

On Teaching Logic Programming in the Era of Generative AI

Paul Tarau¹

¹University of North Texas, Denton, USA

Abstract

After fact finding on the disruption brought by today's Generative AI tools to the education system, we outline the advantages of joining the disruption, motivated by the natural synergies between today's Generative AI and Logic Programming Languages, derived from both their similar and complementary reasoning mechanisms.

We explore engagement strategies for AI assistants with specifics on Prolog teaching and also in the wider context of online teaching and automation of teaching processes, including AI-assisted evaluation strategies for student success. In particular, we overview and explore in some detail AI-based Prolog code generators, gamification, computational thinking in logic-focussed natural language, and the use of multi-agent AI systems.

Keywords

AI as a disruptor of education systems, AI as a disruptor of Logic Programming teaching process, engagement strategies for AI assistants, synergies between Logic Programming and Generative AI, AI-assisted Prolog coding

1. Introduction

We will focus on technical aspects for engaging the AI disruption. It is however of paramount importance to parallel and evaluate our pragmatic fact finding and entailed reasoning steps with alignment to legal, ethical and moral values, integrity, fairness and equity which are not covered in this paper.

2. The Facts on the Generative AI Disruption of Education

2.1. The Domain-independent disruption

Our current educational technologies have over the last 50 years incrementally refined the closely interrelated aspects of training, testing and student evaluation and, to a much lesser extent, the real-world outcomes of the teaching process. At the same time, leftovers for its 2000+ years baggage focussed on replicating memorized content rather than actionable generalization and creativity are still present in the evaluation mechanisms of educational processes.

With the focus on verifiable metrics, our often over-automated grading processes, driven by market-dictated cost reduction on human grader involvement, have created a perfect breeding

ICLP 2024: N¹¹ Workshop PEG 2.0, October, 2024, Dallas, USA.



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



CEUR Workshop Proceedings (CEUR-WS.org)

ground for Generative AI induced gaming opportunities against typical grading systems¹.

Some key facts on the impact on evaluation of student success can be summarized as follows:

- The advanced capabilities of AI models make it nearly impossible to detect AI-generated essays or multiple choice problems, rendering traditional plagiarism detection methods ineffective.
- Educators might need to reconsider the purpose and structure of assignments, including essays and problem sets, in light of AI's ability to generate content and solve complex problems.
- The rapid changes brought by AI in education demand immediate adjustments from instructors and the creation of clear policies regarding AI use to preserve educational integrity while embracing its potential benefits.

Let's follow up with an instance of a pragmatic (notwithstanding its ethical aspects!) decision that a typical average student faces.

The cheating paradox The following case study (with kind assistance from GPT-4o on obvious details) puts on the table an aspect of the a commonly used grading process, that an average student focusing on passing a course with as high grades as possible faces:

Example 1. *After submitting the following set of assumptions to the GPT-4o assistant, actually as a simple math word problem to solve:*

*Assume that we grade courses with
40% home work,
30% Midterm Exam
and 30% Final Exam.*

*It is known that AI-assisted cheating on homework reduces
exam performance by around 5%.*

*Assuming that AI-assisted cheating on home works gives 95% on them and
that under honest hard work the student would obtain a 70% grade.*

*Which of the AI-assisted cheating and the honest strategy would give the
highest grade to the student? Give me a short final answer just citing
the grades and their difference in %.*

we get:

With AI-assisted cheating:

- *Homework:*
- *Exams (Midterm and Final): 65 % (due to a 5% reduction)*
- *Final grade: $0.40 * 95 + 0.30 * 65 + 0.30 * 65 = 76%$*

¹<https://www.oneusefulthing.org/p/post-apocalyptic-education>

With honest hard work:

- Homework: 70%
- Exams (Midterm and Final): 70%
- Final grade: $0.40 * 70 + 0.30 * 70 + 0.30 * 70 = 70\%$

Difference:

- AI-assisted strategy gives 6% higher final grade than the honest strategy.

