Improving Efficiency of Answer Set Planning with Rough Solutions from Large Language Models for Robotic Task Planning

Xinrui Lin[1] , **Yangfan Wu**[1] , **Huanyu Yang**[1] , **Yuting Huang**[1] , **Yu Zhang**[1,2] , **Jianmin Ji**[1,2*] and **Yanyong Zhang**[1,2]

[1]University of Science and Technology of China, China

[2]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, China

{xinruilin, uuyf, yanghuanyu, yutinghuang}@mail.ustc.edu.cn,
{yuzhang, jianmin, yanyongz}@ustc.edu.cn

Abstract

Answer Set Programming (ASP) planning can be used to refine the rough solutions generated by Large Language Models (LLMs) to handle specific restrictions of actions, i.e., reconstruct the rough solutions to be executable, for robotic task planning. However, it is still challenging to efficiently solve ASP programs that have multiple variables with large domains, which prevents the above application of ASP planning from real-world task planning problems. In this paper, we consider how to reduce the domains of variables without losing possible solutions for ASP planning, while given these rough solutions from LLMs. Based on the above reduction, we introduce CLMASP, an approach that couples LLMs with ASP for robotic task planning. We evaluate CLMASP on the VirtualHome platform for common indoor tasks, demonstrating a significant improvement in the executable rate from under 10% to nearly 90% and reducing average ASP planning time from over 2 hours to under 5 seconds. Code is available at https://github.com/CLMASP/CLMASP.

1 Introduction

Answer Set Programming (ASP) [Lifschitz, 2019] has been applied to robotic task planning [Chen *et al.*, 2013; Tran *et al.*, 2023]. However, answer set planning still faces challenges in real-world applications due to the extensive domain knowledge required and the limited computational efficiency in large-scale problems.

Conversely, Large Language Models (LLMs), such as GPT-4 [OpenAI, 2023] and Llama 3 [Meta AI, 2024], possessing broad foundational knowledge, are well-suited for task planning in open-world scenarios [Kambhampati *et al.*, 2024]. It is hard for LLMs to directly generate executable plans that follow specific constraints of the robot [Wu *et al.*, 2023]. For instance, an LLM-generated plan might skip essential actions like "plugin washing_machine" before "switch_on washing_machine". However, these
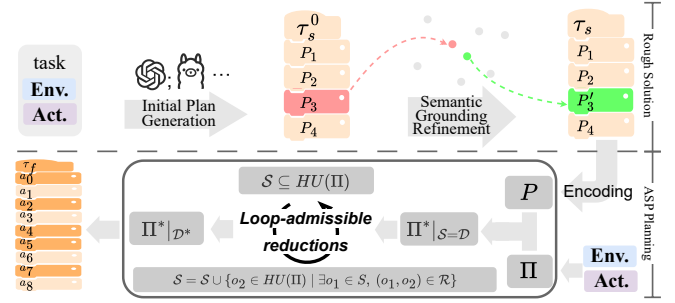
---

*Corresponding author

Figure 1: CLMASP workflow: An LLM generates an initial plan which undergoes Semantic Grounding Refinement to produce a rough solution, which is then enhanced by ASP to generate a detailed, executable plan.

LLM-generated rough solutions serve as valuable guidance.

In this paper, we consider how to utilize these rough solutions to speed up answer set planning. Rough solutions from LLMs can improve the computational efficiency of answer set planning in two different ways. Firstly, inspired by the TLPLAN system [Bacchus and Kabanza, 2000] that utilizes domain-specific control knowledge to speed up the planning process, we encode the ASP program to refine the rough plan to accomplish the task. In this sense, we define the planning problem with a rough solution and provide an ASP encoding to generate executable plans for the problem. Secondly, we consider how to reduce the domains of variables to speed up the grounding process in ASP. However, the ground ASP program obtained by a reduction may generate different results from the original program. To address this, we introduce the notions of *admissible reduction* and *safe reduction* that preserve possible solutions for the original program, while *admissible reductions* are more helpful for answer set planning. It is hard to identify *admissible* or *safe reductions*. Then we propose a sufficient condition with the notions of loops and loop formulas [Lin and Zhao, 2004] for identifying *admissible reductions* and apply the condition in our encoding for answer set planning.

For a typical task planning problem with hundreds of objects and requiring dozens of actions to accomplish the task, the

above methods can reduce the computing time from more than 2 hours to less than 5 seconds. Building on the two speedup methods, we introduce **CLMASP** (see Figure 1), an approach that couples LLMs with ASP for robotic task planning. Our contributions are as follows:

- We consider leveraging rough solutions from LLMs to accelerate answer set planning. To enhance ASP planning efficiency, we propose two methods: (1) defining the planning problem using a rough solution refined by an ASP program, and (2) introducing *admissible* and *safe reductions*, along with a sufficient condition based on loops and loop formulas to reduce variable domains.

- We provide a framework, CLMASP, that leverages the accelerated ASP planning methods to improve both computational efficiency and the executability of plans, as demonstrated through experiments on the Virtual-Home platform.

2 Related Work

2.1 Improve Efficiency of ASP

Reducing domains of variables to reduce the size of ground programs has not been considered much in ASP. Lazy grounding [Dal Palu *et al.*, 2009] requires a new ASP solver implementation, rather than using existing solvers such as clingo [Gebser *et al.*, 2019]. Although methods such as rewriting [Pearce, 2004; Ji *et al.*, 2015a; Ji *et al.*, 2016], splitting [Lifschitz and Turner, 1994; Ji *et al.*, 2015b; Fandinno and Lierler, 2023], and forgetting [Ji *et al.*, 2015c; Saribatur and Woltran, 2024] have been extensively explored, they often require some sense of equivalence during the simplification of logic programs. Abstraction [Saribatur and Eiter, 2021; Saribatur *et al.*, 2021] approximates answer sets without losing any, by simplifying logic programs so that every answer set of the original program corresponds to one in the abstract program. However, it does not explicitly consider the grounding process. In contrast, our *admissible* and *safe reductions* operate at the grounding level by pruning constants to reduce ground program size while preserving relevant answer sets. To our knowledge, such reductions have not been explored in ASP.

