ASPembly: Neuro-Symbolic Knowledge Extraction of Assembly Instructions into ASP

Mario Alviano[1], Lorenzo Grillo[2] and Pierpaolo Sestito[3]

University of Calabria, Department of Mathematics and Computer Science

Abstract

Robotic assembly tasks require systems to interpret unstructured visual instructions, derive action sequences, and support consistency checking before downstream execution. While Vision-Language Models (VLMs) excel at extracting semantic information from visual manuals, they cannot reliably solve complex combinatorial reasoning tasks, enforce action preconditions, or guarantee global consistency. To bridge this gap, we present *ASPembly*, a neuro-symbolic pipeline that instantiates a visual variant of the Logic-Guided Extraction (LGX) framework for image-based assembly manuals. Unlike baseline pipelines that query a model independently and validate the output post-hoc, *ASPembly* interleaves VLM-based extraction with non-monotonic Answer Set Programming (ASP) reasoning. In this architecture, the VLM reads visual instruction pages to propose candidate robot commands, while ASP acts as an active control mechanism rather than a passive checker. Guard conditions over the evolving symbolic state dynamically determine which predicates can be extracted, rejected, delayed, or logically inferred without additional model calls. By shifting logic from post-processing to active process control, *ASPembly* helps improve the coherence of the generated assembly trace, mitigating spurious outputs and reducing unsupported commands during extraction.

Keywords

Answer Set Programming, Robotic Assembly Representation, Vision-Language Models, Neuro-Symbolic AI, Logic-Guided Extraction, Visual Instruction Understanding

CEUR-WS.org/Vol-4311/CARLA_regular_02.pdf

1. Introduction

Answer Set Programming (ASP) is a declarative programming paradigm based on nonmonotonic logic and stable model semantics [1, 2, 3, 4, 5]. It is particularly suitable for modeling combinatorial search problems, complex constraints, incomplete information, and reasoning tasks in which admissible solutions must satisfy domain-specific requirements. For this reason, ASP has been successfully used in knowledge-intensive domains and in robotic settings where actions, preconditions, effects, and feasibility constraints must be represented explicitly.

Robotic assembly planning is a natural application domain for this kind of reasoning. A feasible assembly plan is not only a sequence of manipulation actions, but a structured object constrained by the availability of components, spatial relations, ordering dependencies, physical preconditions, and task-specific restrictions. Declarative encodings are well suited to represent these aspects, because they allow the domain theory and the search process to be specified independently from low-level procedural control. However, many ASP-based planning approaches assume that the symbolic representation of the task is already available. In practical assembly scenarios, this assumption is often too strong.

Assembly instructions are frequently provided as visual manuals composed of diagrams, arrows, exploded views, part layouts, and step-wise illustrations. These documents are designed for human interpretation and may contain little or no textual explanation. As a consequence, the system must first construct a symbolic representation of the task before any logic-based planning or verification can be applied. This requires extracting components, relations, manipulation cues, candidate actions, and ordering information from visual pages whose semantics are often implicit, partial, or ambiguous.

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026), co-located with the Federated Logic Conference (FLoC 2026), July 18–19 and 24–25, 2026, Lisbon, Portugal.

mario.alviano@unical.it (M. Alviano); lorenzo.grillo@unical.it (L. Grillo); pierpaolo.sestito@unical.it (P. Sestito)

0000-0002-2052-2063 (M. Alviano); 0009-0002-6099-3895 (L. Grillo); 0009-0004-7685-6152 (P. Sestito)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-03

