Answer Set Programming and Large Language Models interaction with YAML: Preliminary Report

Mario Alviano*, Lorenzo Grillo

DeMaCS, University of Calabria, 87036 Rende (CS), Italy

Abstract

Answer Set Programming (ASP) and Large Language Models (LLMs) have emerged as powerful tools in Artificial Intelligence, each offering unique capabilities in knowledge representation and natural language understanding, respectively. In this paper, we combine the strengths of the two paradigms to couple the reasoning capabilities of ASP with the attractive natural language processing tasks of LLMs. We introduce a YAML-based format for specifying prompts, allowing users to encode domain-specific background knowledge. Input prompts are processed by LLMs to generate relational facts, which are then processed by ASP rules for knowledge reasoning, and finally the ASP output is mapped back to natural language by LLMs, so to provide a captivating user experience.

Keywords

Answer Set Programming, Large Language Models, Knowledge Representation, Natural Language Generation

1. Introduction

Large Language Models (LLMs) and Answer Set Programming (ASP) are two distinct yet complementary paradigms that have emerged in Artificial Intelligence (AI) LLMs such as GPT [1], PaLM [2] and LLaMa [3], have revolutionized natural language processing (NLP) by achieving unprecedented levels of fluency and comprehension in textual data. Conversely, ASP [4, 5], a declarative programming paradigm rooted in logic programming under answer set semantics [6], excels in knowledge representation and logical reasoning, making it a cornerstone in AI systems requiring robust inference capabilities. Individually, LLMs and ASP offer distinct advantages in their respective domains. LLMs effortlessly handle several NLP tasks [7, 8], including language generation, summarization, and sentiment analysis, leveraging the power of deep learning and vast pre-trained language representations. Meanwhile, ASP empowers AI systems with the ability to reason over complex knowledge bases, derive logical conclusions, and solve intricate combinatorial problems, thereby facilitating decision-making in domains ranging from planning and scheduling [9, 10] to diagnosis and configuration [11, 12]. Recognizing the complementary nature of LLMs' linguistic prowess and ASP's reasoning capabilities, this paper proposes an approach that harnesses the synergies between these two paradigms, in the spirit of other recent works in the literature [13, 14]. Our goal is to develop a cohesive system that seamlessly integrates natural language understanding and logical inference, thereby enabling AI applications to navigate the intricate interplay between textual data and logical structures.

In this paper, we present a comprehensive framework for combining LLMs and ASP, leveraging the strengths of each paradigm to address the limitations of the other. We introduce a methodology for encoding domain-specific knowledge into input prompts using a YAML-based format, enabling LLMs to generate relational facts that serve as input to ASP for reasoning. Subsequently, the reasoned output from ASP is mapped back to natural language by LLMs, thereby facilitating a captivating user experience and enhancing the interpretability of the results. The YAML format serves as a flexible and intuitive mechanism for specifying the various components essential to our integrated LLM-ASP

CILC 2024: 39th Italian Conference on Computational Logic, June 26-28, 2024, Rome, Italy

*Corresponding author.
mario.alviano@unical.it (M. Alviano)
https://alviano.net/ (M. Alviano)
0000-0002-2052-2063 (M. Alviano)

system. Preprocessing prompts within the YAML specification provide instructions to the LLM on how to interpret and map input text into ASP facts. These prompts enable users to encode domain-specific knowledge and contextual information, guiding the generation of relational facts that serve as input to the ASP reasoning process. Moreover, the YAML format allows users to define a separate knowledge base, represented as an ASP program, which encapsulates logical rules, constraints, and background knowledge relevant to the task at hand. By decoupling the input prompts from the knowledge base, our approach facilitates modularity and reusability, empowering users to seamlessly adapt the system to diverse domains and problem instances. Additionally, postprocessing instructions specified in the YAML format provide guidance to the LLM on how to map atoms in the computed answer set generated by ASP reasoning process into human-readable text, thereby enhancing the interpretability and usability of the system's results. Overall, the YAML format serves as a comprehensive and versatile specification framework, enabling users to orchestrate the entire workflow of our LLM-ASP integration seamlessly.

