PSPACE Planning With Expressivity Beyond STRIPS: Plan Constraints via Unordered HTNs, ILPs, Numerical Goals, and More

Pascal Lauer[1,2], **Yifan Zhang**[1], **Patrik Haslum**[1], **Pascal Bercher**[1]

[1]School of Computing, The Australian National University, Canberra, Australia
[2]Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
firstname.lastname@anu.edu.au

Abstract

To better capture real-world problems, Hierarchical Task Network (HTN) planning and numerical planning provide enhanced modeling capabilities over classical planning. However, the plan existence problem in these formalisms is generally undecidable. We identify restricted fragments that remain PSPACE-complete, matching the complexity of classical planning, while being more expressive. The most important result proves that plan existence in unordered HTN planning, i.e. ignoring all ordering relations, is PSPACE-complete. The result motivates a strong preference for unordered HTN models, a largely ignored fragment that deserves more attention. To bridge the gap between the tractable fragments of numerical and HTN planning, we introduce new formalisms that use Integer Linear Programs, Presburger formulas, and grammatical constraints to express action sequence restrictions within PSPACE, offering practical alternatives when the HTN structure is too complex to model.

1 Introduction

Automated planning is a branch of artificial intelligence focusing on generating operator sequences (plans) from a user-provided problem specification. The specification must be based on a planning formalism. A widely supported formalism is referred to as classical planning (Fikes and Nilsson 1971; Bäckström and Nebel 1995), which is restricted to a set of simple features to make finding a plan more feasible.

While the restriction helps planners run efficiently, users often need more flexibility to model a problem. To address this, there are formalisms extending classical planning. We focus on: (1) Numerical planning (Helmert 2002), allowing numerical variable values instead of propositional ones; and (2) Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996), allowing to constrain plans through a hierarchical task structure. The flexibility comes at a cost. Plan existence in both formalisms is undecidable (Helmert 2002; Erol, Hendler, and Nau 1996), but PSPACE-complete for classical planning (Erol, Nau, and Subrahmanian 1995).

It is clearly desirable to have a formalism that combines both benefits, i.e., remains computationally tractable while offering greater modeling flexibility. A good example, showing that this is possible, is totally ordered HTN planning, which imposes a pre-defined order between all tasks. The restriction still allows modeling a meaningful hierarchical structure, but drops the complexity of plan existence to EXPTIME (Erol, Hendler, and Nau 1996). The drop in complexity is reflected in practice. E.g., the results of the HTN track in the latest International Planning Competition (Taitler et al. 2024) show a clear gap between the ability to solve totally ordered versus partially ordered problems.

In this paper, we show that there exists an alternative ordering constraint, namely unordered, which does not allow any order at all, and reduces the complexity of plan existence to PSPACE. To reach this result, we reconsider a fragment of numerical planning for which Helmert (2002) established decidability. It allows only numerical goal conditions, but no numerical preconditions. We improve the result by showing that plan existence in this fragment is PSPACE-complete. The fragment is closely related to HTN planning as the added constraint, imposed on plans by the hierarchical structure, closely compares to the added constraint from the numeric goal. To formalize this connection, we introduce three new planning formalisms that add constraints on classical planning plans through: (1) Integer Linear Programs, (2) Presburger formulas and (3) Counting constraints over context-free grammars. We prove PSPACE membership by establishing a chain of encodings from unordered HTN planning through the formalisms (3) to (1), ending in the numerical fragment. Hardness follows from the fact that these formalisms extend classical planning.

To objectively show that each of the five formalisms meaningfully extends classical planning, we use a language analysis (Höller et al. 2014; 2016; Lin and Bercher 2022). The set of plans for a planning problem is its language, and the language class of a formalism groups all such languages for problems it can represent. The analysis nicely complements our complexity results, as it favors bigger classes, including more languages, whereas complexity theory favors lower classes. To demonstrate that our formalisms are significantly more expressive, we show they include the classical planning language as well as each a regular, context-free, and context-sensitive language that classical planning can not express.

Our results strongly suggest moving away from partial-order HTN models in favor of unordered ones and point at new alternative formalisms that can be used when other formalisms make modeling too complicated.

2 Background

Numerical (and Classical) Planning In numerical planning, operator preconditions and effects are evaluated through rational functions. Restrictions on these functions impact the complexity of plan existence (Helmert 2002). Therefore, we fix families of rational functions to define distinct numerical planning formalisms.

Definition 2.1 (Numerical Planning Formalism)**.** A function $f : \mathbb{Q}^n \to \mathbb{Q}$ is called an n-ary *rational function*. A *family of rational functions* F is a set of rational functions, i.e., $F \subseteq \bigcup_{n \geq 1}(\mathbb{Q}^n \to \mathbb{Q})$. A *numerical planning formalism* (G, P, E) consists of three families of rational functions G, P, E. Here G contains the *goal condition functions*, P the *precondition functions*, and E the *effect functions*.

As we build on results by Helmert (2002) in Section 4, we also reproduce his numerical planning problem definition.

Definition 2.2 (Numerical Planning Problem)**.** A *numerical planning problem* $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ over numerical planning formalism (G, P, E) consists of: The *propositional variables* V_P and *numerical variables* V_N, which are disjoint finite sets. The *states* S are:

$$S := \{(\alpha, \beta) \mid \alpha : V_P \to \{\bot, \top\},\ \beta : V_N \to \mathbb{Q}\}.$$

For each state $(\alpha, \beta) \in S$, α is called *propositional state*. The set of all propositional states is denoted by S_α. β is called *numerical state*. The set of all numerical states is denoted by S_β. *Init* is a state, called *initial state*.

A *propositional condition* is a propositional variable $v \in V_P$, also written as $v = \top$. For a family of rational functions $F \in \{G, P\}$, a *numerical condition* over F is given by an n-ary function $f \in F$, numerical variables $v_1, \ldots, v_n \in V_N$, and a relational operator $\mathit{relop} \in \{=, <, \leq, \geq, >, \neq\}$ denoted as $f(v_1, \ldots, v_n)\ \mathit{relop}\ 0$.

A *propositional effect* is given by a variable $v \in V_P$ and a truth value $t \in \{\top, \bot\}$, written as $v \leftarrow t$. A *numerical effect* over E is given by a function $f \in E$ and variables $v_1, \ldots, v_n \in V_N$. It is written as $v_1 \leftarrow f(v_2, \ldots, v_n)$. We say the effect *assigns* v_1. An operator over $o = (\mathit{pre}, \mathit{eff})$ consists of two finite sets *pre* and *eff*, where: *pre* contains propositional conditions and numerical conditions over P and *eff* contains propositional effects eff_P and numerical effects eff_N over E with pairwise distinct assigned variables. *Ops* is the finite set of operators over P and E of Π. The *goal condition Goal* is a finite set containing propositional and numerical conditions over G.

$\mathcal{NUM}(G, P, E)$ is the set of all numerical planning problems over numerical planning formalism (G, P, E).

Helmert (2002) defines plans using the full transition graph. For brevity, we only define the progressive successor states using the notation by Hoffmann and Nebel (2001).

