LEXICON: a Benchmark for Planning under Temporal Constraints in Natural Language

Periklis Mantenoglou, Rishi Hazra, Pedro Zuidberg Dos Martires
Örebro University, Sweden
{periklis.mantenoglou, rishi.hazra, pedro.zuidberg-dos-martires}@oru.se

Luc De Raedt
Örebro University, Sweden & KU Leuven, Belgium
luc.deraedt@kuleuven.be

Abstract

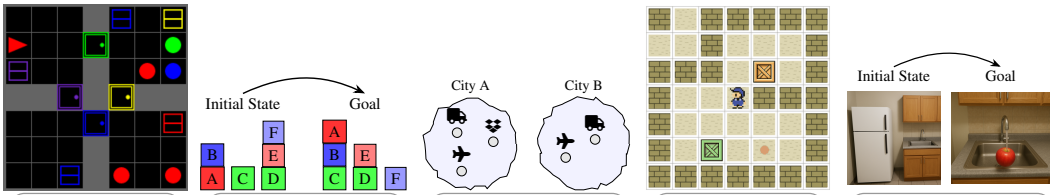
Owing to their reasoning capabilities, large language models (LLMs) have been evaluated on planning tasks described in natural language. However, LLMs have largely been tested on planning domains without constraints. In order to deploy them in real-world settings where adherence to constraints, in particular safety constraints, is critical, we need to evaluate their performance on constrained planning tasks. We introduce LEXICON—a natural language-based (LEXI) constrained (CON) planning benchmark, consisting of a suite of environments, that can be used to evaluate the planning capabilities of LLMs in a principled fashion. The core idea behind LEXICON is to take existing planning environments and impose temporal constraints on the states. These constrained problems are then translated into natural language and given to an LLM to solve. A key feature of LEXICON is its extensibility. That is, the set of supported environments can be extended with new (unconstrained) environment generators, for which temporal constraints are constructed automatically. This renders LEXICON future-proof: the hardness of the generated planning problems can be increased as the planning capabilities of LLMs improve. Our experiments reveal that the performance of state-of-the-art LLMs, including reasoning models like GPT-5, o3, and R1, deteriorates as the degree of constrainedness of the planning tasks increases.

1 Introduction

Planning with constraints is commonly required in problem-solving settings, ranging from resource allocation and scheduling [32] to ensuring safety in reinforcement learning [1, 15, 56, 14]. Several planning specification languages have been proposed [13, 38, 30, 20], including formalisms with constraints [16]. However, specifying the complex, possibly compositional, constraints of an environment in a formal language is rather intricate, as it requires, inter alia, significant domain expertise. Other common ways of integrating constraints in planning problems is via penalties in a reward function [28, 41, 10] or through the physics engine of the environment [7, 46]. These solutions are also challenging for non-experts, while, after tightly integrating constraints into an environment, they are often difficult to alter if needed. We address these limitations by enabling the human user to communicate constraints *directly* to the planning agent, via natural language (NL).

The advent of large language models (LLMs), trained on vast textual corpora, has made NL-based planning increasingly feasible. However, whether LLMs possess the reasoning capabilities required for effective planning remains an open question. Some works argue that LLMs can perform reasoning, and even act as *zero-shot planner* [52, 26, 23], while others critically highlight their limitations [50,

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.



Goal: Reach a red ball.
Constraints: Do not unlock the green door & get the purple box in the top-right room.

Goal: Get block A on top of block B.
Constraints: Never place block B on the table & sometime, clear block E.

Goal: Move the package to the airport of City B.
Constraints: Never use the airplane that is in City A.

Goal: Move all boxes on goal squares.
Constraints: Sometime, grid square (3,3) must be occupied.

Goal: Hold a washed apple.
Constraints: Sometime before washing the apple, place a plate on the kitchen counter.

Figure 1: Constrained problems on environments supported in LEXICON. From left to right: BabyAI [6], Blocksworld [19], Logistics [30], Sokoban [12] and AlfWorld [43]. A constrained planning task is specified by an initial state, a goal, and a set of constraints to be respected.

11, 44]. In particular, LLM-based planning methods are often inefficient, lack formal guarantees, and incur high computational costs due to the generation of numerous "thinking tokens" [25, 51].

As LLMs are increasingly deployed in domains such as robotics [24, 29], travel planning [54], tool use [42], scientific discovery [55], and healthcare [45]—all of which demand planning and reasoning under constraints—it becomes crucial to rigorously assess their constrained planning capabilities. To this end, we make the following contributions.

1. **Extensible Benchmark.** We introduce LEXICON, an extensible NL-based benchmark for planning with temporal constraints specified on state-trajectories, which is publicly available[1]. It comprises two core components: a symbolic reasoning engine and a translator, which together enable the following functionalities.

   – **Constrained Problem Generation.** This module takes as input an unconstrained planning problem (described in a formal language) and introduces constraints to it, while making sure that it remains solvable. The reasoning engine generates ***task-aware*** **constraints** so as to complicate the original problem—resulting in longer solutions compared to its unconstrained version—while guaranteeing that constraints do not subsume one another, which would make them redundant. This leads to a challenging LLM planning benchmark. Crucially, the reasoning engine operates orders of magnitude faster than LLM-based planning, enabling scalable problem generation and evaluation. To interface with LLMs, the translator module converts formal planning problems with constraints into NL, leveraging the compositional structure of these problems to produce NL representations in a systematic manner.

   – **Automated Plan Verification.** The planning capabilities of LLMs are evaluated on the generated NL-representation of constrained planning problems. Subsequently, the reasoning engine automatically verifies whether the LLM-generated plans are correct and/or optimal.

2. **Experimental Evaluation.** We evaluated several state-of-the-art LLMs, including reasoning models like OpenAI o3 [35], DeepSeek R1 [9], Gemini 2.5 Pro [18], Claude 3.7 Sonnet [2], and GPT-5 [34], on benchmarks generated by LEXICON. Using constrained problems of increasing compositional complexity, we found that LLM performance consistently declines with the number of constraints, suggesting that current models do not yet match the performance of formal planning algorithms.

LEXICON supports five environments (Figure 1) and is designed to be extensible. We expect it to remain valuable even as more capable LLMs emerge. As LLMs improve, LEXICON can adapt by generating problems with increased constraint complexity or encorporating new environments, resulting in a flexible, future-proof benchmark that does not rely on static planning problems. Unless LLMs truly acquire algorithmic planning abilities—generalizing across problem instances like symbolic planners—we expect LEXICON to continue serving as an effective tool for assessing their planning capabilities.

[1] `https://github.com/Periklismant/lexicon_neurips`

Benchmark	Constraints	NL Interface	Automated Curation	Suite Extensibility	Environment Diversity
BabyAI [6]	✗	✓	✓	✓	✗
AlfWorld [43]	✗	✓	✓	✗	✗
PlanBench [49]	✗	✓	✓	✓	✓
ACPBench [27]	✗	✓	✓	✓	✓
BALROG [37]	✗	✓	✓	✓	✓
Safety Gym [41]	✓	✗	✓	✗	✗
TravelPlanner [54]	✓	✓	✗	✗	✗
Natural Plan [57]	✓	✓	✗	✗	✓
LEXICON	✓	✓	✓	✓	✓

Table 1: Comparison of simulation benchmarks. "Automated curation" indicates the ability to automatically generate new planning problem instances and verify solutions for those instances. "Suite Extensibility" requires that new planning domains can be added to the benchmark without rewriting its code. "Environment Diversity" indicates that the benchmark supports more than one type of planning domain (e.g., it is not restricted solely to 2D gridworld problems).

2 Related Work

Table 1 compares LEXICON with state-of-the-art planning benchmarks. Benchmarking the planning capabilities of LLMs requires an NL interface, which limits the applicability of traditional constrained environments such as Safety Gym [41] that lack NL support. While simulators such as BabyAI [6], gComm [21], and AlfWorld [43] support NL interaction, they do not model constraints and are limited to narrow domains (e.g., 2D grids or household settings). Constrained planning benchmarks like NaturalPlan [57] and TravelPlanner [54] also support NL, but their tasks are either manually curated or carefully constructed offline, resulting in limited extensibility. Additionally, verifying LLM-generated plans in these settings typically requires exhaustively enumerating all valid solutions, which is prohibitively expensive. In contrast, LEXICON supports the generation of a potentially unbounded number of constrained tasks and can automatically verify agent outputs using its reasoning engine. This enables rigorous, scalable evaluation without needing exhaustive (manual) plan enumeration.

While one might consider augmenting planning benchmarks such as PlanBench [49] or BALROG [37] with constraint-handling functionalities, these systems lack the infrastructure to synthesize, solve, and validate constrained tasks in an integrated manner. In contrast, LEXICON was built from the ground up to support automated constraint generation, enforcement, and verification. As LLMs continue to improve in their reasoning capabilities [22], LEXICON provides a principled platform for evaluating them on increasingly complex planning tasks with compositional constraints. Moreover, its reasoning engine is domain-agnostic, facilitating seamless extension to new environments (i.e., suite extensibility). In what follows, we illustrate the planning formalism in LEXICON, and describe its architecture.

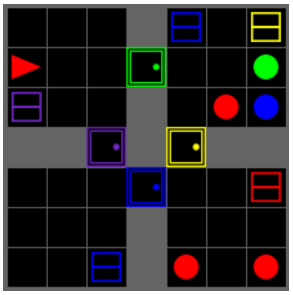
3 The LEXICON Simulator

3.1 Planning Specification Language

LEXICON supports planning problems expressed in PDDL3.0, an extension of the PDDL formal planning language that includes constraints [16].