2.2. The Domain Specific Disruption : Prolog

Let us now focus, by taking advantage of GPT-4o's Prolog programming skills, to solve a beginner-level Prolog assignment testing list processing.

Example 2. We start with the query:

Write a Prolog program that removes each element in an odd position of a list of terms. Sort the result in reverse order. Show me how it works on [l, o, g, c, is, fun]

we get a first shot correct program:

```
% Base case: An empty list has no elements
% to remove, so return an empty list.
remove_odd_positions([], []).

% Recursive case: If the list has at least two elements,
% remove the first element and keep the second.
remove_odd_positions([_, X | T], [X | Rest]) :-
    remove_odd_positions(T, Rest).

% If the list has only one element left,
% skip it (since it's in an odd position).
remove_odd_positions([_], []).

% To get the desired result, first remove odd positions,
% then sort in reverse order.
remove_and_reverse(List, Result) :-
    remove_odd_positions(List, ListWithoutOdds),
    reverse(ListWithoutOdds, Result).
```

with the results on the given example list, as returned by the LLM:

You can use the program on the list [l, o, g, c, is, fun] as follows:

```
?- remove_and_reverse([l, o, g, c, is, fun], Result).
Result = [fun, c, o].
```

Next, let's push ahead with a midlevel problem.

Example 3. *This example shows clear, efficient and well explained Prolog code on a less trivial problem:*

Write a Prolog program that finds the first 5 pairs of twin primes and show the Result it returns.

The LLM returns the following code:

```
% Define a predicate to check if a number is prime
is_prime(2).
is_prime(3).
is_prime(N) :-
    N > 3,
    N mod 2 =\= 0,
    \+ has_factor(N, 3).
```

```
% Helper predicate to check if a number has a factor
has_factor(N, F) :-
    N mod F =:= 0.
has_factor(N, F) :-
    F * F < N,
    F2 is F + 2,
    has_factor(N, F2).

% Define a predicate to find twin primes
twin_prime(X, Y) :-
    is_prime(X),
    Y is X + 2,
    is_prime(Y).

% Find the first N pairs of twin primes
find_twin_primes(0, _, []) :- !.
find_twin_primes(N, X, [(X, Y)|Rest]) :-
    N > 0,
    twin_prime(X, Y),
    N1 is N - 1,
    X2 is X + 1,
    find_twin_primes(N1, X2, Rest).
find_twin_primes(N, X, Result) :-
    X2 is X + 1,
    find_twin_primes(N, X2, Result).

% Predicate to get the first 5 pairs of twin primes
first_5_twin_primes(Result) :-
    find_twin_primes(5, 2, Result).
```

with the correct results computed by the AI's implicit Prolog reasoning engine:

```
?- first_5_twin_primes(Result).
Result = [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31)].
```

Note also a limitation here: the program will not manage a very large number of twin prime pairs as AI's math skills are still imperfect on computations with large values.

The AI-assisted code generation quality extends to puzzles stated in natural language, reachability problems, transitive closure, graph coloring and combinatorial problems, with often first shot correct and elegant, well documented Prolog programs.

Our conclusion here is that *today's top Generative AI models (e.g. OpenAI GPT-4o, Anthropic's Claude, Google's Gemini, etc.) know more than enough Prolog coding for covering at least the teaching domain and specifically its student evaluation subdomain.*

We encourage the reader to try out similar examples for intuition on the domain coverage of the Prolog Generative AI coder, both when restricted to teaching and also on snippets of industrial strength code covering Prolog based knowledge graphs, Definite Clause Grammars, networking, planning and constraint-solving problems, Prolog-based Web apps, and so on.

2.3. On the logic of joining the disruption

Generative AI-denial is a natural, partly rational, partly emotion-driven reaction of practitioners of symbolic AI, Natural Language Processing and classic Machine Learning that are prime disruption targets of it, as it suddenly deprecates years of hard work on significant academic and industrial achievements.

