Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

A Map–Summarize Framework for Answer Set Verbalization

Mario Alviano , Matteo Capalbo , Sebastiano A. Piccolo*
University of Calabria, Department of Mathematics and Computer Science (DeMaCS)
{mario.alviano, matteo.capalbo, sebastiano.piccolo}@unical.it

Abstract

We study answer set verbalization: generating readable natural-language descriptions of ASP solver outputs. Unlike standard data-to-text settings, answer sets often lack an explicit schema and may involve structured, non-relational representations through complex terms rather than flat records. We propose a modular map–summarize framework that first maps atoms into predicate-aware textual fragments (via schema guidance, LLM-based mapping, or deterministic templates) and then uses a language model primarily for fluent aggregation. We evaluate the framework through a quantitative benchmark on structured ASP request, and a human evaluation of solver-generated explanation graphs. Results show that predicate-aware structured methods improve faithfulness and efficiency over direct LLM prompting.

1 Introduction

Answer Set Programming (ASP) is a well-established paradigm for symbolic knowledge representation and combinatorial problem solving (Baral 2010; Brewka, Eiter, and Truszczynski 2011). At the same time, the rapid development of large language models (LLMs) has renewed interest in natural language interfaces for reasoning systems (Brown et al. 2020; d'Avila Garcez et al. 2015). Most existing efforts focus on translating natural language *inputs* into formal representations (Santana, Kareem, and Ricca 2024; Szeider 2025). In contrast, we consider the complementary challenge of making solver *outputs* intelligible to humans.

ASP solvers (Alviano et al. 2023; Gebser et al. 2019) compute *answer sets*, i.e., finite sets of ground atoms describing models of a program. While such symbolic interpretations are logically precise and machine-readable, they are often difficult to inspect directly and unsuitable for presentation in human-facing settings. This motivates the task of *answer set verbalization*: generating short natural-language descriptions of ASP databases that communicate solver results in a readable and faithful way.

From a natural language generation perspective, answer set verbalization can be seen as an instance of data-to-text generation (Gatt and Krahmer 2018). However, unlike standard relational settings, answer sets typically lack an explicit schema and often encode structured objects through complex terms rather than flat records. As a result, producing faithful verbalizations requires making predicate-level semantics explicit, rather than relying on the language model to infer them implicitly from raw atoms. A particularly demanding instance arises in explainability scenarios. Recent ASP explanation tools such as XCLINGO (Cabalar and Muñiz 2024), XASP (Trieu, Son, and Balduccini 2022; Alviano et al. 2024b), and UCOREXPLAIN (Alviano et al. 2024a) enrich solver outputs with justifications, often in the form of explanation graphs. For example, in a Sudoku instance, UCOREXPLAIN may produce atoms of the form `node(A,Truth,Reason)` and `link(A,B,Rule)` describing how a conclusion such as `assign((1,2),2)` is supported or constrained by other atoms and rules. While these representations capture valuable information, they are typically too large and low-level to be communicated directly to users.

In this work, we study a simple *map–summarize* pipeline for ASP verbalization. In the mapping phase, atoms are transformed into predicate-aware textual fragments, either through deterministic templates or LLM-based predicate

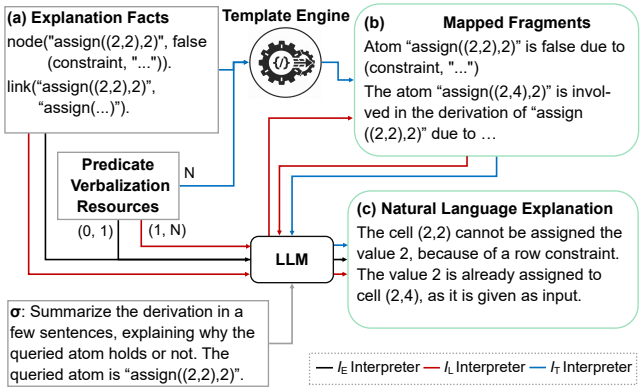

Figure 1: From solver explanation facts to a short natural-language explanation. (a) UCOREXPLAIN exports an ASP database encoding a derivation graph. (b) **Mapping Phase:** In modular strategies, atoms are independently transformed into text fragments via an interpreter: I_L (LLM-based) or I_T (Template-based). (c) **Summarization Phase:** An LLM aggregates these fragments (or raw atoms via I_E, or via I_L—in this case with schema descriptions) into a concise explanation.