Example 3.1: Constrained Planning in BabyAI

BabyAI contains problems where an agent needs to navigate the rooms of 2D gridworld, while interacting with objects, to complete some task [6]. Figure 2 (left) shows the initial state of a problem from BabyAI. BabyAI problems are grounded in PDDL; a domain file specifies the object types, the (time-varying) state atoms, and the actions of the domain, while a problem file denotes the objects of the puzzle, the initial state, the goal, and the constraints. In this case, the domain file defines atom `locked(d)`, expressing that door `d`



```
1. gotodoor purple_door_1 room_1 room_2
2. toggle purple_door_1
3. gotoobject purple_box_1 room_1
4. pick purple_box_1 room_1
5. gotoroom room_1 room_2 purple_door_1
6. gotodoor blue_door_1 room_2 room_4
7. toggle blue_door_1
8. gotoroom room_2 room_4 blue_door_1
9. gotodoor yellow_door_1 room_4 room_3
10. toggle yellow_door_1
11. gotoroom room_4 room_3 yellow_door_1
12. gotoempty
13. drop purple_box_1 room_3
14. gotoobject red_ball_1 room_3
```

```
1. gotodoor
   green_door_1 room_1
   room_3
2. toggle green_door_1
3. gotoroom room_1
   room_3 green_door_1
4. gotoobject
   red_ball_1 room_3
```

```
1. (:goal (and (exists (?v - ball) (and (objectcolor ?v red) (at ?v)))))
2. (:constraints (always (locked green_door_1))
3.               (sometime (agentinroom room_1))
4.               (sometime-after (agentinroom room_1) (objectinroom purple_box_1 room_3)))
```

Figure 2: Left: The initial state of the constrained planning problem in Example 3.1. The red triangle represents the agent. Bottom: The goal and the constraints of the problem in PDDL3.0. Middle: Optimal plan for this problem. Right: Optimal plan for the corresponding unconstrained problem.

is locked, and action `pick`, outlining the conditions for and the effects of picking up and holding an object. Figure 2 (bottom) outlines the goal and the constraints of the problem. The goal is to reach a red ball, while the constraints dictate that (i) the agent must never unlock the green door, (ii) at some point, the agent must visit room 1 (top-left room), and (iii) some time after visiting room 1, purple box 1 needs to be in room 3 (top-right room).

In LEXICON, we are interested in **optimal planning**, i.e., finding a plan that (1) reaches the goal while satisfying all constraints and (2) has minimum length. Optimal planning on the problem described in Example 3.1 is easy if the constraints are ignored—an optimal plan for the unconstrained problem consists of 4 actions (see Figure 2 (right)). However, the constrained version is significantly more challenging. For example, to satisfy the constraint that the green door must always remain locked, the agent must take a longer path through the purple and blue doors to reach the room containing the red ball—resulting in 14 actions (cf., Figure 2 (middle)).

3.2 The LEXICON Architecture

Figure 3(left) illustrates the architecture of LEXICON. The modules between the "Sampler" and the "Translator" implement the constrained problem generator functionality of LEXICON, while the "Verifier" module realises the automated plan verification functionality. We first outline constrained problem generation in PDDL, then its translation into natural language, and lastly our plan verifier.

Constrained Planning Problem Generator. We developed a constrained PDDL problem generator, extending the literature with a **task-aware** method for producing constraints for arbitrary PDDL problems. Its task is to generate constrained planning problems along with their optimal cost. The generator first samples an unconstrained problem using a domain file and a (unconstrained) state-goal pair generator, and then computes an optimal plan for the problem using the state-of-the-art planner `SymK` [47]. This plan, along with the unconstrained problem, is passed to LEXICON's constraint generator, which synthesizes task-aware constraints that (1) preserve feasibility, i.e., the problem still has a solution, and (2) increase the optimal cost relative to the unconstrained version.

Example 3.2: Constraint Generation in BabyAI

Consider the unconstrained plan in Figure 2. Using LEXICON, we can automatically construct an $\mathsf{Always}(\phi)$ constraint by analyzing the state transitions induced by this plan. The system samples domain atoms and evaluates their suitability for inclusion in ϕ based on problem complication, consistency, and non-redundancy. For example, atom `at(red_ball_2)` is excluded since it does not hold in the initial state and thus cannot "always" hold. Similarly, `objectInRoom(red_ball_1, room_3)` is not selected as it holds in all states of the unconstrained plan, thus offering no added difficulty. In contrast, `locked(green_door_1)`

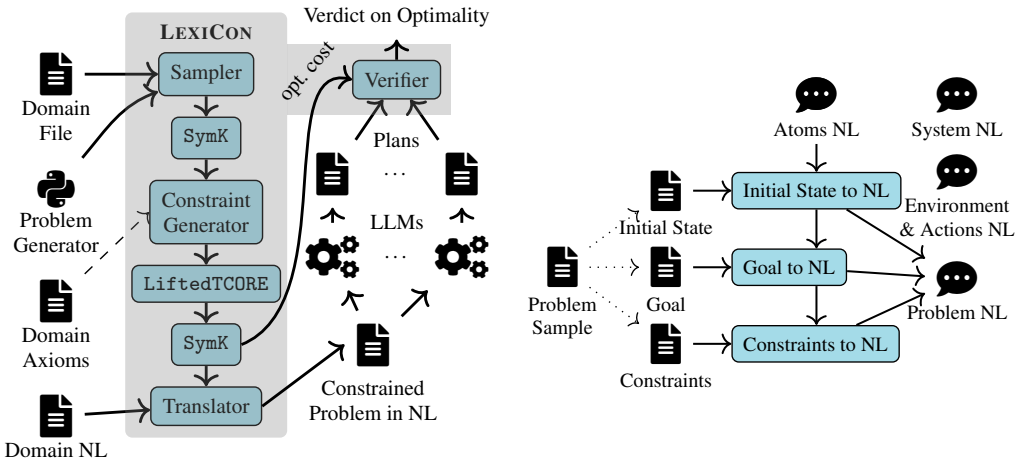


Figure 3: Left: The architecture of LEXICON. Solid arrows denote input/output data transfers. Dashed arrows denote optional input. Right: The translator of LEXICON. Dotted arrows express content extraction.

is included in ϕ, as enforcing it prevents use of the green door—forcing a detour through the purple and blue doors—which increases the plan's optimal cost (see Figure 3 (left)).

In LEXICON, users can optionally provide atemporal **domain axioms** to guide the constraint generator toward meaningful, non-conflicting constraints. For example, given the axiom $\forall d$: $\neg(\texttt{locked}(d) \wedge \texttt{unlocked}(d))$ and an existing constraint $\mathsf{Always}(\texttt{locked}(\texttt{green\_door\_1}))$, the generator avoids sampling $\mathsf{Sometime}(\texttt{unlocked}(\texttt{green\_door\_1}))$ since it would be unsatisfiable.

Next, we compute the optimal cost of the constrained problem generated by LEXICON, which is necessary to evaluate LLM outputs against ground-truth optimal plans. However, no existing planner supports constrained planning problems with actions that have conditional effects, which are often essential to specify certain domains, such as BabyAI. To overcome this, we compile the constraints away [53, 39, 5], producing an equivalent problem without constraints, which can be solved by SymK. LEXICON uses the TCORE compiler [4] for this translation. To avoid the cost of grounding, we apply a lifted variant of TCORE. Solving the compiled problem with SymK yields a formally verified optimal cost for the original constrained planning problem.

Our constrained problem generator is **compositional**, allowing users to control the number and complexity of constraints, enabling the generation of increasingly challenging benchmarks for future LLMs. It is also **extensible**: to support a new domain, users need only provide a PDDL domain file and an automated initial state–goal generator, avoiding manual problem construction. Domains can also be specified in Python via the Unified Planning framework [31], easing use for non-experts.

PDDL to Natural Language Translator. To evaluate LLMs on constrained problems, we first translate them into natural language (NL). As shown in Figure 3 (right), our translator extracts the instance-specific elements—initial state, goal, and constraints—and composes a problem prompt in NL. Since these instance-specific elements are built compositionally from domain atoms, their NL descriptions are generated by combining predefined NL templates for each atom. Domain-level descriptions (e.g., environment and action semantics) are carefully handcrafted per domain.