While keeping this in mind, we will explore here a rational and pragmatic path for joining the disruption motivated by the natural synergies that Logic Programming and Generative AI have together. In fact, it became obvious to this author much earlier, at the time of pure text completion models like GPT3 and declarative image builders like DALL.E that the disruption enables the use of Prolog tools like Definite Clause grammars as generators for prompt and image synthesis [1].

As a recent proof of concept, illustrating the fast progress in the field, we refer to [2] as a prototypical example for the impressive performance gains brought by interleaving Prolog and AI steps on a set of difficult reasoning benchmarks.

The more than 100 entries of the leaderboard at lmsys.org of a competition between LLMs, based on an ELO tournament that pairs them to be judged by thousands of user votes on the quality of their output, testifies on the accelerated evolution of today's Generative AI systems².

3. Engagement strategies with AI Assistants

We will next explore several engagement strategies with Generative AI-based teaching assistants.

3.1. A basic AI engagement strategy

- Learn underlying concepts, try to solve the problem.
- If stuck, iterate a few times with alternative ways to look at it.
- If still stuck, call the AI-assistant using the instructor-provided prompt or your own.

²<https://lmarena.ai/>

- After reading the solution, answer to a set of multiple choice questions about the AI's code. The instructor can use the results of your answers for grading.

Note that this could be still automated by the instructor's code asking the LLM to grade the resulting code if submitted and grading the answers to the follow-up AI generated question. See our WebApp³ for a (recursive) follow-up question generator.

In the case of Prolog, this AI-assisted coding pipeline can be refined by requiring the students to run the AI's code and explain the results of a few spy points by tracing the execution.

On more interactive online engagement on platforms like Canvas⁴, engaging in discussion groups on the AI generated code can provide a richer learning experience that is also gradable by the instructor or an AI assistant.

3.2. Gamification

Gamification⁵ relies on borrowing elements from the gaming world students are naturally exposed to. An example of AI assisted gamified learning process looks as follows:

- an assignment requests a small group of students to compare a Prolog problem given by each to a different AI of their choice
- the students will be timed on testing and possibly debugging the AI-provided solutions and gain points, banners etc. for their performance
- the students discuss the merits and demerits of each in a Canvas forum with a partial grade originating from their participation in the forum, possibly evaluated by an AI-agent if we want a fully automated grading process.

A refinement to the grading would be that allowing AI-assistance (seen as the equivalent of "magic spells" in a game) will have a point cost. Thus, it will need, like in the gaming world, some extra work to refill the player's "strength level", with clear learning benefits in this case.

Another approach, easy to implement in practice, would be organizing student coding contests with participation of (possibly artificially weakened) AI antagonists. This is particularly effective when the problems' solution is already known to the learner or it has even been previously implemented in a different language.

3.3. Engagement with Natural Language and Logic Programming

Given its syntactic and semantic features, it makes sense, when teaching Prolog, to focus on closeness to natural language while keeping in mind the importance of learnability, presence of flexible execution mechanisms and exposure to Prolog's highly expressive language constructs.

Good barometers for learnability are the learning curves of newcomers (including children in their early teens), the hurdles they experience and the projects they can achieve in a relatively short period of time. Another observable to watch is how well one can pick up the language inductively, simply by looking at coding examples.

³<https://auto-quest.streamlit.app/>

⁴<https://www.instructure.com/canvas>

⁵<https://uwaterloo.ca/centre-for-teaching-excellence/catalogs/tip-sheets/gamification-and-game-based-learning>

When thinking about what background can be assumed in the case of newcomers, irrespectively of age, natural language pops up as a common denominator. As logic notation originates in natural language, there are conspicuous mappings between verbs and predicates and nominal groups as their arguments, especially relevant when solving knowledge representation problems.

That hints toward learning methods and language constructs easily mapped syntactically and semantically to natural language equivalents.

