Symmetries and Other Variations of "End-Recursive" HTN Problems: Mapping the Border Between Decidable and Undecidable Restrictions

Hadyn Tang, Pascal Bercher
The Australian National University
{hadyn.tang, pascal.bercher}@anu.edu.au

Abstract

In this paper, we investigate the complexity of determining if various restricted forms of hierarchical task network (HTN) planning have a plan. We perform a systematic analysis of new restrictions formed by applying symmetries and relaxations to two existing restrictions called regularity and tail-recursiveness. By doing so, we confirm that many variations on common restrictions do not affect the complexity of the plan existence problem. However, we also obtain the counter-intuitive result that combining some of these seemingly inert relaxations together renders the plan existence problem undecidable. Additionally, we unearth a critical difference in definitions between an early paper on HTN planning and modern formalisms that appears to have gone unnoticed.

Variation	Regular		Tail-recursive
	with tn_I	w/o tn_I	
None	PSPACE*(1)	PSPACE(1)	EXPSPACE*
Reversed	PSPACE (4)	PSPACE(4)	EXPSPACE(10)
Mixed	PSPACE*(6)	undec. (9)	undec. (12)

Table 1: Complexity of deciding the plan existence problem for each restricted subset of HTN planning. All results are completeness results. Asterisks denote previously known results, and bracketed numbers indicate relevant theorems or corollaries. Undecidable classes are still semi-decidable.

1 Introduction

One formalism for AI planning problems which has been studied extensively is hierarchical task network planning. In hierarchical task network planning, we aim to get from a given start state to some goal state by applying tasks from a partially-ordered collection of tasks called a *task network*: tasks may either be primitive tasks, which describe an atomic state transformation; or compound tasks, which specify one or more *decompositions* which may be used to replace them with specific sub-networks of tasks.

Hierarchical task network planning is known to be undecidable in general (Erol, Hendler, and Nau 1996), yet many restrictions are known under which the problem becomes decidable. Our work builds upon two such common restrictions, *regularity* (Erol, Hendler, and Nau 1996) and *tail-recursiveness* (Alford et al. 2012), which both restrict the positions at which compound tasks can occur in the initial task network and in decompositions, to avoid creating the overly complex task networks which lead to undecidability. We are interested in what tweaks we can make to these various restrictions, and how they affect the complexity of determining if a solution exists to planning problems following these restrictions. In particular, we wish to contribute to a refined understanding of when a problem class of HTN problems becomes undecidable.

For instance, determining whether a plan exists for a 'regular' HTN problem is PSPACE-complete (Erol, Hendler, and Nau 1996), and in this paper we present two independent adjustments to that restriction which still result in a PSPACE-complete problem. However, if we combine these two adjustments together, it results in a problem which is undecidable, just as bad as the general case for HTN problems. We summarise our results in Table 1.

2 Preliminaries

To begin with, we give a formalisation of HTN planning which closely mirrors the formalisations given by Geier and Bercher (2011) and Bercher, Alford, and Höller (2019). We then outline the two common HTN problem classes of which we wish to study variations.

2.1 HTN Planning

We start with the definition of a *task network*, which captures the idea of a collection of tasks (possibly with repetition) which are partially ordered. Tasks are assigned to distinct ids so that we can consider multiple copies of the same task.

Definition 1. A *task network* over a set N of task names is a tuple $\langle T, \prec, \alpha \rangle$, where T is a finite (and possibly empty) set of task ids, $\prec$ is a strict partial order over T, and $\alpha : T \to N$ maps task ids to task names.

Since the task ids are placeholders and not particularly relevant to our discussion, we regard two task networks $\langle T, \prec, \alpha \rangle$ and $\langle T', \prec', \alpha' \rangle$ as *isomorphic* if there is an order-preserving bijection $\sigma : T \to T'$ with $\alpha(t) = \alpha'(\sigma(t))$ for

every $t \in T$. We let $\mathbb{T}_N$ denote the set of task networks over a given set of task names N.

In a slight abuse of notation, given two task networks tn_1 and tn_2, we write $tn_1 \prec tn_2$ to denote the combined task network where all tasks in tn_2 are ordered after those in tn_1. We also let a single task u denote the task network with just u, so for example $tn \prec u$ denotes 'the task network tn with an extra task u ordered after all tasks in tn'.

Tasks come in two types – *primitive* and *compound*. A task network is said to be *primitive* if it does not contain compound tasks. Primitive tasks represent 'actions':

Definition 2. Let F be a finite set, which we call the *facts*. A *state* is a subset $s \in 2^F$. An *action* is a tuple $\langle \text{pre}, \text{eff}^+, \text{eff}^- \rangle \in 2^F \times 2^F \times 2^F$ consisting of *preconditions*, *add effects* and *delete effects* respectively. Such an action is *executable* in a state s if pre $\subseteq s$, in which case acting by a on s results in the new state $(s \setminus \text{eff}^-) \cup \text{eff}^+$.