*Corresponding author

interpretation. In the summarization phase, a language model is used primarily to produce a fluent and coherent natural-language description by aggregating the mapped fragments. This separation enables controllable verbalization while using LLMs mainly for linguistic realization. To make this workflow concrete, we provide an executable ASP CHEF recipe illustrating how solver-generated atoms can be mapped into predicate-level fragments and summarized into a short user-facing explanation; the recipe mirrors Figure 1 and is publicly available at https://asp-chef.netlify.app/s/KR2026/verbalizing-asp. To sum up, we (i) formalize a map–summarize framework for answer set verbalization, (ii) instantiate several concrete mapping strategies, and (iii) provide an initial empirical comparison highlighting trade-offs between faithfulness, readability, and efficiency.

2 Preliminaries and Problem Setting

Answer Set Programming. We assume basic familiarity with the ASP-Core-2 format (Calimeri et al. 2020). Let $\mathcal{P}$ be a finite set of predicate symbols, each with fixed arity. A ground atom is an expression $p(c_1, \ldots, c_n)$ with $p \in \mathcal{P}$. An ASP database is a finite set D of ground atoms. We denote by $\mathcal{D}$ the set of all databases. For $p \in \mathcal{P}$, we denote by $D|_p$ the restriction of D to facts whose predicate is p: $D|_p = \{\alpha \in D : \alpha = p(\overline{c})\}$. This notation naturally extends to sets of predicates $P \subseteq \mathcal{P}$ by defining $D|_P = \bigcup_{p \in P} D|_p$. A program Π may have zero, one, or multiple answer sets, denoted by $AS(\Pi) \subseteq \mathcal{D}$. Each answer set is an ASP database.

Strings and Lists. Let Σ be a finite alphabet. We write Σ^* for the set of finite strings over Σ, and $\Sigma^{[*]}$ for the set of finite lists of such strings. We write $s_1 \cdot s_2$ for string concatenation, and $L_1 +\!\!+ L_2$ for list concatenation. We assume a fixed encoding function $\mathsf{enc} : \Sigma^{[*]} \to \Sigma^*$ that converts a finite list of strings into a single string (e.g., by joining items with space or newline separators). In particular, when applied to a database D, $\mathsf{enc}(D)$ denotes a textual serialization of its facts (e.g., one ground atom per line).

Language Models and Templates. We abstract a large language model as a function $LLM : \Sigma^* \to \Sigma^*$ mapping an input prompt string to an output string. A template for a predicate symbol p of arity n is a string $\tau \in \Sigma^*$ containing placeholders of the form $\{x_i\}$, for $1 \leq i \leq n$. Given a ground atom $p(c_1, \ldots, c_n)$, we denote by $\mathsf{inst}(\tau, p(c_1, \ldots, c_n)) \in \Sigma^*$ the string obtained from τ by replacing each placeholder $\{x_i\}$ with the corresponding constant c_i.

Verbalizing ASP Databases. The task considered in this work is to generate textual descriptions of ASP databases, typically answer sets produced by declarative reasoning processes such as configuration or planning. Accordingly, we model an ASP verbalization method as a function

$$f : \mathcal{D} \to \Sigma^*$$

mapping a database D into an output string $s \in \Sigma^*$ that communicates the content of D in natural language.

3 A Map–Summarize Framework for ASP Verbalization

In this section, we formalize the ASP verbalization strategies studied throughout the paper. All approaches share a common two-phase architecture: (i) a mapping phase that associates ground facts with textual fragments, and (ii) a summarization phase that aggregates these fragments into a coherent description. The strategies differ in how predicate-level verbalization resources are interpreted.