Along these lines, our Natlog system⁶ [3, 4] enforces closeness to Natural Language by a simplified, flat Prolog syntax while opening access to Python's rich finite set and coroutining primitives, while ensuring its smooth integration in the Python-based deep learning ecosystem.

The reader is invited to try out Natlog programs online⁷.

The following code snippets give a hint about Natlog's code focusing on learnability while keeping in mind a focus on expressiveness as a key programming language feature.

Example 4. *Natlog as a syntactically lighter Prolog equivalent:*

```
mother of X M: parent of X M, female M.  
father of X M: parent of X M, male M.  
grand parent of X GP: parent of X P, parent of P GP.
```

```
ancestor of X A : parent of X P, parent or ancestor P A.  
parent or ancestor P P.  
parent or ancestor P A : ancestor of P A.
```

Example 5. *Natlog as expressive code defining at source level a the primitive findall/3:*

```
stack S : `list S.  
push S X : #meth_call S append (X).  
pop S X : `meth_call S pop () X.  
findall X G Xs: listof X G S, to_cons_list S Xs.  
listof X G S: stack S, collect_ X G S.  
collect_ X G S : call G, `copy_term X CX, push S CX, fail.  
collect_ X_G_S.
```

As saliently emphasized in [5] there's an important distinction to be made between Prolog as a programming language and Prolog as knowledge representation language. This will become relevant both for deciding the content of teaching materials and the AI-assistant teaching strategies.

⁶<https://github.com/ptarau/natlog>

⁷<https://natlog.streamlit.app>

When focusing on the latter, AI-assistance will express its results in the form of a Propositional Horn Clause program, with facts and the rules connecting them consisting of natural language statements or noun groups describing concepts and their details. This involves a recursive descent processes starting with a knowledge exploration initiator expressed as a sentence or a key concept, that returns executable Prolog programs that can be evaluated with a simple fixpoint iteration or in tabling-enabled Prolog system like XSB [6] or SWI-Prolog [7].

We refer to [8] and [9] for details of this process and invite the reader to interact with our online deep11m streamlet app⁸ illustrating it.