Our prototype system orchestrates a seamless interaction between a LLM and an ASP solver Upon receiving an input text, the system interfaces with the LLM, providing predefined prompts, and enriching the prompts specified in the YAML format with additional predefined sentences. Subsequently, the responses generated by the LLM are processed to extract factual information, which serves as input to the ASP solver. The ASP solver then executes logical inference over the combined input facts and the specified knowledge base, computing an answer set that encapsulates reasoned conclusions and insights. To bridge the gap between logical output and natural language comprehension, the system once again engages with the LLM, providing predefined prompts and enriched specifications to map the computed answer set to coherent natural language sentences. Finally, the collected response sentences are summarized by the LLM, providing users with a concise and informative overview of the computed insights and conclusions.

2. Background

2.1. Large Language Models

Large Language Models (LLMs) are sophisticated artificial intelligence systems designed to understand and generate human-like text. These models are typically based on deep learning architectures, such as Transformers, and are trained on vast amounts of text data to learn complex patterns and structures of language. In this article, LLMs are used as black box operators on text (functions that take text in input and produce text in output). At each interaction with a LLM, the generated text is influenced by all previously processed text, and randomness is involved in the process. The text in input is called *prompt*, and the text in output is called *generated text* or *response*.

Example 1. Let us consider the following prompt:

> Encode as Datalog facts the following sentences: Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.

A response produced by Google Gemini is reported below. It is a very good starting point, but the LLM must be instructed on a specific format to use in encoding facts.

Here's the encoding of the sentence as Datalog facts:

```
wants_pizza(you). pizza_topping_preference(marco, onions).
dinner_companions(you, marco). dinner_companions(you, alessio).
```

This translates the information into facts: [bullet list omitted] ■

2.2. Answer Set Programming

All sets and sequences considered in this paper are finite. Let $\mathbf{P}$, $\mathbf{C}$, $\mathbf{V}$ be fixed nonempty sets of *predicate names*, *constants* and *variables*. Predicates are associated with an *arity*, a non-negative integer. A *term* is any element in $\mathbf{C} \cup \mathbf{V}$. An *atom* is of the form $p(\bar{t})$, where $p \in \mathbf{P}$, and $\bar{t}$ is a possibly empty sequence of terms. A *literal* is an atom possibly preceded by the default negation symbol `not`; they are referred to as positive and negative literals. An *aggregate* is of the form

$$\#\text{SUM}\{t_a, \overline{t'} : p(\overline{t})\} \odot t_g \tag{1}$$

where $\odot \in \{<, \leq, \geq, >, =, \neq\}$ is a binary comparison operator, $p \in \mathbf{P}$, $\overline{t}$ and $\overline{t'}$ are possibly empty sequences of terms, and t_a and t_g are terms. Let $\#\text{COUNT}\{\overline{t'} : p(\overline{t})\} \odot t_g$ be syntactic sugar for $\#\text{SUM}\{1, \overline{t'} : p(\overline{t})\} \odot t_g$. A *choice* is of the form

$$t_1 \leq \{atoms\} \leq t_2 \tag{2}$$

where $atoms$ is a possibly empty sequence of atoms, and t_1, t_2 are terms. Let $\perp$ be syntactic sugar for $1 \leq \{\} \leq 1$. A *rule* is of the form

$$head \ \texttt{:-} \ body. \tag{3}$$

where $head$ is an atom or a choice, and $body$ is a possibly empty sequence of literals and aggregates. (Symbol `:-` is omitted if *body* is empty. The head is usually omitted if it is $\perp$, and the rule is called *constraint*.) For a rule r, let $H(r)$ denote the atom or choice in the head of r; let $B^\Sigma(r)$, $B^+(r)$ and $B^-(r)$ denote the sets of aggregates, positive and negative literals in the body of r; let $B(r)$ denote the set $B^\Sigma(r) \cup B^+(r) \cup B^-(r)$.

