Prolog: Past, Present, and Future

Gopal Gupta[1]
Elmer Salazar[1], Farhad Shakerin[1], Joaquín Arias[2], Sarat Chandra Varanasi[1], Kinjal Basu[1], Huaduo Wang[1], Fang Li[1], Serdar Erbatur[1], Parth Padalkar[1], Abhiramon Rajasekharan[1], Yankai Zeng[1], and Manuel Carro[3]

[1]Department of Computer Science, UT Dallas, Richardson, USA
[2]CETINIA, Universidad Rey Juan Carlos, Madrid, Spain
[3]Universidad Politecnica de Madrid and IMDEA Software Institute

Abstract. We argue that various extensions proposed for Prolog—tabling, constraints, parallelism, coroutining, etc.—must be integrated seamlessly in a single system. We also discuss how goal-directed predicate answer set programming can be incorporated in Prolog, and how it facilitates development of advanced applications in AI and automated commonsense reasoning.

1 Introduction

The year 2022 was celebrated as the 50th anniversary of the founding of logic programming (LP) and Prolog [16]. Prolog harnesses the power of logic and provides a new declarative paradigm for computing. Initially, Prolog was based on Horn clauses with some built-ins added. Over time more features were added to make the language more powerful as well as efficient. These features include constraints over various types of domains (reals, booleans, finite domains, etc.), negation-as-failure, coroutining, tabling, parallelism, etc. Prolog has been applied to many (innovative) applications. Today, Prolog is a highly sophisticated language [34] with a large user base. Over the last fifty years, as one would expect, research in logic programming has flourished in three main areas: making Prolog more efficient, making Prolog more expressive, and developing applications that make use of logic programming technology. We will focus on the first two issues, as any discussion of applications of logic programming will take significantly more space. However, one of the applications of LP that we will discuss in this paper is automating *commonsense reasoning* with the aim of building systems that can emulate the thought process of an (unerring) human [37, 18, 47, 25]. We believe that logic programming is indispensable for this purpose. We assume that the reader is familiar with Prolog, logic programming, and answer set programming. An excellent, brief exposition is given in the introductory chapter of this book [51]. More detailed expositions can be found elsewhere [48, 39, 23].

1.1 Making Prolog more Efficient

Given a call, a Prolog interpreter finds matching clauses and tries them one by one via backtracking. Once a matching clause is selected, subgoals in the body of the clause are executed. The strategy for rule selection and subgoal selection

is called the *computation rule* [39]. Prolog uses a computation rule that tries clauses in textual order, while subgoals in a clause are tried left to right. A considerable amount of research has been undertaken over the past decades to improve the rule selection process. Executing Prolog programs in or-parallel—trying multiple matching clauses simultaneously on multiple processors—can be regarded as a strategy to improve rule-selection. Tabled logic programming can also be viewed as a rule-selection strategy, where rules are (optionally) selected in a non-textual order to ensure termination of left-recursive programs, for example. Research has also been undertaken over the past several decades to improve the subgoal selection process. Research in goal selection strategies includes constraint logic programming (where the *generate and test* strategy is flipped into *test and generate*), coroutining, concurrent logic programming, and and-parallelism. These efforts in improving rule selection and subgoal selection have resulted in Prolog systems that are highly effective [34].

Unfortunately, various strategies developed to make execution more efficient are not all available in a single Prolog system *where they work seamlessly with each other*. We believe that future research in Prolog must focus on building a unified system that supports tabling, constraints, coroutining, parallelism, and concurrency [26]. These enhancements must also work seamlessly with each other. Many groups have been steadfastly working towards this goal. These include, among others, efforts by Arias and Carro to integrate constraints in tabled logic programming [4] and by Rocha and Santos Costa to combine tabling and or-parallelism [44]. Research has also been conducted on adding concurrency at the user level to Prolog [12, 3, 17]. Some of these ideas have been incorporated in systems such as SWI-Prolog [52] and Ciao-Prolog [30], nevertheless, research to realize a system where all these advanced features are efficiently and seamlessly integrated and available in a single system must continue. Our hope is that such a logic programming system will be realized in the future.

1.2 Making Prolog More Expressive

We next consider research on making Prolog more expressive. Prolog is a Turing-complete language, so by greater expressiveness we mean the ability to represent and solve problems more elegantly, i.e., the logic program developed to solve a problem is "close" to the problem specification. A large segment of this research is dedicated to adding negation-as-failure (NAF) to logic programming, though, considerable research has been done in devising other extensions, e.g., constraint handling rules [20], functional-logic programming [29], higher order LP [14], coinductive logic programming [46], and adding assertions [31]. Due to lack of space, we will primarily focus on the incorporation of NAF and coinduction into logic programming, one of the reasons being that they help in realizing (predicate) ASP within Prolog, critical for automating commonsense reasoning. [37, 18, 47].