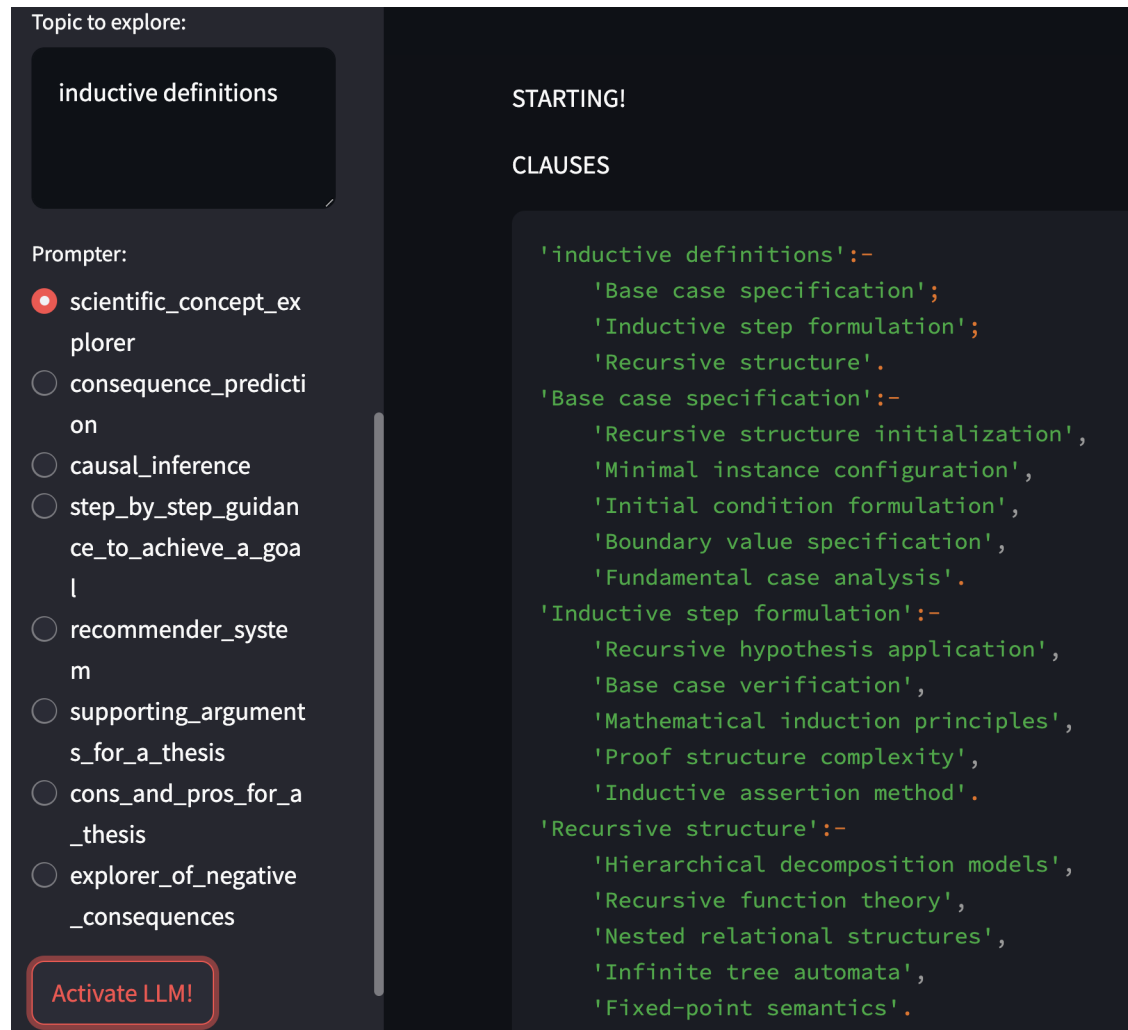


Figure 1: Exploring a concept as a Propositional Horn Clause Program

Fig.1 shows an example exploring the concept of “inductive definition”. Note that, besides the uses for actually introducing a student to Prolog programming, the close connection to natural

⁸<https://deep11m.streamlit.app/>

language exposes the student to a Prolog-inspired computational thinking processes, involving task decomposition, with its complementary aspects of refinement and exploration of alternatives.

As an additional instructional tool, the deep11m app also builds an extended concept graph to expose deeper, LLM-generated relations to similar concepts.