Example 2. Let us consider the following rules:

```
index(1). index(2). index(3). succ(1,2). succ(2,3). empty(2,2).
grid(X,Y) :- index(X), index(Y).
0 <= {assign(X,Y)} <= 1 :-  grid(X,Y), not empty(X,Y).
:- #count{A,B : assign(A,B)} != 5.
:- assign(X,Y), succ(X,X'), assign(X',Y ).
:- assign(X,Y), succ(Y,Y'), assign(X ,Y').
```

The first line above comprises rules with atomic heads and empty bodies (also called *facts*). After that there is a rule with atomic head and nonempty body (also called a *definition*), followed by a choice rule, and three constraints. If r is the choice rule, then $H(r) = \{\texttt{0} <= \{\texttt{assign(X,Y)}\} <= \texttt{1}\}$, $B^+(r) = \{\texttt{grid(X,Y)}\}$, $B^-(r) = \{\texttt{not empty(X,Y)}\}$, and $B^\Sigma(r) = \emptyset$. ■

A variable X occurring in $B^+(r)$ is a *global variable*. Other variables occurring among the terms $\overline{t}$ of some aggregate in $B^\Sigma(r)$ of the form (1) are *local variables*. And any other variable occurring in r is an *unsafe variable*. A *safe rule* is a rule with no *unsafe variables*. A *program* Π is a set of safe rules. A substitution σ is a partial function from variables to constants; the application of σ to an expression E is denoted by $E\sigma$. Let $instantiate(\Pi)$ be the program obtained from rules of Π by substituting global variables with constants in $\mathbf{C}$, in all possible ways; note that local variables are still present in $instantiate(\Pi)$. The Herbrand base of Π, denoted $base(\Pi)$, is the set of ground atoms (i.e., atoms with no variables) occurring in $instantiate(\Pi)$.

Example 3. Variables A, B are local, and all other variables are global. Let Π be the program comprising all rules in Example 2 (which are safe). If $\mathbf{C} = \{1, 2, 3\}$, then $instantiate(\Pi)$ comprises, among others, the following rules:

```
grid(1,1) :- index(1), index(1).
0 <= {assign(1,1)} <= 1 :-  grid(1,1), not empty(1,1).
:- #count{A,B : assign(A,B)} != 5.
```

Note that the local variables A, B are still present in the last rule above. ■

An *interpretation* is a set of ground atoms. For an interpretation I, relation $I \models \cdot$ is defined as follows: for a ground atom $p(\overline{c})$, $I \models p(\overline{c})$ if $p(\overline{c}) \in I$, and $I \models \texttt{not}\ p(\overline{c})$ if $p(\overline{c}) \notin I$; for an aggregate α of the form (1), the aggregate set of α w.r.t. I, denoted $aggset(\alpha, I)$, is $\{\langle t_a, \overline{t'}\rangle\sigma \mid p(\overline{t})\sigma \in I$, for some

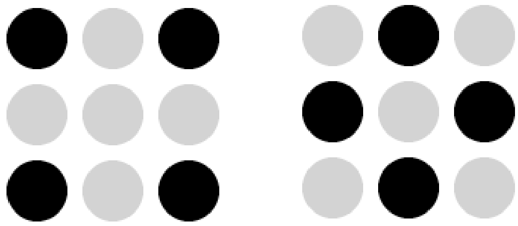

Figure 1: The solutions encoded by the two answer sets of program Π in Example 2.

substitution $\sigma\}$, and $I \models \alpha$ if $(\sum_{\langle c_a, \overline{c'}\rangle \in aggset(\alpha,I)} c_a) \odot t_g$ is a true expression over integers; for a choice α of the form (2), $I \models \alpha$ if $t_1 \leq |I \cap atoms| \leq t_2$ is a true expression over integers; for a rule r with no global variables, $I \models B(r)$ if $I \models \alpha$ for all $\alpha \in B(r)$, and $I \models r$ if $I \models H(r)$ whenever $I \models B(r)$; for a program Π, $I \models \Pi$ if $I \models r$ for all $r \in instantiate(\Pi)$.

For a rule r of the form (3) and an interpretation I, let $expand(r, I)$ be the set $\{p(\overline{c})\ \texttt{:-}\ body. \mid p(\overline{c}) \in I \text{ occurs in } H(r)\}$. The *reduct* of Π w.r.t. I is the program comprising the expanded rules of $instantiate(\Pi)$ whose body is true w.r.t. I, that is, $reduct(\Pi, I) := \bigcup_{r \in instantiate(Pi),\ I \models B(r)} expand(r, I)$. An *answer set* of Π is an interpretation A such that $A \models \Pi$ and no $I \subset A$ satisfies $I \models reduct(\Pi, A)$.