Example 3.3: NL Translation for BabyAI Problem

Consider the constrained planning problem in Example 3.1. Figure 4 (top) shows a fragment of the NL description generated for this problem by our translator. Lines 2–5 describe the initial state by listing NL descriptions of atoms that hold initially. The goal—"*reach a red ball*"—is represented by the logical formula $\exists \texttt{v}$: $\texttt{typeof}(\texttt{v}, \texttt{ball}) \wedge \texttt{objectColor}(\texttt{v}, \texttt{red}) \wedge \texttt{at}(\texttt{v})$, which we translate recursively into NL (lines 7–8). This involves mapping the quantifier to "*There is a ball v such that*", followed by "*The following*

```
1. The original state of the world is:
2.  'you are in room_1'
3.  'purple_box_1 is in room_1'
4.  'blue_box_1 is in room_2'
5.  <Description of the remaining atoms that hold initially>
6.
7. The task is to bring about the following situation:
8.  'There is a ball v such that 'The following conditions are all true: 'v is red', 'you
    are in front of v'''
9.
10. A valid plan for the abovementioned problem must abide by the following constraints:
11.  'The following expression must hold in every state: 'green_door_1 is locked''
12.  'The following expression must hold in at least one state: 'you are in room_1''
13.  'If expression 'you are in room_1' holds in some state s, then expression '
     purple_box_1 is in room_3' must hold at s or at some state after s'
```

```
1. Provided a planning problem, consisting of an initial state of the world, a final goal
   and a set of constraints, your task is to provide a valid sequence of actions that
   solves the planning problem, i.e., bringing about the goal of the problem while
   satisfying all constraints.
2. You need to provide an optimal plan, i.e., a valid plan whose length is equal or less
   than the length of any other valid plan.
```

Figure 4: Top: Fragment of our natural language description of the constrained problem of Example 3.1. Bottom: System role prompt.

conditions are all true", and then enumerating atom-level descriptions. Constraints are translated similarly using this recursive procedure (see lines 10–13).

Our translator is also **extensible**: to support a new planning domain, one only needs to provide (i) an NL description of the environment and actions, and (ii) NL descriptions for each atom. This eliminates the need for instance-specific NL annotations, allowing the translator to operate directly on any generated constrained problem within the domain.

Automated LLM Plan Verifier. With LEXICON's modules for generating constrained planning problems in NL in place, we now evaluate LLMs on these problems. Each LLM is given the NL description of a problem along with a fixed system role prompt (Figure 4 (bottom)), instructing it to act as an optimal planner. This prompt is used consistently across all domains.

To assess LLM outputs, LEXICON includes a verifier module (Figure 3 (left)) with three steps: (1) LLM-generated plans are mapped to PDDL actions using the prescribed output format; deviations are corrected by matching the LLM action to the closest domain action, according to the edit distance [33]. (2) The plan is validated using an automated plan validator on the compiled version of the constrained problem produced by `LiftedTCORE`, leveraging the guarantee that a plan valid for the compiled problem also satisfies the original constrained problem [4]. (3) If valid, the plan is checked for optimality by comparing its length to the optimal cost, which was computed at the problem generation phase.

A rigorous formulation of the constrained planning problem (i.e., with temporal constraints) along with how constrained plans are generated and verified through our reasoning engine is provided in Appendix A.

Figure 5 displays LLM-generated plans for the constrained problem in Example 3.1. The plan on the left was generated by o3; this plan is invalid because it violates the preconditions of the `pick` action, i.e., the agent attempts to pick up a purple box at a time when it is not facing that box (cf. line 1 of Figure 5 (left) and the starting state in Figure 2 (left)). The plan in the middle was suggested by Claude 3.7 Sonnet with extended thinking. This plan is invalid because the agent attempts to drop an object at a time when it is facing a door instead of an empty position, as required by the preconditions of the `drop` action (cf. lines 6 and 7 in Figure 5 (middle)). This type of error may be due to LLM state hallucination or loss of state tracking. The plan on the right was produced by R1. This plan ignores the constraint stipulating that the purple box must be placed in the top-right room, and is thus invalid. Next, we present a thorough evaluation of LLMs on benchmarks generated by LEXICON.

```
1. pick purple_box_1 room_1
2. gotodoor purple_door_1
   room_1 room_2
3. toggle purple_door_1
4. gotoroom room_1 room_2
   purple_door_1
5. gotodoor blue_door_1
   room_2 room_4
6. toggle blue_door_1
7. gotoroom room_2 room_4
   blue_door_1
8. gotodoor yellow_door_1
   room_4 room_3
9. toggle yellow_door_1
10. gotoroom room_4 room_3
    yellow_door_1
11. drop purple_box_1 room_3
12. gotoobject red_ball_1
    room_3
```

```
1. gotoobject purple_box_1
   room_1
2. pick purple_box_1 room_1
3. gotodoor purple_door_1
   room_1 room_2
4. toggle purple_door_1
5. gotoroom room_1 room_2
   purple_door_1
6. gotodoor blue_door_1
   room_2 room_4
7. drop purple_box_1 room_2
8. toggle blue_door_1
9. gotoroom room_2 room_4
   blue_door_1
10. gotoobject red_ball_2
    room_4
```

```
1. gotodoor purple_door_1
   room_1 room_2
2. toggle purple_door_1
3. gotoroom room_1 room_2
   purple_door_1
4. gotodoor blue_door_1
   room_2 room_4
5. toggle blue_door_1
6. gotoroom room_2 room_4
   blue_door_1
7. gotoobject red_ball_2
   room_4
```

Figure 5: Invalid plans suggested by LLMs for the constrained problem in Example 3.1.

4 LLM Evaluation on LEXICON

4.1 Evaluation Setup

Figure 1 displays the domains supported in LEXICON, which are:

- **BabyAI** [6]: an environment with minigrid problems, like our running example.
- **Blocksworld**: a puzzle where an agent rearranges blocks into a target configuration. Constraints may forbid placing certain blocks on the table or require specific sequences of block manipulations.
- **Logistics**: a world consisting of several locations, possibly including packages, trucks and airplanes, where the task is to move all packages to their designated destinations. Constraints may, e.g., forbid the usage of a specific truck or an airport.
- **Sokoban**: a gridworld where an agent has to move a collection of boxes onto target locations. Constraints may indicate, e.g., that a grid square must be occupied or cleared.
- **AlfWorld**: an environment for executing household task, like putting a book in a drawer, washing and slicing an apple, or turning on a lamp. Constraints may, e.g., prohibit the use of certain utensils, or impose a (partial) ordering among sub-tasks.

LLMs were tasked with optimal planning on constrained problems generated by LEXICON. Our experiments ran on a standard PC (Ubuntu 22, Ryzen 7 5700U, 16GB RAM), using each LLM's official API and the maximum allowed token limits for completions and reasoning. The LLM execution parameters for all our experiments are provided in Appendix C.

4.2 Evaluation Results

We evaluated 5 LLMs with thinking token generation capabilities, i.e., DeepSeek R1 [9], OpenAI o3 [35], Gemini-2.5 Pro [18], Claude 3.7 Sonnet (Extended Thinking) [2], and GPT-5 [34]. We also tested 4 LLMs that that do not support thinking capabilities, i.e., GPT 4.1 [36], DeepSeek V3 [8], Claude 3.7 Sonnet (no extended thinking), and Gemini 2.0 Pro [17], on benchmarks generated by LEXICON. Each environment consisted of 150 problems, with the number of constraints in $\{1, 3, 5, 7, 10\}$.

Can LLMs perform constrained planning? Figure 6 displays our results. All data points were produced over 30 executions. For the sake of comparison, we also included performance measurements on unconstrained problems. **LLMs without explicit thinking typically failed to produce an optimal—or even valid—plan for problems involving more than one constraint**. For this class of models, we only report the best-performing configuration, which was GPT-4.1 with Chain-of-Thought (CoT) prompting [52]. For the performance of the remaining models, see Appendix B. In contrast, reasoning models frequently succeeded in producing optimal plans for problems with a few constraints. However, **the capabilities of reasoning LLMs deteriorated sharply as the number of constraints increased**. In the majority of problems including 10 constraints, most LLMs failed to even produce a suboptimal plan. In the case of o3, e.g., in Blockworld, the optimal planning accuracy over constraint range $\{1, 3, 5, 7, 10\}$ was [76%, 30%, 26%, 10%, 0].

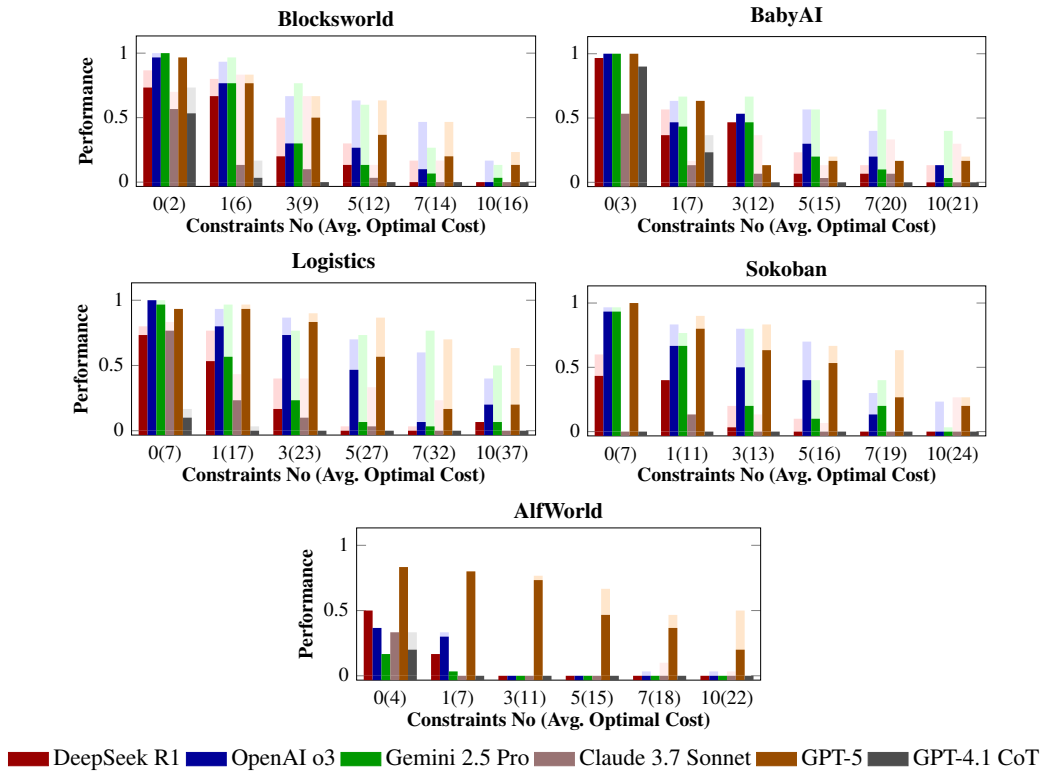


Figure 6: **Performance vs. number of constraints (average optimal cost)**. Performance denotes the percentage of problems solved with an optimal plan (colored bars) or with a valid, but possibly suboptimal plan (background faded-out bars).