Compound tasks may have any number of 'decomposition methods' and can be 'decomposed' by them as follows:

Definition 3. A *decomposition method* (or just *decomposition*) is a pair $m = \langle c, tn \rangle$ of a task name c and a task network tn. Given another task network tn_1 and some id $t \in T_1$ with $\alpha_1(t) = c$, we can *decompose* tn_1 by m at t into $\langle T_2, \prec_2, \alpha_2 \rangle$ by taking some network $tn' = \langle T', \prec, \alpha \rangle$ isomorphic to tn such that we have $T' \cap T_1 = \emptyset$, and letting $T_2 = (T_1 \setminus \{t\}) \cup T'$, $\alpha_2 = \alpha_1|_{T_1 \setminus \{t\}} \cup \alpha'$, and

$$\begin{aligned} \prec_2 = {} & (\prec_1 \cap (T_1 \setminus \{t\})^2) \cup \{(u, v) \in T_1 \times T' \mid u \prec_1 t\} \\ & \cup \quad \prec' \quad \cup \{(u, v) \in T' \times T_1 \mid t \prec_1 v\}. \end{aligned}$$

We are now ready to define the notions of an HTN planning domain and an HTN problem:

Definition 4. An *HTN planning domain* is a tuple $\langle F, N_P, N_C, \delta, M \rangle$ consisting of a finite set of facts F, finite disjoint sets N_P and N_C of primitive and compound task names respectively, a map $\delta : N_P \to (2^F)^3$ from primitive task names to actions, and a finite set $M \subseteq N_C \times \mathbb{T}_{N_C \cup N_P}$ of decomposition methods.

Definition 5. An *HTN (planning) problem* is a tuple $\langle D, tn_I, s_I, g \rangle$ over some domain $D = \langle F, N_P, N_C, \delta, M \rangle$ where $tn_I \in \mathbb{T}_{N_C \cup N_P}$ is the *initial task network*, $s_I \in 2^F$ is the *initial state*, and $g \subseteq F$ is the *goal*.

Now, we address what it means to solve a problem:

Definition 6. A primitive task network $\langle T, \prec, \alpha \rangle$ is a *solution* to an HTN planning problem if it can be obtained from tn_I via a sequence of decompositions by methods in M, and such that there is some linearisation $(t_1, \ldots, t_k)$ of $\prec$ for which it is possible to execute $\alpha(t_1), \ldots, \alpha(t_k)$ in order to get from s_I to some state $s \supseteq g$.

We briefly digress to mention the classical formalism for planning, as introduced by Fikes and Nilsson (1971):

Definition 7. A *STRIPS planning problem* is a tuple $\langle F, A, s_I, g \rangle$ consisting of a finite set of facts F, a finite set of actions $A \subseteq (2^F)^3$, *initial state* $s_i \in 2^F$ and *goal* $g \subseteq F$. A finite sequence $(a_1, \ldots, a_k)$ of actions is a *solution* to such a problem if it is possible to execute $a_1, \ldots, a_k$ in order to get from s_I to some state $s \supseteq g$.

2.2 Common Restrictions on HTN Problems

In this paper, we are interested in the decidability and computational complexity of computing whether a given HTN problem has a solution, the so-called *plan existence problem*. Unfortunately, as mentioned earlier, this problem is in general undecidable, though it is semi-decidable (Erol, Hendler, and Nau 1996). (This result implies plan existence for all subclasses of HTN planning problems are also semi-decidable.) Hence we often consider restricted problem structures, trading expressivity for decidability.

The main cause of the undecidability is the ability to interleave tasks. One class of problems which seeks to address this is the following:

Definition 8. A task network is *regular* (or *right-linear*) if it is either primitive or of the form $tn \prec C$ for some primitive task network tn and compound task C. An HTN planning problem is *regular* if its initial task network tn_I is regular and each decomposition $\langle c, tn \rangle \in M$ has tn regular.

This is similar to the definition of a right-linear grammar: a context-free grammar where each rule produces at most one non-terminal symbol, and if it exists that non-terminal is after all the terminal symbols.

The definition we have given above is used by all recent papers we could find, spanning from Alford et al. (2012) to Zhang and Bercher (2025). It is 'well-known' in the planning community that the plan existence problem for regular HTN problems is PSPACE-complete, however all sources that we could find cite Erol, Hendler, and Nau (1996), who use the term "regular" to refer to a broader class of HTN problems, which we expand on in section 4 under the name "linear". For completeness, we show that plan existence for regular problems is PSPACE-complete in Theorem 1.

The second class of problems that we will consider variations of is the following generalisation of regular problems that allows for a more complicated hierarchy:

Definition 9. Given a total preorder $\leq_r$ over N_C (called a *stratification*), we call a decomposition $\langle c, \langle T, \prec, \alpha \rangle \rangle$ *tail-recursive under* $\leq_r$ if, for any compound task id $t \in T$,

- If t is a last task id, we have $\alpha(t) \leq_r c$, and
- If t is not a last task id, we have $\alpha(t) \leq_r c$ but $c \not\leq_r \alpha(t)$,

where a *last task id* is a task id $u \in T$ for which $v \prec u$ for any $v \in T \setminus \{u\}$. An HTN planning problem is *tail-recursive* if there is some stratification $\leq_r$ such that every decomposition in the problem is tail-recursive under $\leq_r$.

Any regular problem is also tail-recursive, since we can take the trivial stratification where $c \leq_r c'$ for every pair of compound tasks $c, c' \in N_C$. The plan existence problem for tail-recursive HTN planning problems is known to be EXPSPACE-complete (Alford, Bercher, and Aha 2015).

We provide a diagram depicting how problems in the above two classes decompose in Figure 2 (which also contains references for the other classes we will introduce).

