On the Translation of ASP Rules to (Controlled) Natural Language Sentences

Simone Caruso[1,*], Carmine Dodaro[2], Fabrizio Lo Scudo[2], Marco Maratea[2] and Kristian Reale[2]

[1]DIBRIS, University of Genoa, Viale Causa, 13, 16145, Genoa (GE)
[2]DeMaCS, University of Calabria, Via P. Bucci, 87036, Rende (CS)

Abstract

Answer Set Programming (ASP) is a well-known declarative paradigm within the field of knowledge representation and reasoning, extensively used in both academic and industrial settings. The success of ASP in critical decision-making applications makes it crucial to explain its complex logic and reasoning processes to non-experts. Recently, Large Language Models (LLMs), like ChatGPT, have demonstrated remarkable capabilities in explaining complex code segments, particularly in widely-used imperative languages such as Python. Notably, these models are able to interpret ASP rules as well. However, a significant challenge with LLMs is their tendency to "hallucinate", meaning they can generate misleading or incorrect outputs. This issue is particularly problematic in applications where precision is critical. The paper introduces two novel tools, ASP2CNL and CNL2NL, which can be combined together to translate ASP rules into natural language with the goal of mitigating the errors of LLMs when explaining ASP rules. The effectiveness of these tools has been validated during a review phase involving three external ASP experts, demonstrating a significant reduction in the occurrence of wrong outputs from LLM tools.

Keywords

Answer Set Programming, Controlled Natural Language, Natural Language, Large Language Models

1. Introduction

Answer Set Programming (ASP) [1, 2, 3] is a declarative programming paradigm within the field of knowledge representation and reasoning, designed for solving complex combinatorial problems. The main strengths of ASP range from its solid theoretical foundations to the existence of powerful solving tools such as CLINGO [4] and WASP [5]. For these reasons, in the recent years, ASP has been used in a wide range of both academic and industrial scenarios [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16].

The development of novel and complex applications in various fields, including healthcare, necessitates that systems are inherently explainable. In these contexts, the importance of explainability and comprehensibility of Artificial Intelligence (AI) systems, including those based on logical frameworks like ASP, cannot be underestimated. In an era where AI is increasingly integrated into critical decision-making processes, ensuring that these systems are transparent and understandable to end-users is crucial. For logical systems such as ASP, which are inherently complex, making the underlying logic and reasoning processes accessible to non-experts can increase the trust in the technology, enhance user confidence, and encourage broader adoption.

Large Language Models (LLMs) [17, 18], such as ChatGPT [19], offer a promising approach for explaining ASP programs in a more accessible manner to users. By translating complex logical constructs into natural language, they can significantly make ASP accessible also to non-experts. Moreover, they have a wide range of potential applications. Indeed, they can be integrated into debugging and explainability systems, and they can offer support to students new to ASP. Within software development, they can enhance automated code review systems by providing natural language explanations of code logic, thus helping code reviewers in understanding the intent of code more efficiently. Moreover,

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

*Corresponding author.
simone.caruso@edu.unige.it (S. Caruso); carmine.dodaro@unical.it (C. Dodaro); fabrizio.loscudo@unical.it (F. Lo Scudo); marco.maratea@unical.it (M. Maratea); kristian.reale@unical.it (K. Reale)
0000-0002-5617-5286 (C. Dodaro); 0000-0002-9034-2527 (M. Maratea)

in industries with strict compliance and regulatory requirements, they can simplify the verification process, ensuring that ASP-based systems conform to the necessary rules and guidelines by translating their logic into a format that is easier to review.

However, a notable limitation of LLMs is their tendency to *hallucinate*, or generate information that is inaccurate or not present in the input data. This characteristic poses a challenge for reliability, especially in applications where precision and factual accuracy are crucial. Balancing the explanatory power of LLMs with measures to mitigate their propensity for hallucination is therefore essential in leveraging them for explaining ASP programs effectively.

In this paper, we provide a practical contribution in the aforementioned context. Specifically, we present two novel and open-source tools, namely ASP2CNL and CNL2NL, which address this challenge by facilitating a two-stage translation process aimed at making ASP rules accessible to a broader audience. The idea is to first translate ASP rules into an intermediate and reliable textual form, and then to use LLMs tools to improve the readability of the produced text. In more details, ASP2CNL serves as the initial phase, where ASP rules are converted into Controlled Natural Language (CNL) sentences in the format supported by the tool CNL2ASP [20]. This transformation applies a reliable and reproducible translation of ASP rules in a more interpretable form. Subsequently, CNL2NL operates as the second phase, translating the CNL sentences into natural language sentences using state-of-the-art LLM tools, such as ChatGPT [21]. The two tools, ASP2CNL and CNL2NL, can be then combined together in a pipeline, referred to as ASP2NL, that transforms a given ASP program into a set of sentences expressed in a natural language.