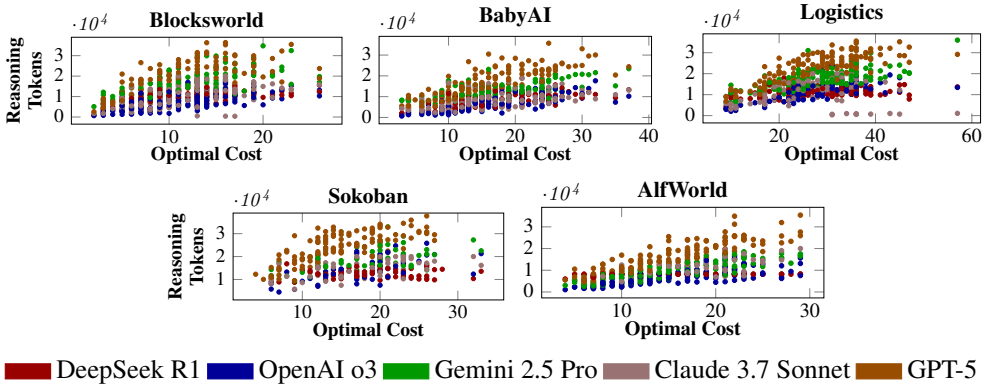
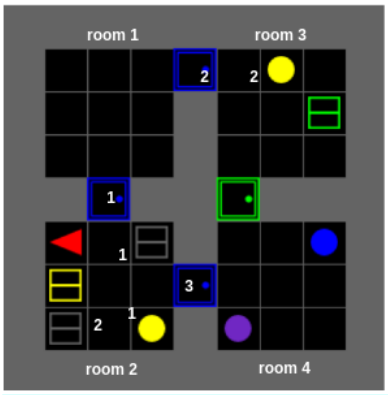


Figure 7: **Correlation between Reasoning Tokens and Optimal Cost**. The Pearson correlation coefficients are as follows. **Blocksworld:** [0.48, 0.6, 0.67, 0.66, 0.63]; **BabyAI:** [0.52, 0.68, 0.7, 0.74, 0.8]; **Logistics:** [0.16, 0.37, 0.73, 0.71, 0.77]; **Sokoban:** [0.53, 0.12, 0.57, 0.62, 0.72]; **AlfWorld:** [0.46, 0.74, 0.86, 0.55, 0.87].

Interestingly, as shown in Figure 7, we observe that the **number of thinking tokens increases with the optimal plan length** (i.e., optimal cost), suggesting that the reasoning models engage in deeper reasoning when the task demands it. However, as highlighted in Figure 5, their *soundness* declines for longer plans, due to: invalid actions from precondition violations (e.g., attempting to pick up a box without facing it), hallucinated states (e.g., perceiving an empty space where a door exists), misinterpreted constraints, and loss of state tracking.



Goal: agent at yellow ball

Constraints:

1. at some state, must hold some object
2. at some state, agent at blue door 3
3. at some state, agent in room 3 and purple ball 1 in room 3
4. at some state, either at yellow box 1 or grey box 1 in room 4
5. at some state, agent in room 1
6. at some state, agent at grey box 1 and green door 1 is unlocked

Unconstrained Plan: The yellow ball 1 is in room 2. So maybe I can just go to that ball.

Bilevel Planning.
Subgoal Generation: To satisfy constraint 4, I must carry purple ball from room 4 to room 3.
Subgoal Planning: How to get to room 4? Through blue door 3, which is locked. So first, unlock blue door 3. But to unlock blue door 3, I need to be in front of it. So I can do gotodoor blue door 3. Then toggle it.

Backtracking: Now in room 4. Need to get purple ball 1 to room 3. unlock green door 1, gotodoor green door 1, toggle green door 1, gotoroom room 3. Drop purple ball 1 here. **But wait,** we need to pick up purple ball 1 first. So in room 4: gotoobject purple ball 1 in room 4; pick purple ball 1 from room 4. Then carry it to room 3. So after entering room 4, pick the ball, then go to green door 1, unlock it, go to room 3, then drop it.

Self Reflection: need to hold an object at some point (constraint 1). So picking up purple_ball_1 satisfies that. **But then, after dropping it, we are not holding anything. So need to pick something else? Or maybe just once is enough.**

Bilevel Planning:
1. To satisfy constraint 5, I must go to room_1
gotodoor blue door 2 between room 3 and room 1; toggle blue door 2 (unlock); gotoroom room 1 (now in room 1). That satisfies constraint 5. ...
2. Now, need to go back to room 2 to handle other constraints.
gotodoor blue door 1 b/w room 1 and room 2; toggle blue door 1; gotoroom room 2 (now in room 2)

Lookahead Search: gotoobject yellow box 1 in room_2 (satisfies constraint 4's first condition)
Alternatively, maybe moving grey box 1 to room 4 would also satisfy constraint 4. Let's see: If we pick up grey box 1 in room 2, carry it to room 4, and drop it there. Then grey box 1 would be in room 4, satisfying constraint 4. **But that would require additional steps.**

Self Reflection: Now, is this the optimal plan? Let's count the steps. The current plan is 19 steps. Maybe there's a shorter way.
Alternative approach: after unlocking blue_door 3 and green door 1, maybe there's a way to combine steps. Let's see.
Alternatively, after step 10 (dropping purple ball 1 in room 3), we can go to room 4 via green door 1, but that's not needed. Wait, maybe after dropping, we can go to room 4 again, but that's unnecessary.
Alternatively, after step 10, go to room 4, pick grey box 1 and move it. But that's more steps.

Figure 8: **Planning Traces from R1**. The white box displays the goal and constraints. On the left is the initial observation from the BabyAI environment. Colored boxes indicate model behaviours: cyan for bilevel planning, green for self-reflection, yellow for lookahead search, and violet backtracking. We also show the 1-step unconstrained plan generated by R1 for the same goal in salmon - - -.

Do LLMs show structured planning behaviour? We qualitatively analysed the reasoning traces of R1 by annotating its "thinking" steps and mapping them to classical planning strategies. (The reasoning traces of the LLMs that performed better on our benchmark were not available to us.) Figure 8 shows a representative example. We observed the following behaviours:

- **Bilevel Planning**. R1 decomposes the goal into high-level subgoals and performs subgoal planning.
- **Backtracking**. R1 can backtrack and generate a more optimized subgoal plan (e.g, instead of having to go back to pick an object, carry it with you).
- **Lookahead Search**. R1 generates multiple rollout paths and selects the optimal action.
- **Self-Reflection**. R1 frequently re-evaluates the state and the selected actions, checking constraint satisfaction and exploring alternatives, towards optimising the plan.

Despite these interesting behaviours, **R1 does not show the structured search behaviour necessary for optimal planning**, like maintaining different search paths simultaneously. Instead, it tries to generate a valid plan that satisfies all constraints, and subsequently attempts to shorten that plan, towards finding an optimal one; this is not guaranteed to work for arbitrary planning problems.

Can LEXICON enable real-time evaluation of LLM Planners? A key advantage of LEXICON is its ability to generate arbitrary constrained planning problems on demand. This allows for on-the-fly evaluation of LLMs on problems of varying complexity, rather than relying on static, offline

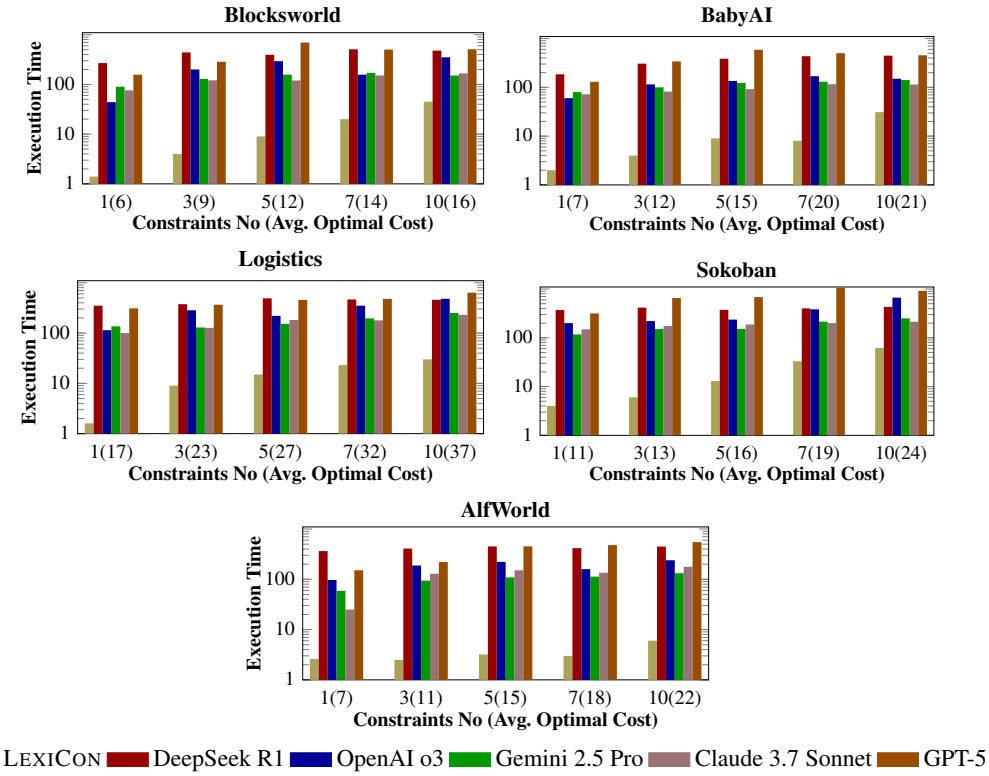


Figure 9: **Execution Time vs. number of constraints (average optimal cost)**. The vertical axes show execution time in seconds. Standard deviations are small, and thus omitted.