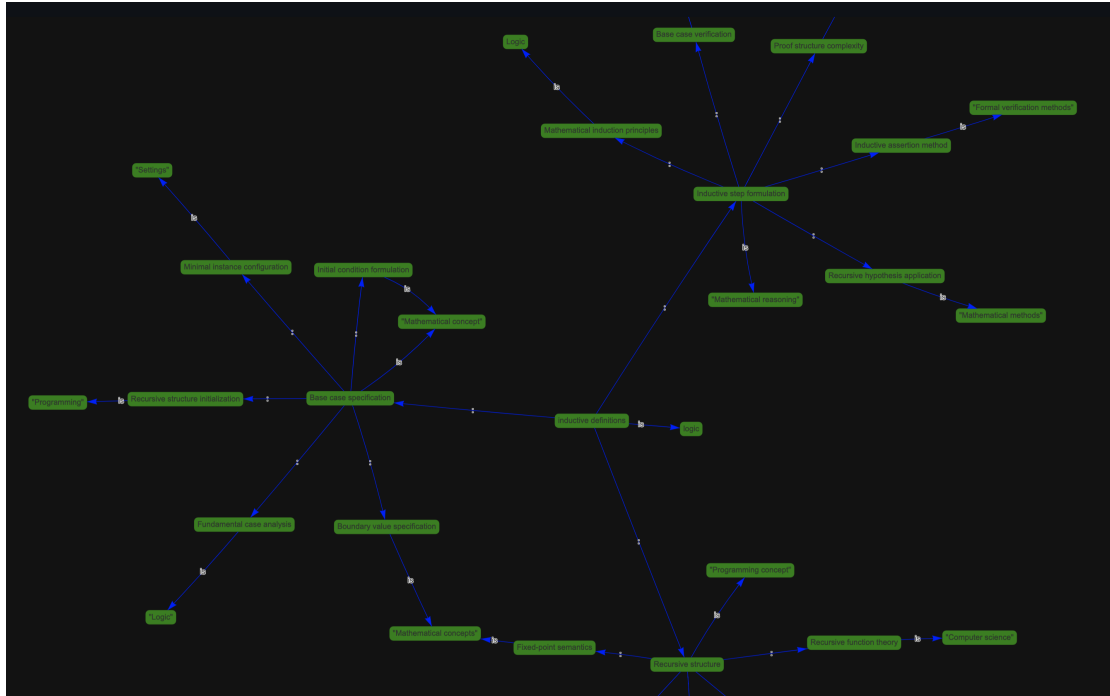


Figure 2: Visualization of the concept graph related to “inductive definitions”

Fig.2 illustrates this relation graph for the concept of “inductive definition” as extracted from the minimal model of the Propositional Horn Clause program.

3.4. Teaching with multi-agent AI systems

A strong trend in today’s large scale, industrial strength AI API based applications is the the use of multi-agent architectures⁹ chaining together custom code, vector store and database calls as well as LLM API calls.

An AI-driven multi-agent online course architecture could involve a society of specialized AI agents, that in its Prolog instance would look like the following:

- Prolog code generator assistant
- Prolog code critique and bug finder
- natural language to/from executable Prolog translator
- conversation animator AI bot as part team projects or forums
- instructor representative as performance evaluator

⁹<https://python.langchain.com/v0.1/docs/modules/agents/>

The multi-agent society's members could watch and react to unblock students based on the Agents' specific skills and available tools using a publish-subscribe architecture while keeping track of each student progress in a student-specific memory track.

3.5. AI-assisted teaching with human in the loop

We have focussed so far on scenarios involving AI-assisted automated and scalable online teaching tools that are (slowly but firmly) becoming prevalent in the education system. However even in the context, the presence of the *human in the loop*¹⁰ is of paramount importance.

Real time or asynchronous presence of the human instructor can be naturally integrated when unexpected events are likely to occur in the process. Ideally, in a large scale online teaching system this could be an instant real-time intervention by an instructor, possibly assisted by an AI-generated summary of the blockers a given student faces. At a smaller scale, help ticket-style event queues and blackboards could be made available soliciting instructor intervention with well-defined status details.

3.6. Some “out of The box” ideas on on reshaping upcoming AI-assisted Teaching Methodologies

We sketch here a few possible refinements and adaptations given the availability of large scale generative AI to the two sides of the educational process: students and their instructors.

Transition from penalty-based to reward-based teaching systems Similarly to the success of Reinforcement Learning in the training loops of the today's user-facing LLMs, possibly also seen as part of a gamification process, reward-based engagement has plenty of room to replace or complement today's penalty-based student-evaluation mechanisms.

Spontaneous social network study groups among remote online friends exposed to a diversity of teaching systems With physical colocation loosing some of its relevance to joint interests on social network co-presence and exposure to a wider diversity of teaching methods, often crossing country borders, is an opportunity to acquire intuitions about key concepts learned in the conventional school systems and synchronize terminological barriers.

Facilitate curiosity-driven AI-assisted self study With personalized, user aware generative AI (e.g., Character AI's agents¹¹), individual curiosity on a school topic can be explored directly and turned into a learning process while interacting with the customized agent aware of past interactions with its owner.