ASP2NL was evaluated by three independent ASP experts on five different domains, ranging from classical simple problems to complex scheduling problems in the context of healthcare, and compared to an LLM baseline executed on the original ASP encoding. The results of this analysis demonstrate that ASP2NL significantly reduces the number of hallucinations produced by the baseline, especially on complex domains.

2. Preliminaries

In this section we describe the main concepts behind the tool CNL2ASP [20], which translates sentences expressed in a controlled natural language (CNL) to ASP rules and is used as baseline for the ASP2CNL tool described in this paper. In the following, we assume the reader to be familiar with syntax and semantics of ASP [2] and with the ASP-Core 2 standard [22].

CNL2ASP is made of three components: the *Parser*, the *Concepts Data Structures*, and the *ASP Rewriter*. The role of the Parser is to first process CNL sentences in the input file, then to create the Concepts Data Structures, and finally to tokenize the CNL statements which are later on processed by the ASP Rewriter. Specifically, the Parser is able to process three types of CNL propositions, which are referred to as explicit definition propositions, implicit definition propositions, and (standard) CNL propositions. The first two types of propositions are used to define the *concepts*, where a concept is a thing, a place, a person or an object that is used to model entities of the application domain of the CNL. Standard CNL propositions are sentences describing the rules of the application domain.

Example 1. *Let the following be a set of CNL propositions in the context of the well-known graph coloring problem:*

```
A node is identified by an id.
A edge is identified by a firstnode, and by a secondnode.
A color is identified by an id.
An assignment is identified by a node, and by a color.
Whenever there is a node X then we can have an assignment to exactly 1 color.
It is prohibited that C1 is equal to C2 whenever there is an assignment with node X, and
    with color C1, whenever there is an assignment with node Y, and with color C2,
    whenever there is an edge with firstnode X, and with secondnode Y.
```

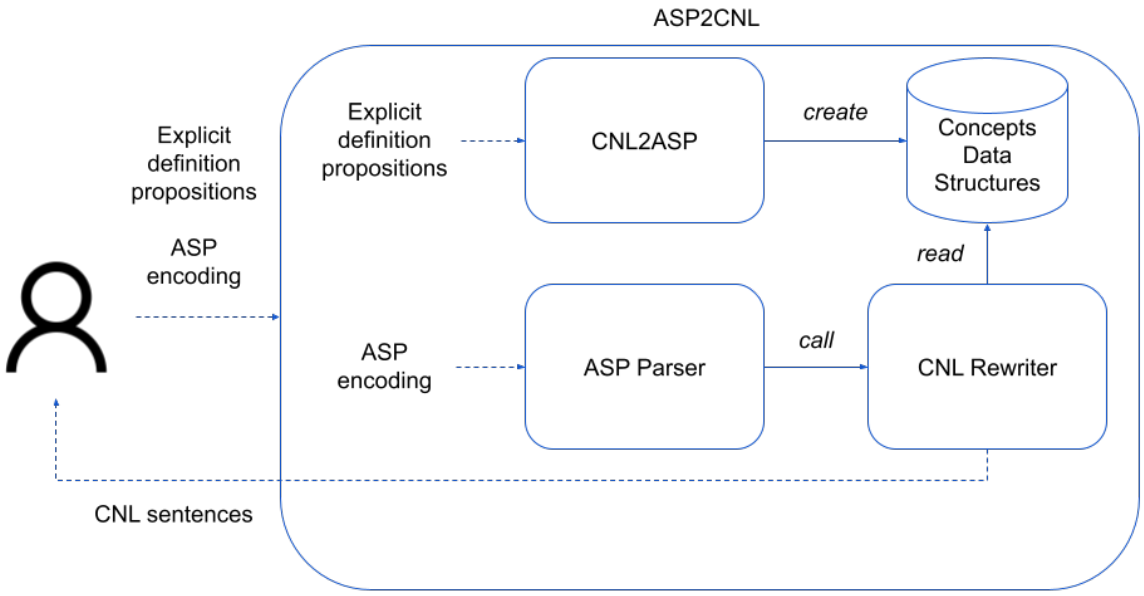

Figure 1: Architecture of the tool ASP2CNL.

Lines 1 to 4 contain explicit definitions for the concepts of node, edge, color, and assignment. Lines 5 and 6 contain (standard) CNL propositions, specifying that each node must be assigned to exactly one color and that two connected nodes cannot have the same color, respectively. Additionally, if the explicit concept definition in line 4 is absent, the proposition in line 5 could implicitly define the concept of assignment.