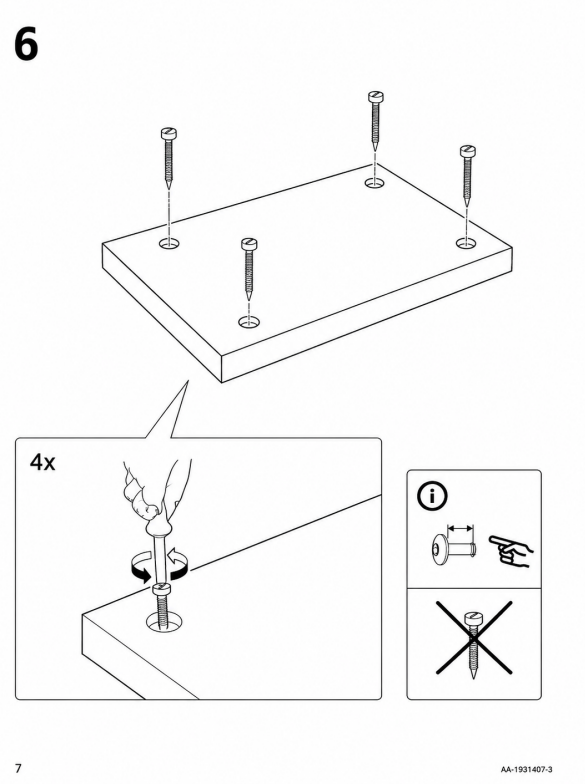


Figure 1: Example of a visual assembly instruction page. The page illustrates an assembly step through parts, spatial cues, and manipulation indications.

This paper presents *ASPembly*, a neuro-symbolic approach for constructing symbolic assembly traces from image-based instruction manuals. The approach builds on a Logic-Guided Extraction (LGX) framework in which generative models propose candidate symbolic facts, while ASP validates the evolving representation, derives implicit information, and controls which extraction steps are admissible. Rather than querying the model independently for all target predicates, the system evaluates the current logical state and uses guard conditions to determine whether a predicate can be extracted, skipped, or inferred. Prior ASP-based robotic assembly systems usually start from symbolic task descriptions, whereas ASPembly addresses the upstream problem of constructing such descriptions from visual instruction manuals.

The contribution of the paper is a controlled extraction pipeline in which ASP is not only used after extraction to check a completed representation, but also during extraction to regulate the construction of the symbolic state required for robotic assembly planning. This design separates visual-semantic interpretation from logical control, reducing unsupported model calls and limiting the introduction of facts that are inconsistent with the current assembly state.

2. Background

2.1. Vision-Language Models

Vision-Language Models (VLMs) extend language-based interpretation to visual inputs by jointly processing images and textual prompts. This capability is central in assembly domains, where instructions are often conveyed through diagrams, exploded views, arrows, part layouts, spatial relations, and sequential visual cues rather than explicit textual descriptions [6, 7]. In a case such as Figure 1, a VLM can inspect the instruction page and identify relevant elements such as parts, target positions, arrows, alignments, and manipulation cues.

In visual instruction understanding tasks, VLMs can be used to produce intermediate structured

descriptions from image-based inputs. Their role is not necessarily to generate a complete symbolic program or a verified assembly plan, but to identify candidate entities, relations, actions, and parameters that can later be processed by a symbolic reasoning layer. For instance, from the visual instruction shown in Figure 1, involving a repeated screw insertion into target holes, a VLM may extract candidate commands such as:

Example 1. A representative natural-language description generated by the VLM for the assembly step in Figure 1 is:

```
"Use a screwdriver to drive four screws into the board, ensuring they are not
    over-tightened or driven too deep."
```

Although VLMs excel at generating locally plausible natural language descriptions, they lack the capacity to guarantee global consistency, action preconditions, or domain-specific constraints [8, 9], particularly when processing multi-page visual sequences. Robotic assembly planning demands far more than superficial textual accuracy: the extracted actions must form a logically coherent sequence where operational preconditions and physical effects strictly align with an evolving assembly state [10]. Without symbolic grounding, a standalone VLM may correctly identify a manipulated object yet leave its target destination underspecified, omit critical intermediate maneuvers, or generate source and destination arguments that directly contradict the actual state of the environment. For this reason, VLM outputs should be treated as candidate observations rather than verified plans. They can provide useful perceptual and semantic information from visual inputs, but the validation of their outputs requires an explicit reasoning layer able to check admissibility, consistency, and missing information.