Industry-inspired project-driven learning with AI assistance It is by now standard practice in industrial code development to use tools (e.g., Github Copilot¹²) that revise or even create and inject new code in the coder's development pipeline.

¹⁰<https://hai.stanford.edu/news/humans-loop-design-interactive-ai-systems>

¹¹<https://character.ai/>

¹²<https://github.com/features/copilot>

This is particularly useful in project-based courses like terminal undergrad classes or in-depth graduate classes involving software artifacts as deliverables.

Focus on spending teaching time on the “why” besides the “how” in problem solving

The “why” part has often been skipped over in the teaching process partly to keep the focus on the topic at hand and partly because its extent is often too wide to be presented under the time-constraints of the teaching time and the student evaluation process. Nevertheless, presenting the “ontology” covering not only the taxonomical relations between concepts but also their historical origins can be enlightening and Generative AI assistants can complement the course content with them, without a significant burden on instructor time.

Extend the narrow domain-specific teaching to encompass the spectrum of related concepts to which students might have been already exposed

This ranges from text simplification and a jargon removal using classic prompts like “explain me X such that a 9 years old can understand it” to using AI-generated relation graphs exposing similarities, generalizations and analogies to concepts in nearby fields. In particular, one can use AI-assistants to help “connect the dots” by forming solid intuitions of the concepts hiding beyond domain-specific taxonomies.

Moving from canned routines and memorized factoids to actionable knowledge outcomes

This is an uphill battle against a 2000+ years tradition, but it might be facilitated by appropriate AI-assistance to encourage immersive content-driven learning. It also involves rethinking of deadline-driven assignments, with focus on quality vs. quantity, as well as prioritizing of the learning outcomes vs. the convenience of assigning and grading the work.

There are also the usual market pressures exercised on “teaching as a business”, but dropping the AI-cheatable canned multiple-choice questions in favor of AI-assisted engagement in actual content creation looks feasible given the technical competence of today’s a generative AI systems.

Design course materials to include video presentations of AI-assisted teaching

Finally, the wide availability of high quality multi-media teaching materials suggest producing content that describes not just the topic at hand, but also the instructor’s interaction with Generative AI agents, to build, clarify and present the content creation process and its outcomes.

4. A brief Related Work Overview

The domain is fast shifting to the point that even a few months old related work, covering different tools or assuming deprecated limitations and features of now out of use AI models loses its informational value. We focus here on some work on Logic Programming and Generative AI that show a long term vision and development plans or some possibly out of the box thinking about the topic.

Goal-directed ASP systems like *s(CASP)* [10], have provided forms of non-monotonic and co-inductive reasoning. With enhanced constructive constraint programming algorithms, going

back to [11] and presenting also explanations covering the reasons of negative outcomes, they show synergies between these extensions to Prolog and Generative AI induced logic program snippets [12].

A more direct approach is recursion on LLM queries, by chaining the LLM’s distilled output as input to a next step and casting its content and interrelations in the form of logic programs, to automate and focus this information extraction with minimal human input [8, 9]. Like in the case of typical RAG architectures [13, 14], this process can rely on external ground truth but it can also use new LLM client instances as “oracles” deciding the validity of the synthesized rules or facts. We refer to [8] for an extensive list of LLM-generated Horn clause programs and [9] for a wider variety of them, including DCG encodings of recursive descents driven by AI-generated follow-up questions on a given topic as well as nicely visualized knowledge representation outputs.

5. Conclusion and Future Work

We have, hopefully, made it clear that it is widely beneficial to join the disruption that Generative AI assistance brings into teaching processes and also discussed the specifics of teaching Prolog in the context.

In fact, we can see Prolog teaching as a “Goldilocks” case, given the natural, reasoning-induced synergies between Prolog and Generative AI-based tools.

With this in mind, automated online teaching and student-testing tools based on Prolog code generation and natural language dialog threads expressed as logic representations ranging from propositional Horn clause or Datalog forms to specialized Prolog code and Definite Clause Grammars have the potential to be extended to similar teaching and student-testing techniques for logic languages like ASP and s(CASP) as possibly also refined to support other programming paradigms.

