Two Constraint Compilation Methods for Lifted Planning

Periklis Mantenoglou[1], Luigi Bonassi[2], Enrico Scala[3], Pedro Zuidberg Dos Martires[1]

[1]Örebro University, Sweden
[2]Oxford Robotics Institute, University of Oxford, UK
[3] University of Brescia, Italy
periklis.mantenoglou@oru.se, luigibonassi@robots.ox.ac.uk, enrico.scala@unibs.it, pedro.zuidberg-dos-martires@oru.se

Abstract

We study planning in a fragment of PDDL with qualitative state-trajectory constraints, capturing safety requirements, task ordering conditions, and intermediate sub-goals commonly found in real-world problems. A prominent approach to tackle such problems is to compile their constraints away, leading to a problem that is supported by state-of-the-art planners. Unfortunately, existing compilers do not scale on problems with a large number of objects and high-arity actions, as they necessitate grounding the problem before compilation. To address this issue, we propose two methods for compiling away constraints without grounding, making them suitable for large-scale planning problems. We prove the correctness of our compilers and outline their worst-case time complexity. Moreover, we present a reproducible empirical evaluation on the domains used in the latest International Planning Competition. Our results demonstrate that our methods are efficient and produce planning specifications that are orders of magnitude more succinct than the ones produced by compilers that ground the domain, while remaining competitive when used for planning with a state-of-the-art planner.

1 Introduction

Planning involves identifying a sequence of actions that brings about a desired goal state, and is vital in several real-world tasks (Ropero et al. 2017; Shaik and van de Pol 2023). In addition to final goals, a planning task may require the satisfaction of certain temporal constraints throughout the execution of the plan (Bacchus and Kabanza 1998). These constraints may enforce, e.g., a set of conditions that must be avoided (Steinmetz et al. 2022), or the ordered resolution of certain tasks (Hoffmann et al. 2004), both of which are common in planning problems (Huang et al. 2024; Jackermeier and Abate 2025). Many planning approaches supporting these types of specifications have been considered in the literature (Bacchus and Kabanza 1998; Patrizi et al. 2011; Micheli and Scala 2019; Camacho and McIlraith 2019; Mallett et al. 2021; Bonassi et al. 2022).

In this paper, we focus on the fragment of PDDL that introduced and standardised constraints over trajectories of states (Gerevini et al. 2009). Some approaches handle these types of constraints directly in the search engine (Benton et al. 2012; Hsu et al. 2007; Edelkamp 2006), while others remove the constraints using compilation approaches (Percassi and Gerevini 2019; Bonassi et al. 2024). Alternatively, these constraints can be translated into Linear Temporal Logic (LTL) (Pnueli 1977), or one of its variants (De Giacomo et al. 2020), and handled by existing compilers (Baier and McIlraith 2006; Torres and Baier 2015; Bonassi et al. 2025). The compilation step yields an equivalent constraint-free problem that can be solved by off-the-shelf planners (Helmert 2006; Speck et al. 2020).

For PDDL qualitative state-trajectory constraints, TCORE (Bonassi et al. 2021) constitutes the state-of-the-art compilation approach. Specifically, TCORE employs a so-called regression operator (Rintanen 2008) that expresses the circumstances under which an action may affect constraint satisfaction. However, TCORE's regression operator only works on ground actions, requiring compilation over the fully grounded problem, which can be orders of magnitude larger than the original (non-ground) problem.

Typically, the cost of grounding increases with the number of objects. This becomes a critical issue in domains with high-arity actions, which have to be grounded once for each valid combination of objects in their arguments. Consider, e.g., the following variant of the BLOCKSWORLD domain.

Example 1 (BLOCKSWORLD2)**.** BLOCKSWORLD involves rearranging stacks of blocks on a table to reach a final configuration by moving one block at a time. States are described by atoms such as $\texttt{clear}(b)$, expressing that there is no block on top of block b, and $\texttt{on}(b_1, b_2)$, expressing that block b_1 is on top of block b_2. BLOCKSWORLD2 extends BLOCKSWORLD by allowing operations on towers consisting of two blocks. Action `pickup2`, e.g., allows the agent to pick up a two-block tower. Grounding a BLOCKSWORLD2 problem introduces one ground instance of the `pickup2` action for each (ordered) pair of blocks in the problem. ♢

Although large instances of such domains can be handled by lifted planners (Corrêa and Giacomo 2024; Horcík et al. 2025; Horcík and Fiser 2021, 2023; Fiser 2023; Chen et al. 2024), these approaches do not consider constraints. In BLOCKSWORLD2, these constraints might be, for example, that block b_1 must never be on the table, or that placing b_1 on top of b_2 is only allowed after placing b_3 on the table.

To address this issue, we propose two constraint com-

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

pilers that are *lifted*, in the sense that they avoid unnecessary grounding. We summarise our **contributions** as follows. **First**, we propose LiftedTCORE, a constraint compiler that employs a lifted variant of the regression operator used by TCORE in order to bypass grounding. **Second**, we propose the Lifted Constraint Compiler (LCC) that compiles away constraints without grounding and without the use of a regression operator. **Third**, we demonstrate the correctness of LiftedTCORE and LCC, and outline their worst-case time complexity. **Fourth**, we present an empirical evaluation on challenging planning domains from the latest International Planning Competition (Taitler et al. 2024). Our results demonstrate that LiftedTCORE and LCC are efficient and lead to compiled specifications that are significantly more succinct than the ones produced by compilers that ground the domain, while remaining competitive when used for planning with a state-of-the-art planner.

The proofs of all propositions, along with details on the benchmark we employed and additional experimental results can be found in the technical appendix (Mantenoglou et al. 2025).

2 Background

2.1 Planning with Constraints in PDDL

A planning problem is a tuple $\Pi = \langle F, O, A, I, G, C \rangle$, where F is a set of atoms, O is a set of objects, 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 list of arguments $Arg(a)$, 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 $\forall \tilde{z}\colon c \triangleright e$, where c is a formula, e is a literal—both constructed based on the atoms in F—and $\tilde{z}$ is a set of variables. $\forall \tilde{z}{:}c \triangleright e$ expresses that, for each possible substitution θ_z of the variables in $\tilde{z}$ with objects from O, if the condition $c|_{\theta_z}$ is true, then effect $e|_{\theta_z}$ is brought about (McDermott 2000), where $x|_\theta$ denotes the application of substitution θ in formula x. We use $\mathit{Eff}^+(a)$ (resp. $\mathit{Eff}^-(a)$) to denote the set of effects of an action a where literal e is positive (negative), and $a^{\doteq}$ to denote an action whose arguments are grounded to objects in O, i.e., $Arg(a^{\doteq})$ does not contain variables. A state $s \subseteq F$ contains the atoms that are true in s. A ground action $a^{\doteq}$ is applicable in state s if $s \models Pre(a^{\doteq})$, and its application yields state $s[a^{\doteq}] = (s \setminus \bigcup_{c \triangleright e \in \mathit{Eff}^-(a^{\doteq}):\, s \models c} e) \cup \bigcup_{c \triangleright e \in \mathit{Eff}^+(a^{\doteq}):\, s \models c} e$.

