Education for Logical Thinking in the Age of AI

Robert Kowalski[0000-0002-1341-8583]

Imperial College London
r.kowalski@imperial.ac.uk

Abstract. The rapid advance of AI is transforming the world. No one knows what the new AI world will be like and how we humans will learn to live with it. I will argue that, no matter where we may be headed, we need to improve our ability to think logically and to use our logical thinking to monitor and control AI.

I will present the work of the Prolog Education Group (PEG), which is working towards this goal, by building upon more than 50 years of expertise of applying logic programming (LP) to knowledge representation and problem solving in AI. I will argue that LP is particularly well suited for teaching logical thinking, because it is much simpler than most other logics, but in many respects it is also more powerful. By developing educational materials in LP, we aim to teach the critical, logical thinking skills needed in the new world of AI.

Keywords: Logical thinking, Logic Programming, Education.

What is the Problem?

Since the early 2010s, neural systems trained on massive amounts of data have dominated the field of AI. These systems include large language models (LLMs) such as ChatGPT and Gemini, which generate natural language text and images in response to user requests. LLMs are arguably close to reaching artificial general intelligence.

Part of the problem is that these neural-based systems are not transparent. They cannot explain their results, and they tend to hallucinate. Despite these problems, the capabilities of these AI systems means that they can do much of our thinking for us, and they have the potential to take control over our lives. This potential is exacerbated by the way that AI is being used to spread misinformation and disinformation, controlled by both known and anonymous actors with their own hidden agendas.

What is the Solution?

The solution is for us, humans, to improve our ability to think for ourselves, and to use that improved ability both to monitor and to control our interactions with AI systems and agents. Here are three reasons why we can be hopeful that this can be done:

- According to the dual-process model of human thinking, we already use conscious, deliberative thinking to monitor our own subconscious, intuitive thinking [8]. So, it would be natural to build a similar relationship with AI systems, using our human intelligence to monitor and control AI.

- AI researchers are coming to a similar conclusion: that neural-based AI systems need to be combined with and controlled by symbolic AI, in hybrid, neural-symbolic AI systems. [5, 17, 19] In many of these hybrid systems, the symbolic component is a logic-based system based on logic programming (LP).
- Thanks in part to the success of LP in symbolic AI, we are now in a much better position than before to teach logical thinking in all subject areas and at all levels of the educational curriculum.

In this short paper, I will argue that LP is particularly well suited for teaching logical thinking, because it is much simpler than most other logics, but in many respects it is also more powerful.

What is logic programming?

Facts and rules. In its basic form, a logic program consists of facts and rules. Facts represent simple relationships among individuals, as in *Alice likes logic*. Rules have the form *if conditions then conclusion*, where the *conclusion* represents a single relationship and the *conditions* represent a conjunction of relationships. For example, *if Alice likes X and X likes Alice then Alice is happy*. Rules can contain variables, like *X*, which stand for any individual.

Like other formal logics, logic programs have a symbolic syntax, as in *likes(alice, logic).* But logic programs can also be written in unambiguous natural language, as in the case of the Logical English [10] syntax used here.

Rules in logic programs can be used for many purposes. For example, they can be used to represent causal relationships, as in *if you press the brake then you stop the car*. They can also be used to represent abstract concepts in terms of more concrete concepts, as in *if a person X is a professor then the person X is an academic*. The conditions-conclusion form of logic programs is especially well suited for representing legal regulations [10].

Note that "facts" and rules are not necessarily true in an absolute sense. They can be regarded as true for the sake of argument. We sometimes use the terms *assumptions*, *beliefs* or *premises*, to draw attention to this tentative commitment to the truth of facts and rules.

Intuitive semantics. Logic programs have an intuitive semantics, which can be introduced to children as early as in primary school [3, 14]. Children can use their intuitive understanding of facts and rules to draw their own logical conclusions. For example, given that *Alice likes logic* and *if X likes logic then Alice likes X*, children can try to answer such questions as *Alice likes which X*. They can compare their answers with the answers generated by a computer implementation of an LP language such as Prolog.

