Exploring Neurosymbolic Systems in Answer Set Programming with an Application to Waste Water Monitoring

Luis Angel Rodriguez Reiners

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

Abstract

Despite their remarkable achievements in tasks such as language generation or sentiment analysis, traditional LLMs face significant challenges when tasked with complex reasoning and problem-solving. Answer Set Programming (ASP) and Large Language Models (LLMs) offer complementary strengths in symbolic reasoning and natural language understanding. Our research interest lay in combining both paradigms, applied to waste water monitoring, to improve the structured representation of complex knowledge encoded in natural language and enable more interactive and adaptive problem-solving workflows. This paper presents the integration between these approaches. LLMASP, a neurosymbolic tool, where the interaction between ASP and LLMs is driven by a YAML file specifying prompt templates and domain-specific background knowledge, combining syntactic structures extracted by LLMs and semantic aspects encoded in the knowledge base. Moreover, within the platform ASP Chef, a web-based environment which provides an intuitive "recipe" framework for operations on answer sets, we present ASP Chef own specific LLM integration and the tool utility to interactive data analysis and visualization.

Keywords

Answer Set Programming, Neurosymbolic, Large Language Models, Visualization, ASP-Chef

1. Introduction

Large Language Models (LLMs) and Answer Set Programming (ASP) represent two distinct but complementary paradigms in Artificial Intelligence (AI). LLMs, such as GPT [1], PaLM [2], and LLaMa [3], have transformed natural language processing (NLP) by achieving unprecedented levels of fluency and understanding of text. In contrast, ASP [4, 5], a declarative programming paradigm rooted in logic programming under answer set semantics [6], excels in knowledge representation and logical inference, making it fundamental for AI systems that require robust logical capabilities. Individually, LLMs and ASP offer unique advantages within their respective domains. LLMs excel at various NLP tasks [7, 8], such as language generation, summarization, and sentiment analysis, utilizing deep learning and extensive pre-trained language models. In contrast, ASP equips AI systems with robust reasoning capabilities, enabling them to process complex knowledge bases, draw logical conclusions, and tackle intricate combinatorial problems. This makes ASP particularly effective in decision-making scenarios, including planning and scheduling [9, 10], as well as diagnosis and configuration [11, 12] tasks in which LLMs do not perform well. Recognizing the complementary strengths and inherent limitations of LLMs, notably their advanced linguistic capabilities but limited formal reasoning, and ASP's logical capabilities, this paper proposes an approach that leverages the synergies between these two paradigms, inspired by recent works in the literature [13, 14]. Our objective is to develop neurosymbolic systems that seamlessly integrates natural language understanding with logical inference, allowing AI applications to adeptly handle the complex interplay between textual data and logical structures.

The LLMASP framework [15] integrates LLMs and ASP to extract structured relational knowledge from natural language and reason over it declaratively. LLMASP is designed as a multi-stage pipeline driven by two YAML configuration files: the *behavior file*, which specifies general prompting strategies

ICLP DC 2025: 21st Doctoral Consortium on Logic Programming, September 2025, Rende, Italy.
luis.reiners@unical.it (L. A. Rodriguez Reiners)
0009-0000-1808-9910 (L. A. Rodriguez Reiners)

and output formats, and the *application file*, which describes the domain-specific predicates to be extracted. Given a user query, the system generates multiple LLM prompts, each tailored to extract a specific predicate, parses the LLM responses as ASP facts, and evaluates them using an ASP solver.

ASP Chef [16], a web-based platform for designing and executing ASP-based pipelines, improves the user experience of ASP and simplifies its use in various computational tasks. ASP Chef has its own integration with LLMs [17] proving particularly usability in three areas. In an educational context, it enables users to quickly generate example data by leveraging the broad knowledge base of LLMs, to facilitate the understanding of ASP concepts and applications. For fast prototyping experience, provides suggestions on ASP syntax, debugging hints and clarification regarding the platform documentation. Finally, in practical problem-solving, LLMs can be used to extract structured data from unstructured sources, which is then processed through ASP Chef recipes for logical reasoning; the results of the reasoning process can subsequently be mapped back into natural language (using LLMs) to improve their interpretability. This last area aligns with the design of LLMASP where the natural language and reasoning abilities of the LLMs and ASP are leveraged, respectively. We also illustrate the application of ASP Chef within the context of the Tech4You project (Technologies for climate change adaptation and quality of life improvement; https://iia.cnr.it/project/tech4you/), where one of the goals is to develop intelligent tools for analyzing environmental data, particularly focusing on water quality monitoring.

