Q\), and \(R\) denote existentially quantified second-order variables (variables that can be bound to predicate symbols), and the letters \(A\), \(B\), and \(C\)

\(^1\)Common-used metarules [30]: 1) identity: \(P(A, B) \leftrightarrow Q(A, B)\); 2) inverse: \(P(A, B) \leftrightarrow Q(B, A)\); 3) precon: \(P(A, B) \leftrightarrow Q(A), R(A, B)\); 4) postcon: \(P(A, B) \leftrightarrow Q(A, B), R(B)\); 5) chain: \(P(A, B) \leftrightarrow Q(A, C), R(C, B)\); and 6) recursive: \(P(A, B) \leftrightarrow Q(A, C), P(C, B)\).
denote universally quantified first-order variables (variables that can be bound to constant symbols). Given the chain metarule, the background parent/2 relation, and examples of the grandparent/2 relation, the learner will try to find the correct substitutions for predicates, and one of the correct solutions would be

\[ \{P/\text{grandparent}, Q/\text{parent}, R/\text{parent} \} \]

and the substitution result is

\[ \text{grandparent}(A, B) \leftarrow \text{parent}(A, C), \text{parent}(C, B). \]

We will discuss the MIL problem after the description of metarules. Before that, we define what MIL input is: an MIL input is a tuple \((B, E^+, E^-, M)\), where \(B\) is a set of Horn clauses denoting background knowledge, \(E^+\) and \(E^-\) are disjoint sets of ground atoms representing positive and negative examples, respectively, and \(M\) is a set of metarules. An MIL problem [69] can be defined from an MIL input.

Given an MIL input \((B, E^+, E^-, M)\), the MIL problem is to return a logic program hypothesis \(H\), such that the following hold.

1. \(\forall c \in H, \exists m \in M, \text{such that } c = m\theta\), where \(\theta\) is a substitution that grounds all the existentially quantified variables in \(m\).
2. \(H \cup B \models E^+\).
3. \(H \cup B \not\models E^-\).

\(H\) can be considered as a solution to the MIL problem. Based on the equation \(c = m\theta\), MIL focuses on searching for \(\theta\) instead of \(H\) through abductive reasoning in second-order logic.

As the grandparent task shown before, MIL could be considered as an FOL rule searching problem based on metarules. Since any first-order predicates could substitute second-order variables, the searching of a logic program hypothesis \(H\) is more flexible than that of ILP: if second-order variables are replaced by those predicates that do not exist in \(B\), new predicates will be invented (predicate invention); if they are replaced by the same predicates in both head and body of the metarules, the recursive definition will be learned.

As we discussed before, the metarules determine the structure of permissible rules, which, in turn, defines the hypothesis space. Deciding which metarules to use for a given learning task is a major open problem and is a trade-off between efficiency and expressivity: we wish to use fewer metarules as the hypothesis space grows given more metarules, but if we use too few metarules, we will lose expressivity. Also, the hypothesis space of MIL highly depends on the metarules we choose. For example, the identity metarule \(P(A, B) \leftarrow Q(A, B)\) cannot be learned from the chain metarule \(P(A, B) \leftarrow Q(A, C), R(C, B)\). Cropper and Muggleton [65] demonstrate that irreducible or minimal sets of metarules can be found automatically by applying Plotkin’s clausal theory reduction algorithm. When this approach is applied to a set of metarules consisting of an enumeration of all metarules in a given finite hypothesis language, they show that, in some cases, as few as two metarules are complete and sufficient for generating all hypotheses. Nevertheless, for expected hypothesis \(H^*\), the learned model \(H'\) is only equivalence with the actual model \(H^*\) semantically, not literally (\(H'\) usually contains more hypothesis than \(H^*\)), so reducing metarules does not necessarily improve the learning efficiency of MIL.

Advances have been made to increase the performance of MIL recently. A new reduction technique, called derivation reduction [69], has been introduced to find a finite subset of a Horn theory from which the whole theory can be derived using SLD resolution. Recent work [67] also shows that adding types to MIL can improve learning performance, and type checking can reduce the MIL hypothesis space by a cubic factor, which can substantially reduce learning times. Extended MIL [66] has also been used to support learning higher-order programs by allowing for higher-order definitions to be used as background knowledge. Recently, Popper [41] supports types, learning optimal solutions, learning recursive programs, reasoning about lists and infinite domains, and hypothesis constraints by combining ASP and Prolog.

C. Differentiable Inductive Logic Programming

The \(\partial\)ILP that combines intuitive perceptual with conceptually interpretable reasoning has several advantages: robust to noise and error, data-efficient, and produces interpretable rules [11]. \(\partial\)ILP implements differentiable deduction over continuous values. The gradient of the loss concerning the rule weights, which we use to minimize classification loss, implements a continuous form of induction [11].