Proofs have the structure of a triangle. Logic programs also have a simple and intuitive notion of proof, as filling in a triangle (or upside-down tree) with facts at the bottom and a goal at the top [13]. Every sentence in the triangle, except for the facts at the bottom, needs to be proved by applying a rule in the logic program. A sentence is proved by applying a rule if there is an instance of the rule whose conclusion is that sentence and whose conditions are sentences which are immediately below the sentence in the triangle. See Fig. 1. Can you guess what rules are in the logic program? (The answer is not unique. It depends on whether the rules are specific to the named individuals, or whether the rules contain variables for any individuals.)

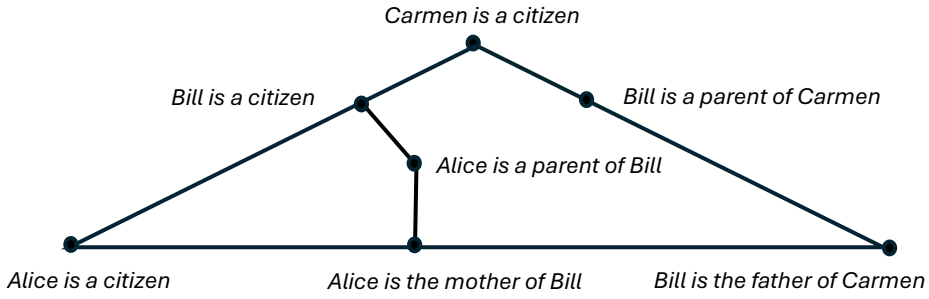

Fig. 1. A proof that Carmen is a citizen, because her father's mother is a citizen.

Bottom-up reasoning formalizes the notion of *synthesis*. For most people, it is probably most natural to construct a proof by filling in the triangle bottom-up, reasoning forwards from given facts and deriving new facts. Bottom-up reasoning is commonly used in Datalog variants of logic programming [4]. It is especially useful when facts and rules are assumed to be true, for the sake of exploring their logical consequences. Bottom-up reasoning can be useful, even when there is no goal to be solved.

Top-down reasoning formalizes the notion of *analysis*. However, when there is a goal to be solved, it is often more efficient to reason backwards, filling in the triangle top down. Top-down reasoning is used in Prolog to interpret rules as goal-reduction procedures: Given a goal that matches an instance of the conclusion of a rule, top-down reasoning reduces the goal to the subgoals that are the conditions of the instance of the rule.

In anticipation of using rules to reason backwards, rules are commonly written backwards in most LP languages. For example, *you stop the car if you press the brake.* Used to reason backwards, the rule is in effect a procedure, which can be expressed as a conditional imperative: *to stop the car, press the brake.* It can also be expressed as a condition-action rule: *if you want to stop the car then press the brake*.

Negation as failure. Logic programs can contain negative conditions, as in *Alice is sad if Alice likes X and it is not the case that X likes Alice.* A negative condition of the form *it is not the case that P* (i.e. *not P*), where *P* is any proposition. can be proved by attempting to prove *P*. If the attempt to prove *P* fails, then the proof of *not P* succeeds. This way of proving *not P* is called *negation as failure*. It can be written as a meta-rule: *it is not the case that P if it cannot be shown that P*.

Negation as failure can be justified by the *closed world assumption* that if a fact cannot be shown to be true then the fact is false. So, in the case that *Alice likes logic*, but it cannot be shown that *logic likes Alice*, then it is not the case that *logic likes Alice*, and therefore *Alice is sad*.

The closed world assumption and negation as failure are a natural way to understand default reasoning. For example: *A person is innocent of a crime if it cannot be shown that the person committed the crime*. If a person is suspected of committing a crime, but it cannot be shown that the person committed the crime, then the rule allows us to conclude that the person is innocent. However, if later, we can show that the person did indeed commit the crime, then we must withdraw the default conclusion.

Abductive logic programming

The closed world assumption is a natural assumption in many, but not all cases. For example, suppose you are driving a car, you see an obstacle ahead, and you want to avoid hitting the obstacle. To achieve the goal (*you avoid hitting the obstacle*), you can reason top-down. If you believe that *you avoid hitting the obstacle if you stop the car,* then you can reduce the goal to the subgoal *you stop the car.* If you also believe that *you stop the car if you press the brake*, then you can reduce the subgoal to the action subgoal *you press the brake*. If your beliefs are true, and *you press the brake* is a fact that is true, then you achieve your goal.