2.2 LLMs in Robotic Task Planning

Recent research has explored the application of LLMs in robotic task planning using prompt-based methods that integrate domain knowledge through natural language [Huang *et al.*, 2022; Song *et al.*, 2023] or pseudo-Python code [Singh *et al.*, 2023; Liang *et al.*, 2023]. However, LLMs often struggle to retain crucial information in long-horizon tasks. To address these limitations, the integration of external tools like knowledge base feedback [Gou *et al.*, 2023], local checkers [Chen *et al.*, 2024], and physical environment interactions [Bhat *et al.*, 2024] has been proposed, alongside fine-tuning LLMs for specific tasks to minimize inaccuracies [Xiang *et al.*, 2024; Mu *et al.*, 2024]. Nevertheless, robotic task plans still struggle to fully conform to action models and environmental constraints, complicating the successful completion of tasks.

In contrast, a well-designed symbolic planner can yield comprehensive and interpretable results within a specific domain [Pan *et al.*, 2023]. Current research is exploring the integration of symbolic planners with LLMs [Liu *et al.*, 2023; Yang *et al.*, 2023]. Compared to our proposed CLMASP, existing approaches primarily use LLMs for translating informal to formal representations, neglecting their planning and commonsense reasoning capabilities, and fail to adapt symbolic planners for effective collaboration with large models.

3 Preliminaries

We first recall the definition of Answer Set Programming (ASP) with variables [Van Harmelen *et al.*, 2008], focusing on key concepts used in our method. We then review loops and loop formulas [Lin and Zhao, 2004], and finally discuss the ASP encoding for robot task planning [Tran *et al.*, 2023].

3.1 Answer Set Programming with Variables

Consider a first-order vocabulary $\Theta = (\mathcal{P}, \mathcal{D})$, where $\mathcal{P}$ and $\mathcal{D}$ are finite, non-empty sets of predicates and constants, respectively. A *term* is a constant from $\mathcal{D}$ or a variable from a set of variables $\mathcal{V}$. An *atom* has the form $p(t_1, \ldots, t_n)$ where $p \in \mathcal{P}$ and each t_i is a term. A *ground atom* contains no variables. An ASP program Π with variables consists of rules of the form:

$$A_0 \leftarrow A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n.$$

where A_i are *atoms* ($0 \leq i \leq n$) and $n \geq 1$. Each rule r is also written as $head(r) \leftarrow body(r)$, where $body(r) = A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n$, $body^+(r) = \{A_1, \ldots, A_m\}$ and $body^-(r) = \{A_{m+1}, \ldots, A_n\}$. Π is *ground* if all its rules are *ground*; Π is *safe* if each variable in rule r appears in $body^+(r)$. Safety is achieved using *domain predicates* - unary predicates given as facts, universally true for all constants used to ground the variables. The *Herbrand universe* of Π, denoted by $HU(\Pi)$, consists of the set of all constants, that is, $HU(\Pi) = \mathcal{D}$.

A *complete variable assignment* σ maps each variable in $\mathcal{V}$ to a constant in $\mathcal{D}$. For a rule r, the ground rule $r\sigma$ replaces every variable in r with its value in σ. The set of all ground instances of r over $\mathcal{D}$, denoted $r|_{\mathcal{D}}$, includes all possible ground rules with constants from $\mathcal{D}$:

$$r|_{\mathcal{D}} = \{\, r\sigma \mid$$
$$\sigma \text{ is a complete variable assignment for } r \text{ over } \mathcal{D}\}.$$

For an ASP program Π, we can construct its ground program $\Pi|_{HU(\Pi)} = \bigcup_{r \in \Pi} r|_{HU(\Pi)}$, referred to as the *grounding result*.

Given an *interpretation* I, i.e., I is a set of ground atoms, I *satisfies* a ground rule r, if $head(r) \in I$ whenever $body^+(r) \subseteq I$ and $body^-(r) \cap I = \emptyset$. An interpretation I is a *model* of Π, if it satisfies every ground rule in $\Pi|_{HU(\Pi)}$. I is a *minimal* model if there is no other model J of Π such that $J \subset I$.

Now, we define the answer sets of an ASP program with variables. Given an ASP program Π and an interpretation I, the Gelfond-Lifschitz reduction [Gelfond and Lifschitz, 1988] of Π on I, written Π^I, is obtained from $\Pi|_{HU(\Pi)}$ by deleting:

1. each rule that has *not A* in its body with $A \in I$, and

2. all *not A* in the bodies of the remaining rules.

For any interpretation I, Π^I is a ground positive ASP program, then Π^I has only one minimal model. Thus, an interpretation I is an *answer set* of Π iff I is the minimal model of Π^I. An ASP program may have zero, one or multiple answer sets. We focus here on normal rules; richer constructs follow in Sec. 3.3.

3.2 Loops and Loop Formulas

With the notions of loops and loop formulas, we can convert an ASP program Π into a propositional formula such that an interpretation I is an answer set of Π iff it is a model of the propositional formula. Notice that, we define the notions of loops and loop formulas on the ground ASP program $\Pi|_{HU(\Pi)}$, as we define the notion of answer sets on $\Pi|_{HU(\Pi)}$, which slightly differs from the first-order loop formulas in [Chen *et al.*, 2006].

Given an ASP program Π, the *positive dependency graph* of Π, written G_Π, is a directed graph whose vertices are atoms in $\Pi|_{HU(\Pi)}$ and there is an arc from A to B if there is a rule $r \in \Pi|_{HU(\Pi)}$ such that $A \in head(r)$ and $B \in body^+(r)$. A set L of ground atoms is a *loop* of Π if the L-induced subgraph of G_Π is strongly connected. Every singleton in $\Pi|_{HU(\Pi)}$ is also a loop of Π.