Predicate Verbalization Resources. A central component of our framework is the association of predicate symbols with textual resources used during the mapping phase. Formally, a *predicate verbalization function*

$$V : \mathcal{P} \to \Sigma^*$$

assigns to each predicate symbol $p \in \mathcal{P}$ a string intended to guide the verbalization of facts of the form $p(\overline{c})$. Intuitively, V plays the role of an explicit predicate-level schema, making the intended meaning of symbols available to the verbalizer. We denote by $\mathcal{V}$ the set of all predicate verbalization functions. We extend V to predicate sets $P \subseteq \mathcal{P}$: $V(P) = \cdot_{p \in P} V(p)$, i.e., $V(P)$ is the concatenation of $V(p)$ for $p \in P$.

Example 1. *Consider a predicate* `assign/2`*. A predicate verbalization function may associate* `assign` *with either: (i) an instruction string* $s =$*"Describe assignments* `assign(cell,value)` *in plain English.", or (ii) a template string* $\tau =$*"Cell* $\{x_1\}$ *is assigned value* $\{x_2\}$*."*

Interpreters. To abstract over different ways of consuming predicate verbalization resources, we introduce the notion of an *interpreter*

$$I : 2^{\mathcal{P}} \times \mathcal{D} \times \mathcal{V} \to \Sigma^{[*]}$$

that, given a set of predicate symbols $P \subseteq \mathcal{P}$, a database D, and a predicate verbalization function V, returns a finite list of strings describing the facts in $D|_P$, called *fragments* in the following. We consider three interpreters, a *echo interpreter* I_{E}, a *language-model interpreter* I_{L} based on LLM prompting, and a deterministic *template interpreter* I_{T}:

$$\begin{aligned} I_{\mathsf{E}}(P, D, V) &= [\,\mathsf{enc}(D|_P) \cdot V(P)\,] \\ I_{\mathsf{L}}(P, D, V) &= [\,LLM(\mathsf{enc}(D|_P) \cdot V(P))\,] \\ I_{\mathsf{T}}(P, D, V) &= [\,\mathsf{inst}(V(p), p(\overline{c})) : p(\overline{c}) \in D|_P\,] \end{aligned}$$

Example 2 (Continuing Example 1)**.** *Consider a database* $D = \{p((1,2),2), p((2,1),3)\}$*, where* $p =$ `assign`*, and let* V_s, V_τ *be such that* $V(p) = s$ *and* $V(p) = \tau$*. We have* $I_{\mathsf{L}}(\{p\}, D, V_s) =$ [*LLM(*"*DB:* `assign((1,2),2). assign((2,1),3).` *Describe assignments in plain English.")*]*, and* $I_{\mathsf{T}}(\{p\}, D, V_\tau) =$ [*"Cell (1,2) is assigned value 2.", "Cell (2,1) is assigned value 3."*].

Mapping. A mapping strategy is determined by selecting a collection of predicate groups $G \subseteq 2^{\mathcal{P}}$. Intuitively, each set $P \in G$ specifies a block of predicates that will be interpreted jointly. We refer to the elements $P \in G$ as predicate groups. We consider two extreme choices of grouping: (i) *all-at-once*, $G_{all} = \{\mathcal{P}\}$; and (ii) *predicate-wise*, $G_{pw} = \{\{p\} : p \in \mathcal{P}\}$. Given an interpreter I and a grouping G, we define the mapping function

$$\begin{array}{rccl} \mathbf{Map}_{I,G} : & \mathcal{D} \times \mathcal{V} & \to & \Sigma^{[*]} \\ & D, V & \mapsto & +\!\!+_{P \in G,\ D|_P \neq \emptyset} I(P, D, V) \end{array}$$

Thus, $\mathbf{Map}_{I,G}(D, V)$ produces a list of fragments obtained by interpreting the facts in D group by group.