Definition 2.3 (Numerical Plans)**.** For a numerical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ over numerical planning formalism $F = (G, P, E)$, a propositional condition $v \in V_p$ is *satisfied* in propositional state $\alpha \in S_\alpha$ iff $\alpha(v) = \top$. A numerical condition $f(v_1, \ldots, v_n)\ \mathit{relop}\ 0$ with $f \in G \cup P$ is satisfied in numerical state $\beta \in S_\beta$ iff $f(\beta(v_1), \ldots, \beta(v_n))\ \mathit{relop}\ 0$.

An operator $o = (\mathit{pre}, \mathit{eff}) \in \mathit{Ops}$ is *applicable* in propositional state $\alpha \in S_\alpha$ iff all propositional conditions in *pre* are *satisfied* in α. In this case the *propositional successor* state is the state replacing the truth assignments of α by *eff*:

$$\mathrm{progr}(\alpha, o) := \{v \mapsto t \in \alpha \mid \nexists v \leftarrow t' \in \mathit{eff}_P\} \cup \mathit{eff}_P$$

Otherwise $\mathrm{progr}(\alpha, o)$ is undefined.

Further, o is *applicable* in numerical state $\beta \in S_\alpha$ iff all numerical conditions in *pre* are satisfied. In this case we set the *numerical successor* as:

$$\begin{aligned}\mathrm{progr}(\beta, o) := \{v \mapsto n \in \beta \mid \nexists e \in \mathit{eff}_N \text{ assigning } v\}\\ \cup \{v_1 \mapsto f(\beta(v_2), ..., \beta(v_n)) \mid v_1 \leftarrow f(v_2, ..., v_n) \in \mathit{eff}_N\}\end{aligned}$$

Otherwise $\mathrm{progr}(\alpha, o)$ is undefined.

Finally, o is *applicable* in state (α, β) if it is applicable in both α and β. In this case we denote the *progressive successor state* as:

$$\mathrm{progr}((\alpha, \beta), o) := (\mathrm{progr}(\alpha, o), \mathrm{progr}(\beta, o))$$

Otherwise $\mathrm{progr}((\alpha, \beta), o)$ is undefined.

We denote $\mathrm{progr}(\mathrm{progr}(... \mathrm{progr}(s, o_1), ...), o_n)$ by $\mathrm{progr}(s, o_1, ..., o_n)$. A plan is an operator sequence $o_1, \ldots, o_n$ so that $\mathrm{progr}(\mathit{Init}, o_1, \ldots, o_n) =: (\alpha', \beta')$ so that all propositional conditions in *Goal* are satisfied in α' and all numerical conditions in *Goal* are satisfied in β'. $\mathrm{sol}(\Pi)$ denotes the set of all plans in Π.

We now replicate Helmert's list of conditions, excluding polynomial functions, which are beyond this paper's scope.

Definition 2.4 (Common Numerical Conditions)**.**
1. $C_\varnothing := \varnothing$
2. $C_0 := \{x \mapsto x\}$
3. $C_c := \{x \mapsto x - c \mid c \in \mathbb{Q}\}$
4. $C_= := \{(x_1, x_2) \mapsto x_1 - x_2\}$

$C_\varnothing$ corresponds to no preconditions. C_0 compares a variable with zero. C_c compares a variable with a constant. $C_=$ compares two variables. For section 4, it will be important to restrict preconditions to $C_\varnothing$, so we give this a name.

Definition 2.5 (No Numerical Preconditions)**.** A numerical planning problem over formalism $(G, C_\varnothing, E)$ with arbitrary families of rational function G and E is said to have *no numerical operator preconditions*.

We also focus on all effects Helmert (2002) uses, except polynomial functions. Helmert describes 10 classes. We summarize them by listing the effects that can add/subtract $E_{\pm c}$, assign constants $E^{=c}$, and their combination $E^{=c}_{\pm c}$

Definition 2.6 (List of Numerical Effects)**.**
1. $E_{\pm c} := \{x \mapsto x + c \mid c \in \mathbb{Q}\}$
2. $E^{=c} := \{x \mapsto c \mid c \in \mathbb{Q}\}$
3. $E^{=c}_{\pm c} := E_{=c} \cup E_{\pm c}$

$E^{=c}_{\pm c}$ includes all other non-polynomial effects Helmert lists.

Definition 2.7 (Constant Effects)**.** A numerical planning problem over numerical planning formalism (G, C, E) so that $E \subseteq E^{=c}_{\pm c}$ is said to have *constant effects*.

Finally, observe that classical planning formalisms, like SAS$^+$ (Bäckström and Nebel 1995) or FDR (Helmert 2009), match our formalism if there are no numerical functions.

Definition 2.8 (Classical Planning)**.** The numerical planning formalism $(C_\varnothing, C_\varnothing, C_\varnothing)$ is called classical planning. We will also refer to $\mathcal{NUM}(C_\varnothing, C_\varnothing, C_\varnothing)$ as $\mathcal{CLASSIC}$.

Unordered HTN Planning We introduce HTN planning based on Geier and Bercher (2011), but restrict it to unordered HTN planning. This means that we remove ordering constraints from any task network in the planning problem.

Definition 2.9 (Unordered HTN planning problem)**.** An *unordered HTN planning problem* $\Pi_{\mathcal{H}} = (\Pi, \mathcal{H})$, consists of a classical planning problem $\Pi = (V_P, V_N, \mathit{Ops}, \mathit{Init}, \mathit{Goal})$ and the hierarchy $\mathcal{H} = (tn_I, \mathcal{M}, \mathcal{C})$ modeling hierarchical restrictions to operator solutions. $\mathcal{C}$ is the set of *compounds tasks*. The set of operators *Ops* is also called *primitive tasks* in this context. A *task network* $tn = (T, \alpha)$ consists of a set of *task IDs* T and a map $\alpha : T \to \mathit{Ops} \cup \mathcal{C}$.

A *method* $m = (c_m, tn_m)$ allows to replace a compound task $c_m \in \mathcal{C}$ with the tasks from task network $tn_m = (T_m, \alpha_m)$. In particular, m defines a relation $tn_1 \to_m tn_2$ between task networks $tn_1 = (T_1, \alpha_1)$ and tn_2, iff there exists a task identifier $tid \in T_1$ such that $\alpha_1(tid) = c_m$, a bijection $\sigma : T_m \to T'$, where T' is a set of fresh task identifiers not in T_1, so that the resulting task network tn_2 is given by:

$$tn_2 = (T_1 \setminus \{tid\} \cup \sigma(T_m), \alpha_1 \setminus \{tid \mapsto c_m\} \cup \sigma^{-1} \circ \alpha_m)$$

$\mathcal{M}$ denotes the set of all methods in $\Pi_{\mathcal{H}}$. The relation $\to_{\mathcal{M}}$ is defined as the union of all equivalence relations $\to_m$, for all $m \in \mathcal{M}$. The transitive closure is denoted by $\to^*_{\mathcal{M}}$. A task network (T, α) is called primitive iff all of its tasks T are primitive. The *solution set* of the hierarchy $\mathcal{H}$ are the primitive task networks that can be derived by decomposition from the initial task network, i.e.:

$$\mathrm{sol}(\mathcal{H}) := \{tn \mid tn_I \to^*_{\mathcal{M}} tn, tn \text{ is primitive}\}$$

A *linearization* of a primitive task network is an arbitrarily ordered sequence of all elements $\alpha(tid)$ for $tid \in T$. A *solution* to an unordered HTN planning task is a set of primitive task networks obtained by decomposing tn_I and have a linearization that is a classical planning solution, i.e.:

$$\mathrm{sol}(\Pi_{\mathcal{H}}) = \{tn \in \mathrm{sol}(\mathcal{H}) \mid \exists \pi \in \mathrm{sol}(\Pi) : \pi \text{ is a linearization of } tn\}$$

$\mathcal{HTN}_{\mathcal{U}}$ is the set of all unordered HTN planning problems.