2.2. Logic-Guided Extraction Framework

Logic-Guided Extraction (LGX, [11]) is a neuro-symbolic framework for constructing symbolic representations from unstructured inputs. Its core principle is the separation between extraction and reasoning: a generative model is used as an extraction oracle for candidate facts, while Answer Set Programming (ASP) is responsible for validation, inference, normalization, and process control. In baseline extraction pipelines, target predicates are typically queried independently, and logical reasoning is applied only after the extraction phase. LGX changes this interaction pattern by interleaving oracle calls with ASP reasoning. At each stage, the current database of extracted and derived facts is evaluated against logic-based admissibility conditions, called guards. The oracle is called only when the guards of the target predicate are satisfied; otherwise, the corresponding query can be avoided or the required information can be derived from the current symbolic state. This selective use of oracle calls reduces unnecessary queries, thereby saving computational resources, while allowing ASP to guide the construction of the symbolic representation rather than merely checking it after completion. More generally, LGX provides a control scheme in which extraction is not performed as a one-shot generation task, but as an incremental process regulated by logical conditions over the evolving set of facts.

3. ASPembly Workflow

ASPembly instantiates the Logic-Guided Extraction framework in the domain of robotic assembly from visual instruction manuals. We refer to this multimodal instantiation as *visual Logic-Guided Extraction* (vLGX), where the core extraction process is entirely driven by a Vision-Language Model (VLM). Within this architecture, the VLM is not tasked with generating a final executable robot plan or an ASP program directly. Instead, it operates on visual instruction pages to extract intermediate structured candidates—such as action-level commands and their associated arguments. These VLM-extracted candidates are subsequently normalized and translated into ASP facts, allowing the symbolic layer to organize, validate, and control the deterministic construction of the assembly trace for downstream reasoning.

Visual input. The workflow starts from an image-based assembly manual processed by the ASPembly pipeline. A representative instruction page is shown in Figure 1. Since such pages are designed for human interpretation, the relevant assembly information must first be extracted from the visual content.

vLGX extraction. The selected instruction page is processed through *vLGX*, an advanced variant designed to overcome the textual limitations of the baseline LGX framework. Specifically, vLGX introduces two key architectural enhancements:

- **Multimodal Input Support:** Unlike the original LGX framework, which operates exclusively on textual data, vLGX natively integrates Vision-Language Models (VLMs). This enables the pipeline to accept raw image-based inputs, such as manual pages, directly into the extraction loop.
- **Inline Vocabulary Constraints:** It introduces the capability to define strict lexical constraints directly within the predicate schema declaration. By appending an explicit set of permitted values to an extraction slot (e.g., `command_name[pick|move|align|...]`), the framework strictly limits the VLM's generative output to that specific vocabulary, effectively neutralizing out-of-domain hallucinations.

The output given by the internal VLM is an intermediate structured representation adhering to a strict JSON Schema. These candidate commands are subsequently translated into ASP facts, which provide a symbolic state that can be inspected with respect to ordering, repetitions, admissibility, and consistency. This follows the core principle that extraction is not a one-shot generation task, but an incremental construction of a symbolic representation regulated by logic-based control.

Example 2 (Continuing Example 1)**.** A fragment of ASPembly output involving a repeated screw insertion and tightening operation is shown below. The representation remains deliberately symbolic and action-level, abstracting away from low-level robot-control parameters such as grasping policies or kinematic trajectories.

```
assembly_page(y).
step(6).
repetition(6,4).
robot_command(6,1,pick,screw_10046802,table,none,
              pick_up_one_of_the_4_screws_part_10046802).
robot_command(6,2,move,screw_10046802,table,screw_hole,
              move_the_screw_towards_the_pre_drilled_hole_on_the_frame).
robot_command(6,3,align,screw_10046802,none,screw_hole,
              align_the_screw_with_the_hole_on_the_frame).
robot_command(6,4,insert,screw_10046802,none,screw_hole,
    insert_the_screw_into_the_hole_on_the_frame).
robot_command(6,5,rotate,screw_10046802,none,none,
              rotate_the_screw_to_tighten_it_using_a_screwdriver).
robot_command(6,6,release,screw_10046802,none,none,
              release_the_screw_after_it_is_fully_tightened).
```

■

Active Process Control & Guard Conditions. To orchestrate this construction, vLGX uses ASP as an active process-control layer to evaluate guard conditions before any model call. For a given target predicate p, the engine evaluates whether the current symbolic state I satisfies its corresponding logical guard G_p (i.e., $I \models G_p$). The VLM acts strictly as a perceptual extraction oracle and is queried if and only if the guard condition is met.