Then, such propositions are translated into the following ASP rules:

```
1 1 <= {assignment(X,CLR_D): color(CLR_D)} <= 1 :- node(X).
2 :- C1 = C2, assignment(X,C1), assignment(Y,C2), edge(X,Y).
```

It is important to emphasize that CNL2ASP defines concepts only by their names, ensuring each concept (i.e., node, edge, color, and assignment in Example 1) is uniquely identified.

3. ASP2CNL and CNL2NL

In this section, we describe two tools, namely ASP2CNL and CNL2NL. Initially, ASP2CNL transforms ASP rules into CNL sentences. This step is crucial for representing the syntax of ASP into a format that, while structured, is closer to the language familiar to people that are not experts of logic programming conventions. Following this, the CNL2NL tool takes the intermediate CNL output and performs a further translation into natural language sentences, with the aim of reducing the gap between technical ASP constructs and everyday understanding. It is important to emphasize here that these tools can be used independently, or combined into a sequential pipeline for a full transformation of ASP rules into natural language.

3.1. ASP2CNL

The architecture of ASP2CNL is shown in Figure 1. Specifically, it takes as input a set of explicit definition propositions and an ASP encoding, and provides as output a set of CNL sentences associated to the ASP encoding. Internally, explicit definition propositions are processed before the ASP encoding and they are provided as input to the CNL2ASP module, which builds the Concepts Data Structures. Intuitively, the Concepts Data Structures associate each of the concept defined by the explicit definition propositions to a list of its attributes.

Example 2 (Continuing Example 1). *After processing the sentences:*

```
A node is identified by an id.
An edge is identified by a firstnode, and by a secondnode.
A color is identified by an id.
An assignment is identified by a node, and by a color.
```

the Concepts Data Structures are extended with a construct that maps each concept to the list of its attributes, i.e., `node` *is mapped to* `[id]`*,* `edge` *is mapped to* `[firstnode, secondnode]`*,* `color` *is mapped to* `[id]`*, and* `assignment` *is mapped to* `[node, color]`*.*

Then, the ASP Parser processes the rules in the ASP encoding and pass them to the CNL Rewriter that uses the Concepts Data Structures to produce the corresponding CNL sentences. In the following, we describe, for each supported ASP construct, how it is translated into a CNL sentence.

Atoms. When the CNL Rewriter processes an ASP rule and encounters an atom, it first looks up the atom's predicate in the Concepts Data Structures. Subsequently, for each term in the atom that is not marked as hidden, the CNL Rewriter finds the associated attribute. Based on this information, it constructs the following:

```
a/an *predicate* with *attribute 1* [equal to] *term 1*, and with *attribute 2* [equal
    to] *term 2*, ..., and with *attribute n* [equal to] *term n*
```

where `*predicate*` is the atom's predicate, `*attribute i*` is the i-th attribute associated to the atom's predicate in the Concepts Data Structures, `*term i*` is the i-th term occurring in the atom, and `equal to` is used when the term is a constant, whereas they are omitted for variables.

Example 3. *The atom* `assignment(X,C1)` *corresponds to the following:*

```
an assignment with node X, and with color C1
```

whereas the atom `assignment(_,blue)` *corresponds to the following:*

```
an assignment with color equal to blue
```

since hidden terms are simply skipped.

Aggregates. Aggregates are converted into sentences according to the following template:

```
the number/the total of *attribute* [for each *attributes*] that have a *atoms* is *
    comparison_operator* *comparison_operand*.
```

for `#count` and `#sum` aggregates, respectively, and

```
the lowest/highest *attribute* of *aggregate_operator* [for each *attributes*] that have a
     *atoms* is *comparison_operator* *comparison_operand*.
```

for `#min` and `#max` aggregates, respectively. The element on which the aggregation is performed is denoted by `*attribute*`. The terms in the aggregate are optionally specified using `for each *attributes*`, while any other element in the aggregate set are denoted by `*atoms*`. The comparison operator is expressed as `is *comparison_operator*`, which can be one of the following: `is different from`, `is equal to`, `is less/greater than`, or `is less/greater than or equal to`, corresponding to the operators supported by the ASP language. Finally, `*comparison_operand*` is the aggregate guard.

Example 4. *The aggregate:*

```
#count{C,X: assignment(X,C)} != 1.
```

is translated as follows:

```
the number of color id, for each node id, that have a assignment with node X, and with
    color C is different from 1.
```

Constraints. ASP constraints are translated into sentences that begin with `It is prohibited that` followed by `there is [not] *atom*`, where `[not] *atom*` is the first (negated) atom occurring in the constraint. Subsequently, for each literal within the constraint, the phrase `whenever there is` is added before the atom. For negated atoms, the translation uses `whenever there is not` instead. Each atom is then translated as described before.