The set \(P\) (positive examples) and \(N\) (negative examples) are used to form a new set \(\Lambda\)

\[ \text{Lambda} = \{(y, 1)|y \in P\} \cup \{(y, 0)|y \in N\}. \]

Each pair \((y, \lambda)\) indicates that whether the atom \(y\) is in \(P\) (\(\lambda = 1\)) or \(N\) (\(\lambda = 0\)). A differentiable model is constructed to implement the conditional probability of \(\lambda\) for a ground atom \(\alpha\)

\[ p(\lambda|\alpha, W, \Pi, L, B) \]

where \(W\) is a set of clause weights, \(\Pi\) is a program template, \(L\) is a language frame, and \(B\) is a set of background assumptions. The goal is to match the predicted label \(p(\lambda|\alpha, W, \Pi, L, B)\) with the actual label \(\lambda\) in the pair \((y, \lambda)\). Minimizing the expected negative log-likelihood is performed

\[ \text{loss} = -E_{(y, \lambda)}[-\lambda \cdot \log p(\lambda|\alpha, W, \Pi, L, B) + (1 - \lambda) \cdot \log(1 - p(\lambda|\alpha, W, \Pi, L, B))]. \]

The conditional probability \(p(\lambda|\alpha, W, \Pi, L, B)\) is calculated by four functions: \(f_{\text{extract}}, f_{\text{infer}}, f_{\text{convert}}\), and \(f_{\text{generate}}\)

\[ p(\lambda|\alpha, W, \Pi, L, B) = f_{\text{extract}}(f_{\text{infer}}(f_{\text{convert}}(B), f_{\text{generate}}(\Pi, L, W, T), \alpha)). \]

The \(f_{\text{extract}}\) extracts the value from an atom, the \(f_{\text{convert}}\) converts the elements of \(B\) to 1 and others to 0, and the \(f_{\text{generate}}\) generates a set of clauses from \(\Pi\) and \(L\). The \(f_{\text{infer}}\) performs \(T\) steps of forward-chaining inference where \(T\) is part of \(\Pi\). To show how inference is performed over multiple time steps, each clause \(c\) has been translated into a function
Each intensional predicate \( p \) is defined by two clauses generated from two rule templates \( \tau_p^1 \) and \( \tau_p^{2,3} \) combining the application of two functions \( F_p^{1,2} \) and \( F_p^{2,3} \) taking the elementwise max \( \gamma \), which equals to

\[
G_p^{j,k}(a) = x
\]

where

\[
x[i] = \max\left(F_p^{1,j}(a)[i], F_p^{2,k}(a)[i]\right).
\]

\( a_t \) depicts conclusions after \( t \) time steps of inference. \( a_0(x) \) equals to 1 if \( \gamma \) belongs to \( B \), and 0 otherwise. Intuitively, \( \gamma^{j,k} \), which equals to \( G_p^{j,k}(a_t) \), is the result of applying one step of forward chaining inference to at using clauses \( C_p^{1,j} \) and \( C_p^{2,k} \). The weighted average of the \( \gamma^{j,k} \) can be defined using the softmax of the weights

\[
b_t^p = \sum_{j,k} e^{W_p[j,k]} \sum_{j',k'} e^{W_p[j',k']}
\]

and the successor \( a_{t+1} \) is the probabilistic sum

\[
a_{t+1} = a_t + \sum_{p \in P} b_t^p - a_t \cdot \sum_{p \in P} b_t^p.
\]

Besides \( \partial \text{ILP} \), several neural program synthesis models have been used to produce explicit human-readable programs: RobustFill [70] performs an end-to-end synthesis of programs from examples by using a modified attention RNN to allow encoding of variable-sized sets of I/O pairs; a differentiable Forth interpreter [71] is an end-to-end interpreter for the programming language Forth, which enables programmers to write program sketches with slots that can be filled with behavior trained from program input–output data; neural theorem provers (NTPs) [72], end-to-end differentiable provers for basic theorems, formulate as queries to a KB and use Prolog’s backward chaining algorithm as a recipe for recursively constructing neural networks that are capable of proving queries to a KB using subsymbolic representations. Payani and Fekri [73] propose a novel paradigm, called differentiable neural logic ILP (dNL-ILP), for solving ILP problems via deep recurrent neural networks. The dNL-ILP, in contrast to the majority of past methods, directly learns the symbolic logical predicate rules instead of searching through the space of possible FOL rules by using some restrictive rule templates.

### IV. Symbolic-Based Integration

ILPs hold certain merits on explainability due to their symbolic nature and also show drawbacks, such as lack of robustness, cannot deal with big data, and being hard to express complex nonlinear decision surfaces. With the development of AI in other fields, the integration of symbolic AI and other ML methods can make use of the strengths and avoid weaknesses of each method. We present the two most important integrations in this section: SRL and NeSy.

2 Rule template \( \tau \) describes a range of clauses that can be generated.

3 \( G_p^{j,k} \) combines the application of two functions \( F_p^{1,j} \) and \( F_p^{2,k} \). \( F_p^{1,j} \) is the valuation function corresponding to the clause \( C_p^{1,j} \).