Extraction Predicates: These facts represent the visual data targeted for extraction. vLGX dynamically gates these queries based on logical prerequisites.

assembly_page(bool): Determines whether the analyzed page contains an actionable assembly step diagram (y) or is empty of instructions (n). *Guard condition:* None. This is the preliminary check for each page.

step(number): Extracts the numerical identifiers of the macro-steps depicted on the current page. *Guard condition:* `assembly_page(y)`. The VLM is queried only if the page is classified as actionable.

repetition(step, count): Identifies whether a specific step involves a repeated action (e.g., "4x", as seen in Example 2) and extracts the exact count. *Guard condition:* `assembly_page(y)`.

robot_command(step, order, command, object, from, to, desc): Decomposes an assembly step into a chronologically ordered sequence of atomic robot commands. The tuple `(command, object, from, to)` encodes the specific action schema, constrained to a predefined vocabulary, while the final argument stores a normalized natural-language description. *Guard conditions:* The vLGX framework triggers this query only if the page is actionable (`assembly_page(y)`) and strictly blocks it if the ASP layer derives any active blocking predicate, namely `inconsistent_repetition(_)` or `invalid_repetition(_)`.

Global Validation Predicates (ASP Knowledge Base): These predicates are never extracted by the VLM. Instead, they are automatically inferred by the non-monotonic ASP reasoning engine within vLGX to evaluate the structural, temporal, and causal validity of the trace.

inconsistent_repetition(S) / invalid_repetition(S): Triggers if a repetition count is assigned to an undeclared step, if a declared step lacks a mandatory repetition assignment, or if the extracted repetition count is mathematically invalid (< 1).

missing_commands(S): Signals a structural omission. It triggers if vLGX records a macro-step S, but the VLM fails to extract any corresponding `robot_command` facts for it.

orders_incomplete(S, ONEXT): Signals a chronological sequencing error. It activates if a gap is detected in the continuous numerical ordering of the micro-commands for step S, and explicitly identifies the missing sequence index ONEXT.

step_not_existent(S): Signals a structural hallucination. It triggers if the VLM proposes manipulation commands for a step S that was never formally declared in the symbolic trace.

Lexical Constraints and Domain Vocabularies. To mitigate out-of-vocabulary generations and support the construction of standardized symbolic traces, vLGX enforces lexical constraints on the VLM output schema. Rather than allowing fully free-form text generation for all predicate arguments, specific slots are bounded to predefined domain vocabularies. For instance, the preliminary page classification is restricted to a boolean-like output (y for actionable pages, n for informational ones). Most importantly, the `robot_command` extraction is typed: the `command` argument is constrained to a predefined set of robotic manipulation primitives (e.g., `pick`, `move`, `align`, `insert`, `rotate`, `place`, `release`), and the `object` argument must match a known inventory of assembly components (e.g., `screw`, `dowel`, `panel`, `leg`, `tool`). These constraints operate at the vocabulary level, ensuring that the extracted symbols are compatible with the command and object definitions used by the Web Simulator. Structural and temporal consistency are instead checked by the ASP validation predicates described above, while full causal and physical reasoning about manipulation feasibility is left to downstream planning and future robotic execution.

