I Always Told My Mom That Order Is Overrated: Unordered HTN Planning is in PSPACE and Models Problems Beyond STRIPS

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

Hierarchical Task Network planning is (in)famous for offering great modeling power but making plan existence undecidable. To lift the computational barrier, it is common to enforce orders between all tasks, which allows for an EXPTIME computation. We show that the opposite extreme, using no ordering constraints, reduces complexity further to PSPACE. This unites benefits of STRIPS and HTN planning in a single formalism: It matches the complexity of STRIPS planning while provably giving modelers more expressive power. We observe the same benefit for more, partially new, planning formalisms along the way to the main result. This motivates that these formalisms have similar merit on their own.

1 Introduction

A planning formalism defines a format users can use to input planning problems into a planner. Naturally, there are two opposing perspectives: (1) The planner prefers a description that is easy to solve. This is usually realized through strong restrictions. A widely used example is STRIPS (Fikes and Nilsson 1971). Here, actions are restricted to trivial conditions and effects on propositional states. (2) Users desire more flexibility to express a problem in a concise way. A prominent example, providing more flexibility, is Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996) which allows the user to constrain plans through a hierarchical task structure.

HTN planning can be useful for encoding knowledge, or problems, that are impossible or difficult to encode in STRIPS planning (Wichlacz, Torralba, and Hoffmann 2019; Jamakatel et al. 2023). But, its expressiveness comes at a cost. Deciding if there exists a plan is undecidable for HTN planning (Erol, Hendler, and Nau 1996), whereas it is just PSPACE-complete for (propositional) STRIPS (Bylander 1994). It is clearly desirable to have a formalism that combines both benefits, i.e., it remains computationally tractable while offering greater modeling flexibility. A good example, showing that this is possible, is totally ordered HTN planning, which imposes an 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 reflects in empirical results. 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 consider the opposite extreme. We show that unordered HTN planning, which does not allow any ordering constraints, 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. We improve the result by showing that plan existence in this fragment is PSPACE-complete. To connect unordered HTN planning to the numerical planning fragment, we introduce three new planning formalisms that add constraints on STRIPS plans through: (1) Linear Constraints, (2) Presburger formulas and (3) Counting constraints over context-free grammars. They allow establishing PSPACE membership by 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 STRIPS.

The decreased complexity is desirable for solving tasks, but only guarantees that there is a problem representation that captures *one* plan. A natural encoding, a modeler would create to reflect the real world, would capture *all* plans. To measure if all plans can be represented, we use a language analysis (Höller et al. 2014; 2016; Lin and Bercher 2022). Our results show that unordered HTN planning, as well as the other four analyzed formalisms, can express classes of problems that STRIPS cannot. Since finding a plan in these formalisms remains in PSPACE, while allowing to model more problems, they may serve as reasonable alternatives when STRIPS is too restrictive. This motivates further research on these formalisms, in particular on unordered HTN planning, to better understand their advantages and how they can be used effectively. We conclude with a discussion that outlines why making use of these theoretical advantages appears feasible in practice.

2 Background

Numerical (and Classical) Planning We closely follow Helmert (2002) to introduce numerical planning. We keep the introduction to one fragment we coin *bounded counter (BC) numerical planning*. Here, variables act as counters that can be incremented/decremented by a constant. The

goal conditions can bound a variable with a constant, or one other variable. But, there are no numerical preconditions. In Helmert's notation this would be the formalism $(C^c_=, \varnothing, E_{\pm c})$ where $C^c_= := C_c \cup C_=$ combines the rational functions

$$C_c := \{x \mapsto x - c \mid c \in \mathbb{Q}\} \text{ and } C_= := \{(x_1, x_2) \mapsto x_1 - x_2\}.$$

Definition 2.1 (Numerical Planning Problem)**.** A *BC numerical planning problem* $\Pi = (V_P, V_N, Init, Goal, Ops)$ 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. $Init = (\alpha_{Init}, \beta_{Init})$ is a state, called *initial state*.

A *propositional condition* is a propositional variable $v \in V_P$ denoted by $v = \top$. A *numerical condition* is $f(v_1, ..., v_n)\ relop\ 0$ where $v_1, ..., v_n \in V_N$ are numerical variables, $f \in C^c_=$, and $relop \in \{=, <, \leq, \geq, >, \neq\}$.

A *propositional effect* is $v \leftarrow t$ for a propositional variable $v \in V_P$ and a truth value $t \in \{\top, \bot\}$. A *numerical effect* is $v \leftarrow v + c$ for a numerical variable $v \in V_N$ and constant $c \in \mathbb{Q}$. Here v is called *assigned* variable. An operator $o = (pre, \mathit{eff})$ consists of two finite sets pre and eff, where: pre contains propositional conditions (and no numerical conditions) and eff contains propositional effects eff_P and numerical effects eff_N with pairwise distinct assigned variables. Ops is a finite set of operators. The *goal condition* $Goal$ is a finite set containing propositional and numerical conditions.

Note that propositional STRIPS (Bylander 1994) matches BC numerical planning without numerical variables.

Definition 2.2 (STRIPS Planning)**.** A BC numerical planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$ with $V_N = \varnothing$ is called STRIPS planning. The set of all STRIPS planning problems is $\mathcal{STRIPS}$.

We start to define propositional plans only, following the standard notation by Hoffmann and Nebel (2001).

Definition 2.3 (Propositional Plans)**.** For numerical planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$, operator $o = (pre, \mathit{eff}) \in Ops$ is *applicable* in propositional state $\alpha \in S_\alpha$ iff $\alpha(v) = \top$ for all conditions $v \in pre$. In this case the *propositional successor* replaces 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.

The repeated application $progr(progr(\alpha, o_1), ..., o_n)$ of operator sequence $\pi = o_1, ..., o_n$ is denoted by $progr(\alpha, o_1, ..., o_n)$. If π reaches $\alpha' := progr(\alpha, o_1, ..., o_n)$ from $Init = (\alpha, \beta)$ so that $\alpha'(v) = \top$ for all conditions $v \in Goal$, then π is a *propositional plan*. A propositional plan for a STRIPS planning problem is also just called *plan*.