4 \( C_p^{j,i} \) is the \( j \)th clause of the \( i \)th rule template \( \tau_p^i \) for intensional predicate \( p \).
B. Neural-Symbolic AI

Neural networks can deal with mislabeled and noisy data but suffer from problems, such as data efficiency and explainability. Symbolic methods are data-efficient along with a number of drawbacks, such as a lack of handling error data and problems in the data. Blending deep neural network topologies with symbolic reasoning techniques creates a more advanced version of AI, i.e., NeSy: the hybrid model aims to combine robust learning in neural networks with reasoning and explainability via symbolic representations for network models [84].

To bring together neural networks and symbolic AI, Kautz [85] indicates a taxonomy including five different types of NeSy at AAAI-2020. In Kautz’s taxonomy, Type 1 is a standard deep learning procedure in which the input and output of a neural network can be made of vectors of symbols, e.g., text in the case of classification, entity extraction, and translation. Type 2 is a neural pattern recognition subroutine within a symbolic problems solver, such as Monte Carlo tree search, e.g., AlphaGo and self-driving car paradigms. Type 3 is a hybrid system that takes symbolic rules, e.g., $A \leftarrow B$, as an input-output training pair $(A, B)$; i.e., the knowledge of symbolic rules will be learned by the neural network. One example uses integration to solve differential equations [86]. Type 4 cascades from the neural network into symbolic reasoner, i.e., the neuro-symbolic concept learner (NS-CL) [87] and conditional theorem provers (CTPs) [88]. Type 5 embeds symbolic reasoning inside a neural engine; e.g., in business AI, when attention to concepts is very high, they are decoded into symbolic entities in an attention schema. A goal that appears in the attention schema indicates that deliberative symbolic reasoning should be initiated.

Table IV summarizes models for the types based on Kautz’s categorization. We now briefly discuss state-of-the-art neural-symbolic models for each type of integration. Note that there are no papers fully regarding type 5. For type 1, symbolic deep reinforcement learning (SDRL) [89] framework conducts high-level symbolic planning based on intrinsic goals using explicitly represented symbolic knowledge and utilizes DRL to learn low-level control policy, leading to improved task-level interpretability for DRL and data efficiency; neural-symbolic stack machine (NeSS) [90] is a differentiable neural network that operates a symbolic stack machine that supports general-purpose sequence-to-sequence generalization, to accomplish compositional generalization; LENSr [91] is a novel approach for improving the performance of deep models by leveraging prior symbolic knowledge. For type 2, generative neuro-symbolic machine (GNM) [92] combines the advantages of distributed and symbolic representation in generative latent variable models; GraIL [93] is a GNN-based framework that predicts relations between nodes that were unseen during training and produces state-of-the-art performances in this inductive setting; a sparse-matrix reified KB [94] is a revolutionary technique for representing a symbolic KB, which enables neural modules that are fully differentiable, faithful to the original semantics of the KB, expressive enough to model multihop inferences, and scalable enough to utilize with realistically large KBs. For type 3, neural-symbolic reader (NeRd) [95] is a scalable integration of distributed representations and symbolic operations for reading comprehension that consists of a reader that encodes text into vector representations and a programmer that generates programs, which will be executed to produce the answer; Know-Evolve [96] is a novel deep evolutionary knowledge network that learns nonlinearly evolving entity representations over time; neural equivalence networks (EQNETs) [97] focus on the challenge of learning continuous semantic representations of algebraic and logical expressions. For type 4, NS-CL [87] learns by looking at images and reading associated questions and answers, without any explicit supervision, such as class labels for objects; CTP [88] is an extension to NTP that uses gradient-based optimization to learn the best rule selection technique; DrKIT [98] is a differentiable module that is capable of answering multihop questions directly using a large entity-linked text corpus.

C. SRL Versus NeSy

Although SRL and NeSy are merging symbolic reasoning with a different fundamental learning paradigm, SRL is the integration of logic and probability, while NeSy is the combination of symbolic reasoning and neural networks. They have a lot in common, and there are interactions between these two fields. De Raedt et al. [99] identify seven dimensions that SRL and NeSy approaches have in common. These seven dimensions are as follows: 1) type of logic; 2) model versus proof-based inference; 3) directed versus undirected models; 4) logical semantics; 5) learning parameters or structure; 6) representing entities as symbols or sub-symbols; and 7) integrating logic with probability and/or neural computation.

We add another dimension, SRL, to represent multiple SRL and NeSy models in Table V in addition to summarizing numerous NeSy structures in [99], along the seven dimensions. Note that, in dimension 1, logic programming and FOL are mostly used for SRL, while models adapt to different types of logic based on the complexity and expressiveness of logic for NeSy. In dimension 4, probabilistic Boolean logic is the most common semantic in SRL. Whereas in NeSy, many probabilistic approaches use neural components to parameterize the underlying distribution, while fuzzy logic is used for computational reasons mostly and to relax rules. In dimension 6, for NeSy, the input and intermediate representations are sub-symbolic, while the output representations can be either

| Type | Category | Models |
|------|----------|--------|
| 1    | [Symbolic Neuro Symbolic] | SDRL, NeSS, LENSr |
| 2    | [Symbolic[Neuro]] | GNM, GraIL, Sparsematrix reified KB |
| 3    | [Neuro $\cup$ compile(Symbolic)] | NeRd, EQNET Know-Evolve |
| 4    | [Neuro $\rightarrow$ Symbolic] | DrDKIT, NS-CL, CTP |
| 5    | [Neuro(Symbolic)] | N/A |
symbolic or sub-symbolic; all representations in SRL are symbolic.