2.3 Progression

One well-known algorithm to solve the plan existence problem is *progression*. It semi-decides all HTN problems, and decides regular and tail-recursive problems in PSPACE and

Algorithm 1: Non-Deterministic Progression

Set $tn := tn_I$ and $s := s_I$.
while tn contains at least one task **do**
  Pick any task t with no predecessor under $\prec$.
  if t is primitive **then**
    Remove it from the network and apply it to s if executable (else reject)
  else
    Pick any of t's decomposition methods (if it has any, else reject) and decompose it via that method.
  end if
end while
Accept if $g \subseteq s$, else reject.

EXPSPACE, respectively (Alford, Bercher, and Aha 2015). Its non-deterministic form can be found in Algorithm 1.

3 Left- and Right-Linear Problems

The first set of restrictions we wish to study are those presenting small modifications on the definition of regularity.

3.1 Right-Linearity (i.e. Regularity)

The specification of a regular HTN problem involves two independent constraints: one on the initial task network, and one on decomposition methods. We would first like to study what occurs if we remove the weaker condition.

Definition 10. We say an HTN problem *has regular* (or *right-linear*) *decompositions* if each decomposition $\langle c, tn\rangle \in M$ has tn regular.

This ensures a problem is regular if and only if its initial task network is regular and it has regular decompositions.

Theorem 1. *The plan existence problem for both of the following classes of problems is PSPACE-complete:*

(1) Regular HTN planning problems
(2) HTN planning problems with regular decompositions

Proof. It suffices to show that the class of problems in (1) is PSPACE-hard and the class of problems in (2) lies in PSPACE, because class (1) is a subset of class (2).

PSPACE-hardness of (1) follows from the PSPACE-hardness of plan existence for STRIPS planning (Bylander 1994). The following reduction was first explained by (Erol, Hendler, and Nau 1996). Let $\langle F, A, s_I, g\rangle$ be a STRIPS planning instance. Consider the HTN planning problem with start state s_I and goal g defined as follows: we have a single compound task C in tn_I which can decompose into the empty task or $t_a \prec C$ for any action $a \in A$, where t_a is a primitive task with action a. This problem is clearly regular, and the hierarchy can decompose to any sequence of actions, producing them from left to right. So the HTN problem has a plan iff the corresponding STRIPS problem did, as desired.

So we only need to show that problem class (2) lies in PSPACE. Suppose the initial task network has p primitive and c compound tasks. Suppose also that the largest task network in any method has size m. Then we claim that in the process of progression planning, at any time our task network only has size at most $p + c + cm$.

Suppose we decompose some compound task C into a task network that contains some other compound task C'. Then by the regularity condition, all the newly added primitive tasks are ordered before C', so we will not decompose C' until all primitive tasks are exhausted. Hence, if we keep track of the number of tasks in our task network that come from a particular initial compound task, this starts as 1 in the initial compound task, increases to at most m after any decomposition, and decreases back to 1 or 0 again before it can be again decomposed. There are at most p extra primitive tasks in the network coming from the initial network, so the size of the network is at most $p + c + cm$ at any point.

So the amount of space required is bounded by a polynomial in the size of the initial task network, and thus this problem lies in PSPACE as required (recalling that PSPACE equals NPSPACE). □

3.2 Left-Linearity

As done in the context of context-free languages, we define the following reflected version of right-linearity:

Definition 11. We call a task network *left-linear* if it is either primitive or of the form $C \prec tn$ for some primitive task network tn and compound task C. We say an HTN problem *has left-linear decompositions* if each decomposition $\langle c, tn\rangle \in M$ has tn left-linear. We call an HTN planning problem *left-linear* if its initial task network tn_I is left-linear and it has left-linear decompositions.

Once again, see Figure 2 for a diagram.

Due to the symmetry of right- and left-linear problems, we would like to show that plan existence for these two problem types is PSPACE-complete by somehow reversing the progression algorithm. In particular, progression has a dual of sorts – *regression* – where instead of looking at minimal primitive tasks and applying them, we look at maximal primitive tasks and 'unapply' (i.e. *regress*) them non-deterministically. Specifically, to regress an action $\langle \text{pre}, \text{eff}^+, \text{eff}^-\rangle$ in state s, we pick some subsets $X^+ \subseteq \text{eff}^+$ and $X^- \subseteq \text{eff}^-$ and take the new state $s' = (s \setminus X^+) \cup X^-$ so long as $\text{pre} \subseteq s'$, $s \cap \text{eff}^- \subseteq X^+$, and $\text{eff}^+ \subseteq s$. If no such subsets X^+ and X^- fulfil these requirements, then the action cannot be regressed. This is not a new concept; regressing a task is mentioned even in the paper by Erol, Hendler, and Nau (1996), though no formal definition is provided and the definition here is a reconstruction based on the ideas in that paper.

Lemma 2. *State s can be regressed to s' under a given action iff that action is executable in state s' to yield state s.*

Proof. Suppose first that we have regressed s to s' under the action $\langle \text{pre}, \text{eff}^+, \text{eff}^-\rangle$ with subset choices $X^+ \subseteq \text{eff}^+$ and $X^- \subseteq \text{eff}^-$. Then in s, we know the action can be applied (because by assumption $\text{pre} \subseteq s'$). Hence $s' \setminus \text{eff}^- = s \setminus X^+$ because $s' = (s \setminus X^+) \cup X^-$, $X^- \subseteq \text{eff}^-$ and also we know $(s \setminus X^+) \cap \text{eff}^- = \emptyset$ (from $s \cap \text{eff}^- \subseteq X^+$). Thus, $(s' \setminus \text{eff}^-) \cup \text{eff}^+ = s$ as $X^+ \subseteq \text{eff}^+ \subseteq s$.