The solution definition clearly shows that plans in HTN planning must meet two criteria: (1) follow the hierarchy, and (2) solve a classical planning task correctly. In sections 5, 6, and 7, we introduce new formalisms that keep this solution structure but replace (1) with different constraints.

Context-Free Grammars

Definition 2.10 (Context-Free Grammar)**.** A *context-free grammar* $G = (N, \Sigma, S, R)$ consists of a finite set of *non-terminals* N, a finite set *terminals* Σ, disjoint from N, the *start symbol* $S \in N$ and a finite set of *production rules* R. A production rule is of the form $A \to \alpha_1 \mid \ldots \mid \alpha_n$, where $A \in N$ is a non-terminal and all $\alpha_1, \ldots, \alpha_n \in (N \cup \Sigma)^*$ are finite sequences of terminals and/or non-terminals.

The language of G are all words derived by repeatedly applying production rules until only terminals remain.

Definition 2.11 (Grammar Language)**.** For a context-free grammar $G = (N, \Sigma, S, R)$, a sequence $s = s_0 s_1 \ldots s_n \in (N \cup \Sigma)^*$ can *derive* a sequence $t \in (N \cup \Sigma)^*$, denoted $s \Rightarrow t$, by replacing some symbol s_i for $i \in \{0, \ldots, n\}$ with α_j for some $j \in \{1, \ldots, m\}$ from a production rule $s_i \to \alpha_1 \mid \cdots \mid \alpha_m$, in R. The relation $\Rightarrow^*$ is the transitive closure of $\Rightarrow$. A *word* (or *string*) is a sequence over Σ^*. The *language* generated by G, denoted $L(G)$, is the set of all words derived from the start symbol:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

Planning Problem Languages A planning problem language represents the set of valid plans. By default terminals are operators whose structure depends on the formalism. To compare languages across different formalisms, we use *exchange functions* that map operators to arbitrary terminals.

Definition 2.12 (Exchange Function)**.** For operators *Ops* the set $\mathrm{exch}(\mathit{Ops})$ contains all bijections $\sigma : \mathit{Ops} \to \Sigma$ for any terminals Σ. We call its elements *exchange function*.

A language for non-HTN planning problems is a plan where each operator is replaced using a fixed exchange function.

Definition 2.13 (Non-HTN Planning Language)**.** For a planning problem Π with operator sequences over operators *Ops* as solutions, the *language* of Π under $\sigma \in \mathrm{exch}(\mathit{Ops})$ is:

$$L_\sigma(\Pi) := \{\sigma(\pi) \mid \pi \in \mathrm{sol}(\Pi)\}.$$

For HTN planning, we adapt the definition to consider a linearization of the primitive task solution.

Definition 2.14 (HTN Planning Language)**.** Let $\Pi_{\mathcal{H}}$ be an HTN planning problem with operators *Ops*. The *language* of $\Pi_{\mathcal{H}}$ under $\sigma \in \mathrm{exch}(\mathit{Ops})$ is:

$$L_\sigma(\Pi_{\mathcal{H}}) = \{\sigma(\pi) \mid \exists tn \in \mathrm{sol}(\mathcal{H}) : \pi \in \mathrm{sol}(\Pi) \text{ is a linearization of } tn\}$$

A language class captures all languages of a set of planning problems under any fixed exchange function.

Definition 2.15 (Language class)**.** For a set $\mathcal{P}$ of planning problems, the *language class* is defined as:

$$L(\mathcal{P}) := \{L_\sigma(\Pi) \mid \Pi \in \mathcal{P}, \sigma \in \mathrm{exch}(\mathit{Ops})\}$$

Integer Linear Programs (ILPs) We define ILPs following Papadimitriou (1981). In our notation $\mathbb{N}$ includes 0.

Definition 2.16. An integer linear program $\mathbb{L} = (M, v)$ is a combination of a matrix $M \in \mathbb{Z}^{m \times n}$ and a vector $v \in \mathbb{Z}^m$. The *solution set* of $\mathbb{L}$ is $\mathrm{sol}(\mathbb{L}) := \{x \in \mathbb{N}^n \mid Mx = v\}$.

3 Formal Criteria for Increased Expressivity

To capture *better expressivity*, we ensure that: A formalism must (1) subsume classical planning, and (2) admit languages beyond the expressive power of classical planning.

Definition 3.1. Let $\mathcal{P}$ be a set of planning problems. We say that $\mathcal{P}$ is *significantly more expressive* than classical planning if all of the following statements hold:

- $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{P})$
- $\{aa\}, \{a^n b^n \mid n \in \mathbb{N}\}, \{a^n b^n c^n \mid n \in \mathbb{N}\} \in L(\mathcal{P})$

where the languages in the latter bullet points are defined over the terminals $\Sigma = \{a, b, c\}$.

The language class for classical planning $L(\mathcal{CLASSIC})$ is a strict subset of regular languages (Höller et al. 2014; 2016). As $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ is a context-sensitive language, and $\{a^n b^n \mid n \in \mathbb{N}\}$ is a context-free language, they are not in $L(\mathcal{CLASSIC})$. $\{aa\}$ is an example of a regular language that is not in $L(\mathcal{CLASSIC})$. The additionally included languages establish $L(\mathcal{CLASSIC}) \subsetneq L(\mathcal{P})$. The inclusion of a regular, context-free and context-sensitive language, members of three and widely studied language classes, demonstrates a significant increase in expressivity.

The three languages also showcase that increased expressivity significantly improves modeling capabilities: In particular, they showcase that a modeler can require operators to occur equally often, without fixing an upper bound. This is impossible in classical planning and relevant in practice. E.g., in a financial market one may require money-earning operators a to be followed by the same number of money-spending operators b. While we keep our analysis to single operators to fit space constraints, our analysis naturally extends to multiple operators, like the language represented by:

$$(\,earn_1 \mid \ldots \mid earn_m\,)^n\,(\,spend_1 \mid \ldots \mid spend_l\,)^n$$

In the following sections, we show that each formalism has better modeling capabilities than classical planning, as they are significantly more expressive, while deciding plan existence remains PSPACE-complete.