Given a loop L of an ASP program Π, a ground rule $r \in \Pi|_{HU(\Pi)}$ is an *external support* of L if $head(r) \in L$ and $L \cap body^+(r) = \emptyset$. We denote $R^-(L, \Pi)$ to be the set of all external support rules of L in $\Pi|_{HU(\Pi)}$. The *loop formula* of L under Π, written $LF(L, \Pi)$, is given by:

$$\bigwedge_{A \in L} A \supset \bigvee_{r \in R^-(L,\Pi)} body(r).$$

Theorem 1. *[Lin and Zhao, 2004] Given an ASP ground program Π and an interpretation I. If I is a model of Π, then I is an answer set of Π if and only if I satisfies $LF(L, \Pi)$ for all loops L of Π.*

3.3 Encoding of Answer Set Planning

A robot task planning problem can be encoded as an ASP program Π [Tran *et al.*, 2023], where each answer set I corresponds to a valid plan. Π comprises the following components:

- A set of fluents F representing states, `h(f,t)` (fluent `f` is true at time step `t`);

- A set of actions A represented as `occurs(a,t)` (action `a` occurs at time step `t`);

- Initial state s_0 using fluents at time 0;

- Action preconditions as constraints on `occurs(a,t)`;

- Effect axioms describing how actions modify fluents;

- Frame axioms for unchanged fluents;

- Goal conditions at the final time step.

Algorithm 1: ASP-based Task Planning Encoding

Input:
Π_{domain}: domain encoding (objects, initial states, maximum allowed timesteps (T_{max}), etc.).
Π_{step}: action encoding (preconditions, effects, etc.)
Π_{check}: goal-checking constraints.
Output: Plan $\tau = \langle s_0, a_1, s_1, \ldots, s_n \rangle$.

1: Set $t \leftarrow 0$ {Initialize time step t to 0}
2: Load $\Pi_{domain}, \Pi_{step}, \Pi_{check}$
3: **while** no solution found and $t < T_{max}$ **do**
4:     $t \leftarrow t + 1$ {Increment time step t}
5:     Ground and solve $\Pi_{base} \cup \Pi_{step}(1..t) \cup \Pi_{check}(t)$
6:     **if** an answer set I is found **then**
7:         Extract plan τ from `occurs(a,t)` and `h(f,t)` in I
8:         **return** τ
9:     **end if**
10: **end while**
11: **return** "No solution found within T_{max} steps."

A *trajectory* τ is a sequence $\langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$, where each s_i is the state at time i ($0 \leq i \leq n$), and a_j ($1 \leq j \leq n$) is the action at time j.[1] An answer set I of Π contains exactly such a trajectory, where states and actions are derived from `h(f,t)` and `occurs(a,t)` present in I respectively. A trajectory τ *satisfies* an encoding Π if there exists an answer set I of Π such that τ can be constructed from I.

We now describe a practical ASP encoding Π for robot task planning, using *clingo*'s syntax [Potassco Team, 2025]. While our preceding discussion focused on normal rules, this encoding also employs constructs like choice rules. To find the shortest plan, we use *clingo*'s incremental mode. Listing 1 shows an ASP encoding of the `wash` and `find` actions, structured per Algorithm 1. Traditional encoding includes Π_{domain} (Lines 3-10) defining objects, initial states, and actions; Π_{step} (Lines 11-22) specifying action preconditions, effects, and state transitions; and Π_{check} (Lines 26-27) detailing goal constraints. When executed, the ASP solver returns an answer set "`occurs(find(2),1).`
`occurs(wash(2),2).`" indicating the robot first `finds` object 2, then `washes` it.

While this encoding can scale to more complex domains by adding new rules, handling numerous constants and actions in indoor task planning can lead to inefficient planning. Section 4 explores methods to enhance computational efficiency.

4 Improve Efficiency of Answer Set Planning with Rough Solutions

Despite advances in ASP solver efficiency, real-world robot task planning remains challenging due to the computational complexity, becoming NP-complete for bounded plan lengths [Tran *et al.*, 2023]. LLMs, with their extensive knowledge, can facilitate the planning process by generating rough solutions. This section shows how to utilize these rough solutions to speed up answer set planning by: (1) incorporat-

---

[1]In this study, only one action is performed at each time step.

Listing 1: ASP Encoding for `wash` and `find`. Our proposed skeleton plans (Section 4.1, lines 23-25, 28) replace the traditional method (Section 3.3, commented out in lines 22, 27), both of which are compared in our experiments.

```
1    #include <incmode>.
2    #const tmax = 30.
3    #program base.
4    num(0..tmax). time(0).
5    is(1, character). is(2, clothesshirt).
6    robot(1). object(2).
7    action_of(wash(O)) :- object(O).
8    action_of(find(O)) :- object(O).
9    h(hand_empty(1),0). h(same_loc(1,2),0).
10   task(wash(O)) :- is(O, clothesshirt), object(O).
11   #program step(t).
12   time(T) :- num(T), T==t.
13   1{occurs(A,t): action_of(A)}1.
14   h(clean(O),t) :- occurs(wash(O),t), object(O).
15   :- occurs(wash(O),t), not h(hand_empty(C),t-1), robot(C
        ), object(O).
16   :- occurs(wash(O),t), not h(found(O),t-1), object(O).
17   h(found(O),t) :- occurs(find(O),t), object(O).
18   :- occurs(find(O),t), not h(same_loc(C,O),t-1), robot(C
        ), object(O).
19   h(hand_empty(C),t) :- h(hand_empty(C),t-1), not -h(
        hand_empty(C),t), robot(C).
20   h(same_loc(C,O),t) :- h(same_loc(C,O),t-1), not -h(
        same_loc(C,O),t), robot(C), object(O).
21   h(clean(O),t) :- h(clean(O),t-1), not -h(clean(O),t),
        object(O).
22   % goal(wash(O),t):-h(clean(O),t), object(O).
23   goal(subtask(p,find(2),1),t) :- occurs(find(2),t).
24   goal(subtask(p,wash(2),2),t) :- occurs(wash(2),t).
25   goal(p,t) :- goal(subtask(p,find(2),1),T1), time(T1),
        goal(subtask(p,wash(2),2),T2), time(T2), T1<=T2,
        T2<=t.
26   #program check(t).
27   % :-query(t), not goal(Task,t), task(Task).
28   :-query(t), not goal(p,t).
29   #show occurs/2.
```

ing *skeleton plans* into existing ASP encoding—an idea inspired by TLPLAN [Bacchus and Kabanza, 2000], which uses domain-specific control knowledge to guide the planner, and (2) reducing variable domains during grounding without losing any valid plans.