We focus on a fragment of PDDL that includes Always, Sometime, AtMostOnce, SometimeBefore and SometimeAfter constraints (Gerevini et al. 2009). Given a sequence of states $\sigma = \langle s_0, \ldots, s_n \rangle$ and first-order formulae ϕ and ψ, these constraint types are defined as follows:

- $\sigma \models \mathsf{Always}(\phi)$ (or $\mathsf{A}(\phi)$) iff $\forall i\colon 0 \leq i \leq n, s_i \models \phi$, i.e., ϕ holds in every state of σ.
- $\sigma \models \mathsf{Sometime}(\phi)$ ($\mathsf{ST}(\phi)$) iff $\exists i\colon 0 \leq i \leq n, s_i \models \phi$, i.e., ϕ holds in at least one state of σ.
- $\sigma \models \mathsf{AtMostOnce}(\phi)$ ($\mathsf{AO}(\phi)$) iff $\forall i\colon 0 \leq i \leq n$, if $s_i \models \phi$, then $\exists j\colon j \geq i$ such that $\forall k\colon i \leq k \leq j, s_k \models \phi$ and $\forall k\colon j < k \leq n, s_k \models \neg\phi$, i.e., ϕ is true in at most one continuous subsequence of σ.
- $\sigma \models \mathsf{SometimeBefore}(\phi, \psi)$ ($\mathsf{SB}(\phi, \psi)$) iff $\forall i\colon 0 \leq i \leq n$, if $s_i \models \phi$ then $\exists j\colon 0 \leq j < i, s_j \models \psi$, i.e., if there is a state s_i where ϕ holds, then there is a state that is before s_i in σ where ψ holds.
- $\sigma \models \mathsf{SometimeAfter}(\phi, \psi)$ ($\mathsf{SA}(\phi, \psi)$) iff $\forall i\colon 0 \leq i \leq n$, if $s_i \models \phi$ then $\exists j\colon i \leq j \leq n, s_j \models \psi$, i.e., if there is a state s_i where ϕ holds, then ψ also holds at state s_i or there is a state that is after s_i in σ where ψ holds.

A plan π for problem $\Pi = \langle F, O, A, I, G, C \rangle$ is a sequence of ground actions $\langle a_0^{\doteq}, \ldots, a_{n-1}^{\doteq} \rangle$ from A and O. Plan π is valid iff, for the sequence $\sigma = \langle s_0, \ldots, s_n \rangle$ such that $s_0 = I$ and $s_{i+1} = s_i[a_i^{\doteq}]$, $\forall i\colon 0 \leq i < n$, we have $s_i \models Pre(a_i^{\doteq})$, $\forall i\colon 0 \leq i < n$, $s_n \models G$, and $\forall q \in C\colon \sigma \models q$.

2.2 Regression

The regression of a formula ϕ through an action a is the weakest condition that must hold to guarantee the satisfaction of ϕ after the application of a. Regression has been used in different contexts, e.g., situation calculus (Levesque et al. 1998), numeric planning (Scala et al. 2020), non deterministic planning (Rintanen 2008), and SAS$^+$ planning (Alcázar et al. 2013). We focus on regression for classical planning as defined by Rintanen (2008) for ground ϕ and $a^{\doteq}$.

Definition 1 (Regression Operator)**.** Consider a ground formula ϕ and a ground action $a^{\doteq}$. Regression $R(\phi, a^{\doteq})$ is the formula obtained from ϕ by replacing every atom f in ϕ with $\Gamma_f(a^{\doteq}) \vee (f \wedge \neg \Gamma_{\neg f}(a^{\doteq}))$, where the gamma operator $\Gamma_l(a^{\doteq})$ for a literal l is defined as: $\Gamma_l(a^{\doteq}) = \bigvee_{c \triangleright l \in \mathit{Eff}(a^{\doteq})} c$. □

TCORE employs regression to compile away constraints (Bonassi et al. 2021). To do this, TCORE first grounds the problem and then calculates $R(\phi, a^{\doteq})$ for each formula ϕ appearing in a constraint and each action $a^{\doteq}$. Subsequently, TCORE introduces $R(\phi, a^{\doteq})$ formulae and 'monitoring atoms'—whose purpose is to track the status of constraints—in action preconditions and effects, capturing the semantics of the constraints in the compiled problem.

3 LiftedTCORE

3.1 Lifted Regression

The idea behind constructing LiftedTCORE is to generalize the regression operator from the ground domain to the lifted domain. Consequently, LiftedTCORE will need to compute the lifted regression operator $R^L(\phi, a)$ for each action a and each formula ϕ appearing in an argument of a constraint. We achieve this by constructing a lifted gamma operator $\Gamma_l^L(a)$ and adapting Definition 1 accordingly.

Similar to $\Gamma_l(a)$, $\Gamma_l^L(a)$ expresses the weakest condition that leads to the satisfaction of l after applying action a, with the difference of $\Gamma_l^L(a)$ operating on non-ground l and a.

We start with the case of a formula ϕ whose truth value may be affected only by $c \triangleright e$ effects of action a, i.e., the only set to consider in $\forall \tilde{z}{:}c \triangleright e$ is the empty set ($\tilde{z} = \emptyset$). For each literal l in ϕ and each effect $c \triangleright e$ of a, we compute the most general unifier $\xi(l, e)$ between l and e (if any) and derive the weakest condition $w_l^c(c \triangleright e)$ under which an action with effect $c \triangleright e$ may bring about l. We derive $\xi(l, e)$ via Robinson's resolution algorithm (Robinson 1965) by using

the most general resolution, in the sense that variables are grounded to constants only when required.

Definition 2 (Weakest Condition $w_l^c(c \triangleright e)$)**.** The weakest condition $w_l^c(c \triangleright e)$ under which effect $c \triangleright e$ brings about literal l is:

$$w_l^c(c \triangleright e) = \begin{cases} c \wedge \bigwedge_{(t_i \doteq u_i) \in \xi(l,e)} (t_i = u_i) & \text{if } \exists \xi(l,e) \\ \bot & \text{if } \nexists \xi(l,e) \end{cases}$$

where t_i and u_i denote the arguments of the literals l and e respectively, and $t_i \doteq u_i$ denotes their unification. □

Example 2 (Weakest Condition $w_l^c(c \triangleright e)$)**.** Consider the BLOCKSWORLD2 domain, and action `putdown2`, which allows the agent to place a block or a two-block tower on the table. `putdown2` has one argument b and its effects are:

- `handEmpty`, i.e., the agent's hand is empty,
- `onTable`(b), i.e., block b is on the table,
- $\neg$`holding`(b), i.e., the agent is not holding b,
- $\neg$`towerBase`$(b) \triangleright$ `clear`(b), i.e., if b is not the base of a block tower, then b is clear.