4 Numerical Planning With No Numerical Preconditions And Constant Effects

In this section, we analyze numerical planning without numerical preconditions and restrict effects to increase, decrease, or assign fixed constants. The goal conditions are comparisons between two variables, or one variable with a constant. This fragment is closely related to (unordered) HTN planning, as it allows counting operator applications and applying conditions on these counts.

We first focus on the complexity of plan existence, which we show to be PSPACE-complete[1]. We build on results of Helmert and so paraphrase the relevant parts of their work in the following, mainly surrounding Helmert's Algorithm 22, which creates plans of a specific format. To explain the format, we introduce properties of operator sequences (paths) based on the propositional states they traverse. In particular, whether states are visited only once or revisited, which would make the path a cycle.

Definition 4.1. For a numerical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ a *path* from s_0 to s_n is a sequence of operators $o_1, \ldots, o_n \in \mathit{Ops}$ resulting in a state $s_i = \mathit{progr}(o_i, s_{i-1})$ for $i \in \{1, \ldots, n\}$.

Let $s_i = (\alpha_i, \beta_i)$ for $i \in \{0, \ldots, n\}$. If all states $\alpha_0, \ldots, \alpha_n$ are pairwise distinct, the path is called *propositionally acyclic*. If $\alpha_0, \ldots, \alpha_{n-1}$ are pairwise distinct and $\alpha_0 = \alpha_n$, the path is called a *propositional cycle* for α_0.

A path is *weakly acyclic* if it can be partitioned into subsequences of paths $p_1, \ldots, p_m$ where p_1 is an acyclic path from s_0 to s_n, and each p_j with $j \in \{2, \ldots, m\}$ is a propositional cycle for some α_i with $i \in \{0, \ldots, n\}$.

We now define the plan format as activator path. This is a concatenation of weakly acyclic paths, each ending with an assignment to a numerical variable. After this assignment, the variable is only modified by increments or decrements.

Definition 4.2. An *activator path*[2] $p = p_1, \ldots, p_n$ for pairwise distinct variables $v_1, \ldots, v_n \in V_N$ is a sequence of weakly acyclic paths p_i, each ending with an assignment to v_i. No v_i is assigned in any later path p_j with $1 \le i < j \le n$.

Helmert (2002) observes that one can always represent plans in this format, as: To satisfy propositional goals it suffices to follow an acyclic path. Since there are no numerical preconditions, it is enough to ensure applicability over propositional states and only consider the final numerical goal values. These values are determined by a single assignment followed by increments or decrements.

Lemma 4.3. *(Helmert 2002) Let Π be a numerical planning problem with no numerical preconditions and only constants effects with $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$.*

If there exists a plan for Π, then there exists a plan for Π that is an activator path for some $v_1, \ldots, v_n \in V_N$.

Helmert (2002) guess a weakly acyclic path and then determine how many times a cycles need to be repeated to obtain a plan. We capture these repetitions by cycle extensions.

Definition 4.4. For a path $p = p_1, p_c, p_2$, where p_c is a propositional cycle and p_1, p_2 are (possibly empty) paths, p_1, p_c, p_c, p_2 is a *cycle extension*.

To determine the required cycle extensions, Helmert (2002) constructs an ILP. The idea is to introduce a constraint for the goal value of each numerical variable $v \in V_N$:

$$\mathit{goal}_v = \mathit{activation}_v + \sum_{c \in \mathit{Cycles}} (\Delta_v(c) \cdot x_c)$$

Here, $\mathit{activation}_v$ is the value assigned by the activator, $\Delta_v(c)$ is the value change to v after one execution of cycle c, and x_c is the number of times c is repeated. By adding the numerical goal constraints, the required cycle extensions can be bounded by the sum of all x_c.

Lemma 4.5. *(Helmert 2002) Let p be an activator path for variables $v_1, \ldots, v_n \in V_N$ in a numerical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ with no numerical preconditions and only constant effects.*

There exists an ILP formulation that computes the number of cycle extensions needed to turn p into a plan, if possible, and is unsolvable otherwise. The ILP has a polynomial number of rows in the size of Π and a number of columns polynomial in the size of Π and the number of cycles in p.

[1]Dekker and Behnke (2024) already claim this result in their Table 3, citing Helmert (2002). While Helmert establishes decidability, there is no PSPACE-completeness proof. To the best of our knowledge also not in other literature. We confirmed this with the main author of Dekker and Behnke (2024). As no proof appears in the literature, we provide it in our work.

[2]Helmert (2002) refer to the last assigning the values as activator sequence. For brevety we combine this with the operators leading up to it as activator path.

This summarizes the main results we needed to restate in order to prove that we can exponentially bound plans.

Lemma 4.6. *Let* Π *be a numerical planning problem with no numerical preconditions and only constants effects with* $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$.

If there exists a plan for Π*, then there exists a plan for* Π *which consists of at most exponentially-many operators bounded in the size of* Π.

Proof. We first bound the number of distinct cycles in a weakly acyclic path: As addition is commutative, the order of operators in a cycle does not matter, only the number of times each operator occurs in it. Thus, cycles with the same underlying multiset of operators are equivalent in this context. Each such multiset has $|\mathit{Ops}|$ entries, each bounded by the maximum cycle length $|S_\alpha| \le 2^{|V_P|}$. Therefore the number of distinct cycles is at most $(2^{|V_P|})^{|\mathit{Ops}|} = 2^{|V_P| \cdot |\mathit{Ops}|}$, which is exponential in the size of Π. As both cycles and acyclic paths are bounded by $\le 2^{|V_P|}$, any weakly acyclic path, where each cycle occurs at most once, is exponentially length-bounded in the size of Π.

Assume a plan exists. By Lem. 4.3, there is an activator path that is a plan. We can construct it by taking an activator path where each cycle occurs at most once, and then applying cycle extensions. The activator path consists of at most $|V_N|$ weakly acyclic paths and thus is also exponentially length-bounded in the size of Π. It remains to bound the operators added by the cycle extensions.

This follows from Papadimitriou (1981), who shows that if an ILP has polynomially many rows in $n \in \mathbb{N}$, and exponentially many columns in n, then if there is a solution, there is one with values bounded exponentially in n. Combined with Lem. 4.5 this implies that the number of cycle extensions required for an activator-path p is exponentially bounded in Π. As the size of each cycle is exponentially bounded ($\le 2^{|V_P|}$), the cycle extensions leading to a plan add at most an exponential operators in the size of Π. □

Theorem 4.7. *Plan Existence for numerical planning problems over* $(G, C_\varnothing, E)$ *is PSPACE-complete for all:*

$$G \in \{C_\varnothing, C_0, C_c, C_=\} \text{ and } E \subseteq E^{=c}_{\pm c}$$

Proof. PSPACE-hardness follows directly from classical planning, which is PSPACE-complete (Erol, Nau, and Subrahmanian 1995) and captured as a syntactic fragment of the formalisms considered.