Stable models semantics [24] led to the paradigm of Answer Set Programming (ASP) [10]. Commonsense reasoning, realized via default reasoning, imposing integrity constraints, and assumption-based reasoning, can be elegantly emulated in ASP [23, 42]. However, a problem faced by ASP is the following: how to execute answer set programs in the presence of predicates? ASP researchers resorted

to allowing only propositional programs. Thus, ASP programs containing predicates have to be grounded first so that they become propositional and then a SAT-solver is used to find its (multiple) models. This leads to a number of restrictions including: (i) programs have to be finitely groundable, (ii) data structures such as lists are not permitted, (iii) program size blows up exponentially during grounding, (iv) real numbers cannot be faithfully represented. Thus, while ASP enhances Prolog, it also restricts it due to the choice of implementation mechanism used. While extremely efficient propositional ASP systems such as CLINGO [22] have been developed, restriction to propositions-only programs make them hard to use for knowledge representation applications, in particular, modeling commonsense reasoning, which is often *query-driven* or *goal-directed*. To overcome this, stable model semantics-based NAF must be incorporated in Prolog. The discovery of coinductive logic programming [46] led to development of query-driven implementations of ASP called s(ASP) [40] and s(CASP) [6]. The s(ASP) and s(CASP) systems allow predicates, do not require grounding, and can be thought of as extending Prolog with stable model semantics-based negation. Thus, default rules with exceptions, integrity constraints, and cyclical reasoning through negation can be elegantly supported in this extended Prolog, thereby supporting automation of commonsense reasoning. The s(ASP) and s(CASP) systems rely on many advanced techniques such as constructive negation, dual rule generation, coinduction, etc., and are scalable.

2 Emulating Human Thinking with Logic Programming

Logic programming was conceived as a language for problem-solving, AI, and emulating human thinking [38, 37]. However, Prolog's inability to effectively model incomplete information limited its use for AI and emulating human reasoning, which became an impetus for significant subsequent research by various groups [25, 18, 23, 37]. Negation-as-failure is an important component in these research efforts and ASP is an important effort in this direction. ASP extends logic programming with stable model semantics-based NAF [10, 23]. ASP allows reasoning with incomplete information through NAF and defaults. ASP also supports integrity constraints, and non-inductive semantics in which multiple models (worlds) are admitted. Thus, ASP is a paradigm that comes close to supporting commonsense reasoning and emulating the human thought process [23, 27]. It is our position that the path to automating commonsense reasoning in a practical manner goes through Answer Set Programming realized via Prolog-like implementations of predicate ASP such as the s(CASP) system [6, 40]. By Prolog-like, we mean that predicates and first order terms are supported, and execution is query-driven, carried out in a top-down manner. Thus, the power of ASP, i.e., negation based on stable model semantics, is supported within Prolog.

We strongly believe that the best path to building intelligent systems is by emulating how intelligent behavior manifests in humans. Humans use their senses (sight, sound, smell, taste, and touch) to acquire information via pattern matching, but to draw conclusions based on this information, humans use (commonsense) reasoning. Note that:

1. Machine learning technologies are akin to human sensing and pattern matching (humans learn by observing patterns through use of various senses). Machine learning technologies have greatly advanced in the last few years.
2. Commonsense reasoning relates to human thinking. ASP and systems such as s(CASP) provide the wherewithal to elegantly automate commonsense reasoning.

Note that sensing and pattern matching corresponds to Kahneman's System 1 thinking and reasoning to Kahneman's System 2 thinking [33]. Just as it is hard for a human to explain the information they have acquired through senses (for example, it will be very hard for someone to explain why they believe that the sound they heard is the sound of a siren), explaining their own decisions has been a problem for machine learning systems. Even understanding natural language involves sensing and pattern matching. Humans hear or read a sentence and are quickly able to understand the knowledge implicit in that sentence. Note that commonsense knowledge may also be used in this sensing and pattern matching process [53]. The knowledge acquired through sensing and pattern matching is represented in some manner in our mind. Next, to draw further conclusions, humans perform reasoning over this knowledge that resides in the mind. This reasoning may also involve using (additional) commonsense knowledge that has been acquired over a period of time and that also resides in our mind in some form. For example, once the sound of the siren is heard, its occurrence is represented in our mind as knowledge. This knowledge may prompt us to find a safe spot as we know that the siren sound is announcing a tornado in the area (commonsense knowledge for those who live in the plains of Texas).