Algorithm 2: Non-Deterministic Regression

Set $tn := tn_I$ and pick some $s \supseteq g$.
while tn contains at least one task **do**
  Pick any task t with no successor under $\prec$.
  if t is primitive **then**
    Remove it from the network and regress it from s if possible (else reject)
  else
    Pick any of t's decomposition methods (if it has any, else reject) and decompose it via that method.
  end if
end while
Accept if s is equal to s_I, else reject.

We can likewise routinely verify that if applying $\langle \text{pre}, \text{eff}^+, \text{eff}^- \rangle$ to s' yields $s = (s' \setminus \text{eff}^-) \cup \text{eff}^+$, then the choices $X^- = s' \cap \text{eff}^-$ and $X^+ = \text{eff}^+ \setminus (s' \setminus \text{eff}^-)$ satisfy $s' = (s \setminus X^+) \cup X^-$ and the other conditions required so that s can be regressed to s'. □

The full regression algorithm is then given in Algorithm 2, which also underlines changes to the original algorithm.

Lemma 3. *Algorithm 2 will accept if and only if the input HTN problem has a solution.*

Proof. Suppose the algorithm accepts. Then consider first performing all the decompositions in the order done in the algorithm. This must result in a primitive task network. It is easy to check that we may then execute the tasks in the reverse order to how they were regressed.

Conversely, suppose a plan exists, and consider its linearisation. Then at each point, take the last primitive task in the linearisation that has not been regressed: it must be possible to decompose the current task network some number of times to produce the relevant primitive task, and we can check that this task can then be regressed. □

Theorem 4. *The plan existence problem for both of the following classes of problems is PSPACE-complete:*

(1) Left-linear HTN planning problems
(2) HTN planning problems with left-linear decompositions

Proof. Again (1) is a subset of (2), so we show (1) is PSPACE-hard and (2) is in PSPACE.

As in Theorem 1, the PSPACE-hardness of (1) again follows from the PSPACE-hardness of plan existence for STRIPS planning (Bylander 1994). The only difference is that to produce all sequences of actions, we have decompositions from C to $C \prec t_a$ (instead of $t_a \prec C$) for each action $a \in A$ to adhere to left-linearity.

The PSPACE-membership proof for (2) is also almost identical to that of Theorem 1. Consider the execution of Algorithm 2. If tn_I has p tasks, c of which are compound tasks, and the largest task network in a decomposition has m tasks, then as before a compound task produced by a decomposition cannot be decomposed until all the newly produced primitive tasks have been regressed away, so similarly we can check that we have at most $p + c + cm$ tasks in our current task network at any time. Thus the plan existence problem for (2) lies in PSPACE as well. □

Note that we could have also provided direct reductions between left- and right-linear problems. We will see a more general version of this reduction in the proof of Theorem 10.

3.3 Mixed-Linearity

A natural question one could ask is what happens if we permit both left-linear and right-linear decompositions to exist simultaneously.

Definition 12. We say an HTN problem *has mixed-linear decompositions* if every decomposition method $\langle c, tn \rangle \in M$ has tn either left- **or** right-linear. We call an HTN problem *mixed-linear* if its initial task network tn_I is left- or right-linear and it has mixed-linear decompositions.

Importantly, we allow both left-linear and right-linear task networks to coexist in different decompositions of the same mixed-linear problem. The definition of mixed-linearity is somewhat unnatural, so in the next section we generalise this restriction to a notion we call 'linearity', while ensuring we can still reduce to our two mixed-linear problem classes.

4 A General Notion of Linearity

A stronger relaxation of the class of regular problems, which would encompass both right- and left-linear problems, would be the class of problems where in each task network there is at most one compound task, which must be 'sandwiched' between two primitive task networks. The case where we have at most one compound task per decomposition method, allowing primitive tasks to be interleaved with the compound task, has been studied extensively by Dekker and Behnke (2024) and Zhang and Bercher (2025). However, to improve our understanding of exactly what factors contribute to the hardness of certain classes of HTN problems, we wish to consider the case where no such interleaving is possible in the decomposition methods.

In the case of context-free grammars, a *linear grammar* is a context-free grammar in which each production rule contains at most one non-terminal. We thus define:

Definition 13. We call a task network *linear* if it is either primitive or of the form $tn_1 \prec t \prec tn_2$ for some primitive task networks tn_1 and tn_2 (which may be empty) and compound task t. We say that an HTN problem *has linear decompositions* if each decomposition $\langle c, tn \rangle \in M$ has tn linear. We call an HTN problem *linear* if its initial task network tn_I is linear and it has linear decompositions.

From this definition it follows that all mixed-linear problems are linear, and likewise all problems with mixed-linear decompositions also have linear decompositions. The above constraints are depicted in Figure 2.