Example 4 (Continuing Example 3)**.** Program Π has two answer sets, which comprise the following instances of predicate `assign/2`:

1. `assign(1,2), assign(2,1), assign(2,3), assign(3,2);`
2. `assign(1,1), assign(1,3), assign(3,1), assign(3,3).`

Figure 1 shows the solutions encoded by the two answer sets. ∎

The language of ASP supports several other constructs, among them binary built-in relations (i.e., `<`, `<=`, `>=`, `>`, `==`, `!=`) which are interpreted naturally.

2.3. YAML

YAML (YAML Ain't Markup Language; https://yaml.org/spec/1.2.2/) is a human-readable data serialization format commonly used for configuration files, data exchange, and representation of structured data. YAML is designed to be easily readable by humans and is commonly used in software development for configuration files, data storage, and data interchange between different programming languages. YAML uses indentation to denote nesting and relies on simple syntax rules, such as key-value pairs and lists, to represent structured data. YAML is often preferred for its simplicity, readability, and flexibility compared to other serialization formats like JSON and XML. In this article, we focus on the following restricted fragment: A *scalar* is any number or string (possibly quoted). A *block sequence* is a sequence of entries, each one starting by a dash followed by a space. A *mapping* is a sequence of key-value pairs, each pair using a colon and space as separator, where keys and values are scalars. A scalar can be written in *block notation* using the prefix `|` (if not a key). Lines starting with # are *comments*.

Example 5. Below is a YAML document:

```
name: Lorenzo
degrees:
- Bachelor
short bio: |
  I'm Lorenzo...
  I'm a student at UNICAL...
```

It encodes a mapping with keys `name`, `degrees` and `short bio`. Key `name` is associated with the scalar `Lorenzo`. Key `degrees` is associated with the list `[Bachelor]`. Key `short bio` is associated with a scalar in block notation. ∎

3. YAML format

The interaction between LLMs and ASP is achieved by means of a YAML specification defined in this section. The specification expects a mapping with keys `preprocessing`, `background knowledge`, and `postprocessing`. The values associated with `preprocessing` and `postprocessing` are mappings where keys are either atoms or the special value _ (used for providing a context), and values are scalars. The value associated with `background knowledge` is an ASP program. Given a YAML specification S, let $pre_S(\alpha)$ be the value associated with α in the `preprocessing` mapping, where α is either an atom or _. Similarly, let $post_S(\alpha)$ be the value associated with α in the `postprocessing` mapping, where α is either an atom or _. Finally, let kb_S be the ASP program in `background knowledge`.

Example 6. Below is an example about catering suggestions.

```
preprocessing:
- _: The user provides a request to obtain catering suggestions. The user
can mention a day, other persons, and their cuisine preferences.
- person("who"): List all the persons mentioned including me if indirectly
   included.
- cuisine_preferences("who", "country"): For each person, list any
   restaurant preferences.
- want_food("who", "what"): For each person, list what they want to eat.

knowledge base: |
  can_go_together(X,Y,Z) :- person(X), person(Y), X < Y,
    want_food(X,Z), want_food(Y,Z).
  can_go_together(X,Y,Z) :- person(X), person(Y), X < Y,
    cuisine_preferences(X,Z), cuisine_preferences(Y,Z).
  #show can_go_together/3.

postprocessing:
- _: Explain the facts in a human readable way, as a paragraph.
- can_go_together("person 1", "person 2", "cuisine preference"): Say that
   "person 1" can go with "person 2" to eat "cuisine preference".
```

The preprocessing aims at extracting data about cuisine preferences of a group of persons. These data are combined with the knowledge base to identify pairs of persons with compatible preferences. Finally, the postprocessing prompts aim at producing a paragraph reporting the identified pairs of persons with compatible preferences. ■

A YAML specification S and an input text T are processed as follows:

P1. The set F of facts and R of responses are initially empty, and the LLM is invoked with the prompt

```
You are a Natural Language to Datalog translator. To translate your
input to Datalog, you will be asked a sequence of questions. The
answers are inside the user input provided with
[USER_INPUT]input[/USER_INPUT] and the format is provided with
[ANSWER_FORMAT]predicate(terms).[/ANSWER_FORMAT]. Predicate is a
lowercase string (possibly including underscores). Terms is a
comma-separated list of either double quoted strings or integers.
Be sure to control the number of terms in each answer!
An answer MUST NOT be answered if it is not present in the user input.
Remember these instructions and don't say anything!
```