Example 3 (Continuing Example 2)**.** *Let us extend D with* `given((2,1),3)`*. Then $\mathbf{Map}_{I,G_{pw}}(D, V)$ performs two interpreter calls: one for predicate* `assign` *and one for predicate* `given`*. In contrast, $\mathbf{Map}_{I,G_{all}}(D, V)$ invokes the interpreter once on the entire database.*

Summarization. The summarization function aggregates the finite list of fragments produced in the mapping phase into a single natural-language text:

$$\begin{array}{rccl} \mathbf{Sum}_{\sigma} : & \Sigma^{[*]} & \to & \Sigma^{*} \\ & F & \mapsto & LLM(\sigma \cdot \mathsf{enc}(F)) \end{array}$$

where $\sigma \in \Sigma^*$ is an instruction string specifying how the fragments should be combined (e.g., style or level of detail). The function $\mathbf{Sum}_{\sigma}$ is intended to rewrite and aggregate the mapped fragments into a more readable description of the underlying database content.

Example 4. *Let F = [“Cell (1,2) is assigned value 2.”, “Cell (2,1) is assigned value 3.”, “Cell (2,1) contains value 3 (given clue).”]. Then $\mathbf{Sum}_{\sigma}(F)$ may aggregate these fragments into a single description, such as “Cell (1,2) has value 2, and cell (2,1) is set to 3 (as indicated by the clue).”*

Framework Summary. The map–summarize framework introduced above provides a uniform abstraction for ASP database verbalization. Given an input database $D \in \mathcal{D}$, a verbalization strategy is determined by: (i) an interpreter I, specifying how predicate-level resources are consumed, (ii) a grouping G, controlling the granularity at which predicates are mapped, (iii) a predicate verbalization function V, encoding domain-specific guidance, and (iv) a summarization instruction σ, specifying how fragments should be aggregated. Together, these components define a family of verbalizers:

$$\begin{array}{rccl} f_{I,G,V,\sigma} : & \mathcal{D} & \to & \Sigma^{*} \\ & D & \mapsto & \mathbf{Sum}_{\sigma}\big(\mathbf{Map}_{I,G}(D, V)\big). \end{array}$$

In the remainder of the paper, we instantiate this framework to obtain several concrete verbalization strategies, differing in their choice of interpreter and mapping granularity, and assess their behavior on representative ASP outputs.

4 Concrete Verbalization Strategies

We now instantiate the map–summarize framework introduced in the previous section with five concrete ASP verbalization strategies. Each strategy corresponds to a particular choice of interpreter I, predicate grouping G, predicate verbalization function V, and summarization instruction σ.

S1: Echo + Summarize (Baseline). As a minimal baseline, we consider a strategy where the mapping phase does not attempt any predicate-level interpretation, but simply exposes the database facts to the language model. This is obtained by using the echo interpreter I_{E} together with an empty predicate verbalization function $V_{\emptyset}$, i.e., $V_{\emptyset}(p) = \epsilon$ for all $p \in \mathcal{P}$. Formally,

$$f_{\mathsf{S1}}(D) = f_{I_{\mathsf{E}},\, G_{all},\, V_{\emptyset},\, \sigma}(D).$$

Intuitively, this corresponds to a standard prompting baseline (zero-shot or few-shot depending on σ), where the database is serialized and the language model is asked to produce a textual description. In Section 5, we refer to the strict zero-shot instance of this baseline (i.e., with no in-context examples included in σ) as S0.

S2: Echo + Schema + Summarize. The baseline strategy above does not exploit any predicate-level information. As a refinement, we consider a schema-guided prompting approach in which predicate verbalization resources are provided directly alongside the database. This is obtained by using the echo interpreter I_{E} together with a non-empty predicate verbalization function V, so that the mapped fragment contains both the serialized facts and the predicate-specific guidance strings. Using the all-at-once grouping, we define:

$$f_{\mathsf{S2}}(D) = f_{I_{\mathsf{E}},\, G_{all},\, V,\, \sigma}(D).$$

Intuitively, this strategy corresponds to a single-call verbalization setting where the language model is prompted with both the database content and an explicit description of predicate semantics, and is asked to directly produce a textual description of D.