2. Background

LLMs are AI systems designed to process and generate human-like text. These models are 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 our neurosymbolic approach, LLMs are used as black box operators on text (functions that take text in input and produce text in output). The text in input is called *prompt*, and the text in output is called *generated text* or *response*. The prompt is a sequence of messages from three roles, namely *system*, *user* and *assistant*. System messages set behavior, tone, and context for the assistant. User messages represent queries to or instructions for the assistant. Assistant messages are responses to user queries.

An ASP program is a finite set of rules. Each rule typically has a head, representing a conclusion (which may be atomic or a choice), and a body, representing a set of conditions that must hold (a conjunction of literals, aggregates and inequalities). Formally, an ASP program Π induces a collection (zero or more) of answer sets (also known as stable models), which are interpretations that satisfy all the rules in Π while also fulfilling the stability condition (i.e., the models must be supported and minimal in a specific formal sense [6]). Quantitative preferences can be expressed in terms of weak constraints. The intended output of a program can be specified using `#show` directives of the form

$$\texttt{\#show}\ p(\overline{t})\ \ \texttt{:}\ \ \mathit{conjunctive\_query}\,.$$

Here, p denotes an optional predicate symbol, $\overline{t}$ is a (possibly empty) sequence of terms, and $\mathit{conjunctive\_query}$ is a conjunction of literals serving as a condition for displaying instances of $p(\overline{t})$. Answer sets are then projected accordingly. For a detailed specification of syntax and semantics, including `#show` and other directives, we refer to the ASP-Core-2 standard format [18].

ASP Chef [16, 19, 20, 21] is a web-based platform where ASP users can create and execute complex workflows, known as recipes, which are executed directly within the browser and can include data manipulation and visualization operations. An *operation* O is a function receiving in input a sequence of interpretations and producing in output a sequence of interpretations. Operations may produce side outputs (e.g., a graph visualization) and accept parameters to influence their behavior. An *ingredient* is an instantiation of a parameterized operation with side output. A *recipe* is a tuple of the form $(\mathit{encode}, \mathit{Ingredients}, \mathit{decode})$, where $\mathit{Ingredients}$ is a (finite) sequence $O_1\langle P_1\rangle, \ldots, O_n\langle P_n\rangle$ of ingredients, and encode and decode are Boolean values. If encode is true, the input of the recipe is mapped to $[[\texttt{\_\_base64\_\_("}s\texttt{")}]]$, where $s = \mathit{Base64}(s_{in})$ (i.e., the Base64–encoding of the input string s_{in}). After that, the ingredients are applied one after another. Finally, if decode is true, every occurrence of $\texttt{\_\_base64\_\_(}s\texttt{)}$ is replaced with (the ASCII string associated with) $\mathit{Base64}^{-1}(s)$.

Works as [22] use the LLMs to transform a context and a question into atomic facts, which are processed by an ASP solver equipped with background knowledge encoded as ASP rules to derive an answer, focusing on *question answering*. Their approach uses *few-shot* examples rather than training datasets. Another argument supporting the use of LLMs for data extraction only rather than for reasoning is given by [23], suggesting that LLMs are suitable for System-1 thinking: LLMs are designed to predict the next word in a sequence without deep comprehension of crucial reasoning concepts like causality, logic, and probability. Another work aiming at generating ASP programs is presented by [24], the main element in their work is a prompt engineering strategy to transform natural language descriptions into ASP incrementally. The proposed pipeline initially identifies the relevant objects and their categories. Subsequently, it forms a predicate that delineates the relationships between objects from various categories. Using these derived data, the pipeline proceeds to build an ASP program following the Generate-Define-Test paradigm. Merging LLMs with logical reasoning into neurosymbolic frameworks is an active research field. In our work, we employ LLMs in information extraction tasks, and in particular, we focus on the extraction of relational facts from text with the aim of addressing subsequent reasoning tasks with formal logic systems. That is the approach followed by LLMASP and ASP Chef, contributing to ease the interface of LLMs and ASP.