P2. If $pre_S(\_)$ is defined, the LLM is invoked with the prompt

```
Here is some context that you MUST analyze and remember.
```
$pre_S(\_)$
```
Remember this context and don't say anything!
```

P3. For each atom α such that $pre_S(\alpha)$ is defined, the LLM is invoked with the prompt

[USER_INPUT]T[/USER_INPUT]
$pre_S(\alpha)$
[ANSWER_FORMAT]α.[/ANSWER_FORMAT]

Facts in the response are collected in F. Everything else is ignored.

P4. An answer set of $kb_S \cup \{\alpha. \mid \alpha \in F\}$ is searched, say I. If an answer set does not exist, the process terminates with a failure.

P5. The LLM is invoked with the prompt

```
You are now a Datalog to Natural Language translator.
You will be given relational facts and mapping instructions.
Relational facts are given in the form [FACTS]atoms[/FACTS].
Remember these instructions and don't say anything!
```

P6. If $post_S(\_)$ is defined, the LLM is invoked with the prompt

```
Here is some context that you MUST analyze and remember.
```
$post_S(\_)$
```
Remember this context and don't say anything!
```

P7. For each atom $p(\bar{t})$ such that $post_S(p(\bar{t}))$ is defined, the LLM is invoked with the prompt

[FACTS]$\{p(\bar{t}') \mid p(\bar{t}') \in I\}$[/FACTS]
Each fact matching $p(\bar{t})$ must be interpreted as follows: $post_S(p(\bar{t}))$

Responses are collected in R.

P8. The LLM is invoked with the prompt

Summarize the following responses: R

The response is provided in output.

Example 7. Let S be the specification given in Example 6. Let T be the following text:

> Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.

The LLM is invoked with the initial fixed prompt of **P1**, and then with the prompt of **P2**:

> Here is some context that you MUST analyze and remember.
> The user provides a request to obtain catering suggestions. The user can mention a day, other persons, and their cuisine preferences.
> Remember this context and don't say anything!

After that, the LLM is invoked seven times with the prompt of **P3** to populate set F. For example,

> [USER_INPUT]Tonight I want to go to eat some pizza with Marco and Alessio. Marco really like the pizza with onions as toppings.[/USER_INPUT]
> List all the persons mentioned including me if indirectly included.
> [ANSWER_FORMAT]person("who").[/ANSWER_FORMAT]

The LLM may provide the response

> [ANSWER_FORMAT]person("me").[/ANSWER_FORMAT]
> [ANSWER_FORMAT]person("marco").[/ANSWER_FORMAT]
> [ANSWER_FORMAT]person("alessio").[/ANSWER_FORMAT]

from which the following facts are extracted and added to F:

```
person("me"). person("marco"). person("alessio").
```

Once all facts are collected, the knowledge base is used to search an answer set (**P4**), say one containing the following atoms:

```
can_go_together("me", "marco", "pizza").
can_go_together("me", "alessio", "pizza").
can_go_together("marco", "alessio", "pizza").
```

The LLM is now invoked with the prompts of **P5**–**P7**. In particular, the prompt of **P7** is the following:

> [FACTS]can_go_together("me", "marco", "pizza"). can_go_together("me", "alessio", "pizza"). can_go_together("marco", "alessio", "pizza"). [/FACTS]
>
> Each fact matching can_go_together("person 1", "person 2", "cuisine preference") must be interpreted as follows: Say that "person 1" can go with "person 2" to eat "cuisine preference".

The response, say

> All three of you, including yourself, Marco and Alessio, enjoy pizza. This means everyone would be happy to go on a pizza outing together. It's a perfect situation for a group pizza party!

is added to R. Finally, the LLM is invoked with the prompt of **P8**:

> Summarize the following responses: All three of you, including yourself, Marco and Alessio, enjoy pizza. This means everyone would be happy to go on a pizza outing together. It's a perfect situation for a group pizza party!