S3: All-at-once Map–Summarize. While S1 and S2 expose the database (and optionally predicate guidance) through the echo interpreter, S3 explicitly delegates the mapping phase to an LLM-based interpreter. In this setting, all predicates are interpreted jointly in a single mapping call, producing mapped fragments that are subsequently aggregated by the summarization step. Formally, we instantiate the framework with the LLM interpreter I_{L} and the all-at-once grouping:

$$f_{\mathsf{S3}}(D) = f_{I_{\mathsf{L}},\, G_{all},\, V,\, \sigma}(D).$$

Intuitively, this strategy separates fact-to-text rendering from aggregation in two language-model invocations: one for mapping and one for summarization.

Strategy	Interpreter	Grouping	Mapping	LLM Calls
S1	I_{E}	G_{all}	Prompt	1
S2	I_{E}	G_{all}	Schema	1
S3	I_{L}	G_{all}	LLM	2
S4	I_{L}	G_{pw}	LLM	$\lvert P_D\rvert + 1$
S5	I_{T}	G_{pw}	Det.	1

Table 1: Summary of the concrete verbalization strategies (Section 4). Mapping types: Prompt = database only; Schema = prompt with V; LLM = model-based mapping; Det. = deterministic template instantiation. $P_D = \{p \in \mathcal{P} : D|_p \neq \emptyset\}$ denotes the set of predicates occurring in the input database.

S4: Predicate-wise Map–Summarize. The all-at-once strategy above performs mapping in a single global call, which may blur predicate-specific semantics. As a refinement, S4 decomposes the mapping phase predicate by predicate, so that each predicate symbol is interpreted independently. Formally, we combine the LLM interpreter I_{L} with the predicate-wise grouping G_{pw}:

$$f_{\mathsf{S4}}(D) = f_{I_{\mathsf{L}},\, G_{pw},\, V,\, \sigma}(D).$$

Intuitively, this strategy performs one mapping invocation per predicate group, followed by a final summarization step that aggregates the resulting fragments into a single textual description. Thus, the number of language-model calls grows with the number of predicates in D.

S5: Deterministic Template Verbalization. The previous strategies rely on language models already in the mapping phase, which may introduce variability at the fact-to-text level. In contrast, S5 performs mapping symbolically through deterministic template instantiation, ensuring that each ground fact is verbalized in a controlled way. Formally, we instantiate the framework with the template interpreter I_{T} and the predicate-wise grouping:

$$f_{\mathsf{S5}}(D) = f_{I_{\mathsf{T}},\, G_{pw},\, V,\, \sigma}(D).$$

Intuitively, this strategy produces deterministic fragments by applying templates to individual facts, and then invokes the language model only once to aggregate these fragments into a readable description.

Discussion. The five strategies above instantiate the proposed framework along three main dimensions: (i) whether predicate verbalization resources are absent (S0–S1), provided as prompt-level schema guidance (S2–S4), or instantiated deterministically as templates (S5); (ii) the granularity at which atoms are processed, ranging from all-at-once grouping (S0–S3) to predicate-wise decomposition (S4–S5); and (iii) the extent to which language models are involved in the mapping phase (S3–S4) versus only in the final aggregation step (S0–S2 and S5). Overall, these strategies correspond to common data-to-text paradigms (from direct prompting to schema-guided and template-based realization) captured uniformly within our map–summarize abstraction. In the following sections, we empirically compare these instantiations in terms of readability, faithfulness to the input atoms, and computational cost.