We now add the numerical progression for BC numerical planning (Helmert 2002, Alg. 22). The result is computed by accumulating all increments per variable.

Definition 2.4 (BC Progression)**.** Let a BC numerical planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$, an operator sequence $\pi = o_1, ..., o_n$ applied in numerical state β reaches numerical state $progr(\beta, \pi) := \beta'$ defined as $\beta'(v) = \beta(v) + \sum_{i \in \{1,...,n\}} incr_v(o_i)$ for $v \in V_N$. Here, $incr_v(o) = c \in \mathbb{Q}$ if operator o has an effect $v \leftarrow v + c$, and 0 otherwise.

Let π be an operator sequence and $\beta' = progr(\beta_{Init}, \pi)$. π is a *plan* for Π if it is a propositional plan and every numerical goal condition $f(v_1, ..., v_m)\ relop\ 0 \in Goal$ is satisfied in β', i.e., $f(\beta'(v_1), ..., \beta'(v_m))\ relop\ 0$ is true.

We define $progr(s, \pi) = (progr(\alpha, \pi), progr(\beta, \pi))$ for state $s = (\alpha, \beta)$. The set $\mathrm{sol}(\Pi)$ consists of all plans in Π.

Unordered HTN Planning We introduce HTN planning like Geier and Bercher (2011), but remove all ordering relations to make it unordered. In turn, our formalism does not consider method preconditions. This fits: The IPC semantics (Höller et al. 2020) define method preconditions through an ordering relation at the beginning of a method's task network

Definition 2.5 (Unordered HTN planning problem)**.** An *unordered HTN planning problem* $\Pi_\mathcal{H} = (\Pi, \mathcal{H})$, consists of a STRIPS planning problem $\Pi = (V_P, V_N, Ops, Init, Goal)$ and the hierarchy $\mathcal{H} = (tn_I, \mathcal{M}, \mathcal{C})$ modeling hierarchical restrictions to operator solutions. $\mathcal{C}$ is the finite set of *compound tasks*. The finite set of operators Ops is also called *primitive tasks*. A *task network* $tn = (T, \alpha)$ consists of a finite set of *task IDs* T and a map $\alpha : T \to Ops \cup \mathcal{C}$.

A *method* $m = (c_m, tn_m)$ can 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 $\to_m$ between task networks. $tn_1 \to_m tn_2$ holds for $tn_1 = (T_1, \alpha_1)$ 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, and tn_2 is constructed as:

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

$\mathcal{M}$ denotes the set of all methods in $\Pi_\mathcal{H}$. The relation $\to_\mathcal{M}$ is the union of all relations $\to_m$ for $m \in \mathcal{M}$. Its 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:

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

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 problem is a primitive task network obtained by decomposing tn_I, which has a linearization that is a propositional plan, 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.

Context-Free Grammars, Linear Constraints, and Presburger Formula We now introduce what we will use in the next section to express constraints on plans.

Definition 2.6 (Context-Free Grammar)**.** A *context-free grammar (CFG)* $G = (N, \Sigma, S, R)$ 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 $A \to \alpha_1 \mid ... \mid \alpha_n$, where $A \in N$ is a non-terminal and all $\alpha_1, ..., \alpha_n \in (N \cup \Sigma)^*$ 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 2.7 (Grammar Language)**.** For a CFG $G = (N, \Sigma, S, R)$, a sequence $s = s_0 s_1 ... s_n \in (N \cup \Sigma)^*$ can *derive* a sequence $t \in (N \cup \Sigma)^*$, denoted $s \Rightarrow t$, by replacing some symbol s_i for $i \in \{0, ..., n\}$ with α_j for some $j \in \{1, ..., m\}$ 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 S:

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

We define linear constraints following Papadimitriou (1981). In our notation $\mathbb{N}$ includes 0.

Definition 2.8. Linear constraints $\mathbb{L} = (M, v)$ are 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 $\mathrm{sol}(\mathbb{L}) := \{x \in \mathbb{N}^n \mid Mx = v\}$.

We define existentially quantified Presburger formulas following Seidl et al. (2004) and Verma, Seidl, and Schwentick (2005). Intuitively speaking, Presburger constraints are linear constraints extended with disjunctions, where all constants are written in unary. E.g., 3 is 1 + 1 + 1.

Definition 2.9 (Presburger Formula)**.** A *Presburger formula* ϕ over variables $X = \{x_1, ..., x_n\}$ is a string derived from the CFG with start symbol ϕ_N and production rules:

$$x \to x_1 \mid ... \mid x_n, \quad t \to 0 \mid 1 \mid x \mid (t + t),$$
$$\phi_N \to (t = t) \mid (t < t) \mid (\phi_N \wedge \phi_N) \mid (\phi_N \vee \phi_N) \mid (\exists x : \phi_N)$$

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 $\sigma: X \to \mathbb{N}$. If the resulting ground formula is satisfied, we write $\sigma \vDash \phi$. The solution set $\mathrm{sol}(\phi) := \{\sigma: X \to \mathbb{N} \mid \sigma \vDash \phi\}$ contains all substitutions satisfying ϕ.

3 New Planning Formalisms

We introduce new planning formalisms to step-wise transform unordered HTN constraints to alternative plan constraints, until we reach BC numerical planning. All constraints are w.r.t. operator counts in a plan. This is motivated by the fact that in unordered HTNs only the counts of tasks in a task network matters. We count terminals in a word using Parikh (1966) vectors:

Definition 3.1. Let Σ be a finite set of terminals. The *Parikh vector* $\Psi(w)$ of a sequence $w = e_1 e_2 ... e_{|w|}$ over terminals Σ is a vector in $\mathbb{N}^{|\Sigma|}$, where each entry $\Psi(w)_i$ at position $i \in \{1, ..., |\Sigma|\}$ counts the occurrences of a terminal, i.e.:

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

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 $\Sigma = Ops$ are its operators. Thus, a Parikh vector contains one entry for the count of each operator. This matches the solution vector of a linear constraint over operator counts. We use this observation to introduce the first formalism.