Example 5 (Continuing Example 1). *The constraint:*

```
:- C1 = C2, assignment(X,C1), assignment(Y,C2), edge(X,Y).
```

is translated as follows:

```
It is prohibited that there is a assignment with node id X, with color id C1 equal to C2,
    whenever there is a assignment with node id Y, with color id C2, whenever there is a
    edge with firstnode X, with secondnode Y.
```

Note that the constraint

```
:- assignment(X,C), assignment(Y,C), edge(X,Y).
```

would be translated as follows:

```
It is prohibited that there is an assignment with node X, and with color C, whenever there
     is an assignment with node Y, and with color C, whenever there is an edge with
    firstnode X, and with secondnode Y.
```

Facts. ASP facts are simply translated into sentences beginning with `There is` followed by the translation of the atom. As example, the fact `edge(1,2).` corresponds to `There is an edge with firstnode equal to 1, and with secondnode equal to 2.`

Normal and disjunctive rules. The structure of a normal rule's body is similar to the one of the constraints. For each atom in the body, the phrase `whenever there is` is used, and for each negated atom, `whenever there is not` is applied. Upon completing the body, the phrase `we must have` is introduced, followed by the atom that constitutes the rule's head.

Example 6. *The following normal rule:*

```
node(X) :- edge(X,_).
```

is translated as

```
Whenever there is a edge with firstnode X then we must have a node with id X.
```

Disjunctive rules differ from normal rules only by the fact that atoms in the head are separated by the word `or`, and the phrase `we can have` is used.

Example 7. *The following disjunctive rule:*

```
assignment(X,"blue") | assignment(X,"green") | assignment(X,"red") :- node(X).
```

is translated as

```
Whenever there is a node with id X then we can have a assignment with node id X, with
    color id equal to blue or a assignment with node id X, with color id equal to green or
     a assignment with node id X, with color id equal to red.
```

Choice rules. In the context of choice rules, the body is treated in the same manner as in normal and disjunctive rules. The head uses the phrase `we can have` followed by the cardinality, the head atom, and, if additional conditions are present, they are specified with `such that`. The cardinality for a choice rule is defined using phrases like `exactly X`, `at most X`, `at least X`, or `between X and Y`, where `X` and `Y` are either numbers or variables.

Example 8. *The following choice rule:*

```
1 <= {assignment(X,CLR_D): color(CLR_D)} <= 1 :- node(X).
```

is translated as

```
Whenever there is a node with id X then we can have exactly 1 assignment with node id X,
    with color id CLR_D such that there is a color with id CLR_D.
```

Weak Constraints. The formulation of weak constraints follows a structure similar to the one of normal constraints, with some distinctions. Specifically, sentences translating weak constraints are structured as follows:

```
It is preferred [as little as possible] with priority *level*, that ... [*term X* is
    minimized/maximized].
```

In this template, `*level*` is a placeholder that is replaced with the actual level of the weak constraint. Both `[as little as possible]` and `[*term X* is minimized/maximized]` are mutually exclusive. In particular, if the weight of the weak constraint is equal to 1, the phrase `as little as possible` is introduced, otherwise, if the weight is a variable, then the phrase `*term X* is minimized` is introduced where the placeholder `*term X*` is replaced with the corresponding variable. When `*term X*` is a variable preceded by minus (-) the phrase becomes `*term X* is maximized`. The other part of the sentence, i.e., the one following `that ...`, is similar to the one created for normal constraints.

Example 9. *The following rule:*

```
:~ numMaxDay(N). [N@4]
```

is translated as

```
It is preferred, with priority 4, that whenever there is a numMaxDay with number N, N is
    minimized.
```

Nevertheless, there are some exceptions in presence of aggregate atoms and arithmetic terms. In particular, in case the weight is an arithmetic term (e.g., `X-Y`) then the placeholder `[*term X* is minimized]` is moved before as follows:

```
It is preferred with priority *level*, that *arithmetic_opration* is minimized/maximized
    ... .
```

Example 10. *The following rule:*

```
:~ numMax(DAY, MAX), numMin(DAY, MIN). [MAX-MIN@5,DAY]
```

is translated as

```
It is preferred, with priority 5, that the difference between MAX, and MIN is minimized,
    whenever there is a numMax with day DAY, with timeslot MAX, whenever there is a numMin
     with day DAY, with timeslot MIN.
```

On the other hand, if an aggregate result has to be minimized/maximized, we use a construct of the form: `It is preferred, with priority *n*, that *aggregate* ... is minimized.`, where `aggregate` follows the rules described for aggregates, as shown in the following example.