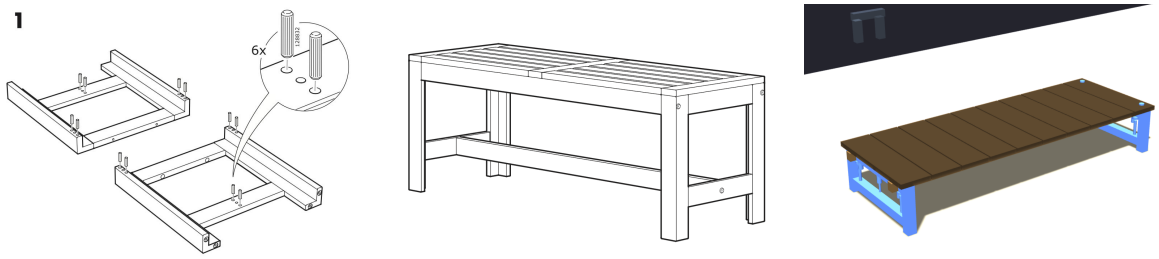


Figure 2: IKEA case study overview. Left: the selected page from the input assembly manual. Center: the finished product as illustrated in the manual view. Right: a simplified 3D rendering of the symbolic trace generated by the ASPembly pipeline shown with the web simulator.

4. Case Study: IKEA Assembly Manual

To illustrate the ASPembly workflow on a realistic visual instruction source, we consider an IKEA assembly manual as a reproducible case study. The manual is provided as a PDF document, publicly accessible [1], and contains image-based assembly steps with limited textual information. This setting is representative of the target scenario of ASPembly, where instructions are mainly communicated through diagrams, arrows, part identifiers, spatial relations, and implicit manipulation cues.
The selected manual page is processed through the ASPembly pipeline. The Vision-Language Model extracts candidate action-level commands from the visual instruction, and ASPembly normalizes them into a symbolic trace. The following fragment shows the ASP representation generated for the selected step:

```
step(1). repetition(1,6). orders_incomplete(1,4).
robot_command(1,1,pick,dowel,table,none,pick_up_one_of_the_6_dowelspart_128832).
robot_command(1,2,move,dowel,table,hole,move_the_dowel_to_the_hole_on_the_frame).
robot_command(1,3,align,dowel,none,none,align_the_dowel_with_the_hole).
robot_command(1,5,release,dowel,none,none,release_the_dowel_in_the_hole).
```

Example 2[2] shows how ASPembly represents the selected visual instruction as a compact symbolic command trace. The fact `step(6)` identifies the assembly step, while `repetition(6,4)` captures the repeated nature of the operation. The `robot_command/7` facts describe the ordered manipulation sequence required to pick, move, align, insert, tighten, and release the screw. Figure 2 shows the same case study from the visual perspective. The left image reports the input manual page, while the right image shows the corresponding case-study-specific 3D rendering. The textual ASP facts constitute the primary output of the ASPembly pipeline, whereas the 3D rendering provides an auxiliary visual interpretation of the symbolic trace using predefined furniture-assembly assets. The goal of this case study is not to claim full robotic executability of the extracted sequence. Rather, it shows that ASPembly can produce a coherent symbolic representation from an image-based manual and expose it through interpretable textual output and an ad hoc visual demonstration. The physical grounding of these commands on real robotic arms is left as future work.

5. Symbolic Trace and Sequence Analysis

In this section, we evaluate the performance of the vLGX framework by conducting a comparative analysis between two VLMs of different scales: `Gemma 4 E4B` (a lightweight 4-billion parameter model) and `Gemma 4 31B` (a larger 31-billion parameter variant). This evaluation aims to investigate how model capacity impacts the syntactic validity and logical consistency of the generated robotic action sequences. To ensure a fair comparison, both models were evaluated using three distinct PDF assembly

[1] https://xiro28.github.io/ASPembly/manual
[2] For reproducibility, the complete test case is available online at: https://xiro28.github.io/ASPembly/IKEA

manuals, encompassing a total of 25 manual steps. Following the methodology outlined previously, the extracted symbolic traces were validated through the ASP layer and classified into four specific categories: fully valid symbolic traces (*Fully Consistent*), sequences with non-critical chronological errors (*Minor: Ordering Issues*), and sequences failing due to either fatal generative loops (*Critical: Repetition Collapse*) or missing actions (*Critical: Omissions*).