4.1 Answer Set Planning with Skeleton Plans

To enhance the efficiency of answer set planning, we introduce *skeleton plans* (e.g., generated by LLMs) that provide guided instructions for tasks.

A *planning problem with a skeleton plan* is a pair (Π, P), where Π is an ASP encoding of the planning domain, and P is a skeleton plan aligned with Π's signature. The signature comprises disjoint sets of action names (from ground terms in `occurs(a,t)`), fluent names (from ground atoms `h(f,t)`), and subtask names, with no circular subtask references. A *skeleton plan* P is recursively defined as:

- An action name a;

- A fluent-specification φ, formed from fluent names using propositional connectives;

- A subtask name p;

- A sequence $P_1; \ldots; P_m$, where each P_i is a skeleton plan.

A trajectory $\tau = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ satisfies P if:

- $P = a$ and $a = a_n$;

- $P = \varphi$ and s_n satisfies φ;

- $P = p$ and τ satisfies the skeleton plan for subtask p;

- $P = P_1; \ldots; P_m$, and there exist $0 \leq n^1 \leq \cdots \leq n^{m-1} \leq n$ such that each sub-trajectory $\langle s_{n^{i-1}}, a_{n^{i-1}+1}, \ldots, s_{n^i} \rangle$ satisfies P_i (with $n^0 = 0$, $n^m = n$).

A trajectory τ is a *solution* to (Π, P) if it satisfies both Π and P. This implies that a plan for Π can be constructed by extending the skeleton in P into an executable plan, rather than exploring all possible action sequences, enhancing efficiency.

For example, a skeleton plan for task "wash object 2" can be:

$$P = \texttt{find(2)}; \texttt{wash(2)}.$$

This plan encodes a high-level instruction that the robot should first `find`s object 2, then `wash`es it. Although execution details are abstracted away, it still constrains the search space and is embedded into ASP via auxiliary rules.

Skeleton plans are incorporated by adding ASP rules for subtask definitions and plan satisfaction constraints, as shown in Listing 1 (Lines 23–25, 28). This enables efficient plan refinement guided by P, compared to unconstrained planning.

Property 1. *Given an encoding of the answer set planning problem Π and a skeleton plan P, a solution τ of (Π, P) always contains a plan for Π and can be computed from the answer set of the encoding of Π appending with the encoding for the skeleton plan P.*

This holds because P constrains the solution space of Π without changing its underlying planning logic. Any answer set satisfying both still represents a valid plan for Π.

Property 1 enables skeleton plans to significantly reduce the search space, enhancing planning efficiency by decreasing the complexity from $O\big((S \times A)^N\big)$ to $O\big(m \times (S \times A)^{N/m}\big)$, where S is the state space size, A is the action space size, N is the maximum number of timesteps, and m is the number of skeleton steps. In our VirtualHome experiments, which involve numerous objects and relations, appropriate skeleton plans can accelerate answer set planning.

4.2 Reduce Domains of Variables

ASP solvers must ground program Π into $\Pi|_{HU(\Pi)}$ before computing answer sets. However, the size of $\Pi|_{HU(\Pi)}$ can be exponentially larger than Π, creating a performance bottleneck. For example, VirtualHome task planning problems with hundreds of objects produce groundings exceeding 16MB at time 0, causing *clingo* to timeout after 2 hours.

Our encoding Π ensures safety through domain predicates (unary predicates given as facts). Each variable in a rule is restricted by a domain predicate in the rule's body. For instance, in rule r at Line 14 of Listing 1, variable O is restricted by predicate `object`.

Since most objects are irrelevant for plan generation, we can omit them during answer set planning. This section discusses how to reduce the constant set for grounding while preserving solutions. We will define *admissible reductions* and *safe reductions*, provide a sufficient condition for *admissible reduction*, and demonstrate its application in speeding up answer set planning.

Definition 1. *Given an ASP program Π (with variables), a set $\mathcal{D}$ of constants with $\mathcal{D} \subseteq HU(\Pi)$ is an* admissible reduction *of Π, if*

- *for every answer set $I_{\mathcal{D}}$ of the ground program $\Pi|_{\mathcal{D}}$, there always exists an answer set I of Π such that $I_{\mathcal{D}} \subseteq I$.*

Definition 2. *An* admissible reduction *$\mathcal{D}$ of Π is also a* safe reduction *of Π, if*

- *for every answer set I of Π, there always exists an answer set $I_{\mathcal{D}}$ of $\Pi|_{\mathcal{D}}$ such that $I_{\mathcal{D}} \subseteq I$.*

Note that for $\mathcal{D} \subseteq HU(\Pi)$, while $\Pi|_{\mathcal{D}} \subseteq \Pi|_{HU(\Pi)}$, $\mathcal{D}$ may neither be an admissible nor a safe reduction of Π. Consider program Π':

$$f(a). \qquad f(b). \qquad g(a,b). \qquad \leftarrow g(x,y), f(x), f(y).$$

where $HU(\Pi') = \{a, b\}$, x, y are variables appearing in Π'. With $\mathcal{D} = \{a\}$, $\Pi'|_{\mathcal{D}}$ has answer set $\{f(a), f(b), g(a,b)\}$, but Π' has none. Thus $\{a\}$ is neither an admissible nor a safe reduction, while $\{a, b\}$ is both.