V. USER-CENTERED EXPLAINABLE ILP

XAI has gotten a lot of attention lately as researchers try to figure out what it means and expects. As AI systems are used by a broader and more diverse audience, researchers have realized that it is crucial to pay enough attention to AI users. From a user-centered perspective, as we surveyed ILP systems and their integrations, we briefly discuss models that could apply to ILP by answering the following questions.

1) How should we create explanations for users?
2) How should a user trust a model?
3) How to provide explanations for symbolic-based integration, e.g., NeSy?

A. User-Centered Explanation

Explainability based on model users’ characteristics is essential, since generating explanations based on different audiences can overcome the difficulties of fulfilling all requirements simultaneously. Ribera and Lapedriza [108] classify users into three main groups (developers or AI researchers, domain experts, and lay users) and design different explanations for each group with a particular purpose, content, and presents it in a specific way, which is a suitable framework for ILP systems. For developers and AI researchers, model inspection and simulation with proxy models are provided. These two types of explanations are great for validating the system, discovering flaws, and providing suggestions for improvement. Because they can understand codes, data representation structures, and statistical variances, the audience is well suited to this mode of communication. For domain experts, who are capable of deciding when and how to question the explanation and are led to their discovery by themselves, explanations through natural language conversations or interactive visualizations are offered. For lay users, outcome explanations with several counterfactuals are used, with which users can interact to select the one most interesting to their particular case.

B. Measurement of Trust

Besides creating explanations for different users, how users trust AI systems is also crucial. Yilmaz and Liu [109] propose a measurement of trust that is also appropriate for ILP. In this measurement, three evaluation models with increasing levels of specificity can be considered: binary evaluation, quantized/ discrete evaluation, and continuous/ spectral evaluation (Fig. 4).

As shown in Fig. 4, binary evaluation shows that model users are at the level of categorizing a model as either trusted or distrusted without any middle ground; quantized evaluation sets a threshold in the unfavorable region distinguishes trust from distrust when the model user lacks trust; continuous evaluation measures trust over a continuous range and gives model users the discretion on the degree of trust. Table VI denotes how these criteria relate to the evaluation models.

C. Explanation for Symbolic-Based Integration

In the current state-of-the-art XAI, models that combine connectionism and symbolism are not frequently represented. The explanation for symbolic-based integration is hard, since the connectionist methods in the hybrid system offer poor explainability. To conquer this problem, explainable neural-symbolic learning [110] learns both symbolic and deep representations, together with an explainability metric to assess the level of alignment of machine and human-expert explanations. Campagner and Cabitza [111] use deep learning for the automatic detection of meaningful, hand-crafted high-level symbolic features, which are then to be used by a standard and more interpretable learning model. Bennetot et al. [112] present a methodology based on the neural network’s learning data that allow us to influence its learning and entirely fix biases while providing a fair explanation from its predictions.

VI. EXPERIMENTS

In this section, we evaluate several ILP systems following the experiment procedures in [41] when varying the following: 1) optimal solution size; 2) optimal domain size; 3) example size; and 4) benchmark ILP problems. We also compare...
these systems from a feature perspective and summarize the comparison results. It is crucial to keep in mind that comparing ILP systems is challenging, since they are typically biased and excel at distinct tasks. Thus, we do not conclude that system $X$ is superior to system $Y$ [41].

A. Robot

This comparison aims to find the optimal solution size for each ILP system. To control the solution size, we use a strategy learning problem; i.e., robot: there is a robot in an $n \times n$ world. The aim is to develop a general plan for moving the robot $i$ steps toward the right in the grid from an arbitrary starting location. For example, for a $5 \times 5$ world, and a start position $(1, 3)$, the goal for the robot is to move to position $(3, 3)$ within $i$ steps where $i = 2$. A solution could be

$$f(A, B) \leftarrow \text{move\_right}(A, C), \quad \text{move\_right}(C, B).$$

The solution indicates that the robot could reach the final position by moving two steps to the right. We fix $n$ to 20 and vary $i$ that corresponds to the optimal solution size. The settings for each system are shown as follows. For Aleph, the maximum number of nodes to be searched is 50,000. For Metagol, five metarules (Table VII) are used to guide the searching process if $i < 3$, and six metarules are used if $i \geq 3$ (metarules in Table VII and another metarule for the current step size). For Popper, there are no restrictions in the experiment.

For each $i$ in $[1, 2, \ldots, 10]$, we generate 100 positive (robot has reached the correct position within $i$ steps) and 100 negative examples (robot has not reached the correct position within $i$ steps). If a system fails to learn a solution within a given time, it will achieve default accuracy (50%).