Example 11. *The following rule:*

```
:~ #count{C: choice(C)} = X . [-X@3]
```

is translated as

```
It is preferred, with priority 3, that the number of id that have a choice is maximized.
```

Terms comparison. When the CNL Rewriter has to process an ASP rule and comes across a comparison between two terms, it translates the comparison operator into natural language in accordance with the following cases:

1. One term of the comparison refers to a specific attribute of an already used concept. In this case, the comparison is specified directly in the `with` condition of the concept itself.

   Example 12. *The following rule:*

   ```
   :- nurse(N), N < 3.
   ```

   is translated as

   ```
   It is prohibited that there is a nurse with id N less than 3.
   ```

2. The left-hand side of a comparison contained in a constraint is an arithmetic operation. In this case, if there is a sum, the phrase `the sum between` is introduced at the beginning of the constraint after `It is prohibited`, or, if it is a subtraction, the phrase `the difference between` is instead introduced.

   Example 13. *The following rule, which contains an arithmetic operation:*

   ```
   :- nurse(N), N-2 > 0.
   ```

   is translated as

   ```
   It is prohibited that the difference between N, and 2 is greater than 0 whenever
       there is a nurse with id N.
   ```

3. In all other cases, the comparison is translated using the `where` keyword followed by the comparison operator.

   Example 14. *The following rules:*

   ```
   :- nurse(N), N < 3, N > 1.
   next(X) :- nurse(N), X=N+1, X < 10.
   ```

   are translated as

   ```
   It is prohibited that there is a nurse with id N less than 3, where N is greater than
       1.
   Whenever there is a nurse with id N then we must have a next with id X, where X is
       equal to N+1, where X is less than 10.
   ```

3.2. CNL2NL

In recent years, Natural Language Processing (NLP) has experienced a huge transformation, which was mainly driven by techniques for training and refining large statistical autoregressive models [23, 24]. These models are usually referred to as large language models (LLMs) [25] and have attracted consistent interest from both the academic and the industrial sector.

In our implementation, we used LLMs to interpret the CNL sentences produced by ASP2CNL. It is important to observe that the development of effective prompts for LLMs (this process is often referred to as prompt engineering), i.e., the ones that produce the most accurate and relevant outputs, is still an interesting and challenging topic of discussion. Generally, there are some rules of thumb that can be applied, such as being specific, using clear and concise language, and providing context whenever possible.

In our case, we created the following prompt template which is later on provided as input to our LLM:

```
Explain in everyday language what the following sentence says, improving the writing style:

*CNL Sentence*
Limit response to 500 characters.
```

where `*CNL Sentence*` is substituted with the sentence that needs to be translated.

Example 15 (Continuing Example 1). *Consider the sentence at line 5, then the following prompt is created to obtain the natural language version of the CNL sentence:*

```
Explain in everyday language what the following sentence says, improving the writing style:

Whenever there is a node X then we can have an assignment to exactly 1 color.

Limit your answer to 500 characters.
```

Then, the output of an LLM can be of the form:

```
Each node X can be assigned only one color.
```

Finally, CNL2NL is not dependent on a specific LLM, since all of them are able to parse plain English text; we can, thus, easily switch from one tool to another one by modifying a few lines of code. Indeed, the typical strategy to interact with LLMs is through their APIs. Consequently, CNL2NL generates an individual request towards the LLM service for each CNL sentence received as input. All the retrieved natural language conversions, produced by the LLM service, are finally gathered and returned to the user. In its current implementation, CNL2NL supports ChatGPT [19] and all the LLMs available through the open-source project ollama (https://github.com/ollama/ollama).

4. Empirical Evaluation

The assessment of the performance of ASP2NL consisted in evaluating the correctness of the sentences produced by ASP2NL executed with the API of ChatGPT version 3.5 as LLM, and its comparison with the baseline approach (referred to as Base), i.e., the same version of ChatGPT executed on the plain ASP encoding. Concerning the baseline approach, we used the prompt:

```
Explain in everyday language what the following rule of an Answer Set Program states,
    improving the writing style:
*RULE*
Limit response to 500 characters.
```

The following example should clarify the prompt used for the baseline approach.