Table 1
Comparative evaluation of vLGX performance using two different VLM sizes across 25 assembly steps. The metrics classify the logical consistency of generated traces, highlighting the specific failure modes identified by the ASP validation layer.

Model Configuration	Fully Consistent	Minor: Ordering Issues	Critical: Repetition Collapse	Critical: Omissions
`Gemma 4 E4B`	20.0% (5/25)	0.0% (0/25)	44.0% (11/25)	36.0% (9/25)
`Gemma 4 31B`	88.0% (22/25)	12.0% (3/25)	0.0% (0/25)	0.0% (0/25)

A complete interactive visualization of the validation outputs, including the analyzed assembly manuals, generated ASP traces, and per-model comparisons, is publicly available at https://xiro28.github.io/ASPembly/validation.

5.1. Preliminary Results

The ASP validation layer exposes a sharp contrast in how VLM scale affects logical consistency, revealing two distinct *Critical Violations* for the lightweight `Gemma 4 E4B` model: *Repetition Collapse* and *Omissions*. Due to limited context retention, the 4B model frequently falls into generative loops, blindly duplicating identical command blocks across entirely different manual steps (flagged via `inconsistent_repetition`). Furthermore, it completely fails to generate outputs for a significant portion of the assembly sequence, resulting in severe step omissions (flagged via `missing_commands`). Conversely, the larger `Gemma 4 31B` variant demonstrates robust state awareness and semantic progression, successfully adapting its vocabulary to changing components and dynamic tool transitions (e.g., from an Allen key to a screwdriver). It entirely eliminates critical structural failures, with its residual inconsistencies bounded exclusively to *Minor: Ordering Issues* triggered by `orders_incomplete` predicates. These findings suggest that while scaling effectively mitigates severe generative loops and omissions, neuro-symbolic grounding remains essential to resolve subtle causal ambiguities.

6. Related Work

Answer Set Programming has been widely used as a declarative reasoning formalism for robotic domains in which actions, preconditions, effects, incomplete knowledge, and feasibility constraints must be represented explicitly. Early work on service and collaborative housekeeping robotics showed how ASP can support semantic reasoning, task selection, and execution in domestic environments [12, 13, 14]. ASP has also been adopted in cognitive architectures for service robots [15], motion and task planning [16, 17, 18], multi-agent plan verification [19], and spatial reasoning for navigation [20]. These works establish ASP as an effective reasoning layer for robotics. However, they typically assume that the relevant symbolic description of the domain, including objects, actions, goals, and constraints, is already available. A closer line of work concerns robotic assembly and manipulation. Rizwan et al. [21] use ASP for human-robot collaborative assembly planning, while Bertolucci et al. [22] apply ASP to the manipulation of articulated objects with dual-arm robots. These approaches are closely related to ASPembly because they address structured manipulation and assembly scenarios. Nevertheless, their focus is on reasoning over an already formalized task representation. ASPembly addresses a complementary upstream problem: constructing such a symbolic representation from visual assembly instructions before planning or execution can take place. Symbolic reasoning has also been combined with perceptual and visual information. Aditya et al. [23] combine perception, semantic parsing, and reasoning for visual commonsense scene understanding. Khan et al. [24] use ASP for event detection from video, transforming visual observations into higher-level event descriptions. Wotawa

and Klampfl [25] rely on ASP to explain object motion from visual observations. These works show that logical reasoning can impose structure on noisy or incomplete perceptual outputs. However, they do not specifically address the extraction of assembly-level symbolic commands from image-based instruction manuals. Recent neuro-symbolic research further supports the idea of combining data-driven models with declarative reasoning. ASP has been identified as a suitable symbolic component in neuro-symbolic AI systems [26], and recent work has explored the use of large generative models to support symbolic planning and reasoning about actions [27, 28]. ASPembly follows this general neuro-symbolic direction, but assigns a different role to the neural component: the Vision-Language Model is used as a visual extraction oracle, while ASP controls the construction of the symbolic representation by validating candidate facts, deriving implicit information, and regulating which extraction steps are admissible. Overall, prior work has demonstrated the effectiveness of ASP for robotic planning, manipulation, perception-based reasoning, and plan verification. ASPembly differs from these approaches by focusing on the construction of the symbolic assembly trace itself. Instead of assuming a manually specified task representation, it extracts candidate commands from visual manuals and uses ASP as an active control layer during extraction. This allows the system to bridge visual instruction understanding and symbolic robotic assembly planning.