We set a timeout of 60 s per task and repeat each experiment ten times to get the mean and the standard error.

Fig. 5 shows that Aleph outperforms both methods when $i > 4$. The learning time of Aleph and Popper increases significantly when the step size reaches 5. The largest program Popper learns is when $i = 5$, which takes 16 s, compared with less than 1 s for Aleph and Metagol. Popper and Aleph share comparable learning curves, while Metagol outperforms the other two, because metarules lead the search efficiently. Note that the accuracies of all three systems are 100% for the experiment.

B. Robot2

The goal of this comparison is to find the optimal domain size for each ILP system. To control the domain size,
we modify the game in VI-A to robot2 [6]: There is a robot in an $n \times n$ world. The goal is to provide a universal method for advancing the robot from any starting position to the rightmost of the grid. The objective for the robot of a $5 \times 5$ world, for instance, is to go to the location $(3, 5)$ with a start position $(3, 1)$. One solution could be

\[
\begin{align*}
  f(A, B) &\leftarrow \text{move_right}(A, B), \quad \text{at_right}(A, B) \\
  f(A, B) &\leftarrow \text{move_right}(A, C), f(C, B).
\end{align*}
\]

The settings for all systems are described below. For Aleph, the maximum number of nodes to be searched is 50,000. For Metagol, five metarules are used (Table VII) in the experiment. There are no restrictions for Popper.

We vary domain size by changing $n$. For each $n$ in $[10, 20, \ldots, 100]$, we generate 100 positive (the starting position is not at the right and the end position is at the right) and 100 negative examples (the starting position and the end position are not at the right). If a system fails to learn a solution within a given time, it will achieve default predictive accuracy (50%). We enforce a timeout of 60 s per task and repeat each experiment ten times to get the mean and the standard error.

Fig. 6 shows that Popper and Aleph reach 100% accuracy for all cases, while Metagol struggles due to its inefficient search. Fig. 7 indicates that Popper is the fastest of all the systems. Aleph struggles to learn recursive programs, as the grid size becomes larger. Metagol struggles due to its inefficient search, while Popper learns restrictions from failure and never attempts failed clauses again.

C. Scalability

The purpose of this comparison is to check the scalability of each ILP system. To do so, we vary the number of examples in the member problem: check if a number is a member of the given list. We produce $n$ positive and $n$ negative examples, for each $n$ in $[1000, 2000, \ldots, 10,000]$. We repeat each experiment ten times to get the mean and the standard error.

The results are shown in Fig. 8 when the number of training instances is varied. Note that the accuracies of all three systems are essentially comparable (100%). All systems share similar curves before 10,000 example size. Aleph and Popper perform better than Metagol after 16,000 examples.

D. Benchmark ILP Problem

The goal of this comparison is to evaluate how well each system performs on different benchmark ILP problems. Note that learning recursion and predicate invention are considered two difficult problems for most ILP systems. We compare Popper, Metagol, Aleph, $\partial$ILP, and dNL-ILP [73].

We select four examples from different domains: grandparent/2 example from the domain family tree, even/succ2 from arithmetic, member from list, and undirected
edge from graphs. The detailed description of each example is shown in Table VIII.

The settings are depicted as follows. For Aleph, the maximum number of nodes to be searched is 50,000. For Metagol, the metarules are the same as those in the robot experiment. The examples should also be provided in increasing sizes to make Metagol find solutions, since Metagol is sensitive to the order of examples. There are no restrictions for Popper.

For each example, we use the same strategy as in the robot problem: We generate 100 positive and 100 negative examples. The default accuracy is, therefore, 50%. We repeat each experiment ten times to get the mean and the standard error.

Table IX indicates that Popper, $\partial$ILP, and dNL-ILP perform perfectly on all the tasks in terms of accuracy. Metagol reaches 100% predictive accuracy except for the even problem. Aleph struggles to learn solutions for even and member problems.

Table X shows that Aleph is the fastest and $\partial$ILP is the slowest within all the systems. Note that the main difference between $\partial$ILP and dNL-ILP is that $\partial$ILP allows for clauses of at most two atoms and only two rules per each predicate to reduce the size of the search space, while in dNL-ILP, the membership weights of the conjunction can be directly interpreted as the flags in the satisfiability interpretation [73].

**E. Feature Comparison**

We compare six ILP systems in terms of the inference rules they employ. In the comparison, we compare important features, including the following: 1) handling recursive rules—a logic program can handle recursive rules if the same predicate appears in the head and body of a rule; 2) robustness—the ability of a system to cope with errors during execution and cope with erroneous input [113]; 3) predicate invention—a way of finding new theoretical terms, or abstract new concepts that are not directly observable in the measured data [24]; and 4) scalability—it describes the capability of a system to cope and perform well under an increased or expanding workload or scope. Although this list can be extended with other features, we aim to include features that we believe are important for different ILP systems. The comparison of the systems is depicted in Table XI.