benchmarks. In our setup, LEXICON generates a new problem while the LLM is still solving the previous one, enabling a seamless and adaptive evaluation pipeline. To test this, we compared the average time LEXICON takes to generate and verify a problem (in natural language and PDDL) with the average time an LLM takes to solve it[2]. As shown in Figure 9, **LEXICON is roughly one order of magnitude faster than LLM-based planning, making real-time evaluation feasible**.

5 Summary & Future Work

We proposed LEXICON, an extensible NL-based benchmark generator for planning under temporal constraints. Our generator is able to produce task-aware constraints for an arbitrary planning problem and verify solutions suggested by LLMs at scale. Our experiments showed that there is a limit of problem constrainedness that LLMs cannot cope with, even for models with reasoning capabilities. We aim to extend LEXICON with partially-observable environments and uncertain observations, as well as a wider class of constraints, including constraints on actions and on continuous states, paving the way for evaluating language-based agents on real reinforcement learning settings.

6 Limitations

Our simulator does not support parallel episode execution, unlike standard RL environments such as MuJoCo [46] or Atari [3], which can be parallelized using tools like `AsyncVectorEnv` (Gymnasium) [48] or `SubprocVecEnv` (Stable-Baselines3) [40]. In our case, multiprocessing is fully utilized for backend tasks such as generating feasible episodes. Furthermore, episode generation is significantly slower ($[1, 100]$ s) due to the complexity of constraint satisfaction and simulation, limiting scalability compared to environments that support fast, parallel rollouts.

[2]Note that LLM solve time also depends on API latency, though LEXICON remains significantly faster.

Acknowledgments and Disclosure of Funding

This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. This work was also supported by the Research Foundation - Flanders (FWO) under contract no G097720N.

References

[1] E. Altman. Constrained markov decision processes. Technical Report RR-2574, INRIA, 1995.

[2] Anthropic. Claude 3.7 sonnet and claude code. 2025. URL `https://www.anthropic.com/news/claude-3-7-sonnet`.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

[4] L. Bonassi, A. E. Gerevini, F. Percassi, and E. Scala. On planning with qualitative state-trajectory constraints in PDDL3 by compiling them away. In *ICAPS*, pages 46–50, 2021.

[5] L. Bonassi, A. E. Gerevini, and E. Scala. Dealing with numeric and metric time constraints in PDDL3 via compilation to numeric planning. In *AAAI*, pages 20036–20043, 2024.

[6] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *ICLR*, volume 105, 2019.

[7] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[8] DeepSeek-AI. Deepseek-v3 technical report, 2024. URL `https://arxiv.org/abs/2412.19437`.

[9] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *CoRL*, pages 1–16, 2017.

[11] N. Dziri, X. Lu, M. Sclar, X. L. Li, L. Jiang, B. Y. Lin, S. Welleck, P. West, C. Bhagavatula, R. L. Bras, J. D. Hwang, S. Sanyal, X. Ren, A. Ettinger, Z. Harchaoui, and Y. Choi. Faith and fate: Limits of transformers on compositionality. In *NeurIPS*, 2023.

[12] D. Feng, C. P. Gomes, and B. Selman. A novel automated curriculum strategy to solve hard sokoban planning instances. In *NeurIPS*, 2020.

[13] R. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[14] J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16:1437–1480, 2015.

[15] A. Gattami, Q. Bai, and V. Aggarwal. Reinforcement learning for constrained markov decision processes. In *AISTATS*, pages 2656–2664, 2021.

[16] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6):619–668, 2009.

[17] Google. Introducing gemini 2.0: Our new ai model for the agentic era. 2024. URL `https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/`.

[18] Google DeepMind. Gemini 2.5: Our newest gemini model with thinking, 2025. URL `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking`.

[19] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artif. Intell.*, 56(2-3): 223–254, 1992.

[20] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. ISBN 978-3-031-00456-8.

[21] R. Hazra and S. Dixit. gcomm: An environment for investigating generalization in grounded language acquisition, 2021. URL `https://arxiv.org/abs/2105.03943`.

[22] R. Hazra, G. Venturato, P. Z. D. Martires, and L. D. Raedt. Have large language models learned to reason? a characterization via 3-sat phase transition. In *COLM*, 2025.

[23] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*, volume 162, pages 9118–9147, 2022.

[24] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu. Do as I can, not as I say: Grounding language in robotic affordances. In *CoRL*, volume 205, pages 287–318, 2022.

[25] M. Katz, H. Kokel, K. Srinivas, and S. Sohrabi. Thought of search: Planning with language models through the lens of efficiency. In *NeurIPS*, 2024.

[26] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. In *NeurIPS*, 2022.

[27] H. Kokel, M. Katz, K. Srinivas, and S. Sohrabi. ACPBench: Reasoning about action, change, and planning. In *AAAI*, 2025.

[28] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. AI safety gridworlds, 2017. URL `https://arxiv.org/abs/1711.09883`.

[29] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: from natural language instructions to feasible plans. *Auton. Robots*, 47(8):1345–1365, 2023.

[30] D. V. McDermott. The 1998 AI planning systems competition. *AI Mag.*, 21(2):35–55, 2000.

[31] A. Micheli, A. Bit-Monnot, G. Röger, E. Scala, A. Valentini, L. Framba, A. Rovetta, A. Trapasso, L. Bonassi, A. E. Gerevini, L. Iocchi, F. Ingrand, U. Köckemann, F. Patrizi, A. Saetti, I. Serina, and S. Stock. Unified planning: Modeling, manipulating and solving AI planning problems in python. *SoftwareX*, 29:102012, 2025.

[32] A. Nareyek, E. C. Freuder, R. Fourer, E. Giunchiglia, R. P. Goldman, H. Kautz, J. Rintanen, and A. Tate. Constraints and ai planning. *IEEE Intelligent Systems*, 20(2):62–72, 2005.

[33] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[34] OpenAI. Gpt-5 is here, 2025. URL `https://openai.com/gpt-5/`.

[35] OpenAI. Introducing openai o3 and o4-mini, 2025. URL `https://openai.com/index/introducing-o3-and-o4-mini/`.

[36] OpenAI. Introducing gpt-4.1 in the api, 2025. URL `https://openai.com/index/gpt-4-1/`.

[37] D. Paglieri, B. Cupiał, S. Coward, U. Piterbarg, M. Wolczyk, A. Khan, E. Pignatelli, Ł. Kuciński, L. Pinto, R. Fergus, J. N. Foerster, J. Parker-Holder, and T. Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *ICLR*, 2025.

[38] E. P. D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *KR*, pages 324–332, 1989.

[39] F. Percassi and A. E. Gerevini. On compiling away PDDL3 soft trajectory constraints without using automata. In *ICAPS*, pages 320–328, 2019.

[40] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22(268):1–8, 2021.

[41] A. Ray, J. Achiam, and D. Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. Technical report, OpenAI, 2019.

[42] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

[43] M. Shridhar, X. Yuan, M.-A. Cote, Y. Bisk, A. Trischler, and M. Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*, 2021.

[44] K. Stechly, K. Valmeekam, and S. Kambhampati. Chain of thoughtlessness? an analysis of cot in planning. In *NeurIPS*, 2024.

[45] K. Swanson, W. Wu, N. L. Bulaong, J. E. Pak, and J. Zou. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*, 2024. URL `https://www.biorxiv.org/content/early/2024/11/12/2024.11.11.623004`.

[46] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[47] Á. Torralba, V. Alcázar, P. Kissmann, and S. Edelkamp. Efficient symbolic search for cost-optimal planning. *Artif. Intell.*, 242:52–79, 2017.

[48] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL `https://arxiv.org/abs/2407.17032`.

[49] K. Valmeekam, M. Marquez, A. O. Hernandez, S. Sreedharan, and S. Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *NeurIPS*, 2023.

[50] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati. On the planning abilities of large language models - A critical investigation. In *NeurIPS*, 2023.

[51] K. Valmeekam, K. Stechly, A. Gundawar, and S. Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of lrm o1, 2024. URL `https://arxiv.org/abs/2410.02162`.

[52] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, volume 35, pages 24824–24837, 2022.