Definition 3.2. A *STRIPS planning problem with linear constraints* $\Pi_L = (\Pi, \mathbb{L})$ consists of a planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$ and an linear constraints $\mathbb{L}$ with $|Ops|$ 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 STRIPS planning problems with linear constraints is denoted by $\mathcal{LIN}$.

Linear constraints are closely related to the goal constraints in BC numerical planning. But it is cumbersome to express disjunctions in linear constraints. Therefore, the next planning formalism constrains operator counts by a Presburger formula. We assign a variable to each operator, representing its count.

Definition 3.3. A STRIPS planning problem with Presburger constraints $\Pi_\phi = (\Pi, \phi)$ consists of STRIPS planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$ and Presburger formula ϕ over variables $X \supseteq Ops$. The plans for Π_ϕ are:

$$\mathrm{sol}(\Pi_\phi) := \{\pi \in \mathrm{sol}(\Pi) \mid \exists \sigma \in \mathrm{sol}(\phi) \quad \forall o \in Ops : \Psi(\pi)_o = \sigma(o)\}$$

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

The last formalism constrains plans through terminal counts in words derived from a CFG. Verma, Seidl, and Schwentick (2005) show that these constraints can be expressed in Presburger Formulas. At the same time, grammars are closely related to (unordered) HTNs (Pantůčková, Ondrčková, and Barták 2023; Lin et al. 2023).

Definition 3.4. A *STRIPS planning problem with grammar amount constraints* $\Pi_G = (\Pi, G)$ consists of STRIPS planning problem $\Pi = (V_P, V_N, Init, Goal, Ops)$ and context-free grammar G with terminals Ops. The plans for Π_G are:

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

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

4 PSPACE-Completeness

In the following we prove PSPACE-completeness for the introduced planning formalisms. Hardness is trivial as all formalisms extend STRIPS planning. We establish membership by providing a PSPACE algorithm for a numerical planning fragment and then creating an encoding chain that ends in unordered HTN planning.

4.1 The Backbone: BC Numerical Planning

Helmert (2002) shows that plan existence in BC numerical planning is decidable[1]. The proof establishes an upper bound on the plan length. In the following we adapt the proof idea to establishing a tighter bound. The proof starts by considering a propositional plan which can be extended to a BC numerical plan by inserting propositional cycles.

Propositional cycles capture action sequences that start and end in the same propositional state. Simple propositional cycles visit no propositional state twice.

Definition 4.1. A propositional cycle $c = (\alpha_c, \pi_c)$ consists of a propositional state α_c and an operator sequence $\pi_c = o_1, ..., o_n$ so that $\alpha_c = progr(\alpha_c, \pi_c)$. c is called *simple* propositional cycle if for all $i \in \{1, ..., n-1\}, j \in \{i+1, ..., n\} : progr(\alpha, o_1, ..., o_i) \neq progr(\alpha, o_1, ..., o_j)$.

We say c occurs in an operator sequence π if there are (potentially empty) operator sequences $\pi_{before}, \pi_{after}$ so that $\pi = \pi_{before}, \pi_c, \pi_{after}$ and $progr(\alpha_{Init}, \pi_{before}) = \alpha_c$. Let π_{before} be the smallest operator sequence such that $\pi = \pi_{before}, \pi_c, \pi_{after}$. We say that we *remove* c from π resulting in $\pi' = \pi_{before}, \pi_{after}$, written as $\pi \ominus c =: \pi'$. Further, we say that we *insert* c into π' resulting in π, written as $\pi' \oplus c =: \pi$. If such $\pi_{before}, \pi_{after}$ do not exist, c can not be inserted in π'. If there is no cycle c so that c occurs in π, π is *acyclic*. We denote the empty cycle for the empty operator sequence ε with $c_\varepsilon = (\alpha_{Init}, \varepsilon)$.

We now construct the propositional plan that is extended to a BC numerical plan and make two additional observations. Addition (2.) bounds the plan length before inserting cycles. Addition (3.) is needed to bound the number of cycles insertions with linear constraints in the next proof step.

Lemma 4.2. *If there exists a plan π in BC numerical planning problem Π, there is a propositional plan π' in Π and simple propositional cycles $c'_1, ..., c'_n$ so that:*

1. *$\pi' \oplus c'_1 \oplus ... \oplus c'_n$ is a BC numerical plan*
2. *π' consists of at most $2^{|V_P|} + 2^{|V_P|} \cdot 2^{|V_P|}$ operators*
3. *Each c'_i for $i \in \{1, ..., n\}$ can be (directly) inserted into π'*

Proof. We construct π' from π by removing simple propositional cycles and only reinserting removed cycles if they visit a new state which is not visited before.

We start defining the removal: Let $\pi_0, ..., \pi_n$ be operator sequences and $c_1, ..., c_n$ simple propositional cycles so that: $\pi_0 = \pi$, π_n is acyclic, and for $i \in \{1, ..., n\}$: $\pi_i = \pi_{i-1} \ominus c_i$. This exists as any operator sequence is either acyclic, or there occurs a simple cycle in the operator sequence.

We now define the reinsertion: Let $visits(\alpha, \pi) = \{progr(\alpha, op_1, ..., op_i) \mid i \in 1, ..., k\} \cup \{\alpha\}$ denote the propositional states operators sequence $\pi = op_1, ..., op_k$ visits from propositional state α. Let $\pi'_n, ..., \pi'_0$ be a sequence so that $\pi'_n = \pi_n$ and for $i \in \{0, ..., n-1\}$: $\pi'_i = \pi'_{i+1}$ if $visits(\alpha_{Init}, \pi'_{i+1}) \subseteq visits(c_i)$ and $\pi'_i = \pi'_{i+1} \oplus c_i$ otherwise.