As alluded to earlier, Erol, Hendler, and Nau (1996) actually define a regular HTN problem in this way, stating in section 2.3 of that paper that a problem "is *regular* if all the task networks in the methods and $[tn_I]$ contain at most one nonprimitive task, and that non-primitive task is ordered with respect to all the other tasks in the network". The definition of

regularity later adopted by the community may come from an erroneous comment on Table 1 of that paper defining it as "$\leqslant 1$ non-primitive task, which must follow all primitive tasks". In any case, we have:

Theorem 5 (Erol, Hendler, and Nau (1996), Theorem 5)**.** *The plan existence problem for linear HTN planning problems is PSPACE-complete.*

The formalism used by Erol, Hendler, and Nau (1996) includes various additional features, such as more constraints on allowable task networks and different solution criteria, so their proof is fairly involved. We translate their proof into the current HTN formalism.

Proof. Since the set of linear HTN planning problems includes the set of regular HTN planning problems and the plan existence problem for regular HTN planning problems is PSPACE-hard, so is the plan existence problem for linear HTN planning problems. Hence, we only need show membership in PSPACE. Consider the following algorithm:

- Set $tn \coloneqq tn_I$ and $s \coloneqq s_I$, and pick some $s' \supseteq g$.
- While tn contains at least one task
  - If all tasks with no predecessor under $\prec$ are primitive, remove one such task from tn and apply it to s if possible, else reject.
  - Else if all tasks with no successor under $\prec$ are primitive, remove one such task from tn and regress it from s' if possible, else reject.
  - Else pick a decomposition method and decompose the compound task once if possible, else reject.
- Accept if s is equal to s', else reject.

Similarly to Lemma 3, this algorithm can be checked to only accept when a valid plan exists. It accepts on a linear problem if a plan exists because it implements regression but with an additional step of applying the tasks in some prefix of the linearisation to 'meet in the middle'.

Furthermore, due to the additional stipulation that we execute or regress all primitive tasks that come strictly before or after the main compound task, the algorithm ensures that once we decompose the unique compound task, we cannot decompose it again until all the added primitive task have been executed or regressed out of the task network. Hence, if p is the number of primitive tasks in the initial task network and m is the size of the largest task network in a method, all task networks encountered during this algorithm have size at most $p + m + 1$. Thus, this provides the required PSPACE membership proof. □

We mentioned earlier that linear problems would be equivalent to mixed-linear problems from a complexity standpoint. Indeed, since all mixed-linear problems are linear, and all regular problems are mixed-linear:

Corollary 6. *The plan existence problem for mixed-linear HTN planning problems is PSPACE-complete.*

As mentioned earlier, the plan existence problem for general HTN planning problems is undecidable (Erol, Hendler, and Nau 1996). However, this reduction follows from the intersection problem from context-free grammars, and context-free grammars admit many structures. We will show the same result for a restricted subset of linear grammars:

Definition 14. We call a context-free grammar *simple linear* if each production rule is of the form $T \to \varepsilon$, $T \to aU$ or $T \to Ua$, where a represents a terminal and T, U represent non-terminals.

Lemma 7. *The following problem is undecidable: Given two simple linear grammars G_1 and G_2, is the intersection of their languages $\mathcal{L}_{G_1} \cap \mathcal{L}_{G_2}$ nonempty?*

Proof. We examine the proof of Theorem 9.20 from the book published by Hopcroft, Motwani, and Ullman (2001). The proof shows by reduction from the Post Correspondence Problem that it is undecidable whether the languages $\mathcal{L}_{G_1}$ and $\mathcal{L}_{G_2}$ of two grammars G_1 and G_2 of the following form have non-empty intersection:

$$S \to w_1 S a_1 \mid \cdots \mid w_k S a_k \mid w_1 a_1 \mid \cdots \mid w_k a_k,$$

where S is the starting symbol, $w_1, \ldots, w_k$ are strings over some alphabet Σ of terminals, and $\{a_1, \ldots, a_k\}$ are a set of distinct terminals disjoint from Σ.

Notice that these are linear grammars, but we can go further and show that any grammar of this form can be rewritten in simple linear form. Rewrite it as

$$S \to w_1 A a_1 \mid \cdots \mid w_k A a_k; \;\; A \to w_1 A a_1 \mid \cdots \mid w_k A a_k \mid \varepsilon,$$

which is clearly equivalent. We wish to replace rules of the form $r : T \to b_1 \ldots b_k U c$ for T, U non-terminals and $b_1, \ldots, b_k, c$ terminals with a set of rules following the simple linear restriction. Indeed, for some new non-terminals $R_{r,1} \ldots, R_{r,k}$ (different for each rule r), replace r with the rules $T \to R_{r,1} c$, $R_{r,i} \to b_i R_{r,i+1}$ for $1 \leq i \leq k-1$, and $R_{r,k} \to b_k U$. It is easy to see this replacement doesn't change the underlying language of the grammar, and that all rules produced follow the simple linear restriction.

So we can computably convert G_1 and G_2 to simple linear grammars G_1' and G_2' with $\mathcal{L}_{G_1} = \mathcal{L}_{G_1'}$ and $\mathcal{L}_{G_2} = \mathcal{L}_{G_2'}$. Thus, it is also undecidable to determine whether two simple linear grammars have non-empty intersection. □

We will use this to show that the plan existence problem for linear HTN problems is also undecidable. We work with the corresponding restriction of linearity in our proof to provide a more applicable result. (Once again, see Figure 2 for a diagram.)

Definition 15. We call a task network *simple linear* if it is either empty, or of the form $p \prec c$ or $c \prec p$ for some compound task c and primitive task p. An HTN problem *has simple linear decompositions* if each decomposition $\langle c, tn \rangle \in M$ has tn simple linear.