Theorem 2. *Given a subset $\mathcal{D}$ of $HU(\Pi)$ for an ASP program Π, deciding whether $\mathcal{D}$ is an admissible reduction or a safe reduction of Π is coNP-hard.*

Proof. We can construct an ASP problem Π from a 3-SAT problem by adding $\{a \leftarrow not\, a'.\ a' \leftarrow not\, a.\}$ for each atom a, without loss of generality, for each clause $\neg a \vee b \vee \neg c$ adding $\{\leftarrow a,\, not\, b,\, c.\}$, and adding $\{f(o_1).\ f(o_2). \leftarrow f(x), f(y), x \neq y.\}$. $HU(\Pi) = \{o_1, o_2\}$, $\Pi|_{HU(\Pi)}$ has no answer sets, $\Pi|_{\{o_1\}}$ has an answer set iff the 3-SAT problem is satisfiable. Then $\{o_1\}$ is an admissible or a safe reduction iff the 3-SAT problem is unsatisfiable. $\square$

It is hard to identify admissible and safe reductions. In the application of answer set planning, rather than finding all solutions, we prefer to find an executable plan for the task. Then we consider the applications of *admissible reduction* in this paper.

With the help of the notions of loops and loop formulas, we provide a sufficient condition for *admissible reduction*.

Definition 3. *Given an ASP program Π (with variables), a set $\mathcal{D}$ of constants with $\mathcal{D} \subseteq HU(\Pi)$ is a* loop-admissible reduction *of Π, if*

1. *for every answer set $I_{\mathcal{D}}$ of $\Pi|_{\mathcal{D}}$, there exists an interpretation I' such that $I_{\mathcal{D}} \cup I'$ satisfies rules in $\Pi|_{HU(\Pi)}$ and loop formulas for every loop L' of Π with $L' \subseteq I'$, and*

2. *there does not exist a loop L of Π such that L is not a loop of $\Pi|_{\mathcal{D}}$ and L contains a loop L' of $\Pi|_{\mathcal{D}}$ with $R^-(L', \Pi|_{\mathcal{D}}) \neq \emptyset$.*

Intuitively, we require that the rules in $\Pi|_{HU(\Pi)} \setminus \Pi|_{\mathcal{D}}$ and any newly introduced loop formulas can be satisfied by $I_{\mathcal{D}} \cup I'$.

Theorem 3. *Given a subset $\mathcal{D}$ of $HU(\Pi)$ for an ASP program Π, if $\mathcal{D}$ is a loop-admissible reduction of Π, then $\mathcal{D}$ is an admissible reduction of Π.*

Proof. Let $I_{\mathcal{D}}$ be an answer set of $\Pi|_{\mathcal{D}}$, from the definition of loop-admissible reduction, there exists the set I' of ground atoms such that $I_{\mathcal{D}} \cup I'$ is a model of Π and $I_{\mathcal{D}} \cup I'$ satisfies loop formulas of loops L of Π with $L \subseteq I_{\mathcal{D}}$ or $L \subseteq I'$.

To prove that $\mathcal{D}$ is an admissible reduction of Π, we need to show that $I_{\mathcal{D}} \cup I'$ is an answer set of Π. By Theorem 1, it suffices to prove that $I_{\mathcal{D}} \cup I'$ satisfies $LF(L, \Pi)$ for all loops L of Π. Consider an arbitrary loop L of Π. We analyze all possible cases:

1. Case $L \nsubseteq I_{\mathcal{D}} \cup I'$: By the the loop formula definition, the formula is vacuously true when the premise $\bigwedge_{A \in L} A$ is false.

2. Case $L \subseteq I_{\mathcal{D}}$: Since $I_{\mathcal{D}}$ is an answer set of $\Pi|_{\mathcal{D}}$ and for any rule $r \in R^-(L, \Pi|_{\mathcal{D}}) \subseteq R^-(L, \Pi)$, $I_{\mathcal{D}}$ satisfies $LF(L, \Pi)$.

3. Case $L \subseteq I'$: By condition 1 of the loop-admissible reduction definition, $I_{\mathcal{D}} \cup I'$ satisfies $LF(L, \Pi)$.

4. Case $L \cap I_{\mathcal{D}} \neq \emptyset \wedge L \cap I' \neq \emptyset \wedge L \subseteq I_{\mathcal{D}} \cup I'$: We prove by contradiction that this case cannot occur. Suppose such a loop L exists. Since $I_{\mathcal{D}}$ is an answer set of $\Pi|_{\mathcal{D}}$, by Theorem 1, there must exist a loop $L' \subseteq L \cap I_{\mathcal{D}} \subseteq L$ that is at least a singleton, such that $R^-(L', \Pi|_{\mathcal{D}}) \neq \emptyset$. However, this contradicts condition 2 of the loop-admissible reduction definition.

So, for every loop L of Π, $I_{\mathcal{D}} \cup I'$ satisfies $LF(L, \Pi)$. Given that $I_{\mathcal{D}} \cup I'$ is also a model of Π, by Theorem 1, $I_{\mathcal{D}} \cup I'$ is an answer set of Π. $\square$

Notice that, a loop-admissible reduction $\mathcal{D}$ does not imply a safe reduction, as for a loop $L \subseteq I_{\mathcal{D}}$ in both $\Pi|_{\mathcal{D}}$ and $\Pi|_{HU(\mathcal{D})}$, $LF(L, \Pi|_{\mathcal{D}}) \supset LF(L, \Pi|_{HU(\mathcal{D})})$ but not vice versa.