For membership we can determine plan existence by a guess-and-check procedure in NPSPACE = PSPACE, since all plans are exponentially length-bounded. □

We now turn to expressivity. We show that numerical planning with additive effects and equality tests can define languages beyond those expressible by classical planning. To ease the upcoming proofs, we introduce a simple classical planning task as a gadget. It has three operators a, b, and c, which must be applied in this order but may be repeated multiple times, before switching to the next operator type. We reuse this gadget throughout the expressivity proofs in the following sections.

Definition 4.8 (Sequential Operator Gadget)**.** The classical planning problem $\Pi^{\mathrm{Seq}} = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ encodes sequence constraints among operators a, b, and c using: The set of propositional variables $V_P = \{\mathit{apply\_a}, \mathit{apply\_b}, \mathit{apply\_c}\}$. The empty set of numerical variables $V_N = \varnothing$. An empty goal condition $\mathit{Goal} = \varnothing$. The initial state $\mathit{Init} = (\alpha, \beta)$, setting the propositional parts to true, i.e., $\alpha(\mathit{apply\_a}) = \top, \alpha(\mathit{apply\_b}) = \top, \alpha(\mathit{apply\_c}) = \top$ and has an empty numerical part $\beta = \varnothing$. Finally, the operator set Ops contains the following operators:

- Operator $a = (\mathit{pre}_a, \mathit{eff}_a)$:
  $\mathit{pre}_a = \{\mathit{apply\_a}\}$, $\mathit{eff}_a = \varnothing$
- Operator $b = (\mathit{pre}_b, \mathit{eff}_b)$:
  $\mathit{pre}_b = \{\mathit{apply\_b}\}$, $\mathit{eff}_b = \{\mathit{apply\_a} \leftarrow \bot\}$
- Operator $c = (\mathit{pre}_c, \mathit{eff}_c)$:
  $\mathit{pre}_c = \{\mathit{apply\_c}\}$, $\mathit{eff}_c = \{\mathit{apply\_a} \leftarrow \bot,\ \mathit{apply\_b} \leftarrow \bot\}$

We can now prove the expressivity result for the numerical planning fragment.

Theorem 4.9. $\mathcal{NUM}(C_=, C_\varnothing, E)$ *is is significantly more expressive than classical planning for* $E \subseteq E^{=c}_{\pm c}$.

Proof. By definition, $\mathcal{CLASSIC} \subseteq \mathcal{NUM}(C_=, C_\varnothing, E)$, so it also holds that $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{NUM}(C_=, C_\varnothing, E))$.

To show that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are in $L(\mathcal{NUM}(C_=, C_\varnothing, E))$, we extend the construction $\Pi^{\mathrm{Seq}} = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ from Def. 4.8 to a numerical planning task $\Pi_N = (V_P, V'_N, \mathit{Ops}', \mathit{Init}', \mathit{Goal}')$. The numeric variables $V'_N = \{v_a, v_b, v_c, v_0, v_2\}$ track how often an operator was applied and simulate constants in the goal condition. We initially set the counters to zero, i.e., $\mathit{Init}' = (\alpha, \beta)$ where α is the propositional state of Init in Π^{Seq} and $\beta = \{v_a \mapsto 0, v_b \mapsto 0, v_c \mapsto 0, v_0 \mapsto 0, v_2 \mapsto 2\}$. Finally, we consider operators $\mathit{Ops}' = \{a', b', c'\}$, extending the original operators $\mathit{Ops} = \{a, b, c\}$ as:

- $a' = (\mathit{pre}_a, \mathit{eff}'_a)$, $\mathit{eff}'_a = \mathit{eff}_a \cup \{v_a \leftarrow v_a + 1\}$
- $b' = (\mathit{pre}_b, \mathit{eff}'_b)$, $\mathit{eff}'_b = \mathit{eff}_b \cup \{v_b \leftarrow v_b + 1\}$
- $c' = (\mathit{pre}_c, \mathit{eff}'_c)$, $\mathit{eff}'_c = \mathit{eff}_c \cup \{v_c \leftarrow v_c + 1\}$

Each operator now increases its count to track how often the operator was applied. We define the goals separately for the two languages.

(1) For Language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ we define the goal:

$$\mathit{Goal}' := (v_a - v_b = 0) \wedge (v_b - v_c = 0)$$

(2) For Language $\{a^n b^n \mid n \in \mathbb{N}\}$ we define the goal:

$$\mathit{Goal}' := (v_a - v_b = 0) \wedge (v_c - v_0 = 0)$$

(3) For Language $\{aa\}$ we define the goal:

$$\mathit{Goal}' := (v_a - v_2 = 0) \wedge (v_b - v_0 = 0) \wedge (v_c - v_0 = 0)$$

Goal (1) requires $v_a = v_b \wedge v_b = v_c$ and so only accepts sequences with equal counts of a, b, and c. Goal (2) requires $v_a = v_b \wedge v_b = 0$ and so only accepts sequences with equal counts of a, b, but no occurrence of c. Goal (3) requires $v_a = 2 \wedge v_b = 0 \wedge v_c = 0$ and so only accepts the sequence aa. □

5 Classical Planning with Linear Constraints

In this and the next two sections, we introduce new planning formalisms that extend classical planning by imposing different solution constraints. This section focuses on Integer Linear Programs (ILPs). We use the numerical fragment from the last section for membership proofs here, creating the first link in a chain connecting all presented formalisms.

To restrict solutions using an ILP, we count the number of occurrences of each operator in a sequence. In language theory, this corresponds to the well-known concept of the Parikh vector, which we then require to satisfy the ILP.

Definition 5.1. Let Σ is a finite set of terminals. The *Parikh vector* $\Psi(w)$ of a sequence $w = e_1 e_2 \dots e_{|w|}$ over the terminals Σ is a vector in $\mathbb{N}^{|\Sigma|}$, where each entry counts the occurrences of the corresponding terminal in w. I.e., for position $i \in \{1, ..., |\Sigma|\}$:

$$(\Psi(w))_i = \sum_{j=1}^{|w|} \begin{cases} 1, \text{ if } \mathrm{id}(e_j) = i \\ 0, \text{ otherwise} \end{cases}$$

For some unique bijective mapping $\mathrm{id} : \Sigma \to \{1, ..., |\Sigma|\}$. For $e \in \Sigma$ we represent $(\Psi(w))_{\mathrm{id}(e)}$ with shorthand $(\Psi(w))_e$.

In the context of planning problems the terminals $\Sigma = \mathit{Ops}$ are simply the operators. We now define the new planning formalism where the solutions are restricted by an ILP.

Definition 5.2 (Planning with Linear Constraints)**.** A *planning problem with linear constraints* $\Pi_L = (\Pi, \mathbb{L})$ consists of a planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ and an ILP $\mathbb{L}$ with $|\mathit{Ops}|$ columns. The plans for Π_L are:

$$\mathrm{sol}(\Pi_L) := \{\pi \in \mathrm{sol}(\Pi) \mid \Psi(\pi) \in \mathrm{sol}(\mathbb{L})\}$$

The set of all planning problems with linear constraints is denoted by $\mathcal{ILP}$.