Theorem 8. *The plan existence problem for HTN planning problems with simple linear decompositions is undecidable, even if the initial task network is restricted to containing only two unordered compound tasks.*

Proof. We proceed via a reduction exploiting Lemma 7. Let G_1 and G_2 be two simple linear grammars with shared set of

terminal symbols Σ. Let their sets of non-terminal symbols be V_1 and V_2 respectively, with starting non-terminals $S_1 \in V_1$ and $S_2 \in V_2$ respectively. Let $\mathcal{L}_{G_1}$ and $\mathcal{L}_{G_2}$ be their corresponding languages.

We now define an HTN planning problem. The set of facts is $F \coloneqq \{\text{done}\} \cup \{\text{echo}_a \mid a \in \Sigma\}$. The primitive tasks are:

- For each $a \in \Sigma$, we have a task on_a with precondition and delete effect $\{\text{done}\}$ and add effect $\{\text{echo}_a\}$; and
- For each $a \in \Sigma$, we also have a task off_a with precondition and delete effect $\{\text{echo}_a\}$ and add effect $\{\text{done}\}$.

We also have start state and goal $\{\text{done}\}$.

Before we define the compound tasks and decompositions, it is useful to consider what the allowable linearisations of primitive tasks look like. It is not too hard to then observe that the tasks must come in ordered pairs '$\text{on}_a, \text{off}_a$' for various $a \in \Sigma$, so that the valid linearisations are exactly $\text{on}_{a_1}, \text{off}_{a_1}, \ldots, \text{on}_{a_k}, \text{off}_{a_k}$ for $k \geq 0$ and $a_1, \ldots, a_k \in \Sigma$.

Now, consider the following definition of compound tasks. For each $T \in V_1$, we have a compound task $C_{1,T}$. For each production in G_1:

- if it is $T \to aU$ for $a \in \Sigma$ and $T, U \in V_1$, we have a decomposition $\langle C_{1,T}, \text{on}_a \prec C_{1,U} \rangle$;
- if it is $T \to Ub$ for $b \in \Sigma$ and $T, U \in V_1$, we have a decomposition $\langle C_{1,T}, C_{1,U} \prec \text{on}_b \rangle$; and
- if it is $T \to \varepsilon$ for $T \in V_1$, we have a decomposition from $C_{1,T}$ to the empty task network.

It is easy to show inductively that the possible networks of primitive tasks that can be produced by decomposing C_{1,S_1} are exactly $\text{on}_{a_1} \prec \cdots \prec \text{on}_{a_k}$ for $a_1 \cdots a_k \in \mathcal{L}_{G_1}$.

Similarly, for each $T \in V_2$, we have a compound task $C_{2,T}$, and for each production in G_2:

- if it is $T \to aU$ for $a \in \Sigma$ and $T, U \in V_2$, we have a decomposition $\langle C_{2,T}, \text{off}_a \prec C_{2,U} \rangle$;
- if it is $T \to Ub$ for $b \in \Sigma$ and $T, U \in V_2$, we have a decomposition $\langle C_{2,T}, C_{2,U} \prec \text{off}_b \rangle$; and
- if it is $T \to \varepsilon$ for $T \in V_2$, we have a decomposition from $C_{2,T}$ to the empty task network.

Decomposing C_{2,S_2} into primitive tasks produces exactly the set of $\text{off}_{a_1} \prec \cdots \prec \text{off}_{a_k}$ for $a_1 \cdots a_k \in \mathcal{L}_{G_2}$.

Hence, combining this with our earlier observation that primitive tasks must be exactly paired up, it follows that if our initial task network contains only the two unordered tasks C_{1,S_1} and C_{2,S_2}, then the possible linearisations of task networks produced are exactly $\text{on}_{a_1}, \text{off}_{a_1}, \ldots, \text{on}_{a_k}, \text{off}_{a_k}$ for $a_1 \cdots a_k \in \mathcal{L}_{G_1} \cap \mathcal{L}_{G_2}$. Thus, a plan exists if and only if $\mathcal{L}_{G_1} \cap \mathcal{L}_{G_2}$ is nonempty, and by reduction from Lemma 7, it follows that the plan existence problem for grammars with simple linear decompositions is undecidable. □

By observation, any HTN planning problem with simple linear decompositions also has mixed-linear decompositions and linear decompositions, so:

Corollary 9. *The plan existence problem for the following two classes of HTN planning problems is undecidable, even if the initial task network is restricted to containing only two unordered compound tasks:*

(1) Problems with mixed-linear decompositions
(2) Problems with linear decompositions

5 Head- and Tail-Recursiveness

Regular problems and tail-recursive problems are often studied together because the tail-recursiveness condition is a weakening of the regularity condition. Hence, we are interested in investigating what occurs if we perform similar modifications to the definition of tail-recursiveness.

We note that the choice of initial task network is essentially irrelevant in the setting of tail-recursive problems (and the related problem classes we define below): in particular, forcing the initial task network to be a single compound task does not affect expressiveness. Informally, this is because we can always add a single compound task C_I which decomposes to the initial task network tn_I and stratify it above all existing compound task names, then take C_I as our new initial task network.