Application to Answer Set Planning. For a planning problem with skeleton plan (Π, P), we define a loop-admissible reduction $\mathcal{D}^*$ as the minimal subset of the $HU(\Pi)$, that satisfies the following conditions: (1) $\mathcal{D} \subseteq \mathcal{D}^*$, where $\mathcal{D}$ is the initial set of relevant elements; and (2) $\mathcal{D}^*$ is closed under the binary relation $\mathcal{R} \subseteq HU(\Pi) \times HU(\Pi)$, such that $(o_1, o_2) \in \mathcal{R}$ and $o_1 \in \mathcal{D}^*$ imply $o_2 \in \mathcal{D}^*$. Formally, $\mathcal{D}^*$ can be expressed as:

$$\mathcal{D}^* = \bigcup \left\{ \mathcal{S} \subseteq HU(\Pi) \;\middle|\; \begin{array}{l} \mathcal{D} \subseteq \mathcal{S} \text{ and } \forall (o_1, o_2) \in \mathcal{R}, \\ o_1 \in \mathcal{S} \implies o_2 \in \mathcal{S} \end{array} \right\}.$$

Method	LLM	$\overline{GCR}$	R_{exec}
[Huang *et al.*, 2022]	GPT-3.5	6.27 ± 3.79%	9.29 ± 8.43%
	GPT-4o	9.94 ± 6.10%	11.90 ± 10.48%
[Singh *et al.*, 2023]	GPT-3.5	20.42 ± 6.02%	16.67 ± 9.60%
	GPT-4o	21.70 ± 8.04%	13.75 ± 11.86%
[Mu *et al.*, 2024]*	Llama-3.1	26.73 ± 8.07%	32.71 ± 22.01%
CLMASP-Clingo (ours)	Llama-3.1	13.69 ± 4.09%	81.04 ± 15.36%
	GPT-3.5	37.07 ± 9.72%	**89.59 ± 9.33%**
	GPT-4o	**41.90 ± 10.75%**	86.99 ± 11.32%

Table 1: **Comparison of CLMASP-Clingo with other baselines.** Results using (Clingo) Full are shown in Table 3. Method marked with * use a fine-tuned LLM.

This expansion primarily affects binary relations (e.g., $\text{on}(o_1, o_2)$, $\text{in}(o_1, o_2)$, $\text{close}(o_1, o_2)$). The resulting $\mathcal{D}^*$ satisfies both conditions for loop-admissible reduction because:

- Different domain predicates restrict variables in each rule of our encoding, ensuring $\Pi|_{HU(\Pi)} \setminus \Pi|_{\mathcal{D}^*}$ and its loop formulas are satisfiable.

- The construction of $\mathcal{D}^*$ ensures the second condition of Definition 3 is met for all relevant loops.

Let Π^* be the combination of encodings for Π and P. We can compute answer sets of $\Pi^*|_{\mathcal{D}^*}$ to obtain the solution of (Π, P).

Property 2. *Given the encoding Π^* of a planning problem with a skeleton plan (Π, P), the expanded set $\mathcal{D}^*$ of constants is a loop-admissible reduction of Π^*. A solution of (Π, P) can be obtained from the answer set of $\Pi^*|_{\mathcal{D}^*}$.*

This is guaranteed by the construction of $\mathcal{D}^*$, which ensures it contains all constants relevant to the planning problem while satisfying loop-admissible reduction conditions.

In our experiments, task planning in VirtualHome involves hundreds of objects. By reducing variable domains, we focus on dozens of objects and relations, greatly reducing ground program size. With appropriate skeleton plans, *clingo* computation time was cut from over 2 hours to under 5 seconds.

5 Couple LLMs with Answer Set Planning

To effectively integrate LLM common sense with ASP's reasoning capabilities for real-world planning tasks, we propose CLMASP as a two-phase planning framework, as illustrated in Figure 1. This framework first leverages the LLM for rough planning and grounding, followed by ASP to elaborate the skeleton plan into a final solution.

5.1 Generating a Skeleton Plan by LLMs

Initial Plan Generation. Given a task description, primitive actions, environmental object categories, and three planning examples, LLMs generate an initial plan, τ_s^0. This plan uses object categories instead of specific IDs due to LLM input capacity limitations.

Semantic Grounding and Refinement (SGR). The SGR step refines the initial plan τ_s^0 by correcting invalid or ambiguous terms (e.g., "walk kitchencounter" to "walk kitchen" because "kitchencounter" is absent). It computes cosine similarities ($\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$) between plan element embeddings (a) and environmental component embeddings (b). Invalid terms are then substituted with the closest valid environmental actions or objects. This yields a semantically refined plan, τ_s, ready for ASP reasoning and execution.

5.2 Enhancing a Skeleton Plan by ASP

The skeleton plan τ_s is refined using ASP to establish action dependencies and specify object locations. First, reusable ASP codes encode robotic action models (e.g., cause-effect relationships). ASP's initial state is then derived from a global environment map—a directed acyclic graph of entity nodes and relationship edges (e.g., $\text{is}(1, \text{character})$, $\text{state}(7, \text{dirty})$, $\text{relation}(\text{in}, 5, 2)$). Finally, a Python module translates τ_s into ASP rules, treating each subtask as a temporally ordered goal to produce the final plan τ_f. This process is detailed in Section 4.

6 Evaluation

6.1 Experiment Setup

Simulator and Dataset.

Experiments utilize the VirtualHome (VH) simulator v2.3.0, where an agent interacts with 250–300 household objects using 42 actions. These objects, linked by approximately 3,000 relationships, form an environment abstracted as a grounded directed acyclic graph. As shown in Fig. 1, task inputs comprise a natural language description (*task*), the environment as a serialized dictionary (*Env.*), and structured possible actions (*Act.*). Each task instance includes a reference plan—the ground truth sequence of verb-object pairs that accomplishes the *task*. (e.g., [WALK] <wardrobe>, [GRAB] <clothes>, [PUTIN] <clothes><clothesbin>for "put clothes in bin"). For evaluation, we sample 269 of 292 VH task instances and reserve 23 for prompting examples.

LLMs and Solvers. Evaluations of CLMASP and baselines utilize the LLMs *gpt-3.5-turbo-1106* [Brown *et al.*, 2020], *gpt-4o-2024-08-06* [Hurst *et al.*, 2024], and *Llama-3.1-8B* [Meta AI, 2024]. For its ASP solver, CLMASP employed *clingo* 5.7.1 [Gebser *et al.*, 2019] and *DLV2* [Calimeri *et al.*, 2022]. The SGR step in CLMASP used the *text-embedding-ada-002* embedding model. To ensure fairness, all experimental results in this paper are obtained from actual measurements rather than citations.