Consider literals `onTable`(b_1) and `clear`(b_5), where b_1 and b_5 are ground. `onTable`(b_1) can only be unified with effect `onTable`(b) of `putdown2` via $b \doteq b_1$, while `clear`(b_5) can only be unified with literal `clear`(b) of effect $\neg$`towerBase`$(b) \triangleright$ `clear`(b) via $b \doteq b_5$. Thus, we have:

$$w^c_{\texttt{onTable}(b_1)}(\texttt{onTable}(b)) = (b = b_1)$$
$$w^c_{\texttt{clear}(b_5)}(\neg\texttt{towerBase}(b) \triangleright \texttt{clear}(b)) = \neg\texttt{towerBase}(b) \wedge (b = b_5) \quad \Diamond$$

Next, we extend Definition 2 to handle $\forall \tilde{z}{:}c \triangleright e$ effects for a non-empty set $\tilde{z}$. In the case of such an effect, a variable u appearing in an argument of e may be: (i) a parameter of a, or (ii) a variable in $\tilde{z}$. Definition 2 handles case (i) by introducing equalities between the action parameters in e and the corresponding arguments of l. Case (ii), however, needs a different type of treatment as the scope of a variable $u \in \tilde{z}$ is only within $\forall \tilde{z}{:}c \triangleright e$, and thus, if we were to introduce an equality between u and an argument of l in $R_l^L(a)$, then u would be a free variable in $R_l^L(a)$.

To tackle this issue, we leverage the semantics of $\forall \tilde{z}{:}c \triangleright e$, according to which $\forall \tilde{z}{:}c \triangleright e$ is equivalent to introducing one effect $c|_{\theta_z} \triangleright e|_{\theta_z}$ for each possible substitution θ_z of the variables in $\tilde{z}$ with domain objects (McDermott 2000). As a result, e can be unified with l iff there is a substitution $\theta_z(l,e)$ such that $e|_{\theta_z(l,e)}$ can be unified with l. If there is such a substitution, then, in order for effect $\forall \tilde{z}{:}c \triangleright e$ to bring about l, there needs to be an assignment to the variables of $\tilde{z}$ that appear in c but not in e, i.e., the variables in set $\tilde{z}^f$, such that $c|_{\theta_z(l,e)}$ becomes true. In other words, formula $\exists \tilde{z}^f \ c|_{\theta_z(l,e)}$ needs to hold. Then, the weakest condition $w_l(\forall \tilde{z}{:}c \triangleright e)$ under which $\forall \tilde{z}{:}c \triangleright e$ brings about l requires for $\exists \tilde{z}^f \ c|_{\theta_z(l,e)}$ to hold under a variable assignment that unifies l and $e|_{\theta_z(l,e)}$.

Definition 3 (z-substitution)**.** Consider a literal l and a conditional effect $\forall \tilde{z}{:}\ c \triangleright e$, such that l and e can be unified and $\xi(l,e)$ is their most general unifier. We define the z-substitution $\theta_z(l,e)$ of l and e as:

$$\theta_z(l,e) = \{u_i \mapsto t_i \mid (t_i \doteq u_i) \in \xi(l,e) \wedge u_i \in \tilde{z}\}$$

where $x \mapsto y$ denotes that term y substitutes term x. □

Definition 4 (Weakest Condition $w_l(\forall \tilde{z}{:}c \triangleright e)$)**.** The weakest condition under which $\forall \tilde{z}{:}c \triangleright e$ brings about l is:

$$w_l(\forall \tilde{z}{:}c \triangleright e) = \begin{cases} \exists \tilde{z}^f \ c|_{\theta_z(l,e)} \wedge \bigwedge_{(t_i \doteq u_i) \in \xi(l,e) \wedge u_i \notin \tilde{z}} (t_i = u_i) & \text{if } \exists \xi(l,e) \\ \bot & \text{if } \nexists \xi(l,e) \end{cases}$$

where $\tilde{z}^f = \{z \in \tilde{z} \mid \nexists t_i : (z \mapsto t_i) \in \theta_z(l,e)\}$. □

According to Definition 4, when $\tilde{z}$ is empty, $w_l(\forall \tilde{z}{:}c \triangleright e)$ coincides with $w_l^c(c \triangleright e)$.

Example 3 (Weakest Condition $w_l(\forall \tilde{z}{:}c \triangleright e)$)**.** Consider that literal l is `clear`(b_5) and suppose that action `putdown2` additionally has the following effect:

- $\forall\{topb\}$: `on`$(topb, b) \triangleright$ `clear`$(topb)$, i.e., any block $topb$ that is on top of block b is clear.

`clear`(b_5) unifies with `clear`$(topb)$ via $topb \doteq b_5$, and, since $topb$ is a $\tilde{z}$ variable of the effect, we have $\theta_z(\texttt{clear}(b_5), \texttt{clear}(topb)) = \{topb \mapsto b_5\}$, while $\tilde{z}^f$ is empty. Therefore, we have:

$$w_l(\forall\{topb\}{:}\ \texttt{on}(topb, b) \triangleright \texttt{clear}(topb)) = \texttt{on}(b_5, b) \quad \Diamond$$

Definition 5 (Lifted Gamma Operator $\Gamma_l^L(a)$)**.** Given an action a and a literal l, the lifted gamma operator is defined as: $\Gamma_l^L(a) = \bigvee_{\forall \tilde{z}:c \triangleright e \in Eff(a)} w_l(\forall \tilde{z}{:}c \triangleright e)$. □

According to Definition 5, $\Gamma_l^L(a)$ is the weakest condition under which action a brings about l via one of its effects.

Example 4 (Lifted Gamma Operator $\Gamma_l^L(a)$)**.** Based on the results in Examples 2 and 3, We have:

$$\Gamma^L_{\texttt{onTable}(b_1)}(\texttt{putdown2}) = (b = b_1)$$
$$\Gamma^L_{\texttt{clear}(b_5)}(\texttt{putdown2}) = (\neg\texttt{towerBase}(b) \wedge (b = b_5)) \vee \texttt{on}(b_5, b) \quad \Diamond$$

Definition 6 (Lifted Regression Operator $R^L(\phi, a)$)**.** Consider a first-order formula ϕ and an action a. The lifted regression $R^L(\phi, a)$ is the formula obtained from ϕ by replacing every atom f in ϕ with $\Gamma_f^L(a) \vee (f \wedge \neg\Gamma^L_{\neg f}(a))$. □

According to Definition 6, $R^L(\phi, a)$ is the weakest condition under which ϕ is true after the execution of action a.