7. Conclusion and Future Work

This paper presented `ASPembly`, a neuro-symbolic pipeline for constructing symbolic assembly traces from image-based manuals. The approach instantiates the Logic-Guided Extraction (LGX) framework in a multimodal setting (`vLGX`), utilizing Vision-Language Models as perceptual extraction oracles to propose structured candidates, and Answer Set Programming to organize, normalize, and actively control the resulting symbolic representation. Rather than passively checking a completed trace post-hoc, `ASPembly` evaluates the evolving logical state via non-monotonic reasoning to dynamically determine which robotic predicates can be extracted, inferred, delayed, or rejected.

Our comparative evaluation highlighted the role of this neuro-symbolic grounding layer across different model scales. The empirical analysis showed that lightweight models, such as `Gemma 4 E4B`, are susceptible to severe context-retention limitations, which manifest as total step omissions (36.0

Future work will extend the evaluation to longer and more heterogeneous assembly manuals, involving a larger number of steps and varied instruction styles, and will include controlled corruption experiments at different noise levels to assess the robustness of the pipeline under degraded or partially inconsistent inputs. Beyond this, future work will advance along three main trajectories. First, we will investigate an explainability layer capable of translating the generated symbolic traces into human-oriented procedural tutorials, thereby supporting both robot execution and human assistance. Second, we will complete a comprehensive web-based inspection environment utilizing a finite library of 3D assets, facilitating the interactive visualization, debugging, and human-in-the-loop validation of the extracted traces without making visualization a strict pipeline prerequisite. Third, we will evaluate the framework on a physical robotic testbed, connecting `ASPembly`'s high-level traces to executable manipulation primitives for real robotic arms. In this direction, the ASP layer will be extended from trace construction to task-level embodied reasoning, explicitly checking physical preconditions, operational effects, and kinematic feasibility constraints during execution.

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 under project NextGenGuides "Innovazione Tecnologica per le Guide Turistiche del Futuro tramite l'ausilio di tecnologie innovative AI e sistemi di

geolocalizzazione avanzata" (CUP J39I24002040005); under project 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).

Declaration on Generative AI

During the preparation of this work, the author(s) used Google Gemini and GPT-5 in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

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

[2] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68.

[3] V. Lifschitz, Answer Set Programming, Springer, 2019.

[4] R. Kaminski, J. Romero, T. Schaub, P. Wanko, How to build your own asp-based system?!, Theory Pract. Log. Program. 23 (2023) 299–361.

[5] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, ASP and subset minimality: Enumeration, cautious reasoning and muses, Artif. Intell. 320 (2023) 103931.

[6] H.-T. Ho, L. Nguyen, M.-T. Pham, Q.-H. Pham, Q.-D. Tran, D. Nguyen, T.-H. Nguyen, A review on vision-language-based approaches: Challenges and applications, Computers, Materials, & Continua 82 (2025) 1733.

[7] Y. Qi, H. Li, Y. Song, X. Wu, J. Luo, How vision-language tasks benefit from large pre-trained models: A survey, IEEE Transactions on Multimedia (2025).

[8] P.-Y. Wang, T.-S. Liu, C. Wang, Z. Li, Y. Wang, S. Yan, C. Jia, X.-H. Liu, X. Chen, J. Xu, et al., A survey on large language models for mathematical reasoning, ACM Computing Surveys 58 (2026) 1–35.

[9] F. Xu, Q. Hao, C. Shao, Z. Zong, Y. Li, J. Wang, Y. Zhang, J. Wang, X. Lan, J. Gong, et al., Toward large reasoning models: A survey of reinforced reasoning with large language models, Patterns 6 (2025).