The insertion is always possible, as cycles are reinserted in reverse order. This ensures that if one cycle was removed from inside another, the outer cycle is added first, or another cycle is added instead, which visits the insertion point.

We show that setting $\pi' = \pi'_0$ proves the claim.

π'_0 is a propositional plan: π'_0 is created by only adding and deleting propositional cycles from propositional plan π.

π'_0 consists of at most $2^{|V_P|} + 2^{|V_P|} \cdot 2^{|V_P|}$ operators: π'_0 extends π_n, which contains $\leq 2^{|V_P|}$ operators as it is acyclic. Each increase is by at most $2^{|V_P|}$ operators, as it adds each one simple propositional cycle. An increase occurs at most $2^{|V_P|}$ times (when $\pi'_i \neq \pi'_{i-1}$), as there must be one of $\leq 2^{|V_P|}$ states that π'_i visits but π'_{i-1} does not.

Choose $c'_i = c_i$ if $\pi'_i = \pi_{i-1}$ and $c'_i = c_\varepsilon$ for $i \in \{1, ..., n\}$. Then, $\pi^* = \pi'_0 \oplus c'_1 \oplus ... \oplus c'_n$ is a BC numerical plan: π^* contains the same operators as π as the cycles are either reinserted by the choice of c'_i or the construction of π'. This means π^* creates the same numerical state as π, on top of it being a propositional plan, making it a BC numerical plan.

c'_i can $\in \{1, ..., n\}$ can be inserted into π': c'_i occurs in π and thus can be inserted in π. As $visits(\alpha_{Init}, \pi) = visits(\alpha_{Init}, \pi'_0)$, it can also be inserted in π'_0. □

To make use of linear constraints, we additionally assume that all goal states have variable values ≥ 0, like solutions in linear constraints.

Definition 4.3. We refer to BC^+ numerical planning for the subset $\mathcal{BCNUM}^+$ of BC numerical planning problems $\Pi = (V_P, V_N, Init, Goal, Ops)$ where the goal $Goal \subseteq \{v \geq 0 \mid v \in V_N\}$ forces all numerical values to be ≥ 0 in a goal state.

We will now upper bound the plan length in BC$^+$ numerical planning using Lem. 4.2.

Lemma 4.4. *There exists a polynomial function $p : \mathbb{N} \to \mathbb{N}$ so that for any solvable BC$^+$ numerical planning problem Π with encoding size $||\Pi||$ there is a plan consisting of at most $2^{p(||\Pi||)}$ operators.*

Proof. Let π', c'_i, n be defined as in Lem. 4.2. We prove that there are choices for $c'_1, ..., c'_n$ so that the number of distinct cycles n_c in $c'_1, ..., c'_n$ and the number n_r of times each such cycle is repeated are bounded by $2^{p'(||\Pi||)}$ for some polynomial $p' : \mathbb{N} \to \mathbb{N}$. By Lem. 4.2 we know that there are at most $2^{|V_P|} + 2^{|V_P|} \cdot 2^{|V_P|}$ operators in π'. Further, there are at most $2^{|V_P|}$ operators in each simple propositional cycle c'_i. I.e., the plan length of $\pi' \oplus c'_1 \oplus ... \oplus c'_n$ is at most:

$$2^{|V_P|} + 2^{|V_P|} \cdot 2^{|V_P|} + n_c \cdot n_r \cdot 2^{|V_P|}$$

Distinct Cycles (n_c): Let $c = (\alpha_c, \pi_c)$ and $d = (\alpha_d, \pi_d)$ be propositional cycles that can be inserted in a sequence of operators π so that π_d is a permutation of π_c. Then, $progr(\alpha_{Init}, \pi \oplus \pi_c) = progr(\alpha_{Init}, \pi \oplus \pi_d)$, as the propositional part of the resulting state is α_{Init} and permutations lead to the same numerical state (addition is commutative).

Thus, w.l.o.g., we can assume that $c'_1, ..., c'_n$ contains no $c'_i = (\alpha_{c'_i}, \pi_{c'_i})$ and $c'_j = (\alpha_{c'_j}, \pi_{c'_j})$ such that $c'_i \neq c'_j$ and $\pi_{c'_i}$ is a permutation of $\pi_{c'_j}$. (Otherwise we were able to replace c'_j with c'_i.) The number of operator sequences of length at

[1] The result is stated for $(C_c, \varnothing, E_{\pm c})$ and $(C_=, \varnothing, E_{\pm c})$, but the proof directly applies to $(C^c_=, \varnothing, E_{\pm c})$, i.e., BC numerical planning.

most $2^{|V^P|}$, up to permutation, is upper bounded by the number of multisets over Ops with at most $2^{|V^P|}$ elements. That is: $n_c \leq \left(2^{|V_P|}\right)^{|Ops|} = 2^{|V_P| \cdot |Ops|}$.

Cycle Repetitions (n_r): To determine the maximum number of times a cycle needs to be repeated, we adapt the constraints used by Helmert (2002, Alg. 22, part 3). We determine the values $\beta'(v)$ for all $v \in V_N$ for the state β' obtained by applying $\pi' \oplus c'_1 \oplus ... \oplus c'_n$ to β_{Init} as:

$$\beta'(v) = \beta_{Init}(v) + \sum_{i \in \{1,...,m\}} incr_v(o_i) + \sum_{i \in \{1,...,n\}} \Delta_v(c'_i) \cdot x_{c'_i}$$

where $\pi' = o_1, ..., o_m$, we reuse the notation of Def. 2.4, and extended it by $\Delta_v(c) = \sum_{i \in \{1,...,k\}} incr_v(op_i)$ to denote the change in variable v after applying propositional cycle $c = (\alpha, (op_1, ..., op_k))$. We consider $\beta'(v)$ for $v \in V_N$ and $x_{c'_i}$ for $i \in \{1, ..., n\}$ as constraint variables and add numerical goal constraints, over $\beta'(v)$. Helmert (2002) also adds constraints $x_{c'_i} \geq 0$ for $i \in \{1, ..., n\}$. We can omit this constrain by Lem. 4.2 (3.). Thus, we only consider $\leq |V_N| + |Goal|$ constraints and $\leq |V_N| + 2^{|V_P| \cdot |Ops|}$ constraint variables that can be transformed into linear constraints. Papadimitriou (1981) shows that all solution values, including the cycle repetitions n_r, are upper bounded by $2^{p'(\|\Pi\|)}$ for a polynomial $p' : \mathbb{N} \to \mathbb{N}$. A detailed conversion and construction of p' is provided in the Appendix (Lauer et al. 2026) for completeness. □

By upper-bounding plan length, we conclude that BC$^+$ numerical planning is in PSPACE using guess and check[2].