Example 5 (Lifted Regression $R^L(\phi, a)$)**.** Based on the results in Example 4, and since $\Gamma^L_{\neg\texttt{onTable}(b_1)}(\texttt{putdown2}) = \bot$ and $\Gamma^L_{\neg\texttt{clear}(b_5)}(\texttt{putdown2}) = \bot$, we have:

$$R^L(\texttt{onTable}(b_1), \texttt{putdown2}) = (b = b_1) \vee \texttt{onTable}(b_1)$$
$$R^L(\texttt{clear}(b_5), \texttt{putdown2}) = \texttt{on}(b_5, b) \vee \texttt{clear}(b_5) \vee (\neg\texttt{towerBase}(b) \wedge (b = b_5)) \quad \Diamond$$

3.2 The LiftedTCORE Compiler

LiftedTCORE follows the same steps as the TCORE compiler (Bonassi et al. 2021), with the exception of using lifted regression. As in TCORE, the main intuition behind LiftedTCORE is to foresee that an action a will affect the truth value of a formula ϕ appearing in a constraint by checking whether $R^L(\phi, a)$ holds. If it does hold, then LiftedTCORE sets a so-called 'monitoring atom', in order to express the status of ϕ

after an execution of a. Contrary to TCORE, LiftedTCORE introduces the necessary regression formulae and monitoring atoms without grounding the problem, thus avoiding the combinatorial explosion induced by grounding.

Algorithm 1 outlines the steps of LiftedTCORE. First, we check if an A or SB constraint (c.f. Section 2.1) is violated in the initial state (line 1). Since their violation is irrevocable, we deem the problem unsolvable (line 2). Next, we introduce the monitoring atoms required to track the constraints of the problem (line 3). $hold_c$ expresses that constraint c is satisfied based on the state trajectory induced so far. $seen_\phi$ expresses that there is a state in the induced trajectory where ϕ holds. $hold_c$ atoms are used to monitor ST and SA constraints, while $seen_\phi$ atoms are required for SB and AO constraints. Constraints of type A do not demand monitoring atoms. We augment the initial state I with the monitoring atoms that are satisfied in I (line 4).

Afterwards, for each action a, we compute the preconditions and the effects that need to be added in a in order to capture the semantics of the constraints within the compiled problem (lines 5–8). For a $\mathsf{A}(\phi)$ constraint, we need to add precondition $R^L(\phi, a)$ in a, stating that ϕ needs to be true after the execution of a (line 13). In the case of $\mathsf{ST}(\phi)$, we activate monitoring atom $hold_{\mathsf{ST}(\phi)}$ if ϕ is true after executing a (line 15). For $\mathsf{AO}(\phi)$, we set $seen_\phi$ when a brings about ϕ (line 17) and forbid the execution of a when $seen_\phi$ holds, ϕ is false and a would bring about ϕ (line 18), thus prohibiting ϕ from occurring more than once. For $\mathsf{SB}(\phi, \psi)$, we set $seen_\psi$ when a brings about ψ, and forbid the execution of a when it would bring about ϕ while $seen_\phi$ does not hold. For $\mathsf{SA}(\phi, \psi)$, we set $hold_{\mathsf{SA}(\phi,\psi)}$ if a brings about ψ, and $\neg hold_{\mathsf{SA}(\phi,\psi)}$ if a brings about a state where $\phi \wedge \neg\psi$ holds. Lastly, we require that all $hold_c$ monitoring atoms need to be true in a goal state, reflecting that all ST and SA must have been satisfied by the end of the plan (line 9).

Example 6 (LiftedTCORE)**.** Consider a BLOCKSWORLD2 problem with the following constraints: $\mathsf{ST}(\texttt{clear}(b_5))$, $\mathsf{AO}(\texttt{onTable}(b_1))$, $\mathsf{SB}(\texttt{clear}(b_5), \exists topb : \texttt{on}(topb, b_3))$. To compile away these constraints, LiftedTCORE introduces monitoring atoms $hold_{\mathsf{ST}(\texttt{clear}(b_5))}$, $seen_\psi$ and $seen_\phi$, where ψ is $\exists topb : \texttt{on}(topb, b_3)$ and ϕ is $\texttt{onTable}(b_1)$, and updates the initial state with these atoms according to line 4. Subsequently, LiftedTCORE introduces a set of action-specific preconditions and effects. For action putdown2, e.g., LiftedTCORE adds preconditions P and effects E, i.e.:

$$\begin{aligned} P = \{&R^L(\texttt{clear}(b_5), \texttt{putdown2}) \rightarrow seen_\psi, \\ &\neg(seen_\phi \wedge \neg\phi \wedge R^L(\texttt{onTable}(b_1), \texttt{putdown2})\} \\ E = \{&R^L(\texttt{clear}(b_5), \texttt{putdown2}) \triangleright hold_{\mathsf{ST}(\texttt{clear}(b_5))}, \\ &R^L(\texttt{onTable}(b_1), \texttt{putdown2}) \triangleright seen_\phi\} \end{aligned}$$

where expressions $R^L(\texttt{onTable}(b_1), \texttt{putdown2})$ and $R^L(\texttt{clear}(b_5), \texttt{putdown2})$ were derived in Example 5. ◊

According to line 20 of Algorithm 1, set E of Example 6 should have included effect $R^L(\psi, \texttt{putdown2}) \triangleright seen_\psi$. The reason for its omission was based on the fact that $R^L(\psi, \texttt{putdown2}) = \psi$, which implies that ψ holds after the execution of action putdown2 in a state s iff ψ holds at s.

Algorithm 1: LiftedTCORE

Require: Planning problem $\Pi = \langle F, O, A, I, G, \mathcal{C}\rangle$.
Ensure: Planning problem $\Pi' = \langle F', O, A', I', G', \emptyset\rangle$
1: **if** $\exists \mathsf{A}(\phi) \in \mathcal{C} : I \models \neg\phi \vee \exists \mathsf{SB}(\phi, \psi) \in \mathcal{C} : I \models \phi$ **then**
2: **return** Unsolvable Problem
3: $F' \leftarrow F \cup \bigcup_{c:\mathsf{ST}(\phi)\in\mathcal{C} \vee c:\mathsf{SA}(\phi,\psi)\in\mathcal{C}} \{hold_c\} \cup \bigcup_{\mathsf{SB}(\phi,\psi)\in\mathcal{C}} \{seen_\psi\} \cup \bigcup_{\mathsf{AO}(\phi)\in\mathcal{C}} \{seen_\phi\}$
4: $I' \leftarrow I \cup \bigcup_{c:\mathsf{ST}(\phi)\in\mathcal{C}} \{hold_c \mid I \models \phi\} \cup \bigcup_{c:\mathsf{SA}(\phi,\psi)\in\mathcal{C}} \{hold_c \mid I \models \psi \vee \neg\phi\}$
$\cup \bigcup_{\mathsf{SB}(\phi,\psi)\in\mathcal{C}} \{seen_\psi \mid I \models \psi\} \cup \bigcup_{\mathsf{AO}(\phi)\in\mathcal{C}} \{seen_\phi \mid I \models \phi\}$
5: **for all** $a \in A$ **do**
6: $P, E \leftarrow$ COMPILECREGRESSION$(a, \mathcal{C})$
7: $Pre(a) \leftarrow Pre(a) \wedge \bigwedge_{p\in P} p$
8: $Eff(a) \leftarrow Eff(a) \cup E$
9: **return** $\langle F', O, A, I', G \wedge \bigwedge_{hold_c \in F'} hold_c, \emptyset\rangle$
10: **function** COMPILECREGRESSION$(a, \mathcal{C})$
11: $P, E = \{\}, \{\}$
12: **for** $c \in \mathcal{C}$ **do**
13: **if** c is $\mathsf{A}(\phi)$ **then** $P \leftarrow P \cup \{R^L(\phi, a)\}$
14: **else if** c is $\mathsf{ST}(\phi)$ **then**
15: $E \leftarrow E \cup \{R^L(\phi, a) \triangleright \{hold_c\}\}$
16: **else if** c is $\mathsf{AO}(\phi)$ **then**
17: $E \leftarrow E \cup \{R^L(\phi, a) \triangleright \{seen_\phi\}\}$
18: $P \leftarrow P \cup \{\neg(seen_\phi \wedge \neg\phi \wedge R^L(\phi, a))\}$
19: **else if** c is $\mathsf{SB}(\phi, \psi)$ **then**
20: $E \leftarrow E \cup \{R^L(\psi, a) \triangleright \{seen_\psi\}\}$
21: $P \leftarrow P \cup \{R^L(\phi, a) \rightarrow seen_\psi\}$
22: **else if** c is $\mathsf{SA}(\phi, \psi)$ **then**
23: $E \leftarrow E \cup \{R^L(\psi, a) \triangleright \{hold_c\}\}$
$\cup \{R^L(\phi, a) \wedge \neg R^L(\psi, a) \triangleright \{\neg hold_c\}\}$
24: **return** P, E