Linear constraints over action counts, even in a very simple form, are useful for modeling (Lauer 2025). E.g., different money-earning and money-spending operators may earn or spend different amounts of money. A linear constraint can express the exact amounts $w_i, w'_i \in \mathbb{Q}$ to ensure a profit:

$$w_1 \cdot \mathit{spend}_1 + ... + w_l \cdot \mathit{spend}_l \;\le\; w'_1 \cdot \mathit{earn}_1 + ... + w'_m \cdot \mathit{earn}_m$$

We now show that plan existence for this formalism is PSPACE-complete by encoding it into numerical planning.

Theorem 5.3. *Deciding whether there exists a plan for a classical planning problem with linear constraints is PSPACE-complete.*

Proof. PSPACE-hardness follows by reduction from classical planning, using the linear constraint $0x = 0 \equiv$ true.

For membership, we encode classical planning problem with linear constraints $\Pi_L = (\Pi, (M, v))$ with $M \in \mathbb{Z}^{m \times n}$ into numeric planning formalism $(C_=, C_\varnothing, E_{\pm c})$, which is PSPACE-complete by Thm. 4.7. If more expressive goal conditions (e.g., supporting multiplication) were allowed, one could simply count operator applications and verify $Mx = v$ line by line in the goal, as in Thm. 4.9. Without such expressiveness, we instead simulate the constraint via auxiliary variables and operators. Specifically, for each constraint line $i \in \{1, \dots, m\}$, we introduce one variable to track each term $M_{i,j} \cdot x_j$ using constant effects, and n auxiliary variables $c_{i,j}$ such that their sum must equal v_i, i.e., $\sum_{j=1}^{n} c_{i,j} = v_i$, enforced by the goal. The goal further requires that each $c_{i,j}$ matches the corresponding value tracking $M_{i,j} \cdot x_j$. The goal conditions imply the original linear constraint $Mx = v$. To allow reaching a valid assignment of values to the $c_{i,j}$ that satisfies the condition, we add distribution actions that transfer units between these variables. This ensures that the constraint can be satisfied if and only if the original system $Mx = v$ holds. ☐

Theorem 5.4. *$\mathcal{ILP}$ is is significantly more expressive than classical planning.*

Proof. The hardness encoding from classical planning in the proof of Thm. 5.3 shows that $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{ILP})$ by allowing arbitrary sequences without constraints. To prove that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are included we extend the sequential planning task Π^{Seq} from Def. 4.8 to a planning problem with linear constraints $\Pi_L = (\Pi, \mathbb{L})$ with $\mathbb{L} = (M, v)$ and:

$$(1) \quad M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{for } \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

$$(2) \quad M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{for } \{a^n b^n \mid n \in \mathbb{N}\}$$

$$(3) \quad M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} \quad \text{for } \{aa\}$$

(1) enforces $x_a = x_b \wedge x_b = x_c$, so only plans with equal counts of a, b, c are accepted. (2) enforces $x_a = x_b \wedge x_c = 0$, so only plans with equal counts of a, b but no c are accepted. (3) admits plans where a occurs twice, and b, c not at all. ☐

6 Classical Planning with Presburger Constraints

We now introduce the next new planning formalism. We extend classical planning by allowing to constrain plans by existentially quantified Presburger formulas. Presburger formulas can be seen as an extension of ILPs, as they additionally allow disjunctions among constraints, but come with the drawback that constants are represented in unary. E.g., 3 is encoded as $1 + 1 + 1$. The connection to ILPs forms the next link in the chain connecting all planning formalisms presented in this paper. Accordingly, we will encode this formalism into ILP constraints for the membership proof. To make use of the work of Verma, Seidl, and Schwentick (2005) in the next section, we reproduce their definition of existentially quantified Presburger formulas, which in turn originates from Seidl et al. (2004).

Definition 6.1 (Presburger Formula)**.** An *existential Presburger formula* ϕ over a set of variables $X = \{x_1, ..., x_n\}$ is a string derived from the context-free grammar with start symbol ϕ_N and production rules:

$$x \to x_1 \mid ... \mid x_n$$

$$t \to 0 \mid 1 \mid x \mid (t + t)$$

$$\phi_N \to (t = t) \mid (t < t) \mid (\phi_N \wedge \phi_N) \mid (\phi_N \vee \phi_N) \mid (\exists x : \phi_N)$$

Here, x, t, and ϕ_N are non-terminals, while all other symbols are terminals. The derived string, i.e. the Presburger formula ϕ, expresses a first-order logic formula with constants 0, 1 and variables x_1, ..., x_n that can be interpreted over the structure $(\mathbb{N}, \leq, +)$. Formally, variables are assigned values via a substitution function $\sigma\colon X \rightarrow \mathbb{N}$. If the resulting ground formula is satisfied, we write $\sigma \vDash \phi$. The solution set $\text{sol}(\phi) := \{\sigma\colon X \rightarrow \mathbb{N} \mid \sigma \vDash \phi\}$ contains all substitutions satisfying ϕ.

Like with ILPs, these formulas simply restrict the amount of operators that occur in the solutions, via the Parikh vector.

Definition 6.2. A classical planning problem with Presburger constraints $\Pi_\phi = (\Pi, \phi)$ consists of classical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ and Presburger formula ϕ. The set of all plans for Π_ϕ is:

$$\text{sol}(\Pi_L) := \{\pi \in \text{sol}(\Pi) \mid \exists \sigma \in \text{sol}(\phi) \\ \forall o \in \mathit{Ops} : \Psi(\pi)_o = \sigma(o)\}$$

The set of all planning problems with Presburger constraints is denoted by $\mathcal{PRESBURGER}$.

We now prove PSPACE-completeness by converting the Presburger formula non-deterministically into an ILP.

Theorem 6.3. *Deciding whether there exists a plan for a classical planning problem with Presburger constraints is PSPACE-complete.*

Proof. Hardness is shown by reduction from Classical planning, which is PSPACE-complete (Erol, Nau, and Subrahmanian 1995). We reduce a classical planning problem Π to a classical planning problem with Presburger constraints $\Pi_L = (\Pi, \phi)$, with trivially true formula $\phi = (0 < 1)$,

For membership, we follow Verma, Seidl, and Schwentick (2005), to encode a Presburger formula ϕ into an ILP. We use a non-deterministic transformation that removes all disjunctions $\phi_1 \vee \phi_2$ by non-deterministically choosing either ϕ_1 or ϕ_2. Let ϕ' be the resulting formula after all such choices. Then ϕ' contains only conjunctions, atomic comparisons, and existential quantifiers. We move all existential quantifiers to the front, yielding:

$$\phi' \equiv \exists x_1 : ...\exists x_n : \bigwedge_{i=1}^{m} \psi_i$$

where all ψ_i are quantifier-free and atomic formulas of shape $t_1 = t_2$ or $t_1 < t_2$. The structures can be encoded in an ILP (M, v). M has n columns so that each row can match constants $c_1, ..., c_n \in \mathbb{Q}$ to the values variables $x_1, ..., x_n$ would take. Each such row, together with a value c_0 of v, forms a constraints of shape $\sum_{i=1}^{n} c_i \cdot x_i \leq c_0$, which can be generated from the Presburger formulas: Given that Presburger formulas are interpreted over $(\mathbb{N}, \leq, +)$ we can use standard algebraic rewriting to bring $t_1 < t_2$ to shape $\sum_{i=1}^{n} c_i^* \cdot x_i < c_0^*$. Here $c_i^* \cdot x_i$ represents $c_i^* \in \mathbb{N}$ additions of x_i and c_0^* is a sum of constants 0, 1. We can represent this as linear constraint $\sum_{i=1}^{n} c_i^* \cdot x_i \leq c_0^* - 1$. For $t_1 = t_2$ we use the same rewriting, but generate $\sum_{i=1}^{n} c_i \cdot x_i \leq c_0 \wedge \sum_{i=1}^{n} c_i \cdot x_i \geq c_0$ where $\geq$ can be represented by negating the according constants in M.