We believe that AI systems can be built better by using (i) machine learning technologies for sensing (seeing, hearing, etc.), and (ii) goal-directed ASP systems such as s(CASP) for reasoning [27]. This is in contrast to using machine learning alone. In this framework, machine learning systems or a neurosymbolic system [53] will translate a picture, video, sound, text, etc., to knowledge expressed as predicates. These predicates capture the relevant knowledge in a manner similar to how humans represent knowledge in their mind. This knowledge gleaned from "senses" and represented as predicates is further augmented with commonsense knowledge expressed in s(CASP) as default rules, integrity constraints, etc. In real life, commonsense knowledge is learned and stored in our mind throughout our life. Next, a question that we may want to answer against this combined knowledge can be treated as a query, and executed using the s(CASP) system. This framework can be used to develop, for example, an autonomous driving system [35] or a goal-oriented interactive conversational agent that can actually "understand" human dialogs [43, 55] .

2.1 Deduction, Abduction, and Induction

A significant part of commonsense reasoning can be emulated with defaults, integrity constraints, and assumption-based reasoning [23, 37, 18, 47]. ASP obviously supports deduction. Default rules can be viewed as inductive generalizations, and assumption-based reasoning can be viewed as abduction. Thus, the

three major modes of reasoning—deduction, abduction, and induction[19]—are naturally supported within ASP. Consider the proposition p, q, and the formula $p \Rightarrow q$.

Deduction: Given premises p and $p \Rightarrow q$, we *deduce* q. Suppose we are given the premises that Tweety is a bird (*bird(tweety)*), and the formula $\forall X bird(X) \Rightarrow flies(X)$. From these two premises, we can deduce that *flies(tweety)* holds, i.e., Tweety can fly. Obviously, such deductive reasoning is easily expressed in ASP.

Abduction: Given the observation q and the premise $p \Rightarrow q$, we *abduce* p. Suppose we observe Tweety flying (*flies(tweety)*), and we know that $\forall X bird(X) \Rightarrow flies(X)$. From these, we can *abduce* that *bird(tweety)* holds, i.e., we assume (or advance the most likely explanation) that Tweety is a bird. Note that there may be other explanations, e.g., Tweety may be the name of an airplane. Generally, the set of abduced literals is fixed in advance. Abductive reasoning in ASP is elegantly modeled via possible worlds semantics. If we make an assumption `p`, i.e., declare `p` to be abducible, then we can assert via an *even loop over negation*:

```
p :- not _notp.        _notp :- not p.
```

where `_notp` is a dummy proposition. This even loop will result in two possible worlds: one in which `p` is true and one in which `p` is false. It should be noted that abductive reasoning is, essentially, assumption-based reasoning. Humans perform assumption-based reasoning all the time [18].

Induction: Given instances of p and corresponding instances of q that may be related, we may induce $p \Rightarrow q$. Thus, given the observations that Tweety is a bird and Tweety can fly, Sam is a bird and Sam can fly, Polly is a bird and Polly can fly, and so on, we may *induce* the formula *bird(X)* $\Rightarrow$ *flies(X)*. Induction, of course, relates to learning associations between data. Induced rules can be elegantly captured as an answer set program. This is because ASP can be used to represent defaults with exceptions, which allows us to elegantly represent inductive generalizations. Consider the following rule:

```
flies(X):- bird(X), not abnormal_bird(X).
abnormal_bird(X):- penguin(X).
```

The above default rule with exception, namely, *normally birds fly unless they are penguins*, elegantly captures the rule that a human may form in their mind after observing birds and their ability to fly. The list of exceptions can grow (ostrich, wounded bird, baby bird, ...). Similarly, the list of defaults rules can grow. For instance, we may have a separate rule for planes being able to fly. Explainable machine learning tools that induce default theories have been developed [49, 50] that are comparable in accuracy to traditionally popular tools such as XGBoost [13] and Multilayer Perceptron [1, 28].

Given that deduction, abduction, and induction fit into the framework of ASP well, it gives us confidence that ASP can be a good means of representing commonsense knowledge and reasoning over it.

2.2 Representing Commonsense Knowledge in ASP/s(CASP)

As stated earlier, a large portion of the human thought process can be largely emulated by supporting (i) default rules with exceptions and preferences, (ii)

integrity constraints, and (iii) multiple possible worlds. As explained earlier, default rules with exceptions express inductive generalizations, and are used by humans for making deductions. Similarly, multiple possible worlds help in abduction, or assumption-based reasoning. Integrity constraints allow us to prune unfruitful paths in our abductive reasoning process. Unfruitful paths in the deductive reasoning process are pruned by adding conditions to rule bodies.