Therefore, if $R^L(\psi, \texttt{putdown2})\psi$ holds, then there is an earlier action a, where $R^L(\psi, a) \neq \psi$, that made ψ true, and thus set atom $seen_\psi$. Thus, it would be redundant to reinstate $seen_\psi$ when putdown2 is executed. By omitting regression formulae such that $R^L(\phi, a) = \phi$, LiftedTCORE compresses the compiled specifications without sacrificing correctness.

3.3 Theoretical Properties

We prove that LiftedTCORE is correct, i.e., a problem has the same solutions as its compiled version produced by LiftedTCORE, and outline its complexity. Towards correctness, we first demonstrate a property of lifted regression.

Lemma 1 (R^L Correctness)**.** Consider a state s, an action a, a closed first-order formula ϕ, and a ground incarnation $a^{\doteq}$ of a such that $a^{\doteq}$ is applicable in s. $\theta_{a^{\doteq}}$ denotes the substitution of the arguments of a with those of $a^{\doteq}$. We have:

$$s \models R^L(\phi, a)|_{\theta_{a^{\doteq}}} \iff s[a^{\doteq}] \models \phi \qquad \text{◊}$$

Proposition 1 (Correctness of LiftedTCORE)**.** If LiftedTCORE compiles a problem Π into problem Π', then a plan π is valid for Π iff π is valid for Π'. ◊

Proposition 2 (Complexity of LiftedTCORE)**.** Assuming that the nesting depth of quantifiers in constraint formulae is bounded by constant b, the worst-case time com-

plexity of compiling a problem Π with LiftedTCORE is $\mathcal{O}(n_c n_f^b + n_a n_c n_f^2 n_k)$, where n_c, n_f, n_a and n_k denote, respectively, the number of constraints, the number of atoms, the number of actions and the maximum atom arity in Π. $\Diamond$

Allowing an unbounded quantifier nesting in constraint formulae makes the step of checking whether the initial state models such a formula (line 1) PSPACE-complete (Vardi 1982), dominating the complexity of LiftedTCORE. In practice, however, the quantifier nesting depth is shallow.

4 The Lifted Constraint Compiler

Next, we discuss the 'Lifted Constraint Compiler' (LCC). LCC does not use lifted regression, and works by computing a set of preconditions P and a set of effects E that are *independent* from actions. Effects E record the status of constraints: when an action a is executed in a state s, E will introduce in the next state s' information regarding the status of the constraints in s. The purpose of preconditions P is to prevent the execution of further actions in a state s when a constraint has been violated. These new preconditions and effects are shared among all actions. In this way, LCC monitors constraint violation without the need for regression, i.e., instead of foreseeing that an action a will affect a formula ϕ of a constraint via the regression of ϕ through a, LCC allows the application of any executable action a but then blocks subsequent state expansion if a led to constraint violation. Following this schema, when a state s where the goal is satisfied is reached, there has not been an earlier check on whether the constraints are satisfied in s. To address this, LCC introduces a new action *fin* whose purpose is to verify that a state of the plan that satisfies the goal also satisfies the constraints. To enforce that *fin* is the last action executed, we use a new atom end that marks the end of the plan.

Algorithm 2 outlines the steps of LCC. Initially, the compilation creates the necessary monitoring atoms to track the status of the constraints (line 1). LCC requires the same monitoring atoms used in LiftedTCORE, plus an additional atom $prevent_\phi$ for every $\mathsf{AO}(\phi)$. This atom express that ϕ is false after having been true, and thus should be prevented from becoming true again in order to satisfy $\mathsf{AO}(\phi)$.

Next, LCC determines the set of preconditions P and the set of effects E to be added to every action (line 2) by iterating over each constraint c (lines 12-23). If c is $\mathsf{A}(\phi)$, LCC adds precondition ϕ in all actions, ensuring that no progress can be made when we have $\neg\phi$ (line 13). If c is $\mathsf{ST}(\phi)$, LCC captures the satisfaction of c by making $hold_c$ true when ϕ holds (line 14). If c is $\mathsf{AO}(\phi)$, LCC brings about $seen_\phi$ when ϕ is true (line 16), and $prevent_\phi$ when ϕ is false after having been true in the past (line 17). Then, LCC prevents the execution of any action when both ϕ and $prevent_\phi$ are true, a situation that violates $\mathsf{AO}(\phi)$ (line 18). If c is $\mathsf{SB}(\phi,\psi)$, LCC sets $seen_\psi$ when ψ becomes true, and prevents further actions when ϕ is true and $seen_\psi$ is false (lines 20-21). Lastly, if c is $\mathsf{SA}(\phi,\psi)$, LCC activates $hold_c$ when ψ is true, and deactivates it when ϕ is true and ψ is false (line 23), thus expressing that $\mathsf{SA}(\phi,\psi)$ is violated only when ϕ held at some point in the past and ψ has not become true since then.

Preconditions P and effects E are added to every action of the problem, including the newly introduced *fin* action (lines 3-7). To ensure that no further action is executable after *fin*, we add precondition $\neg end$ to every action, and end as an effect of *fin*. As a last step, LCC determines the new goal, i.e., G augmented with the conjunction of all $hold_c$ atoms and atom end, and the new initial state, where every $hold_c$ atom for a $\mathsf{SA}(\phi,\psi)$ constraint is set to true, maintaining correctness in the case where both ϕ and ψ are always false.