Metrics. Metrics include *ASP Planning Time* (T_{ASP}), the *ASP runtime* (Fig. 1, lower part); *Solvability Rate* (R_{sol}), the proportion of instances where the final plan τ_f is successfully solved; *Executability Rate* (R_{exec}), the portion of τ_f successfully executed in VH; and *Final Plan Length Stats* (L_{τ_f}), covering average, median, and maximum τ_f length. We also measure *Mean Goal Condition Recall* ($\overline{GCR}$), averaged across all instances. For each instance, *GCR* is the

Solver	Model	T_{ASP}					R_{sol}					L_{τ_f}	
		-S-R	-R	-S	Full	Δ	-S-R	-R	-S	Full	Δ	-S	Full
DLV2	Llama-3.1	>2h	>2h	39.37s	23.33s	-40.7%	0	0	7.4%	65.4%	+58.0%	(4.75/6/13)	(6.08/ 7/29)
	GPT-3.5	>2h	>2h	37.24s	24.38s	-34.5%	0	0	7.4%	93.3%	+85.9%	(4.75/6/13)	(7.53/10/30)
	GPT-4o	>2h	>2h	38.75s	29.29s	-24.4%	0	0	7.4%	91.4%	+84.0%	(4.75/6/13)	(8.23/11/30)
Clingo	Llama-3.1	>2h	>2h	32.06s	4.16s	-87.1%	0	0	7.4%	84.0%	+76.6%	(4.75/6/13)	(5.37/ 7/24)
	GPT-3.5	>2h	>2h	30.52s	4.63s	-84.8%	0	0	7.4%	93.7%	+86.3%	(4.75/6/13)	(7.17/ 9/28)
	GPT-4o	>2h	>2h	31.17s	11.20s	-64.1%	0	0	7.4%	91.4%	+84.0%	(4.75/6/13)	(8.20/11/28)

Table 2: Ablation study on the skeleton plan and loop-admissible reduction techniques in CLMASP. Each variant removes the corresponding technique from CLMASP (Full): removing loop-admissible reduction is denoted as (-R), removing skeleton plan is denoted as (-S), and removing both of them is denoted as (-S-R). The table also shows the relative improvement of Full over -S, denoted as Δ.

LLM		**Method**					
		-SGR-ASP	-ASP	(DLV2)-SGR	(Clingo)-SGR	(DLV2) Full	(Clingo) Full
Llama-3.1	$\overline{GCR}$	$3.83\pm1.48\%$	$11.32\pm3.01\%$	$8.28\pm3.53\%$	$8.61\pm3.51\%$	$11.41\pm3.81\%$	$\mathbf{13.69\pm4.09\%}$
	R_{exec}	$3.72\pm3.58\%$	$27.14\pm19.77\%$	$39.41\pm23.88\%$	$42.01\pm24.36\%$	$62.45\pm23.45\%$	$\mathbf{81.04\pm15.36\%}$
GPT-3.5	$\overline{GCR}$	$14.05\pm6.35\%$	$23.69\pm8.07\%$	$35.93\pm9.98\%$	$35.35\pm9.92\%$	$\mathbf{37.18\pm9.89\%}$	$37.07\pm9.72\%$
	R_{exec}	$8.55\pm7.82\%$	$11.15\pm9.91\%$	$85.87\pm12.13\%$	$86.99\pm11.32\%$	$88.48\pm10.20\%$	$\mathbf{89.59\pm9.33\%}$
GPT-4o	$\overline{GCR}$	$22.55\pm7.41\%$	$28.62\pm8.06\%$	$42.40\pm10.68\%$	$41.76\pm10.94\%$	$\mathbf{42.34\pm10.84\%}$	$41.90\pm10.75\%$
	R_{exec}	$11.52\pm10.20\%$	$15.24\pm12.92\%$	$87.36\pm11.04\%$	$86.25\pm11.86\%$	$\mathbf{87.73\pm10.76\%}$	$86.99\pm11.32\%$

Table 3: Ablation study on the Semantic Grounding and Refinement (SGR) and ASP solving steps in CLMASP. Each variant removes the corresponding step from CLMASP (Full): removing Semantic Grounding and Refinement is denoted as (-SGR), removing ASP solving is denoted as (-ASP), and removing both is denoted as (-SGR-ASP). The ASP solver used is indicated in parentheses.

proportion of goal states and relations achieved:

$$GCR = 1 - \frac{|(\mathcal{D}_{\text{goal}} - \mathcal{D}_{\text{initial}}) - (\mathcal{D}_{\text{method}} - \mathcal{D}_{\text{initial}})|}{|\mathcal{D}_{\text{goal}} - \mathcal{D}_{\text{initial}}|},$$

where $\mathcal{D}_{\text{initial}}$, $\mathcal{D}_{\text{method}}$, and $\mathcal{D}_{\text{goal}}$ are the sets of object states and relations at the initial state, after method application, and in the goal state, respectively.

6.2 Results and Analysis

To verify CLMASP's effectiveness, we compare it with three baselines. As shown in Table 1, CLMASP's R_{exec} (90%) and $\overline{GCR}$ (42%) significantly surpass the best baseline (33% and 27%, respectively), indicating CLMASP effectively improves executability and goal condition fulfillment of LLM-based planning in large domains.

To assess the contribution of CLMASP's two ASP planning acceleration techniques, we conduct an ablation study. As shown in Table 2, removing loop-admissible reduction (-R) results in T_{ASP} exceeding 2 hours; including it drops solving time below 40 seconds. Adding the skeleton plan technique further improves solving efficiency by 24%-87% and significantly increases R_{sol} by approximately 58%-86%. The method is effective for L_{τ_f} across different LLMs and solvers. Therefore, these two techniques, especially the reduction technique, are crucial for accelerating ASP planning and key to its practicality.

To verify the necessity of CLMASP's dual-phase approach,

we also conduct an ablation study. As shown in Table 3, removing the ASP solving step leads to plans with only 3.7%-27.1% R_{exec}. In contrast, removing SGR still maintains 39%-87% R_{exec}. However, only when both components are included does the method achieve 89.59% R_{exec} and around 40% average $\overline{GCR}$. Thus, the ASP solving step is CLMASP's core phase, while SGR serves as a powerful enhancement to fully exploit its potential by correcting the grounding and utilization of objects.