[53] B. Wright, R. Mattmüller, and B. Nebel. Compiling away soft trajectory constraints in planning. In M. Thielscher, F. Toni, and F. Wolter, editors, *KR*, pages 474–483, 2018.

[54] J. Xie, K. Zhang, J. Chen, T. Zhu, R. Lou, Y. Tian, Y. Xiao, and Y. Su. TravelPlanner: A benchmark for real-world planning with language agents. In *ICML*, volume 235, pages 54590–54613, 2024.

[55] Y. Yamada, R. T. Lange, C. Lu, S. Hu, C. Lu, J. Foerster, J. Clune, and D. Ha. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, 2025. URL `https://arxiv.org/abs/2504.08066`.

[56] W.-C. Yang, G. Marra, G. Rens, and L. De Raedt. Safe reinforcement learning via probabilistic logic shields. In *IJCAI*, 2023.

[57] H. S. Zheng, S. Mishra, H. Zhang, X. Chen, M. Chen, A. Nova, L. Hou, H.-T. Cheng, Q. V. Le, E. H. Chi, and D. Zhou. Natural plan: Benchmarking llms on natural language planning, 2024. URL https://arxiv.org/abs/2406.04520.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: LEXICON's reasoning engine (including the plan generator and verifier) are described in Section 3. We also show how LEXICON can be easily extended to other environments in Section 3.2. Through our experiments on a range of environments (Section 4.1) and LLMs, we show that LLM including that of reasoning models struggle with constrained planning (Section 4.2). Furthermore, as part of this submission, we provide a dataset for evaluation generated from LEXICON, along with the environment source files.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: We do not any theoretical claims in the paper.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the data and the code we used in our submission, and reproducibility instructions in the Appendix C. We also provide the LLM parameters in Appendix C.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide open access data and code URLs in our submission. Reproducibility instructions are provided in the Appendix C.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the parameters, e.g., number of constraints, we used for problem generation. The specifics of the planning domains used, the technical details of the simulation, and the LLM hyperparameters used for planning are described in the Appendix A.1, C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Error bars do not apply in Figures 6 and 7 as we do not present average values of individual runs. For Figure 7, we show statistical significance via Pearson correlations. In Figure 9, we omit error bars because they are small.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the specifications of the PC that runs our simulator (Section 4.1). The LLMs are run through their APIs with their maximum allowed token limits. Execution times are recorded (Figure 9).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: No ethics issues.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: While there is no direct societal impact, if constrained planning is see from a safety perspective, our work can be seen as way to evaluate whether LLM-based agents can adhere to safety constraints.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We use simulators from Classical AI domains that are publicly available. These are simple toy-based setups for evaluating decision-making agents.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We provide citations for the planning domains we employed and the models we evaluated. The licenses of the modules used as components in our simulator, like the off-the-self planner, permit our usage.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide documented code with execution scripts and examples in our code submission, and a technical description of our system in the technical appendix (Appendix A.1).

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: No crowdsourcing or research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: While our simulator does not use LLMs, we evaluate LLM planning abilities using episodes/data generated from our simulator. All the LLMs have been adequately cited (Section 4.1), their evaluation results discussed (Section 4.2), and their generation parameters provide for reproducibility (Appendix C.1)

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendix

This document contains supplementary material for our paper, along with code execution and reproducibility instructions for our experiments. Its structure is the following. Appendix A describes LEXICON's reasoning engine. Appendix B exemplifies indicative errors in LLM planning, while also highlighting some experimental results that were omitted from the paper due to space limitations. Appendix C provides the hyperparameters set for each LLM in our experiments, along with the steps for running our code and reproducing our experiments.

A Reasoning Engine

First, we specify the class of planning problems that may be generated and have candidate solutions verified by LEXICON's reasoning engine. Subsequently, we describe its two main modules: the constraint planning problem generator and the automated LLM plan verifier.

A.1 Class of Planning Problems in LEXICON

LEXICON may generate and verify candidate solutions for planning domains expressed in a PDDL fragment that includes the following syntactic components.

- Basic STRIPS, i.e., actions with conjunctive preconditions, and atom addition and deletion effects [13].
- ADL, i.e., equalities, actions with negated, disjunctive and quantified preconditions, as well as conditional and universally quantified effects [38].
- The qualitative state-trajectory constraints found in PDDL3.0 [16].

We formulate this fragment of PDDL, loosely following [4] for the notation, and using the term "constraint" to refer to a qualitative state-trajectory constraint of PDDL3.0 for brevity.

A constrained planning problem is a tuple $\Pi^C=(F, A, I, G, C)$, where F is a set of atoms, A is a set of actions, $I \subseteq F$ is an initial state, G is a formula over F denoting the goal of the problem, and C is a set of constraints. Each action $a \in A$ comprises a precondition $Pre(a)$, which is a formula over F, and a set of conditional effects $\mathit{Eff}(a)$. Each conditional effect in $\mathit{Eff}(a)$ is an expression $c \rhd e$, where c is a formula and e is a set of literals, both constructed based on the atoms in F. We use e^+ (resp. e^-) to denote the positive (negative) literals in e. A state $s \subseteq F$ contains the atoms that are true in s. An action a is applicable in state s if $s \models Pre(a)$, and its application yields state $s'=(s \setminus \bigcup_{c \rhd e \in \mathit{Eff}(a): s \models c} e^-) \cup \bigcup_{c \rhd e \in \mathit{Eff}(a): s \models c} e^+$, which we often denote with $s'=s[a]$.

LEXICON supports the following types of constraints: Always, Sometime, AtMostOnce, SometimeBefore and SometimeAfter. Considering grounded formulas ϕ and ψ over F in negation normal form, and a sequence of states σ over F, these constraint types are defined as follows:

- $\sigma \models \mathsf{Always}(\phi)$ ($\mathsf{A}(\phi)$) iff $\forall s \in \sigma$: $s \models \phi$.
- $\sigma \models \mathsf{Sometime}(\phi)$ ($\mathsf{S}(\phi)$) iff $\exists s \in \sigma$: $s \models \phi$.
- $\sigma \models \mathsf{AtMostOnce}(\phi)$ ($\mathsf{AO}(\phi)$) iff ϕ is true in at most one continuous subsequence of σ.
- $\sigma \models \mathsf{SometimeBefore}(\phi, \psi)$ ($\mathsf{SB}(\phi, \psi)$) requires that, if $\exists s \in \sigma : s \models \phi$, then there is a state s' before s in σ, such that $s' \models \psi$.
- $\sigma \models \mathsf{SometimeAfter}(\phi, \psi)$ ($\mathsf{SA}(\phi, \psi)$) requires that, if $\exists s \in \sigma : s \models \phi$, then $s \models \psi$ or there is a state s' after s in σ such that $s' \models \psi$.

Given a constrained planning problem $\Pi^C=(F, A, I, G, C)$, a plan π for Π^C is a sequence of actions $(a_0, \ldots, a_{n-1})$ from set A. π is a valid plan for Π^C iff there exists a sequence of states $\sigma=(s_0, \ldots, s_n)$ such that $s_0=I$, $\forall i \in \{0, \ldots, n-1\}$ we have $s_i \models Pre(a_i)$ and $s_{i+1}=s_i[a_i]$, $s_n \models G$, and $\forall q \in C$ we have $\sigma \models q$. We define the cost of a plan as the number of actions it includes. An optimal plan π^* for a problem Π^C is a valid plan whose cost is minimal among all valid plans for Π^C, i.e., there is no valid plan for Π^C that has a lower cost than π^*.

A.2 Constrained Planning Problem Generator

We focus on the "Constraint Generator" of LEXICON, as the remaining modules of our reasoning engine that are used for problem generation are off-the-shelf planners and compilers (see Figure 3). Our constraint generator receives as input an unconstrained PDDL problem $\Pi=(F, A, I, G)$ and an optimal plan π^* for Π, and outputs a constrained PDDL problem $\Pi^C=(F, A, I, G, C)$. The challenge here is to construct constraint set C in an informed manner, considering problem Π and plan π^*. In particular, we may add a constraint q in C only if q is a meaningful constraint given Π and π^*, i.e., the inclusion of q makes π^* an invalid plan for Π, potentially complicating the planning problem, while maintaining problem solvability and being non-redundant with respect to the constraints that were previously added in C.

In order to produce such a meaningful constraint q, we proceed as follows.

1. We identify a set of conditions under which a literal is not suitable for inclusion in q, in the sense that its inclusion potentially results in q not being meaningful for the problem.
2. We sample literals that do not satisfy the conditions identified in the previous step, and consider whether they should be included in q. For each sampled literal l, we verify that it is consistent with, and not subsumed by, the literals that were previously added in q, taking into account a (possibly empty) set of domain axioms. If this is the case, then we add l in q. We continue this process until q has reached a specified degree of compositionality, which may be controlled by the user.
3. We verify that the generated constraint q is consistent with, and not subsumed by, the constraints that were previously added in C, in which case we add q in C.

We continue this process until the size of C has reached the number of constraints requested by the user.