Default Reasoning with Exceptions and Preferences: Humans use *default reasoning* [23] to jump to conclusions. These conclusions may be revised later in light of new knowledge. For example, if we are told that Tweety is a bird, and then asked whether Tweety flies, we will immediately answer, yes, it does. However, later if we are told that Tweety is a penguin, we will withdraw the conclusion about Tweety's flying ability, labeling Tweety as an *exception*. Thus, human reasoning is non-monotonic in nature, meaning that conclusions may be withdrawn as new knowledge becomes available. Humans use this sort of *default reasoning* to jump to conclusions all the time, and if they find the assumptions made to jump to this conclusion to be incorrect, they revise their conclusion. Multiple default conclusions can be drawn in some situations, and humans will use additional reasoning to *prefer* one default over another. Thus, default rules with exceptions and preferences capture most of the deductive reasoning we perform. (More details on default reasoning can be found elsewhere [23, 37]). It should be noted that expert knowledge is nothing but a set of default rules about a specialized topic [23].

Classical logic is unable to model default reasoning and non-monotonicity in an elegant way. We need a formalism that is non-monotonic and can support defaults to model commonsense reasoning. ASP is such a formalism. ASP supports both NAF (`not p`) as well as *strong negation* (`-p`), where `p` is a proposition or a predicate. A strongly negated predicate has to be explicitly defined, just as positive predicates are. Combining these two forms of negations results in nuanced reasoning closer to how humans reason:

1. `p`: denotes that `p` is *definitely* true.
2. `not -p`: denotes that `p` *maybe* true (i.e., no evidence that `p` is false).
3. `not p` $\wedge$ `not -p`: denotes that `p` is unknown (i.e., no evidence of either `p` or `-p` being true).
4. `not p`: denotes that `p` *may* be false (no evidence that `p` is true).
5. `-p`: denotes that `p` is *definitely* false.

The above insight can be used, for example, to model the exceptions to Tweety's ability to fly in two possible ways. Consider the rules:

```
flies(X):- bird(X), not abnormal_bird(X).      % default
abnormal_bird(X):- penguin(X).                 % exception
```

which state that if we know nothing about a bird, `X`, we conclude that it flies. This is in contrast to the rules:

```
flies(X):- bird(X), not abnormal_bird(X).      % default
abnormal_bird(X):- not -penguin(X).            % exception
```

which states that a bird can fly only if we can *explicitly* rule out that it is a penguin. So in the latter case, if we know nothing about a bird, we will conclude that it *does not* fly. Which of the two rules one will use depends on how conservative or aggressive one wants to be in jumping to the (default) conclusion. Note that exceptions can have exceptions, which in turn can have their own exceptions, and so on. For example, animals normally don't fly, unless they are birds. Thus, birds are exception to the default of not flying. Birds, in turn, normally fly, unless they are penguins. Thus, a penguin is an exception to the exception for not flying. Defaults, exceptions, exceptions to exceptions, and so on, allow humans to perform reasoning elegantly in an *elaboration tolerant* manner [7, 23].

Integrity Constraints: ASP can also model integrity constraints elegantly. An integrity constraint is a rule of the form:

`false:-` $\mathtt{p}_1$`,` $\mathtt{p}_2$`, ...,` $\mathtt{p}_n$`.`

which states that the conjunction of $\mathtt{p}_1$, $\mathtt{p}_2$, through $\mathtt{p}_n$ is false (the keyword `false` is often omitted). Integrity constraints elegantly model global invariants or restrictions that our knowledge must satisfy, e.g., `p` and `-p` cannot be true at the same time, denoted

```
false:- p, -p.
```

Humans indeed use integrity constraints in their everyday reasoning: as restrictions (two humans cannot occupy the same spot) and invariants (a human must breath to stay alive). Note that integrity constraints are *global* constraints, in that they eliminate possible worlds. Unfruitful paths during deductive reasoning are eliminated by adding appropriate conditions to the rule-bodies. Note that in ASP, integrity constraints may also arise due to *odd loops over negation* (OLON), i.e., rules of the form:

$\mathtt{p}(\bar{t})$ `:- G, not` $\mathtt{p}(\bar{t})$`.`

where $\mathtt{p}(\bar{t})$ is a predicate and `G` is a conjunction of goals. In absence of an alternative proof for $\mathtt{p}(\bar{t})$, the only admissible model for the above rule is $\mathrm{p}(\bar{t}) =$ false, `G` = false, which amounts to the global constraint that `G` must be false.