Proposition 4.5. *Plan existence for BC$^+$ numerical planning problems is PSPACE-complete.*

Proof. As we can explicitly construct the polynomial p from Lem. 4.4, there is a an algorithm that computes $2^{p(\|\Pi\|)}$ in PSPACE and then guesses an action and applies it to the current state up to $2^{p(\|\Pi\|)}$ times. We accept if the current state is the goal state. We only need to store one state at the time and the numerical values do not exceed an exponential number, thus the algorithm is NPSPACE = PSPACE.

PSPACE-hardness follows directly from STRIPS planning, which is PSPACE-complete (Bylander 1994) and a subset of BC$^+$ numerical planning. □

Another way to view our result is that, if we relax PNP (Dekker and Behnke 2024) by removing numerical preconditions and allowing negative variable values in non-goal states, the complexity drops from ACKERMANN-complete to PSPACE-complete. Shleyfman, Gnad, and Jonsson (2023) identify another PSPACE fragment for numerical planning with a similar proof structure. But, their fragment restrictions implicitly enforce $|\text{eff}(o)| \leq 1$ for operators o incrementing a numeric variable. This is unsuitable for our use case, as we create $|\text{eff}(o)| > 1$ to count operator applications.

[2]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, it also does not appear 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.

4.2 Encoding Chain for New Formalisms

We now start the encoding chain leading up to unordered HTN planning. The first encoding is from STRIPS planning with linear constraints to BC$^+$ numerical planning. One row in a linear constraint essentially represents the constraint $M_1x_1 + ... + M_nx_n = v$ where $x_1, ..., x_n$ are variables and $M_1, ..., M_n, v$ are constants. Our encoding tracks the value $M_1x_1 + ... + M_nx_n$ in one numerical variable lhs which is increased by M_i when the i-th operator is applied.

Proposition 4.6. *Plan existence for STRIPS planning problems with linear constraints is PSPACE-complete.*

Proof. For membership, we encode a STRIPS planning problem with linear constraints $\Pi_L = (\Pi, \mathbb{L})$, where $\mathbb{L} = (M, v)$ and $\Pi = (V_P, \varnothing, Init, Goal, Ops)$. We first assume M has one row, i.e., $M = (c_1, ..., c_n) \in \mathbb{Q}^{1 \times n}$ and $v \in \mathbb{Q}$, where $n = |Ops|$. We denote a solution vector as $x = (x_1, ..., x_n)$. I.e., $\mathbb{L}$ encodes $M_1x_1 + ... + M_nx_n = v$.

We use one numerical variable lhs that is initially assigned 0 via $Init_{LHS} = \{lhs \mapsto 0\}$. To adjust lhs' value, we construct $Ops' = \{o'_1, ..., o'_n\}$ for $Ops = \{o_1, ..., o_n\}$ s.t.

$$o'_i = (pre'_i, \mathit{eff}'_i) \text{ and } \mathit{eff}'_i = \mathit{eff}(o_i) \cup \{lhs \to lhs + M_i\}$$

for each $o_i = (pre_i, \mathit{eff}_i)$. We will enforce the constraint required by $\mathbb{L}$ through $Goal_N = \{lhs = v\}$. The final BC$^+$ numerical planning problem is constructed as:

$$(V_P, \{lhs\}, Init \cup Init_{LHS}, Goal \cup Goal_N, Ops')$$

For any state $s = (\alpha, \beta)$ it holds that $\beta(lhs) = M_1x_1 + ... + M_nx_n$ where x_i is the number of times the i-th operator is applied. Thus the goal enforces $M_1x_1 + ... + M_nx_n = v$.

To encode linear constraints with multiple rows, we can simply start to encode one line, rename the variables and respective dependencies to not be called lhs, rhs anymore and repeat until no constraint is left. PSPACE-hardness follows by reduction from STRIPS planning, using the linear trivially satisfied constraint $0x = 0$. □

We now encode a Presburger formula into a linear constraint.

Proposition 4.7. *Plan existence for STRIPS planning problems with Presburger constraints is PSPACE-complete.*

Proof. For membership, we follow Verma, Seidl, and Schwentick (2005), to encode a Presburger formula ϕ into linear constraints. 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 to get a formula $\exists x_1 : ... \exists x_n : \bigwedge_{i=1}^{m} \psi_i$ that is equivalent to ϕ', and all ψ_i are quantifier-free and atomic formulas of shape $t_1 = t_2$ or $t_1 < t_2$. This can be encoded in linear constraints by a standard transformation. We provide the transformation in the Appendix (Lauer et al. 2026) for completeness.

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 satisfiable linear constraints.

Hardness is shown by reduction from STRIPS planning, which is PSPACE-complete (Bylander 1994). We reduce a STRIPS planning problem Π to a STRIPS planning problem with Presburger constraints $\Pi_L = (\Pi, \phi)$ with trivially true formula $\phi = (0 < 1)$. □

We now encode a Presburger formula to linear constraints.

Proposition 4.8. *Plan existence for STRIPS planning problem with grammar amount constraints is PSPACE-complete.*

Proof. For membership, we encode a STRIPS planning problem with grammar amount constraints $\Pi_G = (\Pi, G)$ into a STRIPS planning problem with Presburger constraints $\Pi_\phi = (\Pi, \phi)$. To construct ϕ, we use 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 an (existential) Presburger formula ϕ. So, by Prop. 4.7 we can use the PSPACE verifier for Π_ϕ.