Algorithm 1 Always Constraint Generator

Require: State changes σ induced by executing plan π^*, unconstrained problem $\Pi=(F, A, I, G)$, constraint set so far C, domain axioms D, possible user parameter values cfg
Ensure: New constraint set $C \cup \{q\}$
1: $op, l\_no \leftarrow sample\_parameters(cfg)$, $literals \leftarrow \emptyset$
2: **for** l_no iterations **do**
3: $\quad l \leftarrow sample\_literal(F)$
4: $\quad$ **if** $l \rightarrow G$ **or** $G \rightarrow \neg l$ **or not** $(I \models l)$ **or** $\forall s \in \sigma : s \models l$ **then goto** 3
5: $\quad$ **for** l' **in** $literals$ **do**
6: $\quad\quad$ **if** $l{=}l'$ **or** $(op{=}\wedge$ **and** $D \models \neg(l \wedge l'))$ **then goto** 3
7: $\quad literals.\mathsf{append}(l)$
8: **if** $op{=}\wedge$ **then** $\phi \leftarrow \bigwedge_{l \in literals} l$ **else** $\phi \leftarrow \bigvee_{l \in literals} l$
9: $q \leftarrow \mathsf{A}(\phi)$
10: **for** $q' \in C$ **do**
11: $\quad$ **if** $q'{=}\mathsf{A}(\phi')$ **and** $(D \models (\phi \rightarrow \phi')$ **or** $D \models (\phi' \rightarrow \phi)$ **or** $D \models \neg(\phi \wedge \phi'))$ **then goto** 1
12: $\quad$ **else if** $q'{=}\mathsf{S}(\phi')$ **and** $(D \models (\phi \rightarrow \phi')$ **or** $D \models \neg(\phi \wedge \phi'))$ **then goto** 1
13: $\quad$ **else if** $q'{=}\mathsf{AO}(\phi')$ **and** $(D \models (\phi \rightarrow \phi')$ **or** $D \models \neg(\phi \wedge \phi'))$ **then goto** 1
14: $\quad$ **else if** $q'{=}\mathsf{SB}(\phi', \psi')$ **and** $(D \models (\phi \rightarrow \phi')$ **or** $D \models (\phi \rightarrow \psi')$
15: $\quad\quad\quad\quad\quad\quad\quad\quad$ **or** $D \models \neg(\phi \wedge \phi')$ **or** $D \models \neg(\phi \wedge \psi'))$ **then goto** 1
16: $\quad$ **else if** $q'{=}\mathsf{SA}(\phi', \psi')$ **and** $(D \models (\phi \rightarrow \psi')$ **or** $D \models \neg(\phi \wedge \phi')$ **or** $D \models \neg(\phi \wedge \psi'))$ **then**
17: $\quad\quad$ **goto** 1
18: **return** $C \cup \{q\}$

The above procedure for generating a constraint is adapted for each possible type of constraint. Constraint consistency and subsumption, e.g., is defined differently for each constraint type. As an example, Algorithm 1 outlines the procedure for constructing an Always constraint $\mathsf{A}(\phi)$. We start by sampling a Boolean operation op and a number of literals l_no for ϕ, taking into account the parameters that are optionally provided by the user (see line 1 of Algorihtm 1). Subsequently, we generate l_no literals that are suitable for inclusion in ϕ (lines 2–7). We sample a literal l based on the atoms of the problem F (line 3), and then evaluate a set of conditions such that, if l satisfies one

of them, then including l in ϕ is not meaningful with respect to constraint $\mathsf{A}(\phi)$. For instance, it is not meaningful to include l in $\mathsf{A}(\phi)$ if (i) l implies the goal G, as, in non-trivial problems where $I \not\models G$, $l \rightarrow G$ implies that l does not hold in the initial state I, and thus cannot "always" hold; (ii) G implies $\neg l$, because then l cannot hold in the final state of a plan that brings about G; (iii) if l does not hold in I; or (iv) if l is satisfied in every state in the sequence σ induced by executing plan π^*, as, in that case, $\mathsf{A}(l)$ is satisfied by optimal plan π^* of the unconstrained problem, and thus adding l in $\mathsf{A}(\phi)$ may not lead to a more complicated problem. If any of the above conditions holds, then we drop l and sample another literal for our constraint (line 4). Additionally, we resort to resampling if l has already been added to q in a previous step, or the selected operation op is a conjunction and l is inconsistent with some other literal l' in q, taking into account a (possibly empty) set of atemporal domain axioms (see lines 5–6). If none of the above conditions is satisfied, then we add l to the set of literals that will be used to construct q (line 7).

After identifying l_no literals that are suitable for constraint $\mathsf{A}(\phi)$, we construct ϕ and $\mathsf{A}(\phi)$ using the sampled operation op (see lines 8–9 of Algorithm 1). Next, we need to verify whether $\mathsf{A}(\phi)$ is inconsistent or redundant with respect to the constraints that are already present in C. To do this, for each constraint q' in C, we check if $\mathsf{A}(\phi)$ is compatible with q' (lines 10–17). If $\mathsf{A}(\phi)$ is not compatible with some constraint in C, then we drop $\mathsf{A}(\phi)$ and generate another constraint. Otherwise, if $\mathsf{A}(\phi)$ is compatible with every constraint in C, then we add it in C (line 18). For example, q=$\mathsf{A}(\phi)$ is compatible with a Sometime constraint q'=$\mathsf{S}(\phi')$ if, according to the atemporal domain axioms, (i) ϕ does not imply ϕ', as $\phi \rightarrow \phi'$ would imply that constraint $\mathsf{S}(\phi')$ is redundant given $\mathsf{A}(\phi)$; and (ii) ϕ and ϕ' are consistent, because if they were inconsistent, it would be impossible to satisfy $\mathsf{A}(\phi)$ given that $\mathsf{S}(\phi')$ holds, i.e., in the case where ϕ' is true in at least one state of a valid plan.

A.3 Automated LLM Plan Verifier

Algorithm 2 LLM Plan Verifier

Require: LLM plan π_{NL}, compiled problem Π^{cm}, optimal cost c^*
Ensure: Plan validation outcome: Invalid, Suboptimal or Optimal
1: $s \leftarrow I$
2: **for** $a_{NL} \in \pi_{NL}$ **do**
3: **if** $pddl\_format(a_{NL})$ **then** $a \leftarrow extract\_action(a_{NL})$
4: **else** $a \leftarrow closest\_action\_edit\_distance(a_{NL}, \Pi)$
5: $s \leftarrow simulate\_action(a, s, \Pi)$
6: **if** s=None **then return** Invalid
7: **if** $s \not\models G$ **then return** Invalid
8: **if** $length(\pi_{NL}) > c^*$ **then return** Suboptimal
9: **return** Optimal

Algorithm 2 outlines LEXICON's automated LLM plan verifier. Its input is an LLM-generated plan π_{NL}, a PDDL problem Π^{cm} that has been compiled with `LiftedTCORE` and the optimal cost c^* of Π^{cm} that was discovered by LEXICON during constrained problem generation. (Recall that a plan that is valid for the compiled version of a problem satisfies all the constraints in the original problem.) Given this input, Algorithm 2 reports whether π_{NL} is an invalid plan, a valid but suboptimal plan, or a valid, optimal plan for problem Π^{cm}. This is achieved in two steps: (i) simulating plan π_{NL} over Π^{cm} to verify its validity, and, if π_{NL} is valid, (ii) comparing the length of π_{NL} to the optimal cost c^* of Π^{cm} in order to check whether π_{NL} is optimal.

To initiate the simulation of plan π_{NL} over problem Π^{cm}, we set variable s, tracking the state of the problem, to the initial state I of Π^{cm} (line 1 of Algorithm 2), and iterate over the actions in π_{NL}, in order to sequentially simulate the effects of each one over Π^{cm} (line 2). The prompt we use for LLM plan generation requests a specific format for LLM actions, so that they can be mapped directly to domain actiin in PDDL (line 3). In practice, however, LLM actions may deviate from this format; we handle such cases by mapping the LLM-generated action a_{NL} to the PDDL domain action yielding the shortest edit distance from a_{NL} (line 4). Both cases map a_{NL} to a PDDL domain action a, which we apply on the current state s of our plan simulation (line 5). If the application of a does not lead to a new state, then we deduce that either the preconditions of a are not met in state s, or that the application of a over state s led to the violation of a constraint of the original problem. Thus, in this

case, we deduce that plan π_{NL} is invalid. If the simulation of all LLM-generated actions over Π^{cm} succeeds, then we check whether the goal of the problem is satisfied in the final state s reached in the simulation. If the goal is not satisfied in s, then plan π_{NL} is invalid (line 7). Otherwise, if the goal is satisfied in s, then π_{NL} is valid, and we proceed with checking whether π_{NL} is optimal or not. If the length of π_{NL} is greater than c^*, i.e., the optimal cost of Π^{cm}, then plan π_{NL} is suboptimal (line 8). Otherwise, the length of π_{NL} is equal to c^*, and thus π_{NL} is an optimal plan for the problem (line 9).

B Additional Results

B.1 Performance of Non-Reasoning LLMs

Model	Blocksworld		BabyAI		Logistics		Sokoban		AlfWorld	
	Opt.	Val.	Opt.	Val.	Opt.	Val.	Opt.	Val.	Opt.	Val.
DeepSeek V3	0	7%	9%	17%	0	5%	0	0	15%	21%
Claude 3.7 Sonnet (no thinking)	0	0	12%	20%	5%	5%	0	0	3%	12
Gemini 2.0 Pro	0	7%	0	0	0	0	0	0	0	0
OpenAI GPT-4.1	3%	16%	17%	26%	0	3%	0	0	0	0

Table 2: Performance of non-reasoning LLMs on problems with one constraint. We measured the percentage of problems solved with an optimal plan (Opt.) and the percentage of problems solved with a valid, but possibly suboptimal, plan (Val.).