Example 16 (Continuing Example 1). *Consider the sentence at line 5, then the following prompt is created to obtain the natural language version of the CNL sentence:*

```
Explain in everyday language what the following rule of an Answer Set Program states,
    improving the writing style:

1 <= {assignment_to(X,CLR_D): color(CLR_D)} <= 1 :- node(X).

Limit response to 500 characters.
```

Then, the output can be of the form:

```
Each node must be assigned exactly one color from a set of available colors.
```

To evaluate the correctness, we selected five benchmark tests: Graph Colouring, Hamiltonian Path, Hanoi Tower, Chemotherapy Treatment Scheduling [26], and Nurse Scheduling [27]. The first three benchmarks are traditional problems frequently encountered in ASP. The latest two benchmarks are complex scheduling problems proposed in the literature. It is important to observe that these last two benchmarks are less likely to have been part of ChatGPT's training data, whereas Graph Colouring, Hamiltonian Path, and Hanoi Tower are classical benchmarks, commonly used in educational settings

Table 1
Correctness test: percentage of wrong rules as provided by the three ASP experts. Best results highlighted in bold.

		Rev. 1		Rev. 2		Rev. 3	
Benchmark	#Rules	Base	ASP2NL	Base	ASP2NL	Base	ASP2NL
Graph Colouring	2	0%	0%	0%	0%	0%	0%
Hamiltonian Path	5	40%	40%	40%	**20%**	40%	**20%**
Hanoi Tower	17	**29%**	41%	53%	**24%**	**12%**	35%
CTS	11	45%	**18%**	73%	**27%**	64%	**45%**
Nurse Scheduling	19	37%	**21%**	26%	**0%**	63%	**0%**

and as illustrative examples, suggesting that ChatGPT may have been exposed to similar data during its training.

Then, we designed a questionnaire and distributed it to three independent evaluators and ASP experts: Giuseppe Galatà from SurgiQ s.r.l., Daniela Inclezan from Miami University, and Francesco Ricca from the University of Calabria. They received both the original ASP rule and a detailed description of it, and they evaluated the two translated natural language sentences, rating them as either Wrong, Acceptable, or Correct. To ensure that the evaluations were conducted impartially, the reviewers were unaware of whether a given sentence was generated by the Base ChatGPT or by our specialized ASP2NL tool. The results of this evaluation are summarized in Table 1, which shows the percentage of sentences each reviewer marked as Wrong for each benchmark.

First, note that it might be possible to have divergent opinions among the reviewers about the correctness of the sentences since the distinction between what is considered correct or incorrect is often subjective. Despite these differences, there is an agreement among the reviewers that the combined approach of ASP2NL reduces the incidence of errors compared to using ChatGPT alone in three out of five benchmarks, particularly in the complex scenarios of Chemotherapy Treatment Scheduling and Nurse Scheduling. The two approaches are basically on par on Graph Colouring where they do not report any wrong answers. A special mention is reserved to the case of the Hanoi Tower benchmark, where opinions between reviewers vary. Indeed, two reviewers found that ChatGPT performed better, while one reviewer observed superior performance from ASP2NL.

5. Discussion and Conclusions

The tool ASP2CNL presented in this paper uses the Concept Data Structures, and the controlled natural language proposed by [20], implemented by the open-source tool CNL2ASP. Therefore, improvements in the language supported by CNL2ASP might lead to direct improvements in our ASP2CNL tool. As far as we know, ASP2CNL is the first tool that is able to take as input an arbitrary ASP program producing sentences expressed in a CNL form. A similar attempt was proposed by [28], who defined the language PENG^{ASP}, a CNL that is automatically converted into ASP. The grammar of PENG^{ASP} is designed for allowing a conversion from the CNL to ASP and then back in the other direction. However, as far as we know, this possibility was only possible for ASP rules produced by PENG^{ASP}, therefore it was not possible to process an arbitrary ASP program. PENG^{ASP} should be already compatible as is with the tool CNL2NL, since our tool does not make any assumption on the input text, however its implementation, as well as a binary executable, is not yet public, which makes it impossible to perform some practical tests. It is worth to observe that CNL2ASP was also used by [29] for creating ASP programs from English sentences using neural networks. ASP2CNL can be used in this context to automatically create the sentences used during the training phase of the neural network.

LLMs represent a specific category of models mainly adopted to solve natural language related tasks. Following the introduction of word2vec [30], a pioneering method for learning distributed word representations using shallow neural networks, the field of NLP was profoundly transformed

by the emergence of a highly scalable architecture known as Transformer [31]. Subsequent to this, Pre-trained Language Models (PLM), such as BERT [32], introduced an effective paradigm that involves the pre-training of language models on extensive unlabeled corpora and eventually fine-tuning on some downstream task in order to have versatile semantic features expressed by word representations [33, 34]. Since then, the prevalent approach has been to increase the dimensions of the model [17]. At present, the term *large* in LLM is indicative of the count of the weights and bias parameters within the network, which can reach approximately one trillion. With the growth of language models, the requirement for fine-tuning has lessened, and generative language models are now capable of addressing a wide variety of tasks merely through interaction based on text which is termed as prompt engineering. Present state-of-the-art models, including GPT, have shown extraordinary characteristics across a multitude of tasks [35]. CNL2NL benefits from advancements in LLMs and is not restricted to any specific one. Consequently, any progress in the LLM field can be easily integrated into our tool. Additionally, it is worth noting that CNL2ASP operates using sentences in a controlled natural language, making it compatible with any LLM trained on English sentences. This compatibility means there is no requirement for the LLMs to directly process ASP rules in their training data.