3. Goal of the research

The primary goal of this research is to develop and validate practical neurosymbolic frameworks that synergistically leverages the strengths of Answer Set Programming (ASP) for robust logical reasoning and Large Language Models (LLMs) for advanced natural language understanding and generation; to make declarative programming paradigms more accessible, intuitive, and powerful for addressing complex real-world problems. We address this objective by:

- Developing methods, exemplified by the LLMASP system, to accurately extract structured facts from natural language to support robust ASP reasoning and enable coherent translation of ASP outputs back into human-readable form.
- Enhancing ASP Chef into a more flexible web tool that uses a "recipe" approach to simplify practical ASP use and support interactive workflows with LLM integration, advancing accessible neurosymbolic AI for researchers and practitioners.
- Making declarative programming more accessible by enabling natural language interaction, intuitive visualizations, and transparent reasoning, empowering users to better understand, control, and trust AI systems.

4. Current Status

The architecture of LLMASP is shown in Figure 1 [25, 26, 15]. LLMASP takes in input two YAML files, namely the behavior file B and the application file A, together with a database file D (comprising facts) and a request text T (expressed by the user in natural language). A set F of facts is populated and a database is given in output. This section defines B and A, as well as the the LLMASP pipeline.

The **behavior file** specifies global behavior settings for LLMASP, such as tone, style and general instructions for the LLM, while the **application file** contains domain-specific guidelines, as a description of the context and mappings between facts and their corresponding natural language translation.

The pipeline implemented by LLMASP to map natural language into ASP facts follows a structured four-step process, namely **P1**–**P4**. First, the system initializes the language model using a predefined prompt from the behavior file (**P1**). Next, if a general context prompt is specified in the application file, it is inserted into a template from the behavior file to extend the model's input (**P2**). Then, for each specific instruction defined in the application file, the LLM is queried using a customized prompt that includes the target input, the atom being defined, and its corresponding instruction; only the resulting ASP facts are extracted and stored (**P3**). Finally, these collected facts, along with predefined background knowledge, are used to compute an answer set (**P4**).

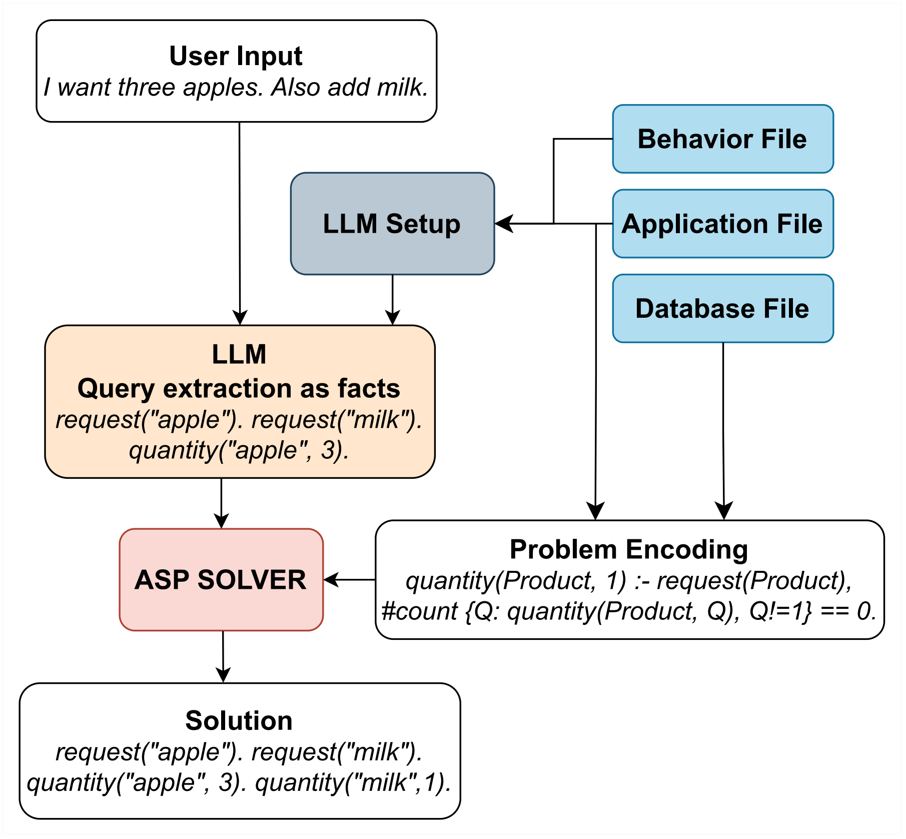
User Input
I want three apples. Also add milk.
Behavior File
LLM Setup
Application File
Database File
LLM
Query extraction as facts
request("apple"). request("milk").
quantity("apple", 3).
ASP SOLVER
Problem Encoding
quantity(Product, 1) :- request(Product),
#count {Q: quantity(Product, Q), Q!=1} == 0.
Solution
request("apple"). request("milk").
quantity("apple", 3). quantity("milk",1).