Algorithm 2: LCC

Require: Planning Problem $\Pi = \langle F, O, A, I, G, C\rangle$.
Ensure: Planning Problem $\Pi = \langle F', O, A', I', G', \emptyset\rangle$.
1: $F' \leftarrow F \cup \bigcup_{c:\mathsf{ST}(\phi)\in\mathcal{C} \vee c:\mathsf{SA}(\phi,\psi)\in\mathcal{C}} \{hold_c\} \cup \bigcup_{\mathsf{AO}(\phi)\in\mathcal{C}} \{seen_\phi, prevent_\phi\} \cup \bigcup_{\mathsf{SB}(\phi,\psi)\in\mathcal{C}} \{seen_\psi\} \cup \{end\}$
2: $P, E \leftarrow$ COMPILEC($\mathcal{C}$)
3: **for all** $a \in A$ **do**
4: $\quad Pre(a) \leftarrow Pre(a) \wedge \bigwedge_{p\in P} p \wedge \neg end$
5: $\quad Eff(a) \leftarrow Eff(a) \cup E$
6: $Pre(fin) \leftarrow P \wedge \neg end$
7: $Eff(fin) \leftarrow E \cup \{end\}$
8: $A' \leftarrow A \cup \{fin\}$
9: **return** $\langle F', O, A', I \cup \bigcup_{\mathsf{SA}(\phi,\psi)\in\mathcal{C}} \{hold_c\}, G \wedge \bigwedge_{c:\mathsf{ST}(\phi)\in\mathcal{C} \vee c:\mathsf{SA}(\phi,\psi)\in\mathcal{C}} hold_c \wedge end, \emptyset\rangle$
10: **function** COMPILEC($\mathcal{C}$)
11: $\quad P, E \leftarrow \{\}, \{\}$
12: $\quad$ **for all** $c \in \mathcal{C}$ **do**
13: $\quad\quad$ **if** c is $\mathsf{A}(\phi)$ **then** $P \leftarrow P \cup \{\phi\}$
14: $\quad\quad$ **else if** c is $\mathsf{ST}(\phi)$ **then** $E \leftarrow E \cup \{\phi \triangleright hold_c\}$
15: $\quad\quad$ **else if** c is $\mathsf{AO}(\phi)$ **then**
16: $\quad\quad\quad E \leftarrow E \cup \{\phi \triangleright seen_\phi\}$
17: $\quad\quad\quad E \leftarrow E \cup \{(\neg\phi \wedge seen_\phi) \triangleright prevent_\phi\}$
18: $\quad\quad\quad P \leftarrow P \cup \{\neg(\phi \wedge prevent_\phi)\}$
19: $\quad\quad$ **else if** c is $\mathsf{SB}(\phi,\psi)$ **then**
20: $\quad\quad\quad E \leftarrow E \cup \{\psi \triangleright seen_\psi\}$
21: $\quad\quad\quad P \leftarrow P \cup \{\phi \rightarrow seen_\psi\}$
22: $\quad\quad$ **else if** c is $\mathsf{SA}(\phi,\psi)$ **then**
23: $\quad\quad\quad E \leftarrow E \cup \{(\phi \wedge \neg\psi) \triangleright \neg hold_c\} \cup \{\psi \triangleright hold_c\}$
24: $\quad$ **return** P, E

Example 7 (LCC Compiler)**.** LCC compiles the problem in Example 6 using the monitoring atoms $hold_{\mathsf{ST}(\mathtt{clear}(b_5))}$, $seen_\psi$, $seen_\phi$ and $prevent_\phi$, where ψ is $\exists topb : \mathtt{on}(topb, b_3)$ and ϕ is $\mathtt{onTable}(b_1)$, and then extends all actions with the following preconditions P and effects E:

$$P = \{\mathtt{clear}(b_5) \rightarrow seen_\psi, \neg(\mathtt{onTable}(b_1) \wedge prevent_\phi)\}$$
$$E = \{\mathtt{clear}(b_5) \triangleright hold_{c_1}, \exists topb : \mathtt{on}(topb, b_3) \triangleright seen_\psi,$$
$$\mathtt{onTable}(b_1) \triangleright seen_\phi,$$
$$(\neg\mathtt{onTable}(b_1) \wedge seen_\phi) \triangleright prevent_\phi\}$$ □

Proposition 3 (Correctness of LCC)**.** If LCC compiles a problem Π into problem Π', then plan $\langle a_0^{\pm}, \ldots, a_{n-1}^{\pm}\rangle$ is valid for Π iff plan $\langle a_0^{\pm}, \ldots, a_{n-1}^{\pm}, fin\rangle$ is valid for Π', and only plans ending with action *fin* are valid for Π'. $\Diamond$

Proposition 4 (Complexity of LCC)**.** Suppose that n_a and n_c denote, respectively, the number of actions and the number of constraint in some problem Π. The worst-case time

complexity of compiling Π with LCC is $\mathcal{O}(n_a+n_c)$. $\Diamond$

5 Comparing LiftedTCORE and LCC

We highlight the main differences between our compilers.

Introduced Preconditions and Effects. Through the use of regression, LiftedTCORE identifies action-specific preconditions and effects that capture the constraints, while LCC adds the required preconditions and effects in all actions in a uniform way. This action-specific compilation of LiftedTCORE allows us to avoid redundant updates to actions that are irrelevant to a constraint, leading to succinct compiled problems. In contrast, the action-agnostic compilation of LCC does not permit such optimisations.

Constraint Evaluation Delay. LCC monitors constraints with a 1-step delay, i.e., the status of constraint in a state s can only be identified after an action a is applied in s. On the other hand, LiftedTCORE does not delay the evaluation of constraints, in the sense that the status of a constraint in state s was calculated at the time of reaching s through the evaluation of regression formulae.

Monitoring Atoms. LCC uses the same monitoring atoms as LiftedTCORE, plus one $prevent_\phi$ atom for each $\mathsf{AO}(\phi)$. Unlike LiftedTCORE, LCC cannot foresee when an action will cause ϕ to hold for a second time, and thus uses $prevent_\phi$ to block further actions once this occurs.

Initial State. In LiftedTCORE, to express the status of the constraints in the initial state, we need to add to that state the monitoring atoms that hold initially. This is not required in LCC because of its delayed constraint evaluation, i.e., the status of a constraint in the initial state is expressed by the truth values of the monitoring atoms in its subsequent state.

Actions. LCC requires 1 additional action compared to LiftedTCORE, i.e., action *fin*, which is required to check whether the constraints are satisfied in a goal state, due to the 1-step constraint evaluation delay in LCC.

Complexity. While both LiftedTCORE and LCC operate in polynomial time to the size of the problem (see Propositions 2 and 4), LCC runs in time that is *linear* in the number of constraints and the number of actions, as because it determines all preconditions and effects required for the compilation with one pass over the constraints. Instead, LiftedTCORE derives a set of preconditions and effects for each action of the domain, which more computationally involved.