In the context of automated code review tasks, a recent study by [36] presents an investigation to explore the feasibility of using ChatGPT for this purpose. Although their results indicate the promise of ChatGPT in improving code quality, they also reveal certain challenges in comprehending the code. Despite this, ChatGPT has shown good performance in explaining programs written in imperative programming languages [37, 38, 39], indicating some potential ability to also convert raw ASP code into human-readable language. However, in its current state, ChatGPT does not appear to fully capture the semantics of ASP, generating inaccurate translations, as shown in Table 1. This is likely due to the relatively small number of ASP programs in ChatGPT's training data compared to the amount of documented code written in imperative programming languages.

As future work, we plan to analyze further strategies and open source LLMs, and to extend the evaluation of the tools.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. of Logic Programming, MIT Press, 1988, pp. 1070–1080.

[2] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. URL: https://doi.org/10.1145/2043174.2043195. doi:10.1145/2043174.2043195.

[3] V. Lifschitz, Answer Set Programming, Springer, 2019. URL: https://doi.org/10.1007/978-3-030-24658-7. doi:10.1007/978-3-030-24658-7.

[4] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: Proc. of ICLP Technical Communications, volume 52 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:15. URL: https://doi.org/10.4230/OASIcs.ICLP.2016.2. doi:10.4230/OASICS.ICLP.2016.2.

[5] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: Proc. of LPNMR, volume 11481 of *LNCS*, Springer, 2019, pp. 241–255.

[6] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68. URL: https://doi.org/10.1609/aimag.v37i3.2678. doi:10.1609/AIMAG.V37I3.2678.

[7] A. A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. C. Teppan, Industrial applications of answer set programming, Künstliche Intelligenz 32 (2018) 165–176.

[8] M. Alviano, R. Bertolucci, M. Cardellini, C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, M. Mochi, V. Morozan, I. Porro, M. Schouten, Answer set programming in healthcare: Extended overview, in: IPS and RCRA 2020, volume 2745 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020.

[9] C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, I. Porro, An ASP-based solution for operating room scheduling with beds management, in: P. Fodor, M. Montali, D. Calvanese, D. Roman (Eds.), Proc. of RuleML+RR, volume 11784 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 67–81.

[10] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, A two-phase ASP encoding for solving rehabilitation scheduling, in: S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, D. Roman (Eds.), Proc. of RuleML+RR, volume 12851 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 111–125.

[11] G. Grasso, S. Iiritano, N. Leone, V. Lio, F. Ricca, F. Scalise, An asp-based system for team-building in the gioia-tauro seaport, in: M. Carro, R. Peña (Eds.), Proc. of PADL 2010, volume 5937 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 40–42.

[12] C. Dodaro, G. Galatà, M. Maratea, I. Porro, Operating room scheduling via answer set programming, in: AI*IA, volume 11298 of *LNCS*, Springer, 2018, pp. 445–459.

[13] C. Dodaro, G. Galatà, M. Maratea, I. Porro, An ASP-based framework for operating room scheduling, Intelligenza Artificiale 13 (2019) 63–77.

[14] P. Bruno, F. Calimeri, C. Marte, M. Manna, Combining deep learning and asp-based models for the semantic segmentation of medical images, in: Proc. of RuleML+RR, volume 12851 of *LNCS*, Springer, 2021, pp. 95–110.

[15] C. Dodaro, G. Galatà, M. Gebser, M. Maratea, C. Marte, M. Mochi, M. Scanu, Operating room scheduling via answer set programming: Improved encoding and test on real data, Journal of Logic and Computation 34 (2024) 1556–1579.

[16] S. Caruso, G. Galatà, M. Maratea, M. Mochi, I. Porro, Scheduling pre-operative assessment clinic with answer set programming, Journal of Logic and Computation 34 (2023) 465–493.

[17] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, D. Amodei, Scaling laws for neural language models, CoRR abs/2001.08361 (2020). URL: https://arxiv.org/abs/2001.08361. arXiv:2001.08361.

[18] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Comput. Surv. 56 (2024) 30:1–30:40. URL: https://doi.org/10.1145/3605943. doi:10.1145/3605943.