If ϕ is satisfiable by some plan, then for every disjunction in the formula, at least one part must be satisfied. Any other transformation we do preserves equivalence. Therefore, if there exists a plan satisfying ϕ, there is at least one such transformation that results in a satisfiable ILP instance. Moreover, if the ILP instance is satisfiable, this guarantees that for each disjunction in the original formula, at least one part is satisfiable. This concludes correctness. □

We now turn to expressivity, showing $\mathcal{PRESBURGER}$ is more expressive than classical planning.

Theorem 6.4. *$\mathcal{PRESBURGER}$ is significantly more expressive than classical planning.*

Proof. The hardness encoding from classical planning in the proof of Thm. 6.3 shows that $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{PRESBURGER})$ by allowing arbitrary sequences without constraints. To prove that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are included we use the sequential operator gadget Π^{Seq} from Def. 4.8 and extend it with the following Presburger formula ϕ over variables x_a, x_b, x_c to planning problem with Presburger constraints $\Pi_\phi = (\Pi^{\text{Seq}}, \phi)$:

(1) $(x_a = x_b) \wedge (x_b = x_c)$ for $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
(2) $(x_a = x_b) \wedge (x_c = 0)$ for $\{a^n b^n \mid n \in \mathbb{N}\}$
(3) $(x_a = (1{+}1)) \wedge (x_b = 0) \wedge (x_c = 0)$ for $\{aa\}$

(1) accepts exactly the Parikh images where a, b, c occur equally often. (2) accepts the Parikh images where a, b occur equally often, but no c. And (3) accepts only plans where a occurs twice and b, c not at all. □

7 Classical Planning with Grammar Amount Constraints

We now introduce the final new formalism before closing the gap to (unordered) HTN planning. It again constrains the number of operator occurrences in a solution, this time by requiring that a context-free grammar can derive a word with the same Parikh vector, i.e., the same number of terminal occurrences matching an operator. As before, we will encode this formalism into the previously introduced one, continuing the next link in the chain connecting all presented planning formalisms.

Definition 7.1. A *classical planning problem with grammar amount constraints* $\Pi_G = (\Pi, G)$ consists of classical planning problem $\Pi = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ and context-free grammar G with terminals Ops. The plans for Π_G are:

$$\text{sol}(\Pi_G) := \{\pi \in \text{sol}(\Pi) \mid \exists L \in L(G) : \Psi(\pi) = \Psi(L)\}$$

$\mathcal{GAMOUNT}$ is the set of all planning problems with grammar amount constraints.

Again, we show that plan existence under this formalism matches the complexity of classical planning.

Theorem 7.2. *Deciding whether there exists a plan for a classical planning problem with grammar amount restrictions is PSPACE-complete.*

Proof. Hardness is shown by reduction from Classical planning, which is known to be PSPACE-complete (Erol, Nau, and Subrahmanian 1995). This is, we can simply reduce a planning problem Π to classical planning problem with grammar amounts, where the grammar amounts are arbitrary. This can be encoded in a context-free grammar with start symbol S and production rules:

$$S \to S\, o_1 \mid ... \mid S\, o_n \mid \varepsilon$$

Where the terminal $Ops = \{o_1, ..., o_n\}$ represent the operators in the problem and S is the only non-terminal.

For membership, we encode classical planning problem with grammar amount constraints $\Pi_G = (\Pi, G)$ into a classical planning problem with Presburger constraints $\Pi_\phi = (\Pi, \phi)$, by using the result from Verma, Seidl, and Schwentick (2005), stating that there is a linear time construction that encodes a predicate matching the Parikh vector of a CFG G into existential Presburger Logic ϕ. So, by Thm. 6.3 we can use the PSPACE verifier for Π_ϕ. □

It is easy to prove that $\mathcal{GAMOUNT}$ is signficantly more expressive than classical planning. We omit the proof due to space constraints. It is akin to the construction in Thm. 8.2.

8 Unordered HTN Planning

We now analyze unordered HTN planning and show that plan existence is PSPACE-complete. So far, we knew that deciding plan existence is EXPTIME for total-ordering (Erol, Hendler, and Nau 1996), and Ackermann-complete for some restricted partial orders (Dekker and Behnke 2024). PSPACE fragments had only been identified by restricting the hierarchy (Alford, Bercher, and Aha 2015), which is less desirable for modeling. Unordered HTN was studied only for lifted plan verification (Lauer, Lin, and Bercher 2025), but its plan existence complexity was unknown. We close this gap by encoding the hierarchy into a context-free grammar, completing the chain of encodings.

Theorem 8.1. *Deciding whether there exists a plan for an unordered HTN planning problem is PSPACE-complete.*

Proof. For hardness we use a reduction from Classical planning, which is PSPACE-complete. To simulate a classical planning problem with HTN planning we use the construction from Erol, Hendler, and Nau (1996, Sec. 3.3). This is to have one compound task that can be decomposed into an arbitrary operator or twice this compound task. On this construction one can impose arbitrary ordering constraints, without restricting the operator refinements, as any arbitrary sequence remains to be a refinement.

From the HTN structure we create a context-free grammar with start symbol S and the following production rules: There is one production rule for the initial task network $tn_I = (T, \alpha)$ with task IDs $T = \{tid_1, ..., tid_n\}$:

$$S \to \alpha(tid_1)\ ...\ \alpha(tid_n)$$

And one production rule per method $m \in \mathcal{M}$ to replace the head of $m = (c, tn)$ with $tn = (T, \alpha)$, $T = \{tid_1, ..., tid_n\}$:

$$c \to \alpha(tid_1)\ ...\ \alpha(tid_n)$$

The terminals are Ops and non-terminals $\mathcal{C}$.