Possible Worlds: Humans can represent *multiple possible worlds* in parallel in their minds and reason over each. For example, in the real world, birds do not talk like humans, while in a cartoon world, birds (cartoon characters) can talk. Humans can maintain the distinction between various worlds in their minds and reason within each one of them. These multiple worlds may have aspects that are common (birds can fly in both the real and cartoon worlds) and aspects that are disjoint (birds can talk only in the cartoon world). Unlike Prolog, ASP/s(CASP) support multiple possible worlds. (See also the example about Annie and David teaching programming languages in the introductory paper of this volume [51]).

3 The s(CASP) System

The s(CASP) system [6] supports predicates, constraints over non-ground variables, uninterpreted functions, and, most importantly, a top-down, query-driven execution strategy for ASP. These features make it possible to return answers with non-ground variables (possibly including constraints among them) and compute partial models by returning only the fragment of a stable model that is

necessary to support the answer to a given query. The s(CASP) system supports constructive negation based on a disequality constraint solver, and unlike Prolog's negation as failure and ASP's default negation, `not p(X)` can return bindings for `X` on success, i.e., bindings for which the call `p(X)` would have failed.

The s(CASP) system is based on the earlier s(ASP) system [40], and also supports constraints over reals. The s(CASP) system provides support for full Prolog, however, in addition, it also supports coinductive (circular, or assumption-based) reasoning, constructive negation, dual rules, and support for universally quantified variables. These are explained briefly next. More details can be found elsewhere [40, 6, 5].

Coinductive Reasoning: Coinductive reasoning is crucial for s(CASP). Answer set programs may contain circular rules, for example:

```
p:- not q.          q:- not p.
```

If we ask the query `?-p` in Prolog with these rules, execution will loop forever. This is because `p` calls to `not q`, which calls `q`, which calls `not p`, which then calls `p`. If we allow coinductive or circular reasoning[46, 45], then the query `p` should succeed. Essentially, we are stating that `p` succeeds if we *assume* `p` to hold. This yields the answer set in which `p` is true and `q` false. Note also that at least one intervening negation is required between a call and its recursive descendent for coinductive success in s(CASP). This prevents *positive loops*, i.e., loops with no intervening negation, from succeeding, and allows us to stay faithful to stable model semantics (in contrast, such positive loops will not terminate under *completion semantics* as realized in Prolog). Thus, given the rule:

```
p :- p.
```

the query `?- p.` will fail in s(CASP), while the query `?- not p.` will succeed [40, 6]. More details can be found elsewhere [40, 45].

Constraints and OLON rules: Global constraints of the form

`false :-` $p_1(\bar{t}_1)$`, ...,` $p_n(\bar{t}_n)$`.`

are suitably transformed and appended to each top level query to ensure that each constraint is enforced. Global constraints can also implicitly arise due to OLON rules. These are analyzed at compile time and the appropriate constraint generated and appended to a top-level query [40, 6].

Constructive Negation: Since s(CASP) allows general predicates that could be negated, support for constructive negation becomes essential. Consider a program consisting of the simple fact:

```
p(a).
```

If we pose the query `?-not p(X)`, it should succeed with answer `X` $\neq$ `a`. Intuitively, `X` $\neq$ `a` means that `X` can be any term not unifiable with `a`. To support constructive negation, the implementation has to keep track of values that a variable cannot take. The unification algorithm has to be extended, therefore, to account for such disequality-constrained values. The s(CASP) system incorporates [40, 6] constructive negation.

Dual Rules: ASP assumes that programs have been *completed* [39, 23]. To complete a program, the s(CASP) system will add *dual* rules [2] to the program. The procedure to add the dual rules is relatively simple and can be found elsewhere

[6]. An additional complication in computing the dual rules is the need to handle existential variables. Consider the following very simple rule:

```
p(X):- not q(X,Y).
```

This rule corresponds to the Horn clause:

$\forall X(p(X) \Leftarrow \exists Y\ not\ q(X,Y))$

Its dual will be:

$\forall X(not\_p(X) \Leftarrow \forall Y\ q(X,Y))$

which, in s(CASP), will be represented as:

```
not_p(X):- forall(Y, q1(X,Y)).        q1(X,Y):- q(X,Y).
```

Universal quantification in the body of the dual rule is needed because, for example, for the goal `not p(a)` to succeed, we must prove that `q(a, Y)` holds for every possible value of `Y`. The s(CASP) system handles all these issues and produces dual rules for arbitrary programs. The execution of the *forall*, however, is non-trivial, as often times the foralls are nested.