[19] OpenAI, GPT-4 technical report, CoRR abs/2303.08774 (2023). URL: https://doi.org/10.48550/arXiv.2303.08774. doi:10.48550/ARXIV.2303.08774. arXiv:2303.08774.

[20] S. Caruso, C. Dodaro, M. Maratea, M. Mochi, F. Riccio, CNL2ASP: converting controlled natural language sentences into ASP, Theory Pract. Log. Program. 24 (2024) 196–226. URL: https://doi.org/10.1017/s1471068423000388. doi:10.1017/S1471068423000388.

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: Proc. of NeurIPS, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[22] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. URL: https://doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.

[23] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, W. Fedus, Emergent abilities of large language models, Trans. Mach. Learn. Res. 2022 (2022). URL: https://openreview.net/forum?id=yzkSU5zdwD.

[24] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard,

G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, N. Fiedel, Palm: Scaling language modeling with pathways, J. Mach. Learn. Res. 24 (2023) 240:1–240:113. URL: http://jmlr.org/papers/v24/22-1144.html.

[25] M. Shanahan, Talking about large language models, Commun. ACM 67 (2024) 68–79. URL: https://doi.org/10.1145/3624724. doi:10.1145/3624724.

[26] C. Dodaro, G. Galatà, A. Grioni, M. Maratea, M. Mochi, I. Porro, An asp-based solution to the chemotherapy treatment scheduling problem, Theory Pract. Log. Program. 21 (2021) 835–851.

[27] C. Dodaro, M. Maratea, Nurse scheduling via answer set programming, in: Proceedings of LPNMR, volume 10377 of *LNCS*, Springer, 2017, pp. 301–307.

[28] R. Schwitter, Specifying and verbalising answer set programs in controlled natural language, Theory Pract. Log. Program. 18 (2018) 691–705. URL: https://doi.org/10.1017/S1471068418000327. doi:10.1017/S1471068418000327.

[29] M. Borroto, I. Kareem, F. Ricca, Towards automatic composition of ASP programs from natural language specifications, Accepted at IJCAI https://doi.org/10.48550/arXiv.2403.04541 (2024). URL: https://doi.org/10.48550/arXiv.2403.04541. doi:10.48550/ARXIV.2403.04541. arXiv:2403.04541.

[30] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: Proc. of ICLR, 2013. URL: http://arxiv.org/abs/1301.3781.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proc. of NeurIPS, 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[32] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Proc. of NAACL-HLT, Association for Computational Linguistics, 2019, pp. 4171–4186. URL: https://doi.org/10.18653/v1/n19-1423. doi:10.18653/V1/N19-1423.

[33] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, in: Proc. of ACL, Association for Computational Linguistics, 2020, pp. 7871–7880. URL: https://doi.org/10.18653/v1/2020.acl-main.703. doi:10.18653/V1/2020.ACL-MAIN.703.

[34] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized BERT pretraining approach, CoRR abs/1907.11692 (2019). URL: http://arxiv.org/abs/1907.11692. arXiv:1907.11692.

[35] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. M. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, Y. Zhang, Sparks of artificial general intelligence: Early experiments with GPT-4, CoRR abs/2303.12712 (2023). URL: https://doi.org/10.48550/arXiv.2303.12712. doi:10.48550/ARXIV.2303.12712. arXiv:2303.12712.

[36] Q. Guo, J. Cao, X. Xie, S. Liu, X. Li, B. Chen, X. Peng, Exploring the potential of ChatGPT in automated code refinement: An empirical study, in: Proc. of ICSE, ACM, 2024, pp. 34:1–34:13. URL: https://doi.org/10.1145/3597503.3623306. doi:10.1145/3597503.3623306.

[37] P. Bhattacharya, M. Chakraborty, K. N. S. N. Palepu, V. Pandey, I. Dindorkar, R. Rajpurohit, R. Gupta, Exploring large language models for code explanation, CoRR abs/2310.16673 (2023). URL: https://doi.org/10.48550/arXiv.2310.16673. doi:10.48550/ARXIV.2310.16673. arXiv:2310.16673.

[38] T. Ahmed, P. T. Devanbu, Few-shot training llms for project-specific code-summarization, in: Proc. of ASE, ACM, 2022, pp. 177:1–177:5. URL: https://doi.org/10.1145/3551349.3559555. doi:10.1145/3551349.3559555.

[39] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, X. Liao, Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning, in: Proc. of ICSE, ACM, 2024, pp. 39:1–39:13. URL: https://doi.org/10.1145/3597503.3608134. doi:10.1145/3597503.3608134.