Model	BERTScore	ROUGE$_L$	BLEU	chrF	Runtime (s)
LLaMA 3 8b	0.902	0.365	0.099	0.543	84.383
LLaMA 4 16x17b	0.921	0.502	0.172	0.622	367.700
Ministral 3 14b	0.913	0.547	0.178	0.629	106.018
Mixtral 8x22b	0.927	0.505	0.194	0.634	2695.087
Deepseek R1 32b	0.927	0.553	0.206	0.643	1845.587
LLaMA 3 70b	0.926	0.521	0.207	0.649	339.017
Qwen 3 32b	0.934	0.588	0.247	0.668	2049.478
Mistral 3 24b	0.937	0.599	0.257	0.670	136.645

Table 2: Semantic preservation (BERTScore) and syntactic overlap (BLEU, ROUGE-L, chrF) across strategies, averaged by models.

Strategy	BERTScore	ROUGE$_L$	BLEU	chrF	Runtime (s)
S0	0.918	0.51	0.15	0.61	72.37
S1	0.924	0.47	0.17	0.63	*65.05*
S2	*0.951*	*0.70*	*0.34*	*0.71*	**64.79**
S3	0.925	0.51	0.20	0.65	241.23
S4	0.949	*0.70*	*0.34*	0.70	309.34
S5	**0.952**	**0.71**	**0.35**	**0.72**	67.09

Table 3: Semantic preservation and syntactic overlap, by strategy, for Mistral 3. **Bold**: best score; *italics*: second best.

5 Preliminary Evaluations

We evaluate the proposed framework through two complementary studies: (i) a quantitative benchmark on structured ASP databases to assess core verbalization performance, and (ii) a human evaluation of solver-generated explanation graphs to assess utility for Explainable AI (XAI).

We utilize a dataset of fashion retail assistant requests where ASP databases are paired with ground-truth natural language forms. We evaluate strategies S0–S5 across eight open-weight LLMs; namely, Deepseek R1 32b, LLaMA 3 8b and 70b, LLaMA 4 16x17b, Ministral 3 14b, Mistral 3 24b, Mixtral 8x22b, and Qwen 3 32b. We use four standard NLP metrics to measure both semantic preservation and syntactic overlap:

1. BERTScore, which measures the semantic agreement between the ground truth and the verbalized explanation;
2. ROUGE$_L$, which measures sentence-level structural similarity based on the longest common subsequence between the ground truth and the verbalized explanation;
3. BLEU, which measures the n-gram overlap between the ground truth and the verbalized text;
4. chrF, which captures morphological variation by evaluating character-level overlap.

Table 2 shows results on all strategies, aggregated by models. Mistral 3 emerges as the top-performing model, surpassing significantly larger architectures (e.g., LLaMA 3 70b) and reasoning-specialized models (e.g., DeepSeek R1 32b and Qwen 3 32b). Table 3 reports the performance of each strategy with Mistral 3 24b. The results demonstrate that structured mapping (S2, S5) is essential for faithfulness; while baselines S0 and S1 capture general intent, they frequently omit critical constants. Notably, S2 and S5 are the

Strategy	Faithful	Truthfulness			Clarity	Preferred
		C	U	I		
S2	0.73	0.82	0.11	0.07	3.41	28%
S5	0.79	0.92	0.06	0.02	3.78	46%

Table 4: Summary of human evaluation for S2 and S5. Truthfulness indicates the fraction of correct (C), unclear (U), or incorrect (I) answers. Clarity is rated on a Likert scale (1–5).

most computationally efficient, offering a 5x speedup over multi-call strategies like S4.

To test the framework on complex reasoning outputs, we applied S2 and S5 (using Mistral 3 24b) to solver-generated explanation graphs for Sudoku. Seven expert ASP researchers performed a blind, paired comparison of 15 explanation sets, evaluating them for faithfulness (support by ASP facts), truthfulness (logical correctness), and clarity. As shown in Table 4, S5 outperformed S2 across all qualitative dimensions. Experts preferred S5 in 46% of cases, citing its superior truthfulness (92%) and clarity. These results suggest that while schema-guidance (S2) is effective for simple data-to-text tasks, deterministic template mapping (S5) provides the "grounding stability" required for trustworthy XAI, significantly reducing the risk of semantic drift during the verbalization of complex derivation steps.