Figure 1: Graphical representation of the LLMASP pipeline. The pipeline begins with a user query formulated in natural language. The system also receives two YAML files: one specifying the system's behavior and another detailing the context of the specific application under development. The behavior YAML file provides initial prompts for the LLM, which are enriched with contextual details derived from the application YAML file and the user's query. These enriched prompts guide the LLM in extracting relevant information from the user input, transforming it into structured factual representations. These facts are subsequently integrated into an ASP framework, where they are combined with the existing knowledge base and database, and processed by the solver to compute an answer set.

In ASP Chef we introduce operations to register API keys of LLM servers, to configure endpoints and models to use, and to perform remote *chat completion* requests. Such requests use messages stored in ASP facts and can incorporate mustache queries [27] to dynamically include data from other facts. This approach enables prompts for LLMs to be generated from templates, where placeholders are automatically replaced with the results of mustache queries. As a result, ASP and LLMs can be seamlessly combined for dynamic and context-aware interactions. The response generated by the LLM is stored as an ASP fact, allowing seamless integration with other ASP Chef operations. For example, if the LLM outputs a response in comma-separated values (CSV) format, the *Parse CSV* operation can transform each value into a structured fact. Subsequently, the *Search Models* operation can process these facts to derive a meaningful relational representation. Alternatively, the usual Markdown format used by LLMs can be processed by the *Markdown* operation of ASP Chef to visualize the generated response as a side output of the recipe.

Config. The *@LLMs/Config* operation extends each input interpretation with facts representing parameters like server, model and temperature. These facts have the form `__llms_config__(key,"value")`, where `__llms_config__` can be set in the ingredient.

Chat Completion. For each input interpretation I, the *@LLMs/Chat Completion* takes the configuration from instances of `__llms_config__` (or the predicate specified in the ingredient), and messages from atoms of the form `__message__(role("content"))`, where `role` is `system`, `user` or `assistant`, and `content` is a string with *mustache queries*, defined in [27]. The response given by the LLM is

Base64-encoded in the predicate `__base64__` (or the predicate specified in the ingredient), and can be further processed by the subsequent ingredients in the recipe.

Register/Unregister API Keys. Servers usually expect an API key to be provided with each request. In ASP Chef, such API keys are retrieved from the session storage, where they are saved via a *@LLMs/Register API Key* ingredient. API keys enabled in the current session can be disabled via a *@LLMs/Unregister API Keys*, which also lists the permanently stored API keys and gives the possibility to remove them selectively or in block.

Interactive representations of ASP interpretations are now possible thanks to recent integrations of data visualization tools into ASP Chef [27] with mustache templates. The mustache syntax is essential because it offers a logic-less templating system that converts raw ASP outputs into formats that javascript libraries like Tabulator, Chart.js, and vis.js may use directly. The efforts described in [19] are enhanced by new operations like *Tabulator*, *Chart.js*, and *@vis.js/Network*, which make fully browser-based interactive visualizations possible.

5. Preliminary results and Open Issues

Our LLMASP implementation is powered by Ollama, OpenAI API and CLINGO [28]. We focus on two models, Llama 3.1 with 8 and 70 billions parameters. In order to assess LLMASP empirically, we defined a dataset using domains from ASP Competitions [29]. Specifically, we use the description of the domain and format of the facts available online, and systematically generate text representations (of portions) of the instances. We therefore aim at extracting the facts from the generated text, and adopt F1-score as a measure of quality. The dataset consists of 32 test cases for each domain problem, and the generation of text is obtained by randomly applying templates (among several alternatives) to randomly selected facts. The templates are structured to ensure variability in the generated natural language descriptions. For example, the application of some templates of *Incremental Scheduling* (IS) to the facts

```
job(1). job(3). job(14). job(17). deadline(3,14). job_len(1,14). importance(1,2).
precedes(14,17). device(1). job_device(1,1). offline_instance(1,1).
```

may result to the following text:

> The job 3 has a deadline 14. The job 1 has an importance of 2, must be executed by device 1, needs 14 timesteps to be performed. Device 1 has instance one offline. Job 14 must finish before the start time of 17.