Recursive clauses are part of the hypothesis in some learning examples. In a recursive rule, one atom at least shows in both the head and the body. Although the ability to generate recursive rules is important to a system, it is often easier for an ILP system to generalize from small numbers of examples with recursion [114]. All models in Table XI could generate recursive rules, but traditional ILP systems struggle to learn recursive programs, especially from small numbers of training examples [115]. For the robustness of each system, all the traditional ILP systems, as well as Metagol and Popper, cannot be robust to noise in and mislabeled inputs. As PILP combines uncertainty with logic, while dNL-ILP and $\partial$ILP connect ILP with neural networks, they can deal with noise and ambiguity.

Predicate invention in ILP means the automatic introduction of new and hopefully useful predicates during learning from examples. Such new predicates can then be used as part of background knowledge in finding a definition of the target
predicate [24]. Traditional ILP systems have nonsupport for predicate invention. Since Metagol and Popper use high-order metarules to define the hypothesis space, the new predicate will be invented when induce definitions are necessary. \( \partial \)ILP and dNL-ILP also use rules as guidance to support predicate invention.

Existing ILP systems and Metagol cannot be applied effectively for data sets with 10,000 data points for each system’s scalability. Approximative generalization can compress several data points into one example to tackle large datasets in PILP [116]. \( \partial \)ILP is restricted to small datasets because of its memory requirements. The scalability of \( \partial \)ILP is not satisfied due to the predicates constraints (only two predicates in each clause are allowed in \( \partial \)ILP to avoid the memory-consuming problem). It should be good if the memory-consuming problem can be solved, since gradient descent used in \( \partial \)ILP performs well under an increased workload.

F. Summary

We test several systems in this section by adjusting the following: 1) optimal solution size; 2) optimal domain size; 3) example size; and 4) benchmark ILP problems. We broadly summarize the learning times and predictive accuracies of all models in the experiments (see Table XII): Although Aleph is faster than all the other systems in benchmark ILP problems, it only learns accurate solutions for grandparent and undirected edge. Popper can be roughly considered an upgrade of Metagol: by learning from failure, Popper finds accurate solutions with small time-consuming for most of the experiments. The performance of Popper highly relies on initial restrictions, while the performance of Metagol heavily depends on metarules. \( \partial \)ILP and dNL-ILP only handle benchmark ILP problems, although they produce reliable results.

VII. CHALLENGES

In this section, we share challenges facing the community based on the recent developments surveyed in this article.

A. Data Efficiency

Symbolic methods deal with small amounts of data efficiently, while the leading approaches in neural models need large amounts of data to achieve outstanding performance. Meanwhile, symbolic methods have difficulty dealing with large datasets, but neural models address them easily. For SRL, the structure learning, i.e., in MLNs, is not scalable and inefficient for large amounts of data as well. This has triggered the development of NeSy and SRL. To learn the benefits of each model and overcome their complimentary flaws are potential research directions.

B. Generalization

Generalization refers to the capacity of models to adapt to new, previously unseen data derived from the same distribution as the one used to create the model. Understanding generalization is one of the fundamental unsolved problems for all machine learning models. We found that several symbolic-based models suffer from weak generalization. For instance, Metagol sometimes cannot generate valid inference rules when changing the number range from 10 to 20 for positive and negative examples in the even dataset. Although the search method during training can be fully guided and understood, enough attention should be paid to the failure of generalization that may be caused only by a tiny change in the input.

C. Application

From an application point of view, the usage of traditional symbolic-based applications is very limited, and applications in SRL and NeSy are rarely developed. As the integration of symbolic with probability and neural networks, hybrid methods can be adapted to new fields, such as computer vision, natural language processing, and so on. As explainability and logic play crucial roles in many fields, methods in SRL and NeSy will be challenging and have great potential for the field.
D. Explainability

Symbolic AI holds certain merits in explainability due to its symbolic nature. Although AI users will increasingly require explanations, the current research in symbolic AI and the related integrations need to pay more attention to the target users. Recent work [108] shows a new explainability pipeline, which can be applied to symbolic AI. Another work [117] investigates the explanatory effects of a machine-learned theory in the context of simple two-person games and proposes a framework for identifying the harmfulness of machine explanations based on the cognitive science literature. However, more work is required for explanations in machine-learned theories, and the creation of different user-centered explainable solutions is required.

VIII. Conclusion

The increasing interests in symbolic AI and symbolic-based integration have been shown by various communities. This survey presents an overview of the symbolic family of algorithms, especially ILP. Traditional ILP holds certain merits on machine learning algorithms’ explainability due to its symbolic nature and variants of ILP: e.g., PILP, MIL, and δILP maintain interpretations and add powerful new features. The pros and cons of related research areas, such as SRL and NeSy, are also investigated. Combining user-centered explainable AI with ILP is also reviewed, as we consider user-centered design the next frontier of AI. We believe that the character and development of integrated symbolic AI will have a significant impact on AI in the future.