Hardness is shown by reduction from STRIPS planning problem Π with operators $Ops = \{o_1, ..., o_n\}$. Consider the context-free grammar G, where the start symbol S is the only non-terminal, there is one production rule

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

and terminals are Ops. Then, in the planning problem with grammar amounts (Π, G), grammar amounts are arbitrary. □

4.3 Reaching The End: Unordered HTN Planning

To conclude that plan existence for unordered HTN planning is PSPACE-complete, we exploit the similarities between the HTN structure and CFGs to complete the final link in the chain of encodings.

Theorem 4.9. *Plan Existence for unordered HTN planning problems is PSPACE-complete.*

Proof. For membership, 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 word 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 number of non-terminals, representing compound tasks, and terminals, representing operators, are the same as by the method replacement. This concludes that the Parikh image of the HTN structure and constructed CFG are the same, allowing us to construct a STRIPS planning problem with grammar amount constraints and exploit the PSPACE-verifier from Prop. 4.8.

For hardness we use a reduction from STRIPS planning, which is PSPACE-complete. To simulate a STRIPS planning problem with HTN planning we slightly adapt the construction of Erol, Hendler, and Nau (1996, Sec. 3.3). This is to have one compound task that can be decomposed into an arbitrary operator, twice this compound task, or no task. We do not enforce any ordering, thus this is an unordered HTN planning problem. Any action sequence can be obtained as linearization of a primitive task networks obtained by decomposing the initial task. Thus, the solution set matches exactly the one of the one of the STRIPS problem. □

Previous PSPACE fragments were obtained through strong restrictions on the hierarchy (Alford, Bercher, and Aha 2015; Zhang and Bercher 2025). Unordered HTNs already received some attention, e.g., for grounding (Behnke et al. 2020) and computing heuristics (Höller, Bercher, and Behnke 2020) when combined with the delete relaxation. But also from a theoretical viewpoint, where so far, the focus was mostly towards plan verification (Behnke, Höller, and Biundo 2015; Lauer, Lin, and Bercher 2025; Brand et al. 2025). However, the complexity of plan existence itself remained open. By closing this gap, we show that unordered HTN planning is, from a complexity standpoint, better than total-order HTN planning. But, the lowered complexity also tells us that there will be total-order problems that unordered HTN planning can not express.

This naturally raises the question of how the modeling power of unordered HTN planning compares to other PSPACE planning formalism. Thus, we addresses this question in the next section. We show that unordered HTN indeed can model relevant tasks and more than the commonly considered STRIPS planning formalism.

5 Expressivity Analysis

The complexity of plan existence measures how hard it is to find *one* plan. But, if a modeler encodes real world dynamics, they encode *all* plans. Encoding all plans is also a requirement when learning planning models: The usual objective is to match all real-world plans (Gösgens, Jansen, and Geffner 2025; Stern et al. 2025). In the following we measure how expressive a formalism is in this regard, by looking at sets of plans that problems in the formalism entail.

5.1 The Measure for Expressivity

We start by defining the language a problem entails. Previous analyses (Höller et al. 2014; 2016; Lin and Bercher 2022) define this as plans where operators are represented by arbitrary operator ids. To match this, we introduce *exchange functions* that map operators to arbitrary terminals. This allows for a comparison independent of the operator structure.

Definition 5.1 (Exchange Function)**.** For operators Ops the set $\mathrm{exch}(Ops)$ contains all bijections $\sigma : Ops \to \Sigma$ for any terminals Σ. We call its elements *exchange functions*. We apply σ to $\pi = o_1, ..., o_n \in Ops$ as $\sigma(\pi) := \sigma(o_1), ..., \sigma(o_n)$.

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

Definition 5.2. For $\mathcal{P} \in \{\mathcal{STRIPS}, \mathcal{PRESBURGER}, \mathcal{LIN}, \mathcal{GAMOUNT}, \mathcal{HTN}_{\mathcal{U}}, \mathcal{BCNUM}^{+}\}$ the set

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

is the *language class* of $\mathcal{P}$. Here Ops_{Π} are the operators of $\Pi \in \mathcal{P}$. And $L_{\sigma}(\Pi)$ is the *language* of Π under $\sigma \in \text{exch}(\mathit{Ops}_{\Pi})$. I.e., for $\mathcal{P} \neq \mathcal{HTN}_{\mathcal{U}}$:

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

For $\mathcal{P} = \mathcal{HTN}_{\mathcal{U}}$ with $\Pi = (\Pi', \mathcal{H})$ the language is:

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

To contrast the matching complexity of STRIPS with better expressiveness ,we ensure: (1) The formalism subsumes STRIPS planning. (2) The formalism admits languages beyond the expressive power of STRIPS.

Definition 5.3. A set of planning problems $\mathcal{P}$ is *significantly more expressive* than STRIPS if both:

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

Here a, b, c are terminals.

This implicitly establishes $L(\mathcal{STRIPS}) \subsetneq L(\mathcal{P})$. The reason is that the language class $L(\mathcal{STRIPS})$ is a strict subset of regular languages (Höller et al. 2014; 2016). As $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ and $\{a^n b^n \mid n \in \mathbb{N}\}$ are not regular, they are not in $L(\mathcal{STRIPS})$. $\{aa\}$ is an example of a regular language that is not in $L(\mathcal{STRIPS})$. The additionally included languages are common examples for a regular, context-free, and context-sensitive language, and thus serve as a compact proxy to demonstrate that $\mathcal{P}$ likely includes different types of languages. Intuitively, the languages capture the ability to count, which can be required in a planning problem. E.g., consider the language represented by:

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

modeling a finance problem where money-spending operators must be preceded by the same number of money-earning operators. The solution space for this problem cannot be captured by STRIPS. The solution space for this problem cannot be captured by STRIPS. But, it is simple to extend the following analysis, which considers less operators (a, b, and c).