But, under the closed world assumption, if the action subgoal is not a fact, then you will fail to achieve your goal, and you will not avoid hitting the obstacle.

Open predicates and abductive reasoning. However, in abductive logic programming (ALP) [9], it is possible to combine the closed world assumption for some predicates (such as *likes*) with an open world assumption for other predicates (such as *pressing the brake*).

Under the *open world assumption* for a predicate, if a fact that is an instance of the predicate cannot be shown to be true, then the fact is neither true nor false. As a result, if a predicate is open, then an instance of the predicate can be assumed to be true for the purpose of *satisfying* a goal, by making the goal true. In this case, if *pressing the brake* is an open predicate, then the goal *you avoid hitting the obstacle* can be satisfied by assuming that *you press the brake*. In general, the process of satisfying (or solving) a goal by generating facts that are assumed to be true is called *abduction*, and the open predicates that are candidate assumptions are said to be *abducible.*

The most typical use of abduction is to generate assumptions to explain observations. But as can be seen in the example above, abductive reasoning can also be used to

generate actions to achieve a desired effect. In both cases and more generally, there can be several alternative abductive solutions for the same goal.

Integrity constraints. In database management systems, integrity constraints monitor and restrict the data that can be stored in the database. In ALP, integrity constraints serve a similar purpose, restricting the instances of open predicates that can be assumed to solve or satisfy goals.

Consider the goal of explaining why the *grass is wet* when you wake up in the morning. Suppose that you believe: *the grass is wet if it rained at night* and *the grass is wet if the sprinkler was on at night.* If the conditions of the two rules are both abducible predicates, then there are two alternative explanations: either *it rained at night* or *the sprinkler was on at night.*

Suppose, however, that there is an integrity constraint: *it is not the case that both the sprinkler was on at night and the sprinkler is broken.* If you know *the sprinkler is broken*, then given what you believe (and only what you believe), there is only one logical explanation, namely the assumption that *it rained at night.*

In the LP languages ASP and s(CASP) [6], integrity constraints are written with the same syntax as rules, but with the conclusion *false*. For an integrity constraint *false if conditions*, written in this form, a candidate *assumption* satisfies the integrity constraint, if making the *assumption* true does not make the *conditions* true.

Alternatively, and for many practical cases, it is more natural, when one of the conditions is an open predicate *O*, to express the constraint with the same syntax as a rule, but with the negative conclusion *not O*. For example, *it is not the case that the sprinkler was on at night if the sprinkler is broken*.

For such an integrity constraint of the form *not O if conditions*, there is a simple way to check whether a candidate assumption that is an instance of *O* satisfies the integrity constraint: The assumption *O* satisfies the constraint if the corresponding instantiated *conditions* of the constraint cannot be shown.

Active integrity constraints. Integrity constraints in ALP generalize the notion of goal in simple LP. Integrity constraints can be passive goals, which prohibit the addition of facts that make the constraints false. But they can also be active goals, which make themselves true by generating their own abductive assumptions.

These two uses of integrity constraints serve the same purposes as prohibitions and obligations in deontic logic [7, 12, 16]. Their use as active goals generalizes condition-action rules in production systems [11].

For example, the integrity constraint *if there is an obstacle ahead then you avoid hitting the obstacle* can be understood as a *maintenance goal*, which generates the *achievement goal* of making *you avoid hitting the obstacle* true, whenever *there is an obstacle ahead* becomes true. It can also be understood as an obligation to avoid hitting obstacles.

What does it mean to be rational?

ALP opens the door to addressing the bigger issue of what it means to be rational. To be rational, it is not enough to reason logically to solve goals, it is also necessary to have good reasons for choosing one solution over another [1, 2, 15].

If the goal is to explain an observation, it is necessary to decide which explanations have the most evidential support. If the goal is to achieve a desired effect, then it is necessary to compare different plans of actions to decide which plans have the most likely chance of producing the most useful outcomes.