We complement the experimental results in Figure 6 of the paper with the performance of LLMs that do not use explicit thinking, i.e., DeepSeek V3, Claude 3.7 Sonnet (no extended thinking) and Gemini 2.0 Pro, on constrained planning problems. Table 2 displays their performance on each domain, in terms of plan optimality and plan validity, on problems that included one constraint. None of these models was able to produce an optimal plan for a problem from our benchmark that included more than one constraint.

B.2 LLM Action Format Compliance

Model	Blocksworld			BabyAI			Logistics			Sokoban			AlfWorld		
	1	5	10	1	5	10	1	5	10	1	5	10			
DeepSeek R1	0	50%	47%	6%	10%	23%	3%	7%	10%	3%	3%	3%	6%	0	0
OpenAI o3	0	0	0	0	3%	3%	0	0	3%	0	0	3%	0	0	3%
Gemini 2.5 Pro	0	0	0	0	0	0	0	3%	3%	0	0	0	70%	78%	81%
Claude 3.7 Sonnet (with thinking)	27%	33%	47%	10%	12%	27%	33%	42%	60%	36%	45%	96%	15%	33%	39%
GPT-5	3%	3%	3%	0	0	0	3%	0	0	0	0	0	0	3%	0

Table 3: Percentage of LLM-generated plans that could not be mapped directly into PDDL for problems with 1, 5 and 10 constraints.

During LLM plan verification, we measured the number of times an LLM-generated plan did not comply with the format we instructed the LLMs to follow via our prompt. Table 3 displays our results on LLMs with reasoning capabilities, when instructed to suggest plans for problems with 1, 5 and 10 constraints. Our results show that the responses of o3 and Gemini 2.5 Pro included, in almost all cases, a plan that conformed with the format of PDDL actions, and could thus be mapped directly into a PDDL plan, without needing to resort to distance calculations between LLM-generated actions and domain actions. In contrast, the responses of R1 and Claude 3.7 Sonnet often deviated from the requested plan format, in which cases we needed to map some of the actions in the suggested plans into PDDL action via distance minimisation, in order to be able to verify these plans.

C Experimental Setup and Reproducibility

First, we outline the hyperparameter values used in LLM executions. Second, we provide a set of instructions for running LEXICON. Third, we outline the steps for reproducing our experiments.

C.1 Execution Parameters

Model	max tokens	temperature
DeepSeek R1	40K	0.2
OpenAI o3	100K	1
Gemini 2.5 Pro	64K	0.2
Claude 3.7 Sonnet (with thinking)	64K	1
GPT-5	128K	0.2
GPT-4.1	32K	0.2
DeepSeek V3	8K	0.2
Gemini 2.0 Pro	8K	0.2
Claude 3.7 Sonnet (no thinking)	64K	0.2

Table 4: LLM hyperparameters.

Table 4 displays the values for the upper limit on generated tokens (including both completion and reasoning tokens) and the temperature hyperparameters used for each LLM. For all models, we set the upper limit for generated tokens on the maximum value allowed by their developers. We chose to use a temperature of 0.2, i.e., a low value that enables structured, deterministic thinking, while also being higher than zero, allowing a certain degree of exploration. In the case of o3 and Claude 3.7 Sonnet, we used the temperature value 1, because this was the only temperature value allowed for these models when thinking is enabled.

C.2 Code Execution Instructions

Our code is publicly available on Github[3]. We provide instructions on executing our constrained planning problem generator and our LLM plan verifier on the domains that are present in our repository. You may add a custom domain by providing a PDDL domain file, an initial state-goal pair generator and NL descriptions of the actions and the atoms of the domain, following the structure of the domains in our repository.

Installation. You may install LEXICON by following these steps:

1. Install conda on an Ubuntu machine.
2. Clone our repository with Git.
3. Create a conda environment with the necessary package dependencies installed. To do this, visit the root directory of our repository and run:
   `conda env create --name lexiconenv --file=environment.yml`
4. Activate your new conda environment with: `conda activate lexiconenv`
5. Make sure that the following packages are installed: [`anthropic==0.51.0, dotmap==1.3.30, gym==0.26.2, gymnasium==1.0.0, hydra-core==1.3.2, matplotlib==3.7.1, minigrid==3.0.0, numpy==2.2.6, omegaconf==2.3.0, openai==1.81.0, protobuf==6.31.0, pyprover==0.6.2, tqdm==4.67.1, unified_planning==1.2.0`]

All the instructions that follow require that you have the `lexiconenv` environment activated.

Constrained Planning Problem Generation. To generate a constrained planning problem for a specified domain, you may use script `generate_benchmark.py`.

This script receives as input:

[3] `https://github.com/periklismant/lexicon_neurips`

- a domain name ("blocksworld", "babyai", "logistics", "sokoban", or "alfworld"),
- an integer denoting the random seed for generating the first problem in the benchmark,
- the number of problems to be generated, and
- the number of constraints in each problem.

The output of the script is:

- a constrained problem for the domain (in both PDDL and NL), located in folder:
  `domains/{domain_name}/data_{constraints_no}/{seed_no}`

In order to run our problem generator, follow these steps:

1. Move into the root directory of our repository.
2. Construct a directory with the name `intermediate_sas`, which is a required folder for SymK to store intermediate computations, with the following command:
   `mkdir intermediate_sas`
3. Run command:
   `python3 generate_benchmark.py {domain_name} {initial_seed} {problems_no} {constraints_no}`

Example executions:

- `python3 generate_benchmark.py blocksworld 100 1 2`
  → Starting from seed 100, construct a blocksworld problem with 2 constraints.
- `python3 generate_benchmark.py logistics 50 3 4`
  → Starting from seed 50, construct 3 logistics problems with 4 constraints each.

LLM Plan Verification. To validate an LLM-generated plan for a constrained planning problem, you may use script `verify_plan.py`.

This script receives as input:

- a domain name ("blocksworld", "babyai", "logistics", "sokoban", or "alfworld"),
- a folder number (corresponding to the number of constraints in the generated problem),
- a data number (corresponding to the seed used to generate the problem), and
- an llm name ("deepseek", "o3", "gemini-2.5", "claude_37_sonnet", "gpt_5"), where "deepseek" verifies a plan produced by R1.

The output of the script is:

- an indication on whether the plan stored in
  `domains/{domain_name}/data/data_{folder_no}/{data_no}/{llm}_plan`
  is invalid, suboptimal or optimal.

In order to run our LLM plan verifier, follow these steps:

1. Move into the root directory of our repository.
2. Run command:
   `python3 verify_plan.py {domain_name} {folder_no} {data_no} {llm}`

Example executions (on pre-generated, packed LLM plans):

- `python3 verify_plan.py babyai 1 1 o3`
  → Verifies that the plan in the corresponding directory is optimal.
- `python3 verify_plan.py babyai 3 1 o3`
  → Verifies that the plan in the corresponding directory is invalid.
- `python3 verify_plan.py blocksworld 5 1 o3`
  → Verifies that the plan in the corresponding directory is suboptimal.

C.3 Experiment Reproducibility Instructions

Reproducing our experiments requires three main steps:

1. Generating benchmarks with problems having an increasing number of constraints for each domain.
2. Evaluating LLMs on the generated benchmarks.
3. Verifying the LLM plans produced in the previous step.

Steps 1 and 3 are described in Appendix C.2.

In order to run LLMs on constrained problems generated by LEXICON, follow these steps:

1. Get API keys by OpenAI, Deepseek, Google and Anthropic, and store them in conda environment variables as follows:

   ```
   conda env config vars set OPENAI_API_KEY=yourkey
   conda env config vars set DEEPSEEK_API_KEY=yourkey
   conda env config vars set GEMINI_API_KEY=yourkey
   conda env config vars set ANTHROPIC_API_KEY=yourkey
   ```

   You have to deactivate and reactivate your conda environment for the variable changes to take effect. In order to use some models, such as o3, you may need to elevate your subscription to a certain tier level.
2. Open file `cfg/config.yaml` with a text editor and make the following changes:
   - set the value of `mode` to `evaluation`.
   - set the value of `folder_no` to the `constraints_no` used to generate the problems you want the LLMs to solve.
   - set the value of list `evaluation_data` to the ids of the problems you want to evaluate LLMs on. These problem ids can be found in:
     `domains/{domain_name}/data/data_{folder_no}/`
   - add a new key-value pair "`llm: evaluation`".
3. Evaluate DeepSeek R1, OpenAI o3, Gemini 2.5 Pro, Claude 3.7 Sonnet (with extended thinking) and GPT-5 on the problems selected in the previous step on, e.g., the Blocksworld domain:
   (a) Create a file named `run_blocksworld.py` and add to it the following code:

   ```python
   from omegaconf import OmegaConf
   from domains.blocksworld.blocksworld import main
   if __name__ == "__main__":
       cfg = OmegaConf.load("cfg/config.yaml")
       main(cfg)
   ```

   (b) Run the following command:
   `python3 run_blocksworld.py`

   You may evaluate these LLMs on a different domain by replacing "blocksworld" with the name of another domain in the above steps.
4. In order to use different LLMs, open file `lexicon.py` with a text editor, make the following changes and then go back to the previous step.
   - Go to the definition of `evaluate_llms` and adjust the elements of list `model_names_and_strategies`.