6 Experimental Evaluation

Our aim is to evaluate LiftedTCORE and LCC against state-of-the-art approaches. To do this, we generated a new benchmark based on the 7 domains of the latest International Planning Competition (IPC) (Taitler et al. 2024), including problems with many objects and high-arity actions that are challenging for grounding-based compilations. Specifically, we created two datasets, each with 20 tasks per domain: one with ground constraints, denoted by "Ground", and one with constraints containing quantifiers ($\forall$, $\exists$), denoted by "Non-Ground". These datasets contained 280 problems in total and were generated as follows. First, we used the IPC generators to construct problem instances without constraints. Second, we introduced ground constraints that complicate an optimal solution of the problem using a custom constraint generator, yielding the Ground dataset. Third, we modified the produced ground constraints by introducing quantifiers, leading to our Non-Ground dataset. Additional details on this benchmark are provided in (Mantenoglou et al. 2025).

	Domain	LCC	LiftedTCORE	TCORE	LTL-C
Ground	Folding	**20**	**20**	**20**	**20**
	Labyrinth	17	17	**18**	16
	Quantum	**19**	**19**	18	17
	Recharging	**19**	**19**	**19**	N.A.
	Ricochet	**20**	**20**	**20**	**20**
	Rubik's	**20**	**20**	**20**	N.A.
	Slitherlink	**18**	**18**	17	**18**
Non-Ground	Folding	**18**	**18**	**18**	**18**
	Labyrinth	16	15	**18**	15
	Quantum	**19**	17	18	16
	Recharging	18	17	**19**	N.A.
	Ricochet	12	11	**14**	7
	Rubik's	16	**17**	14	N.A.
	Slitherlink	**14**	10	12	11
	Total	**246**	238	245	158

Table 1: Coverage achieved by all systems in all domains.

We used TCORE as our baseline, as it has been proven more effective (Bonassi et al. 2021) than other methods (Edelkamp 2006; Benton et al. 2012; Hsu et al. 2007; Torres and Baier 2015) for handling PDDL constraints. We also included LTL-C, a lifted compiler that supports quantified LTL constraints (Baier and McIlraith 2006). Unlike our encodings, LTL-C compiles the constraints by first translating them into finite-state automatons. All compiled instances were solved using LAMA (Richter and Westphal 2010), a state-of-the-art satisficing planner. We stopped LAMA after finding the first solution and disabled its invariant generation. All experiments were run on an Intel Xeon Gold 6140M 2.3 GHz, with runtime and memory limits of 1800s and 8GB, respectively. LiftedTCORE[1] and LCC[2] were implemented using the Unified Planning library (Micheli et al. 2025); their code and our benchmark[3] are publicly available.

6.1 Experimental Results

Table 1 presents the coverage (number of solved problems) per domain achieved by LAMA when planning over the compiled specification of each system, We observe that TCORE, LiftedTCORE, and LCC performed comparably. Overall, LCC achieved the best performance, with TCORE being a close runner-up—these IPC domains were designed to be compatible with traditional grounding-based planners like LAMA, and thus this grounding-based planner performed well on tasks compiled by TCORE. Nonetheless, both LCC and LiftedTCORE were competitive with TCORE and outperformed LTL-C, the state of the art among lifted compilation approaches. LTL-C failed in the 'Recharging'

[1]https://github.com/Periklismant/LiftedTCORE

[2]https://github.com/LBonassi95/NumericTCORE

[3]https://github.com/Periklismant/aaai26-pddl3-benchmark

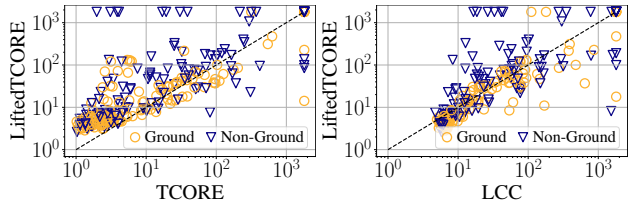

Figure 1: Runtime comparison between LiftedTCORE vs. TCORE and between LiftedTCORE vs. LCC.

and 'Rubik's' domains, as it does not support universally quantified effects, which are present in these domains.

In terms of coverage, LCC outperformed LiftedTCORE; LiftedTCORE's lifted regression operator introduced complex precondition and effect formulas, especially in cases with Non-Ground constraints, hindering LAMA's performance. On the other hand, this operator allows constraint evaluation without the 1-step delay of LCC, allowing LAMA to perform a more efficient search. Table 2 reports the average number of nodes expanded by LAMA when planning over the compiled specifications of LCC, LiftedTCORE and TCORE. On average, LiftedTCORE led to fewer node expansions than LCC. TCORE and LiftedTCORE led to a similar number of node expansions, as they both employ regression to foresee redundant expansions.

The reduction in expansion nodes also affects runtime; Figure 1 shows that LiftedTCORE led to more efficient planning than LCC in several instances. Specifically, constraint compilation with LiftedTCORE led to more efficient planning than LCC compilation in 161 out of the 234 instances that were solved by both approaches. LiftedTCORE performed better on instances with Ground constraints, where both approaches introduce formulas with comparable complexity. In the Non-Ground case, the size of the largest formula introduced by LiftedTCORE and LCC per instance was, on average, 26 and 18 atoms, thus leading to slower planning in the case of LiftedTCORE. Our results show that LiftedTCORE compilation led to slower planning than TCORE compilation, although there is complementarity.

We conclude with statistics on the size of the compiled problems and their compilation times, reported in Table 3. On average, TCORE produced problems with thousands of actions and effects, reaching up to 411K actions and 1.8M effects, while both LiftedTCORE and LCC produced problems that are more succinct by orders of magnitude. This not only makes the compiled problems more understandable and easier to debug, but also facilitates the use of planners that do not need to perform grounding. Currently, the support of complex preconditions and conditional effects is limited in these planners (Horcík et al. 2022; Horcík and Fiser 2023; Corrêa and Giacomo 2024). We expect that the benefits of our lifted compilers will become more prominent as more expressive heuristics for lifted planning are developed.

Limitations: We observed that lifted constraint compilers may not be beneficial for problems with a large number of constraints. For each constraint, lifted methods update all

	Domain	LCC	LiftedTCORE	TCORE
Ground	Folding	150.45	**132.10**	139.55
	Labyrinth	**769.69**	1453.44	1464.94
	Quantum	338.24	379.82	**307.59**
	Recharging	9259.16	**1629.53**	1796.47
	Ricochet	2412.70	2153.00	**1956.75**
	Rubik's	230268.65	**180999.25**	189079.45
	Slitherlink	19483.47	13454.12	**11437.53**
Non-Ground	Folding	43.89	**37.61**	40.33
	Labyrinth	**623.57**	3028.00	3230.93
	Quantum	408214.12	224.18	**216.18**
	Recharging	1305.18	**459.24**	673.18
	Ricochet	327.73	**121.27**	156.73
	Rubik's	782161.29	**316887.57**	382046.86
	Slitherlink	**478.40**	964.00	780.40

Table 2: Average number of nodes expanded by LAMA on tasks compiled by TCORE, LiftedTCORE, and LCC. Averages are computed among instances solved by all systems.