REFERENCES

[1] A. Adadi and M. Berrada, “Peeking inside the black-box: A survey on explainable artificial intelligence (XAI),” IEEE Access, vol. 6, pp. 5218–5240, 2018.
[2] A. B. Arrieta et al., “Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” Inf. Fusion, vol. 58, pp. 82–115, Jun. 2020.
[3] D. Gunning and D. Aha, “Darpa’s explainable artificial intelligence (XAI) program,” AI Mag., vol. 40, no. 2, pp. 44–58, 2019.
[4] J. Furnkranz, T. Klieg, and H. Paulheim, “On cognitive preferences and the plausibility of rule-based models,” Mach. Learn., vol. 109, no. 4, pp. 1–46, 2019.
[5] R. Calegari, G. Ciato, E. Denti, and A. Omicini, “Logic-based technologies for intelligent systems: State of the art and perspectives,” Information, vol. 11, no. 3, p. 167, Mar. 2020.
[6] A. Cropper and S. Dumančić, “Inductive logic programming at 30: A new introduction,” 2020, arXiv:2008.07912.
[7] U. Schmid, “Inductive programming as approach to comprehensible machine learning,” in Proc. 7th Workshop Dyn. Knowl. Belief 6th Multi-Strategy Mach. Learn. Workshop, 1991, pp. 175–191.
[8] A. Cropper and S. D. Dumančić, “Learning large logic programs by going beyond entailment,” 2020, arXiv:2004.09855.
[9] S. H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad, “Meta-interpretable learning: Application to grammatical inference,” Mach. Learn., vol. 94, no. 1, pp. 25–49, Jan. 2014.
[10] L. De Raedt and K. Kersting, Probabilistic Inductive Logic Programming, Springer, 2008, pp. 1–27.
[11] R. Evans and E. Grefenstette, “Learning explanatory rules from noisy data,” J. Artif. Intell. Res., vol. 61, pp. 1–64, Jan. 2018.
[12] S. Muggleton, “Inductive logic programming,” New Gener. Comput., vol. 8, no. 4, pp. 295–318, Feb. 1991.
[13] E. H. Shortliffe and B. G. Buchanan, “A model of inexact reasoning in medicine,” Math. Biosci., vol. 23, pp. 351–379, Apr. 1975.
[14] M. S. Fox and J. McDermott, “The role of databases in knowledge-based systems,” in On Knowledge Base Management Systems. Springer, 1986, pp. 407–430.
[15] S. Slocombe, K. Moore, and M. Zelouf, “Engineering expert system applications,” in Proc. Annu. Conf. BCS Spec. Group Expert Syst., 1986.
[16] J. Hayes-Michie, “News from brainware,” Pragmatica, vol. 1, pp. 10–11, Jan. 1990.
[17] P. R. Quinlan, “Discovering rules from large collections of examples: A case study,” in Expert Systems in the Microelectronic Age, 1979.
[18] G. Plotkin, “Automatic methods of inductive inference,” Tech. Rep., 1972.
[19] E. Y. Shapiro, “Algorithmic program diagnosis,” in Proc. 9th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang., 1982, pp. 299–308.
[20] C. Sammut and R. B. Banerji, “Learning concepts by asking questions,” in Machine Learning: An Artificial Intelligence Approach, vol. 2, 1986, pp. 167–192.
[21] J. R. Quinlan, Learning Relations: Comparison of a Symbolic and a Connectionist Approach. Sydney, NSW, Australia: Univ. Sydney, 1989.
[22] S. Muggleton et al., Efficient Induction of Logic Programs. Citeseer, 1990.
[23] S. Muggleton and W. Buntine, “Machine invention of first-order predicates by inverting resolution,” in Machine Learning Proceedings, Amsterdam, The Netherlands: Elsevier, 1988, pp. 339–352.
[24] S. Muggleton et al., “ILP turns 20,” Mach. Learn., vol. 86, no. 1, pp. 3–23, Jan. 2012.
[25] J.-U. Kietz, “Some lower bounds for the computational complexity of inductive logic programming,” in Proc. Eur. Conf. Mach. Learn. Springer, 1993, pp. 115–123.
[26] S.-H. Nienhuys-Cheng and R. De Wolf, Foundations of Inductive Logic Programming, vol. 1228. Springer, 1997.
[27] A. Srivivasan, (1999). The Aleph Manual. [Online]. Available: http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph
[28] J. R. Quinlan, “Learning logical definitions from relations,” Mach. Learn., vol. 5, no. 3, pp. 239–266, 1990.
[29] I. Bratko, Prolog Programming for Artificial Intelligence. London, U.K.: Pearson, 2001.
[30] A. Cropper and S. H. Muggleton. (2016). Metagol System. [Online]. Available: https://github.com/metagol/metagol
[31] R. M. Smullyan, First-Order Logic. Courier Corporation, 1995.
[32] H.-D. Ebbinghaus, J. Flum, and W. Thomas, Mathematical logic. Springer, 2013.
[33] S. Muggleton and L. de Raedt, “Inductive logic programming: Theory and methods,” J. Log. Program., vol. 19, pp. 629–679, May 1994.
[34] L. De Raedt and M. Bruynooghe, “CLINT: A multi-strategy interactive concept-learner and theory revision system,” in Proc. Multi-Strategy Learn. Workshop, 1991, pp. 175–191.
[35] S. Muggleton, “Inverse entailment and Progol,” New Gener. Comput., vol. 13, nos. 3–4, pp. 245–286, Dec. 1995.
[36] N. Lavrač, S. Džeroski, and M. Grobelnik, “Learning nonrecursive definitions of relations with LINUS,” in Proc. Eur. Work. Session Learn. Springer, 1991, pp. 265–281.
[37] P. Idestam-Almquist, “Generalization under implication by using or-introduction,” in Proc. Eur. Conf. Mach. Learn. Springer, 1993, pp. 56–64.
[38] P. Idestam-Almquist, “Learning missing clauses by inverse resolution,” in Proc. FoCS, 1992, pp. 610–617.
[39] L. De Raedt, A. Kimmig, and H. Toivonen, “ProbLog: A probabilistic Prolog and its application in link discovery,” in Proc. IJCAI, Hyderabad, India, vol. 7, 2007, pp. 2462–2467.
[40] R. Manhaeve, S. S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt, “DeepProbLog: Neural probabilistic logic programming,” in Proc. Adv. Neural Inf. Process. Syst., vol. 31, 2018, pp. 3749–3759.
[41] A. Cropper and R. Morel, “Learning programs by learning from failures,” Mach. Learn., vol. 110, no. 4, pp. 801–856, Apr. 2021.
[42] L. De Raedt and S. Džeroski, “First-order jk-clausal theories are PAC-learnable,” Artif. Intell., vol. 70, nos. 1–2, pp. 375–392, Oct. 1994.
[43] G. D. Plotkin, “A note on inductive generalization,” Mach. Intell., vol. 5, no. 1, pp. 153–163, 1970.
[44] D. Koller and A. Pfeffer, “Learning probabilities for noisy first-order rules,” in Proc. IJCAI, 1997, pp. 1316–1323.
[45] L. Getoor, N. Friedman, D. Koller, and B. Taskar, “Learning probabilistic models of link structure,” J. Mach. Learn. Res., vol. 3, pp. 679–707, Dec. 2003.
[70] J. Devlin, J. Uesato, S. Bhupatiraju, R. Singh, A.-R. Mohamed, and T. Rocktäschel and S. Riedel, “End-to-end differentiable proving,” in Proc. Conf. Inductive Log. Program. Springer, 2004, pp. 270–287.