[10] L. Wang, S. Keshavarzmanesh, H.-Y. Feng, R. O. Buchal, Assembly process planning and its future in collaborative manufacturing: a review, The International Journal of Advanced Manufacturing Technology 41 (2009) 132–144.

[11] M. Alviano, L. Grillo, N. Leone, F. Lo Scudo, Logic-guided data extraction with answer set programming and large language models, Theory Pract. Log. Program. 26 (2026). URL: https://www.dropbox.com/scl/fi/67tvxu84caen4z6djsk3f/ICLP26_LGX.pdf?rlkey=zb2fmv7tjakx5s05drxhyw3zn&dl=1, accepted at (ICLP 2026).

[12] E. Erdem, E. Aker, V. Patoglu, Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution, Intelligent Service Robotics 5 (2012) 275–291.

[13] E. Aker, V. Patoglu, E. Erdem, Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics, IFAC Proceedings Volumes 45 (2012) 77–83.

[14] E. Aker, V. Patoğlu, E. Erdem, Collaborative housekeeping robotics using answer set programming, in: 2013 21st Signal Processing and Communications Applications Conference (SIU), IEEE, 2013, pp. 1–4.

[15] J. Ji, A cognitive architecture for a service robot: an answer set programming approach, in: Proceedings of the 25th international conference on logic programming, volume 5649, 2009, pp. 532–533.

[16] J. J. Portillo, C. L. García-Mata, P. R. Márquez-Gutiérrez, R. Baray-Arana, Robot platform motion planning using answer set programming., in: LA-NMR, 2011, pp. 35–44.
[17] F. Yang, P. Khandelwal, M. Leonetti, P. Stone, Planning in answer set programming while learning action costs for mobile robots., in: AAAI Spring Symposia, 2014.
[18] D. Lu, Y. Zhou, F. Wu, Z. Zhang, X. Chen, Integrating answer set programming with semantic dictionaries for robot task planning, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), 2017, pp. 4361–4367.
[19] S. Opfer, S. Niemczyk, K. Geihs, Multi-agent plan verification with answer set programming, in: Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering, 2016, pp. 32–39.
[20] Y. Izmirlioglu, Hybrid reasoning about relative position and orientation of objects and navigating agents using answer set programming, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 14997–15004.
[21] M. Rizwan, V. Patoglu, E. Erdem, Human robot collaborative assembly planning: An answer set programming approach, Theory and Practice of Logic Programming 20 (2020) 1006–1020.
[22] R. Bertolucci, A. Capitanelli, C. Dodaro, N. Leone, M. Maratea, F. Mastrogiovanni, M. Vallati, Manipulation of articulated objects using dual-arm robots via answer set programming, Theory and Practice of Logic Programming 21 (2021) 372–401.
[23] S. Aditya, Y. Yang, C. Baral, C. Fermüller, Y. Aloimonos, Visual commonsense for scene understanding using perception, semantic parsing and reasoning., in: AAAI Spring Symposia, 2015.
[24] A. Khan, L. Serafini, L. Bozzato, B. Lazzerini, et al., Event detection from video using answer set programming, in: CEUR WORKSHOP PROCEEDINGS, volume 2396, CEUR-WS, 2019, pp. 48–58.
[25] F. Wotawa, L. Klampfl, Explaining object motion using answer set programming, in: International Symposium on Methodologies for Intelligent Systems, Springer, 2020, pp. 298–307.
[26] M. Borroto, A. Ielo, G. Mazzotta, F. Ricca, Answer set programming and neurosymbolic ai: Applications and future perspectives (invited talk), in: International Workshop on Hybrid Models for Coupling Deductive and Inductive Reasoning, Springer, 2024, pp. 1–21.
[27] X. Lin, Y. Wu, H. Yang, Y. Huang, Y. Zhang, J. Ji, Y. Zhang, Improving efficiency of answer set planning with rough solutions from large language models for robotic task planning, in: Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, 2025, pp. 4570–4578.
[28] A. Ishay, J. Lee, Llm+ al: Bridging large language models and action languages for complex reasoning about actions, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 24212–24220.