A call for help!

Now, when logical thinking skills are needed more than ever, the academic community needs to join forces and urgently promote logic as a learnable skill that can be taught at all levels of the educational curriculum. For this purpose, we need to agree on a core curriculum and try to adopt a common approach. The Prolog Education Group [18] is working towards this end, and it welcomes your support.

Acknowledgments. Many thanks for helpful comments from Veronica Dahl, Jacinto Dávila, Wlodek Drabent, Gopal Gupta and Fariba Sadri.

Disclosure of Interests. The author has no competing interests to declare.

References

1. Audi, R.: *The architecture of reason: The structure and substance of rationality*. Oxford University Press (2001)
2. Baron, J.: Thinking and Deciding. Cambridge University Press (2023)
3. Cecchi, L.A., Rodríguez, J.P. and Dahl, V.: Logic programming at elementary school: why, what and how should we teach logic programming to children?. In *Prolog: the next 50 years* (pp. 131-143). Cham: Springer Nature Switzerland (2023)
4. Ceri, S., Gottlob, G. and Tanca, L.: What you always wanted to know about Datalog (and never dared to ask). *IEEE transactions on knowledge and data engineering*, *1*(1), pp.146-166 (1989)
5. Fabiano, F., Ganapini, M.B., Loreggia, A., Mattei, N., Murugesan, K., Pallagani, V., Rossi, F., Srivastava, B. and Venable, K.B.: Thinking fast and slow in human and machine intelligence. *Communications of the ACM*, *68*(8), pp.72-79 (2025)
6. Gupta, G., Salazar, E., Varanasi, S.C., Basu, K., Arias, J., Shakerin, F., Min, R., Li, F. and Wang, H.: Automating common sense reasoning with ASP and s(CASP). In *Proceedings of 2nd Workshop on Goal-directed Execution of Answer Set Programs* (2022)
7. Gupta, G., Rajasekharan, A., Tudor, A.R., Salazar, E. and Arias, J.: Modeling Deontic Modal Logic in the s(CASP) Goal-directed Predicate Answer Set Programming System. *arXiv preprint arXiv:2507.05519* (2025)
8. Kahneman, D.: *Thinking, Fast and Slow*. Farrar, Straus and Giroux (2011)
9. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. Journal of logic and computation, 2(6), 719-770 (1992)

10. Kowalski, R., Davila, J., Sartor, G. and Calejo, M.: Logical English for law and education. In *Prolog: The Next 50 Years* (pp. 287-299). Cham: Springer Nature Switzerland (2023)
11. Kowalski, R. and Sadri, F.: Abductive logic programming agents with destructive databases. *Annals of Mathematics and Artificial Intelligence*, *62*(1), pp.129-158 (2011)
12. Kowalski, R., Satoh, K.: Obligations as optimal goal satisfaction, Journal of Philosophical Logic, 47(4), 579-609 (2018)
13. Kowalski, R.: Logic for Problem Solving. North Holland, Elsevier (1979)
14. Kowalski, R.: Logic as a computer language for children. In *ECAI* pp. 2-10 (1982)
15. Kowalski, R.: Computational Logic and Human Thinking: How to be Artificially Intelligent. Cambridge University Press (2011)
16. Kowalski, R.: Satisfiability for First-order Logic as a Non-Modal Deontic Logic. In *Bridging@ CogSci* pp. 84-90 (2017)
17. Marra, G., Dumančić, S., Manhaeve, R., De Raedt, L.: From statistical relational to neurosymbolic artificial intelligence: A survey. Artificial Intelligence Volume 328, 104062, ISSN 0004-3702 (2024)
18. PEG, https://prolog-lang.org/Education/Organization.html, last accessed 2026/3/6
19. Yang, X.W., Shao, J.J., Guo, L.Z., Zhang, B.W., Zhou, Z., Jia, L.H., Dai, W.Z. and Li, Y.F.: Neuro-symbolic artificial intelligence: Towards improving the reasoning abilities of large language models. *arXiv preprint arXiv:2508.13678* (2025)