5.1 Head-Recursive Problems

Similarly to the correspondence between right- and left-linearity, we can define a reversed version of tail-recursiveness as follows:

Definition 16. Given a stratification $\leq_r$, we call a decomposition $\langle c, \langle T, \prec, \alpha \rangle \rangle$ *head-recursive under* $\leq_r$ if, for any compound task id $t \in T$,

- If t is a first task id, we have $\alpha(t) \leq_r c$, and
- If t is not a first task id, we have $\alpha(t) \leq_r c$ but $c \not\leq_r \alpha(t)$,

where a *first task id* is a task id $u \in T$ for which $u \prec v$ for any $v \in T \setminus \{u\}$. We call a HTN problem *head-recursive* if there is some stratification $\leq_r$ such that every decomposition in the problem is head-recursive under $\leq_r$.

See Figure 2 for an illustration.

Theorem 10. *The plan existence problem for head-recursive HTN planning problems is EXPSPACE-complete.*

Proof. The goal is to provide reductions in both directions between the plan existence problem for head- and tail-recursive HTN problems. Since the plan existence problem for tail-recursive problems is EXPSPACE-complete, it will follow that it is also EXPSPACE complete for head-recursive problems.

We start by reducing from head-recursive to tail-recursive problems. For a given stratification $\leq_r$, let a *stratum* be an equivalence class of $\leq_r$, i.e. maximal subset $S \subseteq N_C$ such that $t \leq_r u \leq_r t$ for any tasks $t, u \in S$. Given a task t, we say t *is on a lower stratum than* S if $t \leq_r u$ and $u \not\leq_r t$ for some $u \in S$ (and hence for every $u \in S$).

The aim is to replace the tasks and decompositions for each stratum (one by one in any order) so that the tasks go from being head-recursive under $\leq_r$ to being tail-recursive under a modified stratification. To do this, we will introduce new compound tasks: a task T_c for each $c \in S$, and a task

$U_{c',c}$ for every pair $(c', c) \in S^2$. We will 'insert' these tasks into $\leq_r$ so that they would lie in the same stratum S.

Call a decomposition $\langle c, tn \rangle \in M$ *external* if $c \in S$ but all tasks in tn are on a lower stratum than S, and *internal* if $c \in S$ and tn can be written as $c' \prec tn'$ for some $c' \in S$. Since $\langle c, tn \rangle$ is head-recursive under $\leq_r$, these are the only two options for decompositions if $c \in S$.

For each external decomposition $\langle c', tn' \rangle$ and $c \in S$, we introduce a decomposition from T_c into $tn' \prec U_{c',c}$. For each internal decomposition $\langle c_1, c_2 \prec tn \rangle$ and $c \in S$, we introduce a decomposition from $U_{c_2,c}$ to $tn \prec U_{c_1,c}$. Finally, for each $c \in S$, we introduce a decomposition from $U_{c,c}$ into the empty network $\langle \emptyset, \emptyset, \emptyset \rangle$.

Consider a decomposition in the original problem which starts out with $c_1 \in S$ and ends with a task network which has only tasks on a lower stratum than S. If we pass through the tasks $c_1, \ldots, c_k \in S$, where for $1 \leq i \leq k-1$ the relevant decomposition is $\langle c_i, c_{i+1} \prec tn \rangle$ and the final decompositions $\langle c_k, tn_k \rangle$, the result is $tn_k \prec tn_{k-1} \prec \cdots \prec tn_1$. But we can produce this result from our new tasks by decomposing T_c into $tn_k \prec U_{c_k,c_1}$, and then for each $k-1 \geq i \geq 1$ in decreasing order decomposing U_{c_{i+1},c_1} into $tn_i \prec U_{c_i,c_1}$, all of which we can check are possible by construction.

Similarly, consider a decomposition from T_c into some task network which has only tasks on a lower stratum than S. We must start by decomposing T_c into some $tn_1 \prec U_{c_1,c}$, and there is some $\ell \geq 1$ such that for each $1 \leq i \leq \ell - 1$ we decompose $U_{c_i,c}$ into $tn_{i+1} \prec U_{c_{i+1},c}$, and finally we decompose $U_{c_\ell,c}$ into the empty network (requiring $c_\ell = c$). This produces the network of tasks $tn_1 \prec \cdots \prec tn_\ell$. But we can produce this starting with $c = c_\ell$ by decomposing c_i into $c_{i-1} \prec tn_i$ for each $\ell \geq i \geq 2$ in decreasing order, and finally decomposing c_1 into t_1. Likewise, we can check these decompositions are also possible based on the definition of the T_cs and $U_{c',c}$s.

Hence, the networks of tasks that can be produced from some $c \in S$ are exactly those that can be produced from our new T_c. An example of this bijection is shown in Figure 1. In particular, c and T_c thus can be decomposed into exactly the same set of primitive task networks. Furthermore, if we insert each T_c and $U_{c',c}$ into the stratification $\leq_r$ on the same stratum S to get some new stratification $\leq'_r$, we can verify that all our new tasks are tail-recursive under $\leq'_r$. To finish replacing the tasks, we simply replace each occurrence of any $c \in S$ in a decomposition method $\langle c', tn \rangle$ satisfying $c' \notin S$ (which must necessarily have $c' >_r c$) with T_c, and then we may remove the original tasks $c \in S$ and their corresponding decompositions from consideration.

If we do this for every stratum S, we end up with a new stratification $\leq''_r$ so that every decomposition is tail-recursive under $\leq''_r$, where at every step we ensured the same primitive task networks were producible. So this provides a reduction from plan existence for head-recursive HTN problems to plan existence for tail-recursive HTN problems. Furthermore, since we at most square the number of tasks on each stratum (and modifying one stratum doesn't change the number of tasks on any other stratum) this reduction can be performed in polynomial time.