[75] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville, “Transforming graph data for statistical relational learning,” J. Artif. Intell. Res., vol. 45, no. 1, pp. 363–441, 2012.

[76] D. Koller, “Probabilistic relational models,” in Proc. Int. Conf. Inductive Log. Program. Springer, 1999, pp. 3–13.

[77] J. Chen, S. Muggleton, and J. Santos, “Learning probabilistic logic models from probabilistic examples,” Mach. Learn., vol. 73, no. 1, pp. 55–85, Oct. 2008.

[78] M. Jaeger, “Relational Bayesian networks,” in Proc. 13th Conf. Uncertainty Artif. Intell. Morgan Kaufmann Publishers Inc., 1997, pp. 266–273.

[79] B. Taskar, P. Abbeel, M.-F. Wong, and D. Koller, “Relational Markov networks,” in Introduction to Statistical Relational Learning, 2007, pp. 175–200.

[80] J. Neville and D. Jensen, “Relational dependency networks,” J. Mach. Learn. Res., vol. 8, pp. 653–692, Mar. 2007.

[81] D. Poole, “Probabilistic horn abduction and Bayesian networks,” Artif. Intell., vol. 64, no. 1, pp. 81–129, Nov. 1993.

[82] K. Kersting and L. De Raedt, “Bayesian logic programming: Theory and tool,” in Statistical Relational Learning, 2007, p. 291.

[83] M. Richardson and P. Domingos, “Markov logic networks,” Mach. Learn., vol. 62, no. 1, pp. 107–136, Feb. 2006.

[84] O. B. H. Langseth and T. Nielsen, “Structural learning in object models from probabilistic examples,” in Proc. 4th IEEE Conf. Data Mining, Apr. 2004, pp. 170–177.

[85] F. Riguzzi, “Learning logic programs with annotated disjunctions,” in Proc. Int. Conf. Inductive Log. Prog. Program. Springer, 2004, pp. 270–287.

[86] F. Riguzzi, “Learning logic programs with weighted Boolean formulas,” Theory Pract. Log. Program., vol. 15, no. 3, pp. 358–401, May 2015.

[87] M. Bošnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel, “Programming with a differentiable forth interpreter,” in Proc. 34th Int. Conf. Mach. Learn., vol. 70, 2017, pp. 547–556.

[88] T. Rocktäschel and S. Riedel, “End-to-end differentiable proving,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 3788–3800.

[89] A. Payani and F. Felek, “Inductive logic programming via differentiable deep neural logic networks,” 2019, arXiv:1906.05323.

[90] D. Koller et al., Introduction to Statistical Relational Learning. Cambridge, MA, USA: MIT Press, 2007.