Our first experiment in [15] consisted of testing the ability of the models to extract facts from natural text. For llama3.1 models with 8b and 70b parameters, best results were obtained when including descriptions and formats of the problems with an F1-score of 73.7% and 90.4%, respectively. These initial results suggest that clearly defining the fact format is crucial. We employ LLMASP to test several prompts, by combining behavior and application files with different features. Best result were obtained for the llama3.1 70b model with 91.1% with a behavior file combining extraction example in the context, repeated instructions on the extraction task in the mapping and indication in the context to reply NONE if the answer is missing, while for the 8b version was 80.9%, when also including repeated indication on NONE in the mapping. Our last benchmark involves domains for which part of the generation process can be addressed in ASP after some facts are extracted. We therefore encode such parts in the `knowledge base` section of the application files. We observed a sensible improvement in some domains with discrete results, up to 46% for the 8b model. With a few exceptions, the use of ASP is beneficial for the generation process [15].

The integration of LLMs with ASP Chef was accepted in the Demonstrations Track of the 34th International Joint Conference on Artificial Intelligence (IJCAI) [17]. Moreover, in [30] ASP Chef is used to visualize and analyze water quality data collected by multisensory buoys, which monitor a wide range of chemical and physical parameters. Selected portions of the cleaned data are explored via ASP Chef recipes. The presented results show that ASP Chef enables effective visual exploration of parameter trends, detection of critical values through logic queries, the creation of interactive dashboards for domain experts and illustrates how declarative logic programming can be combined with modern front-end technologies to build practical tools for environmental monitoring and decision support. Extending the work on [30] the recipe https://tinyurl.com/aspchf/ICLP2025-DC/llm leverages the LLMs to automatically generate high-level conclusions based on the visualizations produced by other ingredients in the recipe. In this case, we are plotting the conductivity and pH trends over time, the *@LLMs/Chat Completion* ingredient receives this context and produces human-readable summaries, such as

> The graph shows stable conductivity levels from January 17-19, 2023, with a dramatic simultaneous drop on January 17th and January 19th where both parameters fell sharply. This sudden decline suggests a significant water chemistry change, likely due to dilution from freshwater influx or rainfall.

This approach demonstrates a neurosymbolic integration, where symbolic reasoning provided by ASP handles structured, rule-based data interpretation, while neural models provide language-based explanations. By embedding the LLM within ASP Chef, we enable a more intuitive understanding of complex data patterns, opening up possibilities for interactive analysis and domain expert engagement without requiring deep technical knowledge of logic programming.

Several open issues remain critical for advancing the LLMASP and ASP Chef frameworks. For LLMASP, current efforts include an under submission work presenting an enhanced version of the framework that employs grammar-constrained decoding. Using formal grammars gives precise control over LLM outputs, reducing hallucinations and associated costs. Ongoing work on ASP Chef aims to enhance its interactivity by integrating JavaScript frameworks like SurveyJS for dynamic form creation. This extends its usefulness in domains requiring real-time data input and interactive workflows, complementing its existing input and result visualizations.

Acknowledgments

This work was supported by the Italian Ministry of University and Research (MUR) under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and by the Italian Ministry of Enterprises and Made in Italy under PN RIC project ASVIN "Assistente Virtuale Intelligente di Negozio" (CUP B29J24000200005);

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT-4o in order to: Grammar and spelling check. After using this tool, the author reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] 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: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[2] 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: https://jmlr.org/papers/v24/22-1144.html.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models, CoRR abs/2302.13971 (2023). URL: https://doi.org/10.48550/arXiv.2302.13971. doi:10.48550/ARXIV.2302.13971. arXiv:2302.13971.

[4] V. W. Marek, M. Truszczynski, Stable models and an alternative logic programming paradigm, in: K. R. Apt, V. W. Marek, M. Truszczynski, D. S. Warren (Eds.), The Logic Programming Paradigm - A 25-Year Perspective, Artificial Intelligence, Springer, 1999, pp. 375–398. URL: https://doi.org/10.1007/978-3-642-60085-2_17. doi:10.1007/978-3-642-60085-2_17.

[5] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (1999) 241–273. URL: https://doi.org/10.1023/A:1018930122475. doi:10.1023/A:1018930122475.

[6] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. H. D. Warren, P. Szeredi (Eds.), Logic Programming, Proceedings of the Seventh International Conference, Jerusalem, Israel, June 18-20, 1990, MIT Press, 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] M. Alviano, L. Grillo, F. Lo Scudo, L. A. R. Reiners, Integrating answer set programming and large language models for enhanced structured representation of complex knowledge in natural language, in: IJCAI 2025, Montreal, Canada, August 16-22, 2025, https://tinyurl.com/ijcai25-llmasp, 2025.