	Ground			Non-Ground		
	Lifted TCORE	LCC	TCORE	Lifted TCORE	LCC	TCORE
Actions	7	8	21K	7	8	21K
Effects	58	63	95K	60	67	96K
Comp. Time	0.023	0.003	4.518	0.081	0.003	5.956

Table 3: Average number of actions, effects and compilation time (in seconds) on the problems in our dataset.

(lifted) actions that may affect it in a least one of their instantiations, resulting in numerous new action preconditions and effects that hinder the performance of modern planners like LAMA. In contrast, TCORE updates ground actions sparsely, as only few of them may affect each constraint. We evaluated our compilers on a benchmark including problems with hundreds of constraints (Bonassi et al. 2021), and TCORE proved to be the best performing compiler (Mantenoglou et al. 2025). We believe that further research on lifted planning heuristics could alleviate this issue, while preserving the benefits of lifted compilations, such as compilation efficiency and compiled specification succinctness.

7 Summary and Further Work

We proposed two methods, LiftedTCORE and LCC, for compiling away quantitative state-trajectory constraints from a planning problem without grounding it. We studied both compilers theoretically, proving their correctness and deducing their worst-case time complexity, and qualitatively compared their key features. Moreover, we presented an empirical evaluation of our methods on the domains included in the latest International Planning Competition, demonstrating that our compilers are efficient and lead to significantly more succinct compiled specifications compared to a state-of-the-art compiler that grounds the domain, while yielding competitive performance when used for planning with LAMA.

In the future, we aim to design lifted compilers for numeric and metric time constraints (Bonassi et al. 2024).

Acknowledgements

Periklis Mantenoglou and Pedro Zuidberg Dos Martires were supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Luigi Bonassi was supported by the joint UKRI and AISI-DSIT Systemic Safety Grant [grant number UKRI854].

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting Regression in Planning. In *IJCAI*, 2254–2260. IJCAI/AAAI.

Bacchus, F.; and Kabanza, F. 1998. Planning for Temporally Extended Goals. *Ann. Math. Artif. Intell.*, 22(1-2): 5–27.

Baier, J. A.; and McIlraith, S. A. 2006. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*, 788–795.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *ICAPS*.

Bonassi, L.; Gerevini, A. E.; Percassi, F.; and Scala, E. 2021. On Planning with Qualitative State-Trajectory Constraints in PDDL3 by Compiling them Away. In *ICAPS*, 46–50.

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2022. Planning with Qualitative Action-Trajectory Constraints in PDDL. In *IJCAI*, 4606–4613.

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2024. Dealing with Numeric and Metric Time Constraints in PDDL3 via Compilation to Numeric Planning. In *AAAI*, 20036–20043.

Bonassi, L.; Giacomo, G. D.; Favorito, M.; Fuggitti, F.; Gerevini, A. E.; and Scala, E. 2025. Planning for temporally extended goals in pure-past linear temporal logic. *Artif. Intell.*, 348: 104409.

Camacho, A.; and McIlraith, S. A. 2019. Strong Fully Observable Non-Deterministic Planning with LTL and LTLf Goals. In *IJCAI*, 5523–5531.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. W. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *AAAI*, 20078–20086.

Corrêa, A. B.; and Giacomo, G. D. 2024. Lifted Planning: Recent Advances in Planning Using First-Order Representations. In *IJCAI*, 8010–8019.

De Giacomo, G.; Stasio, A. D.; Fuggitti, F.; and Rubin, S. 2020. Pure-Past Linear Temporal and Dynamic Logic on Finite Traces. In *IJCAI*, 4959–4965.

Edelkamp, S. 2006. On the Compilation of Plan Constraints and Preferences. In *ICAPS*, 374–377.

Fiser, D. 2023. Operator Pruning Using Lifted Mutex Groups via Compilation on Lifted Level. In *ICAPS*.

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

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *J. Artif. Intell. Res.*, 22: 215–278.

Horcík, R.; and Fiser, D. 2021. Endomorphisms of Lifted Planning Problems. In *ICAPS*, 174–183.

Horcík, R.; and Fiser, D. 2023. Gaifman Graphs in Lifted Planning. In *ECAI*, volume 372, 1052–1059.

Horcík, R.; Fiser, D.; and Torralba, Á. 2022. Homomorphisms of Lifted Planning Tasks: The Case for Delete-Free Relaxation Heuristics. In *AAAI*, 9767–9775.

Horcík, R.; Sír, G.; Simek, V.; and Pevný, T. 2025. State Encodings for GNN-Based Lifted Planners. In *AAAI*.

Hsu, C.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences. In *IJCAI*.

Huang, Z.; Liu, H.; Shen, S.; and Ma, J. 2024. Parallel Optimization with Hard Safety Constraints for Cooperative Planning of Connected Autonomous Vehicles. In *ICRA*.

Jackermeier, M.; and Abate, A. 2025. DeepLTL: Learning to Efficiently Satisfy Complex LTL Specifications for Multi-Task RL. In *ICLR*.

Levesque, H. J.; Pirri, F.; and Reiter, R. 1998. Foundations for the Situation Calculus. *Electron. Trans. Artif. Intell.*, 2: 159–178.

Mallett, I.; Thiébaux, S.; and Trevizan, F. W. 2021. Progression Heuristics for Planning with Probabilistic LTL Constraints. In *AAAI*, 11870–11879.

Mantenoglou, P.; Bonassi, L.; Scala, E.; and Martires, P. Z. D. 2025. Two Constraint Compilation Methods for Lifted Planning. arXiv:2511.10164.

McDermott, D. V. 2000. The 1998 AI Planning Systems Competition. *AI Mag.*, 21(2): 35–55.

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

Micheli, A.; and Scala, E. 2019. Temporal Planning with Temporal Metric Trajectory Constraints. In *AAAI*.

Patrizi, F.; Lipovetzky, N.; Giacomo, G. D.; and Geffner, H. 2011. Computing Infinite Plans for LTL Goals Using a Classical Planner. In *IJCAI*, 2003–2008.

Percassi, F.; and Gerevini, A. E. 2019. On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata. In *ICAPS*, 320–328.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*, 46–57. IEEE Computer Society.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Rintanen, J. 2008. Regression for Classical and Nondeterministic Planning. In *ECAI*, volume 178, 568–572.

Robinson, J. A. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1): 23–41.

Ropero, F.; Muñoz, P.; R-Moreno, M. D.; and Barrero, D. F. 2017. A Virtual Reality Mission Planner for Mars Rovers. *SMC-IT*, 142–146.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.*, 68: 691–752.

Shaik, I.; and van de Pol, J. 2023. Optimal Layout Synthesis for Quantum Circuits as Classical Planning. In *ICCAD*, 1–9.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *AAAI*, 9967–9974.

Steinmetz, M.; Hoffmann, J.; Kovtunova, A.; and Borgwardt, S. 2022. Classical Planning with Avoid Conditions. In *AAAI*, 9944–9952.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fiser, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Mag.*, 45(2).

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *IJCAI*, 1696–1703.

Vardi, M. Y. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*, 137–146.