4 Applications

Several advanced applications that automate commonsense reasoning using ASP and s(CASP) have been developed. Most prominent is the CHeF system [15] which emulates the expertise of a cardiologist to automatically generate treatment for congestive heart failure. Our studies show that the CHeF system performs on par with a cardiologist. The s(CASP) system has also been used by others to develop intelligent applications [41, 36]. With respect to the framework above, where we use machine learning for sensing and pattern matching and s(CASP) for commonsense reasoning, two major strands have been pursued.

Image Understanding: A major task in AI is to understand a picture and answer questions about it. Current approaches for visual question answering (VQA) are solely based on machine learning. These approaches train on a collection of images together with the question-answer pairs for each image. Once the model has been learned, a new image along with a question is given, and the expectation is that a correct answer will be generated. Given that a generated answer cannot be justified, or may not be accurate, an alternative approach is to use machine learning for translating an image into a set of predicates that capture the objects in the image, their characteristics and spatial relationships. Commonsense knowledge about the image's domain can be coded in ASP. Next, the question to be answered is translated into an ASP query, which is then executed using s(CASP) against the knowledge (represented as predicates) captured from the image augmented with commonsense knowledge of the domain. 100% accuracy in answering questions is achieved in some of the VQA datasets [8]. The process is similar to how humans answer questions about an image, and can be leveraged to realize reliable autonomous driving systems [35].

Natural Language Understanding (NLU): A combination of machine learning and commonsense reasoning can be used for NLU as well. The idea is to generate predicates from text using large language models such as GPT-3 [11] via the use of *in-context learning* or *fine-tuning*. These predicates represent the meaning of the sentence, i.e., its *deep structure*. Commonsense reasoning can

then be performed over these predicates to draw further conclusions, ask for missing information, and check for consistency of the information in the sentence. We have used this approach for qualitative reasoning, solving simple word problems in Algebra, and developing conversational agents that can interact with a human while "understanding" what he/she is saying [43, 55]. Essentially, we emulate how a person understands sentences and carries on a conversation. Commonsense knowledge is also embedded in the LLM, so just like humans, commonsense knowledge is used at two levels—in "understanding" text as predicates and subsequent reasoning.

5 Conclusion

Learning/pattern-matching and reasoning are crucial to human intelligence. It is our belief that effective AI systems can only be obtained by combining machine learning for sensing/pattern-matching and a Prolog system that encapsulates ASP, such as s(CASP), for commonsense reasoning. While the applications developed so far are in narrow domains, it is our position that the path to building AI applications that perform as well as humans goes through logic programming. Machine learning *alone* cannot be used for modeling human thinking, as fundamentally it is a statistical technique. If this was indeed possible, then we believe that nature would have already produced an intelligent being—as intelligent as humans, or more—based on pattern matching and operating on instincts alone. As we move up the evolutionary chain towards more intelligent life-forms culminating in humans, reasoning abilities improve. "Lower" life forms sometimes do have better sensing capabilities, e.g., dogs can smell better, eagles can see better, etc., but a combination of instinct and reasoning puts humans on top of the evolutionary chain. Therefore, just as humans rely on both learning/sensing/pattern-recognition *and* reasoning, an AI system that aims to achieve human-level performance must follow the same path [53, 21]. This is also evident in large language models such as GPT-3 and ChatGPT [11] that use pattern matching on a massive scale to generate a human-like response. Due to the statistical nature of LLMs, we can never be certain that the generated text is correct, consistent, and useful [9]. Logic is essential for producing a consistent, correct, and assuredly-useful response.

In conclusion, Prolog is an indispensable part of computing's and AI's landscape. We believe that it is an essential component in achieving Artificial General Intelligence (AGI) [54, 32]—AI's purported holy grail for some—whether we agree with that goal or not. Considerable research is still needed to: (i) improve the s(CASP) system (by incorporating tabling, constraints, coroutining, etc.) and making it more efficient; (ii) develop machine learning systems that extract knowledge as predicates from arbitrary text & images; and, (iii) develop methods to automatically extract commonsense knowledge and represent it in s(CASP). We hope that these tasks will be completed in the coming years.

Acknowledgements: We are grateful to the anonymous reviewers and to David S. Warren for insightful comments and suggestions that resulted in significant

improvements to the paper. Authors acknowledge partial support from NSF grants IIS 1910131, IIP 1916206, and US DoD.

References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning - A Textbook*. Springer, 2018.

[2] José Júlio Alferes, Luís Moniz Pereira, and Terrance Swift. "Abduction in well-founded semantics and generalized stable models via tabled dual programs". In: *TPLP* 4.4 (2004), pp. 383–428.

[3] Miguel Areias and Ricardo Rocha. "Multi-dimensional lock-free arrays for multithreaded mode-directed tabling in Prolog". In: *Concurr. Comput. Pract. Exp.* 31.5 (2019).