5.2 Analyzing Expressivity

We will close the section by establishing the property from Def. 5.3 for all analyzed formalisms. To encode languages like $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, where all a must precede all b, and all b must precede all c we construct a task gadget in which this is always the case. 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.

Definition 5.4 (Sequential Operator Gadget)**.** The STRIPS planning problem $\Pi^{\text{Seq}} = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ 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 will now use constraints on top of the task gadget to establish the following result.

Theorem 5.5. *$\mathcal{LIN}$, $\mathcal{PRESBURGER}$, $\mathcal{GAMOUNT}$, $\mathcal{HTN}_{\mathcal{U}}$ and $\mathcal{BCNUM}^{+}$ are significantly more expressive than STRIPS.*

Proof. It is easy to observe that any $\mathcal{P} \in \{\mathcal{GAMOUNT}, \mathcal{PRESBURGER}, \mathcal{LIN}, \mathcal{HTN}_{\mathcal{U}}, \mathcal{BCNUM}^{+}\}$ fulfills $L(\mathcal{STRIPS}) \subseteq L(\mathcal{P})$. In the hardness proofs of Prop. 4.5, Prop. 4.6, Prop. 4.7, Prop. 4.8, and Thm. 4.9 respectively, we provided formal arguments that the formalisms extend STRIPS. This implies $L(\mathcal{STRIPS}) \subseteq L(\mathcal{P})$.

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 in $L(\mathcal{P})$ we extend Π^{Seq} from Def. 5.4 with constraints. We start with $\mathcal{HTN}_{\mathcal{U}}$ and construct unordered HTN planning problem $\Pi_{\mathcal{H}} = (\Pi^{\text{Seq}}, \mathcal{H})$. 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) $m_1 = (S, (\{\}, \{\}))$
$m_2 = (S, (\{t_1, t_2, t_3, t_4\}, \{t_1 \mapsto a, t_2 \mapsto b, t_3 \mapsto c, t_4 \mapsto S\}))$

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

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

In (1), every decomposition of S adds a, b, and c; making it $a^n b^n c^n$ for $n \in \mathbb{N}$ decompositions. In (2), every decomposition of S adds a, b; making it $a^n b^n$ for $n \in \mathbb{N}$ decompositions. (3) yields a task network with two a, matching aa.

Similarly, we construct a planning problem with linear constraints $\Pi_L = (\Pi^{\text{Seq}}, \mathbb{L})$, a planning problem with Presburger constraints $\Pi_{\phi} = (\Pi^{\text{Seq}}, \phi)$, and a planning problem with grammar amount constraints $\Pi_G = (\Pi^{\text{Seq}}, G)$ with the constructions listed in Figure 1. The constraints are different syntactic expressions to enforce that: (1) a, b, c occur the same number of times; (2) a, b occurs the same number of times, but not c; and (3) only a occurs twice. This shows that (1) $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, (2) $\{a^n b^n \mid n \in \mathbb{N}\}$, and (3) $\{aa\}$ are included in $\mathcal{LIN}$, $\mathcal{PRESBURGER}$, and $\mathcal{GAMOUNT}$.

For $\mathcal{BCNUM}^{+}$, we need to add counters to the Π^{Seq}: We extend the construction $\Pi^{\text{Seq}} = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$

(a) Linear Constraint $\mathbb{L} = (M, v)$:

(1) $M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \end{bmatrix}, v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ (2) $M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ (3) $M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, v = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$

(b) Presburger Formula ϕ:
(1) $(x_a = x_b) \wedge (x_b = x_c)$
(2) $(x_a = x_b) \wedge (x_c = 0)$
(3) $(x_a = (1{+}1)) \wedge (x_b = 0) \wedge (x_c = 0)$

(c) Production Rules of grammar G:
(1) $S \to Sabc \mid \varepsilon$ (2) $S \to Sab \mid \varepsilon$
(3) $S \to aa$
Where nonterminal S is the start symbol.

(d) Numerical Goal $Goal'$:
(1) $\{v_a - v_b = 0, v_b - v_c = 0)$
(2) $\{v_a - v_b = 0, v_c = 0\}$
(3) $\{v_a = 2, v_b = 0, v_c = 0\}$

Figure 1: Operator count constraints matching (1) $a^n b^n c^n$, (2) $a^n b^n$, and (3) aa when combined with Π^{Seq}.

from Def. 5.4 to a BC$^+$ numerical planning problem $\Pi_N = (V_P, V'_N, Ops', Init', Goal' \cup \{v \geq 0 \mid v \in V'_N\})$. The numeric variables $V'_N = \{v_a, v_b, v_c\}$ track how often an operator was applied. We initially set the counters to zero, i.e., $Init' = (\alpha, \beta)$ where α is the propositional state of $Init$ in Π^{Seq} and $\beta = \{v_a \mapsto 0, v_b \mapsto 0, v_c \mapsto 0\}$. Finally, operators $Ops' = \{a', b', c'\}$ extend the original operators $o \in Ops = \{a, b, c\}$ as:

$$o' = (pre_o, eff'_o), eff'_o = eff_o \cup \{v_o \leftarrow v_o + 1\}$$

Each operator increases its count to track how often the operator was applied. Thus, the numerical goal $Goal'$ from Figure 1 (d) enforces constraints showing that $\mathcal{BCNUM}^+$ includes $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n \in \mathbb{N}\}$, and $\{aa\}$. □

Note that the expressivity of SAS$^+$ (Bäckström and Nebel 1995) and FDR (Helmert 2009), which are other commonly used classical planning formalisms, is the same as that of STRIPS. The same holds for commonly used lifted variants of STRIPS (Corrêa et al. 2020, 2021, 2022; Lauer et al. 2021, 2025). Thus, this theorem shows that the novel formalisms are more expressive than many commonly used classical planning variants.

6 Discussion

We discuss potential benefits of the analyzed formalisms from different perspectives: Modeling, Solving, and Explainability.