References

- [1] P. Tarau, Reflections on Automation, Learnability and Expressiveness in Logic-Based Programming Languages, Springer Nature Switzerland, Cham, 2023, pp. 359–371. doi:10.1007/978-3-031-35254-6_29.
- [2] N. Borazjanizadeh, S. T. Piantadosi, Reliable reasoning beyond natural language, 2024. URL: <https://arxiv.org/abs/2407.11373>. arXiv:2407.11373.
- [3] P. Tarau, Natlog: a Lightweight Logic Programming Language with a Neuro-symbolic Touch, in: A. Formisano, Y. A. Liu, B. Bogaerts, A. Brik, V. Dahl, C. Dodaro, P. Fodor, G. L. Pozzato, J. Vennekens, N.-F. Zhou (Eds.), Proceedings 37th International Conference on Logic Programming (Technical Communications) , 20-27th September 2021, 2021. doi:10.4204/EPTCS.345.27.
- [4] P. Tarau, Natlog: Embedding Logic Programming into the Python Deep-Learning Ecosystem, in: E. Pontelli, S. Costantini, C. Dodaro, S. Gaggl, R. Calegari, A. D’Avila Garcez, F. Fabiano, A. Mileo, A. Russo, F. Toni (Eds.), Proceedings 39th International Conference on Logic Programming, Imperial College London, UK, 9th July 2023 - 15th July 2023,

- volume 385 of *Electronic Proceedings in Theoretical Computer Science*, Open Publishing Association, 2023, pp. 141–154. doi:10.4204/EPTCS.385.15.
- [5] M. Genesereth, Prolog as a Knowledge Representation Language The Nature and Importance of Prolog, in: D. S. Warren, V. Dahl, T. Eiter, M. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023.
 - [6] T. Swift, S. Warren, David, XSB: Extending Prolog with Tabled Logic Programming, *Theory and Practice of Logic Programming* 12 (2012) 157–187. doi:10.1017/S1471068411000500.
 - [7] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, Swi-prolog, *Theory and Practice of Logic Programming* 12 (2012) 67–96.
 - [8] P. Tarau, Full Automation of Goal-driven LLM Dialog Threads with And-Or Recursors and Refiner Oracles (2023) arXiv:2306.14077. doi:10.48550/arXiv.2306.14077. arXiv:2306.14077.
 - [9] P. Tarau, System description: Deepllm, casting dialog threads into logic programs, in: J. Gibbons, D. Miller (Eds.), *Functional and Logic Programming*, Springer Nature Singapore, Singapore, 2024, pp. 117–134. doi:10.1007/978-981-97-2300-3_7.
 - [10] J. ARIAS, M. CARRO, E. SALAZAR, K. MARPLE, G. GUPTA, Constraint answer set programming without grounding, *Theory and Practice of Logic Programming* 18 (2018) 337–354. doi:10.1017/S1471068418000285.
 - [11] P. Stuckey, Negation and Constraint Logic Programming, *Information and Computation* 118 (1995) 12–33. URL: <https://www.sciencedirect.com/science/article/pii/S0890540185710486>. doi:10.1006/inco.1995.1048.
 - [12] Y. Zeng, A. Rajashekharan, K. Basu, H. Wang, J. Arias, G. Gupta, A reliable common-sense reasoning socialbot built using llms and goal-directed asp, 2024. URL: <https://arxiv.org/abs/2407.18498>. arXiv:2407.18498.
 - [13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive nlp tasks, *Advances in Neural Information Processing Systems* 33 (2020) 9459–9474.
 - [14] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, C. D. Manning, Raptor: Recursive abstractive processing for tree-organized retrieval, 2024. URL: <https://arxiv.org/abs/2401.18059>. doi:10.48550/arXiv.2401.18059. arXiv:2401.18059.