First observe that by our definition, of terminals and non-terminals each derived string only contains operators. Now, we can inductively observe (starting with initial task network, to $n \in \mathbb{N}$ derivation steps) that each application of a production rule mirrors the replacement by a method in a way, so that the amount of non-terminals, representing compound tasks, and terminals, representing operators, are the same as by the method replacement. This concludes that the amounts and so the Parikh image of the HTN structure and constructed CFG are the same, allowing us to construct a planning problem with grammar amounts and exploit the PSPACE-verifier from Thm. 7.2. □

At first, it may seem that unordered HTN is restrictive, since orders between operators are very natural. However, these orders do not necessarily need to be enforced through task network constraints. We implicitly show this in the following by encoding languages such as $\{a^n b^n \mid n \in \mathbb{N}\}$, where a must precede b, into unordered HTN.

Theorem 8.2. *$\mathcal{HTN}_\mathcal{U}$ is significantly more expressive than classical planning.*

Proof. $L(\mathcal{CLASSIC}) \subseteq L(\mathcal{HTN}_\mathcal{U})$ follows from the encoding to classical planning in the proof of Thm. 5.3 where arbitrary classical planning problems can be encoded in HTN planning with matching plans.

To show that $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{aa\}$ are in $L(\mathcal{HTN}_\mathcal{U})$, we construct unordered HTN planning tasks $\Pi_\mathcal{H} = (\Pi^{\text{Seq}}, \mathcal{H})$. The hierarchy $\mathcal{H} = (tn_I, \mathcal{M}, \mathcal{C})$ contains one compound task, i.e., $\mathcal{C} = \{S\}$ and the initial task network as $tn_I = (\{t_0\}, \{t_0 \mapsto S\})$. We define methods:

(1) For $\{a^n b^n c^n \mid n \in \mathbb{N}\}$:

$$m_1 = (S, (\{\}, \{\}))$$
$$m_2 = (S, (\{t_1, t_2, t_3, t_4\}, \{t_1 \mapsto a, t_2 \mapsto b, t_3 \mapsto c, t_4 \mapsto S\}))$$

(2) For $\{a^n b^n \mid n \in \mathbb{N}\}$:

$$m_1 = (S, (\{\}, \{\}))$$
$$m_2 = (S, (\{t_1, t_2, t_3\}, \{t_1 \mapsto a, t_2 \mapsto b, t_3 \mapsto S\}))$$

(3) For $\{aa\}$:

$$m = (S, (\{t_1, t_2\}, \{t_1 \mapsto a, t_2 \mapsto a\}))$$

In (1) every decomposition of S adds the same number of a, b, c; making it $\{a^n b^n c^n\}$. In (2) every decomposition of S adds the same number of a, b; making it $\{a^n b^n\}$. (3) yields the task network with two operators a, matching $\{aa\}$. □

Together, the theorems underline why unordered HTN can combine the hierarchy guidance with state-of-the-art planning techniques. E.g., consider the language represented by:

$$(drive\_to_1 \mid \dots \mid drive\_to_m)^n\ load$$
$$(drive\_back_1 \mid \dots \mid drive\_back_m)^n\ unload$$

Here, a truck drives, loads a package, and returns. Encoding this similar to $a^n b^n$ allows counting the remaining $drive\_back$ (and $unload$) operators in the task network after $load$, to get the exact remaining plan length. But, there is no additional guidance to reach the load step. To provide that, one can likely use techniques from classical planning (Hoffmann and Nebel 2001; Richter and Westphal 2010; Seipp and Helmert 2018), as the plan existence is also PSPACE-complete. From a practical perspective, this is a major advantage, as there are many ongoing lines of research to improve planners for classical planning (Corrêa et al. 2020; 2021; 2022; Lauer et al. 2020; 2021; 2025a; 2025b; Chen et al. 2024a; 2024b; Tollund et al. 2025).

Similar benefits likely exist for the other formalisms. But HTNs are already well-studied, making it easier to reuse existing knowledge and to improve existing applications.

9 Conclusion

We have analyzed five planning formalisms that are significantly more expressive than classical planning, without increasing the complexity of plan existence. Our results open new directions for modeling, both through the introduction of novel formalisms and by highlighting the low complexity of unordered HTN planning. The latter findings strongly advocate a shift towards unordered HTN planning problems.

Acknowledgment

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – ProjectID 232722074 – SFB 1102. Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight Bounds for HTN Planning. In *Proc. of the 25th ICAPS*, 7–15.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Comp. Intell.*, 11: 625–656.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning domain-independent heuristics for grounded and lifted planning. In *Proc. of the 38th AAAI*, 20078–20086.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proc. of the 34th ICAPS*, 68–76.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In *Proc. of the 31st ICAPS*, 94–102.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *Proc. of the 30th ICAPS*, 80–89.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In *Proc. of the 36th AAAI*, 9716–9723.

Dekker, M.; and Behnke, G. 2024. Barely Decidable Fragments of Planning. In *Proc. of the 27th ECAI*, 4198–4206.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Ann. Math. Artif. Intell.*, 18(1): 69–93.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *AIJ*, 76(1-2): 75–88.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proc. of the 2nd IJCAI*, 608–620.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proc. of the 22nd IJCAI*, 1955–1961.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proc. of the 6th AIPS*, 44–53.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AIJ*, 173: 503–535.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14: 253–302.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proc. of the 21st ECAI*, 447–452.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages. In *Proc. of the 26th ICAPS*, 158–165.

Lauer, P. 2025. Arguments in Favor of Allowing a Modeler to Constrain Action Repetitions. In *Proc. of KEPS at ICAPS*.

Lauer, P.; and Fickert, M. 2020. Beating LM-cut with LM-cut: Quick Cutting and Practical Tie Breaking for the Precondition Choice Function. In *Proc. of the 12th HSDIP at ICAPS*.

Lauer, P.; and Fišer, D. 2025. Potential Heuristics: Weakening Consistency Constraints. In *Proc. of the 35th ICAPS*.

Lauer, P.; Lin, S.; and Bercher, P. 2025. Tight Bounds for Lifted HTN Plan Verification and Bounded Plan Existence. In *Proc. of the 35th ICAPS*.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proc. of the 30th IJCAI*, 4119–4126.

Lauer, P.; Torralba, Á.; Höller, D.; and Hoffmann, J. 2025. Continuing the Quest for Polynomial Time Heuristics in PDDL Input Size: Tractable Cases for Lifted hAdd. In *Proc. of the 35th ICAPS*.

Lin, S.; and Bercher, P. 2022. On the Expressive Power of Planning Formalisms in Conjunction with LTL. In *Proc. of the 32nd ICAPS*, 231–240.

Papadimitriou, C. H. 1981. On the complexity of integer programming. *JACM*, 28: 765–768.

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

Seidl, H.; Schwentick, T.; Muscholl, A.; and Habermehl, P. 2004. Counting in Trees for Free. In *Proc. of the 31st ICALP*, 1136–1149.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.

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 Magazine*, 45: 280–296.

Tollund, R. G.; Larsen, K. G.; and Torralba, A. 2025. What Makes You Special? Contrastive Heuristics Based on Qualified Dominance. In *Proc. of the 34th IJCAI*, 8652–8660.

Verma, K. N.; Seidl, H.; and Schwentick, T. 2005. On the Complexity of Equational Horn Clauses. In *Proc. of the 20th CADE*, 337–352.