[16] M. Alviano, D. Cirimele, L. A. R. Reiners, Introducing ASP recipes and ASP chef, in: J. Arias, S. Batsakis, W. Faber, G. Gupta, F. Pacenza, E. Papadakis, L. Robaldo, K. Rückschloß, E. Salazar, Z. G. Saribatur, I. Tachmazidis, F. Weitkämper, A. Z. Wyner (Eds.), Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023, volume 3437 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3437/paper4ASPOCP.pdf.

[17] M. Alviano, P. Macrì, L. A. R. Reiners, Asp chef chats with llms, in: Proceedings of the Thirty-Four International Joint Conference on Artificial Intelligence, IJCAI-25, International Joint Conferences on Artificial Intelligence Organization, 2025. Demo Track.

[18] 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.

[19] M. Alviano, L. A. Rodriguez Reiners, ASP chef: Draw and expand, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. URL: https://doi.org/10.24963/kr.2024/68. doi:10.24963/KR.2024/68.

[20] M. Alviano, P. Guarasci, L. A. R. Reiners, I. R. Vasile, Integrating structured declarative language (SDL) into ASP chef, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11-14, 2024, Proceedings, volume 15245 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 387–392. URL: https://doi.org/10.1007/978-3-031-74209-5_29. doi:10.1007/978-3-031-74209-5_29.

[21] M. Alviano, L. A. R. Reiners, Integrating minizinc with ASP chef: Browser-based constraint programming for education and prototyping, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11-14, 2024, Proceedings, volume 15245 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 174–186. URL: https://doi.org/10.1007/978-3-031-74209-5_14. doi:10.1007/978-3-031-74209-5_14.

[22] Z. Yang, A. Ishay, J. Lee, Coupling large language models with logic programming for robust and general reasoning from text, in: A. Rogers, J. L. Boyd-Graber, N. Okazaki (Eds.), Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023, Association for Computational Linguistics, 2023, pp. 5186–5219. URL: https://doi.org/10.18653/v1/2023.findings-acl.321. doi:10.18653/V1/2023.FINDINGS-ACL.321.

[23] M. I. Nye, M. H. Tessler, J. B. Tenenbaum, B. M. Lake, Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning, in: M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, J. W. Vaughan (Eds.), Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, 2021, pp. 25192–25204. URL: https://proceedings.neurips.cc/paper/2021/hash/d3e2e8f631bd9336ed25b8162aef8782-Abstract.html.

[24] A. Ishay, Z. Yang, J. Lee, Leveraging large language models to generate answer set programs, in: P. Marquis, T. C. Son, G. Kern-Isberner (Eds.), Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023, 2023, pp. 374–383. URL: https://doi.org/10.24963/kr.2023/37. doi:10.24963/KR.2023/37.

[25] M. Alviano, L. Grillo, Answer set programming and large language models interaction with

YAML: preliminary report, in: E. D. Angelis, M. Proietti (Eds.), CILC 2024, Rome, Italy, June 26-28, 2024, volume 3733 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3733/short2.pdf.

[26] M. Alviano, F. L. Scudo, L. Grillo, L. A. R. Reiners, Answer set programming and large language models interaction with YAML: second report, in: L. G. Á. et al. (Ed.), KoDis+CAKR+SYNERGY@KR 2024, Hanoi, Vietnam, November 2-8, 2024, volume 3876 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3876/paper3.pdf.

[27] M. Alviano, W. Faber, L. A. Rodriguez Reiners, ASP Chef grows Mustache to look better, 2025. URL: https://arxiv.org/abs/2505.24537. `arXiv:2505.24537`.

[28] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, TPLP 11 (2011) 821–839.

[29] M. Gebser, M. Maratea, F. Ricca, The seventh answer set programming competition: Design and results, TPLP 20 (2020) 176–204. URL: https://doi.org/10.1017/S1471068419000061. doi:`10.1017/S1471068419000061`.

[30] M. Alviano, L. A. R. Reiners, ASP chef for water waste monitoring, in: D. Guidotti, L. Pandolfo, L. Pulina (Eds.), Proceedings of the 40th Italian Conference on Computational Logic, Alghero, Italy, June 25-27, 2025, volume 4003 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. URL: https://ceur-ws.org/Vol-4003/paper01.pdf.