Modeling Section 5.1 ends with an example that highlights the relevance of the gained expressive power proven in Thm. 5.5. Revisiting the example with linear constraints, allows us to additionally express the exact amounts $w_i, w'_i \in \mathbb{Q}$ operators make or spend to ensure a profit:

$$w_1 \cdot spend_1 + ... + w_l \cdot spend_l \leq w'_1 \cdot earn_1 + ... + w'_m \cdot earn_m$$

Lauer (2025) motivates that such quantitative constraints are an important form of knowledge that are easy to encode but can significantly improve solver performance. This is an advantage linear constraints, Presburger formulas, and BC numerical planning have over grammar amount constraints and unordered HTN planning.

Solving Section 6 shows that, despite its name, *unordered* HTN planning can represent plans with ordering dependencies between operators. This suggests that, even if a user defines an order between tasks, we may be able to compile them away and take advantage of a PSPACE solver by a compilation to unordered HTN. Def. 5.4 is essentially an example construction for orderings in primitive tasks.

To understand how we can benefit from the PSPACE results in practice, consider the language represented by:

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

Here, a truck drives, loads a package, and returns. Encoding this like $a^n b^n$ 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 for STRIPS and other classical planning formalisms (Hoffmann and Nebel 2001; Richter and Westphal 2010; Seipp and Helmert 2018; Fiser, Torralba, and Hoffmann 2024), where the plan existence is also PSPACE-complete.

Instead of counting tasks in a task network, one could simply solve a linear equation to obtain the required number of plan steps. A similar argument could be made for the constraints of the other analyzed formalisms. But, as there are very performant solvers for linear constraints, they may be the easiest to evaluate in practice. On top of that, linear constraints are closely connected to planning, e.g., heuristics that use ILP constraints over operator sequences (Imai and Fukunaga 2014; Pommerening et al. 2014; Lauer and Fišer 2025) could benefit from the added linear constraints.

Explainability One application of HTN is providing more concise explanations of plans (Seegebarth et al. 2012; Bercher et al. 2014; Jamakatel et al. 2023). Here, one uses the hierarchy to group actions together. In some cases, the ordering might not be needed at all. Then, the advantage canonically transfers to unordered HTN with the added advantage of having lower complexity to find a plan.

7 Conclusion

We have analyzed five planning formalisms that are significantly more expressive than STRIPS, 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 advocate for more research on unordered HTN planning.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Projec-

tID 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.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the complexity of HTN plan verification and its implications for plan recognition. In *Proc. of ICAPS*, 25–33.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On succinct groundings of HTN planning problems. In *Proc. of AAAI*, 9775–9784.

Bercher, P.; Biundo, S.; Geier, T.; Hoernle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain—how planning helps to assemble your home theater. In *Proc. of the 24th ICAPS*.

Brand, C.; Ganian, R.; Inerney, F. M.; and Wietheger, S. 2025. A Structural Complexity Analysis of Hierarchical Task Network Planning. In *Proc. of IJCAI*, 4391–4400.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *AIJ*, 69(1-2): 165–204.

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.

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.

Fiser, D.; Torralba, Á.; and Hoffmann, J. 2024. Boosting optimal symbolic planning: Operator-potential heuristics. *AIJ*, 334: 104174.

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

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning lifted strips models from action traces alone: A simple, general, and scalable solution. In *Proc. of ICAPS*, 189–197.

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.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proc. of AAAI*, 9883–9891.

Höller, D.; Bercher, P.; and Behnke, G. 2020. Delete- and Ordering-Relaxation Heuristics for HTN Planning. In *Proc. of the 29th IJCAI*, 4076–4083.

Imai, T.; and Fukunaga, A. 2014. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In *Proc. of the 21st ECAI*, 459–464.

Jamakatel, P.; Bercher, P.; Schulte, A.; and Kiam, J. J. 2023. Towards Intelligent Companion Systems in General Aviation using Hierarchical Plan and Goal Recognition. In *Proc. of the 12th HAI*.

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

Lauer, P.; Zhang, Y.; Haslum, P.; and Bercher, P. 2026. Appendix for paper: "I Always Told My Mom That Order Is Overrated: Unordered HTN Planning is in PSPACE and Models Problems Beyond STRIPS". https://doi.org/10.5281/zenodo.19145107.

Lin, S.; Behnke, G.; Ondrčková, S.; Barták, R.; and Bercher, P. 2023. On total-order HTN plan verification with method preconditions–an extension of the CYK parsing algorithm. In *Proc. of AAAI*, 12041–12048.

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.

Pantůčková, K.; Ondrčková, S.; and Barták, R. 2023. Parsing-Based Recognition of Hierarchical Plans Using the Grammar Constraint. In *Proc. of FLAIRS*.

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

Parikh, R. J. 1966. On context-free languages. *JACM*, 13(4): 570–581.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proc. of the 24th ICAPS*, 226–234.

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

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users-a formal approach for generating sound explanations. In *Proc. of ICAPS*, 225–233.

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.

Shleyfman, A.; Gnad, D.; and Jonsson, P. 2023. Structurally Restricted Fragments of Numeric Planning - a Complexity Analysis. In *Proc. of the 37th AAAI*, 12112–12119.

Stern, R.; Lamanna, L.; Mordoch, A.; Benyamin, Y.; Lauer, P.; Juba, B.; Behnke, G.; Muise, C.; Bercher, P.; Vallati, M.; Xi, K.; Wattad, O.; and Eliyahu, O. 2025. Evaluating Planning Model Learning Algorithms. In *Proc. of KEPS at ICAPS*.

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.

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

Wichlacz, J.; Torralba, Á.; and Hoffmann, J. 2019. Construction-Planning Models in Minecraft. In *Proc. of the 2nd HPLAN at ICAPS*.

Zhang, Y.; and Bercher, P. 2025. Computational complexity of planning for recursive primitive task networks: selective action nullification with state preservation. In *Proc. of IJCAI*, 8696–8704.