The response, say

> Based on the analysis of the facts, it appears you, Marco, and Alessio all enjoy pizza, making a group pizza party a perfect option for your outing tonight.

is provided in output. ■

4. Conclusion

In this paper, we have introduced an approach for combining Large Language Models (LLMs) and Answer Set Programming (ASP) to harness their complementary strengths in natural language understanding and logical reasoning. Our prototype system (https://github.com/Xiro28/LLMASP) is written in Python, and it is powered by the LLM TheBloke/Llama-2-13B-chat-GGUF from HuggingSpace and the CLINGO Python API [15]. By providing predefined prompts and enriching specifications with domain-specific knowledge, our approach enables users to tailor the system to diverse problem domains and applications, enhancing its adaptability and versatility. Future research directions encompass the evaluation of the quality of the answers provided by our integrated LLM-ASP system, and exploring with different prompts to improve the overall quality of the system.

Acknowledgments

This work was supported by Italian Ministry of University and Research (MUR) under PRIN project PRODE "Probabilistic declarative process mining", CUP H53D23003420006 under PNRR project FAIR "Future AI Research", CUP H23C22000860006, under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and under PNRR project SERICS "SEcurity and RIghts in the CyberSpace", CUP H73C22000880001; by Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); and by the LAIA lab (part of the SILA labs). Alviano is member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

[1] T. B. Brown, et al., Language models are few-shot learners, CoRR abs/2005.14165 (2020). URL: https://arxiv.org/abs/2005.14165. arXiv:2005.14165.

[2] A. Chowdhery, et al., 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.

[3] H. Touvron, et al., Llama: Open and efficient foundation language models, CoRR abs/2302.13971 (2023). doi:10.48550/ARXIV.2302.13971. arXiv:2302.13971.

[4] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-year Perspective, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[5] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[6] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. Warren, P. Szeredi (Eds.), Logic Programming: Proc. of the Seventh International Conference, 1990, pp. 579–597.

[7] H. Jin, Y. Zhang, D. Meng, J. Wang, J. Tan, A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods, CoRR abs/2403.02901 (2024). doi:10.48550/ARXIV.2403.02901. arXiv:2403.02901.

[8] W. Zhang, X. Li, Y. Deng, L. Bing, W. Lam, A survey on aspect-based sentiment analysis: Tasks, methods, and challenges, IEEE Trans. Knowl. Data Eng. 35 (2023) 11019–11038. doi:10.1109/TKDE.2022.3230975.

[9] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Logic-based benders decomposition in answer set programming for chronic outpatients scheduling, Theory Pract. Log. Program. 23 (2023) 848–864. doi:10.1017/S147106842300025X.

[10] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, Solving rehabilitation scheduling problems via a two-phase ASP approach, Theory Pract. Log. Program. 24 (2024) 344–367. doi:10.1017/S1471068423000030.

[11] F. Wotawa, On the use of answer set programming for model-based diagnosis, in: H. Fujita, P. Fournier-Viger, M. Ali, J. Sasaki (Eds.), Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices - 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings, volume 12144 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 518–529. doi:10.1007/978-3-030-55789-8_45.

[12] R. Taupe, G. Friedrich, K. Schekotihin, A. Weinzierl, Solving configuration problems with ASP and declarative domain specific heuristics, in: M. Aldanondo, A. A. Falkner, A. Felfernig, M. Stettinger (Eds.), Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021), Vienna, Austria, 16-17 September, 2021, volume 2945 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 13–20. URL: https://ceur-ws.org/Vol-2945/21-RT-ConfWS21_paper_4.pdf.

[13] K. Basu, S. C. Varanasi, F. Shakerin, J. Arias, G. Gupta, Knowledge-driven natural language understanding of english text and its applications, in: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, AAAI Press, 2021, pp. 12554–12563. doi:10.1609/AAAI.V35I14.17488.

[14] Y. Zeng, A. Rajasekharan, P. Padalkar, K. Basu, J. Arias, G. Gupta, Automated interactive domain-specific conversational agents that understand human dialogs, in: M. Gebser, I. Sergey (Eds.), Practical Aspects of Declarative Languages - 26th International Symposium, PADL 2024, London, UK, January 15-16, 2024, Proceedings, volume 14512 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 204–222. doi:10.1007/978-3-031-52038-9_13.

[15] R. Kaminski, J. Romero, T. Schaub, P. Wanko, How to build your own asp-based system?!, Theory and Practice of Logic Programming 23 (2023) 299–361. doi:10.1017/S1471068421000508.