A horizontal comparison in Table 2 and 3 reveals that while clingo and DLV2 offer similar ASP solving capabilities, clingo is more efficient. Regarding LLMs, their performance mainly dictates skeleton plan quality, which impacts $\overline{GCR}$ more significantly (sometimes negatively) than R_{exec}.

7 Conclusion

This paper defines the planning problem with a skeleton plan and encodes the ASP program to refine the skeleton plan to accomplish the task. We introduce *admissible and safe reductions* that preserve solutions, and to address the challenge of identifying them, we propose a sufficient condition for *admissible reductions* using loops and loop formulas. Using these speedup techniques, we present CLMASP, an approach coupling LLMs with ASP for robotic task planning. Experiments on the VirtualHome platform demonstrate CLMASP's improvements in both computational efficiency and plan executable rates.

Acknowledgments

Contribution Statement

The first two authors contributed equally to this work.

References

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial intelligence*, 116(1-2):123–191, 2000.

[Bhat *et al.*, 2024] Vineet Bhat, Ali Umut Kaypak, Prashanth Krishnamurthy, Ramesh Karri, and Farshad Khorrami. Grounding llms for robot task planning using closed-loop state feedback. *arXiv preprint arXiv:2402.08546*, 2024.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[Calimeri *et al.*, 2022] Francesco Calimeri, Giovambattista Ianni, Francesco Pacenza, Simona Perri, and Jessica Zangari. Asp-based multi-shot reasoning via dlv2 with incremental grounding. In *Proceedings of the 24th International Symposium on Principles and Practice of Declarative Programming*, pages 1–9, 2022.

[Chen *et al.*, 2006] Yin Chen, Fangzhen Lin, Yisong Wang, and Mingyi Zhang. First-order loop formulas for normal logic programs. *KR*, 6:298–307, 2006.

[Chen *et al.*, 2013] Xiaoping Chen, Jianmin Ji, Zhiqiang Sui, and Jiongkun Xie. Handling open knowledge for service robots. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 2459–2465, 2013.

[Chen *et al.*, 2024] Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6695–6702. IEEE, 2024.

[Dal Palu *et al.*, 2009] Alessandro Dal Palu, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. Gasp: answer set programming with lazy grounding. *Fundamenta Informaticae*, 96(3):297–322, 2009.

[Fandinno and Lierler, 2023] Jorge Fandinno and Yuliya Lierler. Splitting answer set programs with respect to intensionality statements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6338–6345, 2023.

[Gebser *et al.*, 2019] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080. Cambridge, MA, 1988.

[Gou *et al.*, 2023] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[Huang *et al.*, 2022] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[Hurst *et al.*, 2024] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[Ji *et al.*, 2015a] Jianmin Ji, Hai Wan, Ziwei Huo, and Zhenfeng Yuan. Simplifying a logic program using its consequences. In *IJCAI*, pages 3069–3075, 2015.

[Ji *et al.*, 2015b] Jianmin Ji, Hai Wan, Ziwei Huo, and Zhenfeng Yuan. Splitting a logic program revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[Ji *et al.*, 2015c] Jianmin Ji, Jia-Huai You, and Yisong Wang. On forgetting postulates in answer set programming. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[Ji *et al.*, 2016] Jianmin Ji, Hai Wan, Kewen Wang, Zhe Wang, Chuhan Zhang, and Jiangtao Xu. Eliminating disjunctions in answer set programming by restricted unfolding. In *IJCAI*, pages 1130–1137, 2016.

[Kambhampati *et al.*, 2024] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[Liang *et al.*, 2023] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the Eleventh International Conference on Logic Programming (ICLP-94)*, page 23–37, 1994.

[Lifschitz, 2019] Vladimir Lifschitz. *Answer set programming*. Springer Heidelberg, 2019.

[Lin and Zhao, 2004] Fangzhen Lin and Yuting Zhao. Assat: Computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.

[Liu et al., 2023] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[Meta AI, 2024] Meta AI. Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/, 2024. Accessed: 2025-06-05.

[Mu et al., 2024] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36, 2024.

[OpenAI, 2023] OpenAI. Gpt-4 research and capabilities. https://openai.com/index/gpt-4-research/, 2023. Accessed: 2025-06-05.

[Pan et al., 2023] Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, 2023.

[Pearce, 2004] David Pearce. Simplifying logic programs under answer set semantics. In *International Conference on Logic Programming*, pages 210–224. Springer, 2004.

[Potassco Team, 2025] Potassco Team. Potassco user guides and documentation. https://potassco.org/doc/, 2025. Accessed: 2025-06-05.

[Saribatur and Eiter, 2021] Zeynep G Saribatur and Thomas Eiter. Omission-based abstraction for answer set programs. *Theory and Practice of Logic Programming*, 21(2):145–195, 2021.

[Saribatur and Woltran, 2024] Zeynep G Saribatur and Stefan Woltran. A unified view on forgetting and strong equivalence notions in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10687–10695, 2024.

[Saribatur et al., 2021] Zeynep G Saribatur, Thomas Eiter, and Peter Schüller. Abstraction for non-ground answer set programs. *Artificial Intelligence*, 300:103563, 2021.

[Singh et al., 2023] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[Song et al., 2023] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023.

[Tran et al., 2023] Son Cao Tran, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: a survey. *Theory and Practice of Logic Programming*, 23(1):226–298, 2023.

[Van Harmelen et al., 2008] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*. Elsevier, 2008.

[Wu et al., 2023] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with large language models. *arXiv e-prints*, pages arXiv–2307, 2023.

[Xiang et al., 2024] Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. Language models meet world models: Embodied experiences enhance language models. *Advances in neural information processing systems*, 36, 2024.

[Yang et al., 2023] Zhun Yang, Adam Ishay, and Joohyung Lee. Coupling large language models with logic programming for robust and general reasoning from text. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5186–5219, 2023.