[4] Joaquín Arias and Manuel Carro. "Description, Implementation, and Evaluation of a Generic Design for Tabled CLP". In: *TPLP* 19.3 (2019), pp. 412–448.

[5] Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. "Modeling and Reasoning in Event Calculus using Goal-Directed Constraint Answer Set Programming". In: *TPLP* 22.1 (2022), pp. 51–80.

[6] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. "Constraint Answer Set Programming without Grounding". In: *TPLP* 18.3-4 (2018), pp. 337–354.

[7] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[8] Kinjal Basu, Farhad Shakerin, Gopal Gupta, and Author Unknown. "AQuA: ASP-Based Visual Question Answering". In: *Proc. PADL'20*. Springer LNCS, pp. 57–72.

[9] Ali Borji. *A Categorical Archive of ChatGPT Failures*. Preprint arXiv:2302.03494. 2023.

[10] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. "Answer set programming at a glance". In: *Commun. ACM* 54.12 (2011), pp. 92–103.

[11] Tom Brown, Benjamin Mann, and Others. "Language Models are Few-Shot Learners". In: *Proc. NeurIPS*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[12] Manuel Carro and Manuel V. Hermenegildo. "Concurrency in Prolog Using Threads and a Shared Database". In: *Proc. ICLP*. Ed. by Danny De Schreye. MIT Press, 1999, pp. 320–334.

[13] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proc. 22nd ACM SIGKDD*. KDD '16. 2016, pp. 785–794.

[14] Weidong Chen, Michael Kifer, and David Scott Warren. "HILOG: A Foundation for Higher-Order Logic Programming". In: *J. Log. Program.* 15.3 (1993), pp. 187–230.

[15] Zhuo Chen, Kyle Marple, Elmer Salazar, Gopal Gupta, and Lakshman Tamil. "A Physician Advisory System for Chronic Heart Failure man-

agement based on knowledge patterns". In: *Theory Pract. Log. Program.* 16.5-6 (2016), pp. 604–618.
[16] Alain Colmerauer and Philippe Roussel. "The Birth of Prolog". In: *History of Prog. Lang. Conf. (HOPL-II)*. ACM, 1993, pp. 37–52.
[17] Vítor Santos Costa, Inês de Castro Dutra, and Ricardo Rocha. "Threads and or-parallelism unified". In: *Theory Pract. Log. Program.* 10.4-6 (2010), pp. 417–432.
[18] E.A. Dietz Saldanha, S. Hölldobler, and L.M. Pereira. "Our Themes on Abduction in Human Reasoning: A Synopsis". In: *Abduction in Cognition and Action: Logical Reasoning, Scientific Inquiry, and Social Practice*. 2021, pp. 279–293.
[19] Peter A. Flach and Antonis C. Kakas. *Abductive and Inductive Reasoning: Background and Issues*. Springer Netherlands, 2000.
[20] Thom W. Frühwirth. "Theory and Practice of Constraint Handling Rules". In: *J. Log. Program.* 37.1-3 (1998), pp. 95–138.
[21] Artur d'Avila Garcez and Luis C. Lamb. *Neurosymbolic AI: The 3rd Wave*. 2020. arXiv: 2012.05876 [cs.AI].
[22] Martin Gebser et al. "Potassco: The Potsdam Answer Set Solving Collection". In: *AI Communications* 24.2 (2011), pp. 107–124. DOI: 10.3233/AIC-2011-0491.
[23] Michael Gelfond and Yuliya Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: An Answer Set Programming Approach*. Cambridge Univ. Press, 2014.
[24] Michael Gelfond and Vladimir Lifschitz. "The stable model semantics for logic programming." In: *ICLP/SLP*. Vol. 88. 1988, pp. 1070–1080.
[25] David Gunning, Vinay K. Chaudhri, Peter Clark, Benjamin Grosof, and Others. "Project Halo Update - Progress Toward Digital Aristotle". In: *AI Mag.* 31.3 (2010), pp. 33–58.
[26] Gopal Gupta. *Next Generation of Logic Programming Systems*. Tech. rep. Dept. of Comp. Sci., UT Dallas. 2003.
[27] Gopal Gupta et al. "Automated Commonsense Reasoning". In: *Proc. GDE'22*. https://utdallas.edu/~gupta/csr-scasp.pdf. 2022.
[28] Gopal Gupta et al. "Logic-based Explainable and Incremental Machine Learning". In: *Prolog50: Fifty Years of Prolog*. Springer LNCS 13900, (this proceedings), pp. 16–28.
[29] Michael Hanus. "From Logic to Functional Logic Programs". In: *Theory Pract. Log. Program.* 22.4 (2022), pp. 538–554.
[30] Manuel V. Hermenegildo et al. "An overview of Ciao and its design philosophy". In: *Theory Pract. Log. Program.* 12.1-2 (2012), pp. 219–252.
[31] M.V. Hermenegildo, J.F. Morales, P. Lopez-Garcia, and M. Carro. "Types, modes and so much more – the Prolog way". In: *Prolog - The Next 50 Years*. Ed. by David S. Warren et al. LNCS 13900. Springer, 2023.
[32] Marcus Hutter. *Universal Artificial Intellegence - Sequential Decisions Based on Algorithmic Probability*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005.