To assess experts' reliability, we compute Gwet's AC1 for categorical metrics (Truthfulness, Faithfulness) and Gwet's AC2 (quadratic weights) for the ordinal Likert scale (Clarity). We select Gwet's metrics because our experts reached a high level of consensus (82% agreement), creating a high-prevalence distribution where traditional Kappa is known to be overly conservative. Results show Excellent agreement for Truthfulness (AC1=0.803), confirming logical soundness. Agreement for Faithfulness is 0.591, while Clarity (AC2=0.341) is lower; this is expected as linguistic fluency is more subjective than logical correctness.

6 Conclusion

We proposed a *map-summarize* framework for verbalizing ASP answer sets. Our results show that predicate-aware strategies (S2, S5) improve faithfulness, allowing a 24B model to outperform 70B as well as reasoning baselines. Moreover, human evaluation on solver-generated explanations indicates that deterministic mapping (S5) offers the grounding stability required for trustworthy XAI, motivating future work on revising the upstream UCOREXPLAIN pipeline to produce explanation atoms more targeted for downstream verbalization.

Acknowledgments

This work was supported by the Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by the Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); under PN RIC project ASVIN "Assistente Virtuale Intelligente di Negozio" (CUP B29J24000200005); by Regione Calabria NextGenGuides "Innovazione Tecnologica per le Guide Turistiche del Futuro tramite l'ausilio di tecnologie innovative AI e sistemi di geolocalizzazione avanzata" (CUP J39I24002040005); by MOZART "Modelli e tecniche di mOdernizzazione di applicaZioni e data silos a supporto di clienti esterni, field force e impiegati di backoffice basati su AI GeneRativa e apprendimento auTomatico" (CUP J49I24001740005); by the LAIA lab (part of the SILA labs). Mario Alviano is a member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

Alviano, M.; Dodaro, C.; Fiorentino, S.; Previti, A.; and Ricca, F. 2023. ASP and subset minimality: Enumeration, cautious reasoning and muses. *Artif. Intell.* 320:103931.

Alviano, M.; Hahn, S.; Sabuncu, O.; and Weichelt, H. 2024a. Answer set explanations via preferred unit-provable unsatisfiable subsets. In Dodaro, C.; Gupta, G.; and Martinez, M. V., 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*, 187–199. Springer.

Alviano, M.; Trieu, L. L. T.; Son, T. C.; and Balduccini, M. 2024b. The XAI system for answer set programming xasp2. *J. Log. Comput.* 34(8):1500–1525.

Baral, C. 2010. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language models are few-shot learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Cabalar, P., and Muñiz, B. 2024. Model explanation via support graphs. *Theory Pract. Log. Program.* 24(6):1109–1122.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. ASP-Core-2 input language format. *TPLP* 20(2):294–309.

d'Avila Garcez, A. S.; Besold, T. R.; Raedt, L. D.; Földiák, P.; Hitzler, P.; Icard, T.; Kühnberger, K.; Lamb, L. C.; Miikkulainen, R.; and Silver, D. L. 2015. Neural-symbolic learning and reasoning: Contributions and challenges. In

2015 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 22-25, 2015. AAAI Press.

Gatt, A., and Krahmer, E. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *J. Artif. Intell. Res.* 61:65–170.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.* 19(1):27–82.

Santana, M. A. B.; Kareem, I.; and Ricca, F. 2024. Towards automatic composition of ASP programs from natural language specifications. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, 6198–6206. ijcai.org.

Szeider, S. 2025. Cp-agent: Agentic constraint programming. *CoRR* abs/2508.07468.

Trieu, L. L. T.; Son, T. C.; and Balduccini, M. 2022. xasp: An explanation generation system for answer set programming. In Gottlob, G.; Inclezan, D.; and Maratea, M., eds., *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, 363–369. Springer.