[33] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.
[34] Phillip Körner et al. "Fifty Years of Prolog and Beyond". In: *Theory and Practice of Logic Programming* (2022), 1–83.
[35] Suraj Kothawade, Vinaya Khandelwal, Kinjal Basu, Huaduo Wang, and Gopal Gupta. "AUTO-DISCERN: Autonomous Driving Using Common Sense Reasoning". In: *Proc. ICLP Workshops: GDE'21*. Vol. 2970. CEUR Workshop Proceedings. CEUR-WS.org, 2021.
[36] Robert Kowalski, Jacinto Davila, Galileo Sartor, and Miguel Calejo. "Logical English for Law and Education". In: *Prolog - The Next 50 Years*. Ed. by David S. Warren et al. LNCS 13900. Springer, 2023.
[37] Robert A. Kowalski. *Computational Logic and Human Thinking*. Cambridge University Press, 2011.
[38] Robert A. Kowalski. *Logic for Problem Solving*. North Holland, 1979.
[39] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.
[40] Kyle Marple, Elmer Salazar, Gopal Gupta, and A Unknown. *Computing stable models of normal logic programs without grounding*. Preprint arXiv:1709.00501. 2017.
[41] Jason Morris. "Blawx: User-friendly Goal-Directed Answer Set Programming for Rules as Code". In: *Proc. Prog. Lang. and the Law (ProLaLa)*. 2023.
[42] Erik T. Mueller. *Commonsense reasoning: an event calculus based approach*. Morgan Kaufmann, 2014.
[43] A. Rajasekharan, Y. Zeng, P. Padalkar, and G. Gupta. *Reliable Natural Language Understanding with Large Language Models and Answer Set Programming*. Preprint arXiv:2302.03780; to appear in Proc. ICLP'23 (Tech. Comm.) 2023.
[44] Ricardo Rocha, Fernando M. A. Silva, and Vítor Santos Costa. "On Applying Or-Parallelism and Tabling to Logic Programs". In: *Theory Pract. Log. Program.* 5.1-2 (2005), pp. 161–205.
[45] Elmer Salazar. "Proof-theoretic Foundations of Normal Logic Programs". PhD thesis. Department of Computer Science, Univ. of Texas at Dallas., 2019.
[46] Luke Simon, Ajay Bansal, Ajay Mallya, and Gopal Gupta. "Co-Logic Programming: Extending Logic Prog. with Coinduction". In: *Proc. ICALP'07*. LNCS 4596. Springer, pp. 472–483.
[47] Keith Stenning and Michiel van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, 2008.
[48] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MITPress, 1994.
[49] Huaduo Wang and Gopal Gupta. "FOLD-R++: A Scalable Toolset for Automated Inductive Learning of Default Theories from Mixed Data". In: *Proc. FLOPS'22*. Vol. 13215. LNCS. Springer, 2022, pp. 224–242.
[50] Huaduo Wang, Farhad Shakerin, and Gopal Gupta. "FOLD-RM: Efficient Scalable Explainable AI". In: *TPLP* 22.5 (2022), pp. 658–677.

[51] David S. Warren. "Introduction to Prolog". In: *Prolog - The Next 50 Years*. Ed. by David S. Warren et al. LNCS 13900. Springer, 2023.

[52] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. "SWI-Prolog". In: *Theory Pract. Log. Program.* 12.1-2 (2012), pp. 67–96.

[53] Wikipedia. *Neurosymbolic AI*. https://en.wikipedia.org/wiki/Neuro-symbolic_AI#. retrieved February, 2022.

[54] Wikipedia contributors. *Artificial general intelligence — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-April-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Artificial_general_intelligence&oldid=1148436187.

[55] Yankai Zeng, Abhiramon Rajasekharan, et al. *Automated Interactive Domain-Specific Conversational Agents that Understand Human Dialogs*. Preprint arXiv:2302.08941. 2023.