Finally, observe that we did not refer to executability in this reduction. So if we reversed all $\prec$ relations in the above reasoning, we would likewise get a reduction from plan existence for tail-recursive planning problems to plan existence for head-recursive HTN problems. Hence, as mentioned, since the plan existence problem for the tail-recursive setting is known to be EXPSPACE-complete, and noting that we can verify that the above reductions can be performed in polynomial time, this shows the plan existence problem for the head-recursive setting is also EXPSPACE-complete. □

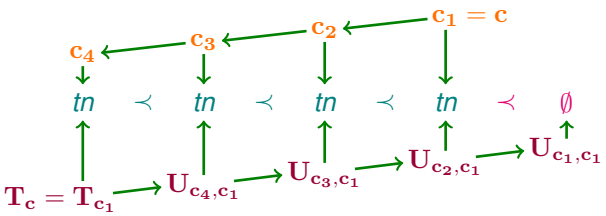


Figure 1: Illustration of an example conversion from head- to tail-recursive decompositions. Arrows denote decompositions, and *tn*s denote arbitrary task networks produced by the decompositions.

As mentioned in Theorem 4, it is possible to convert between left- and right-linear tasks in the same way, noting that left-linear tasks are head-recursive under the trivial stratification $c \leq c'$ for any compound tasks $c, c' \in N_C$, and observing that the above algorithm would convert them to a right-linear form.

5.2 Mixed- and Almost-Tail-Recursive Problems

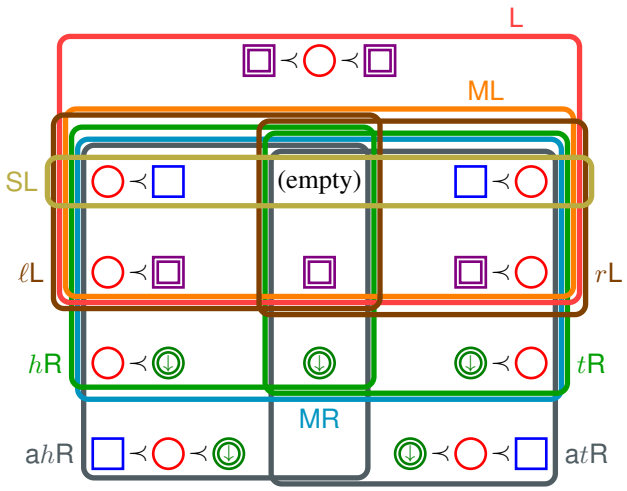
In analogy to mixed-linear problems, we could also consider a notion of mixed-recursiveness, defined as follows:

Definition 17. We call an HTN planning problem *mixed-recursive* if there exists some stratification $\leq_r$ such that every decomposition method $\langle c, tn \rangle \in M$ has tn either head-recursive under $\leq_r$ **or** tail-recursive under $\leq_r$.

One of our motivations for considering tail-recursion comes from a result by Alford et al. (2012), that plan existence for a given set of decompositions is solvable via progression if and only if the decompositions are tail-recursive under some stratification. We will show that this result is, in a sense, tight. However, the definition of mixed-recursiveness is a fairly large relaxation of this condition, so we need a different definition which is only a slight generalisation of the definition of a tail-recursive problem.

Definition 18. Given a stratification $\leq_r$, we call a task network *almost-tail-recursive under* $\leq_r$ if it is in the form $tn' \prec p$ for some task network tn which is tail-recursive under $\leq_r$ and primitive task p. We call an HTN planning problem *almost-tail-recursive* if there exists some stratification $\leq_r$ such that every decomposition method $\langle c, tn \rangle \in M$ is either tail-recursive or almost-tail-recursive under $\leq_r$.

Similarly to how linear problems can be reduced to mixed-linear problems (Corrs. 6, 9), almost-tail-recursive problems can be reduced to mixed-recursive problems. Both these classes are shown in Figure 2.



Key:

L:	linear	hR:	head-recursive
ML:	mixed linear	tR:	tail-recursive
SL:	simple linear	MR:	mixed-recursive
ℓL:	left-linear	ahR:	almost head-recursive
rL:	right-linear	atR:	almost tail-recursive

Figure 2: Diagram showing the various shapes of allowed task networks in the various problem classes discussed throughout in this paper. Blue squares denote primitive tasks, purple double squares denote primitive task networks, red circles denote compound tasks, and green double circles containing the symbol '↓' denote task networks which, if the task network is in a decomposition $\langle c, tn \rangle$, have all their tasks either primitive or on a strictly lower stratum than c. '(empty)' represents the task network $\langle \emptyset, \emptyset, \emptyset \rangle$. We have not depicted the various classes 'with X decompositions' (e.g. with linear decompositions); recall here that the initial task network can be arbitrary and all decomposition methods must obey the relevant constraint X (e.g. linearity, in the case of linear decompositions).

Unfortunately, as a corollary of Theorem 8, and recalling that we can replace tn_I with a single compound task:

Corollary 11. *The plan existence problem for almost-tail-recursive HTN planning problems is undecidable, even if tn_I consists of exactly one compound task.*

Proof. Any HTN planning problem with simple linear decompositions is also an almost-tail-recursive problem under the trivial stratification ($c \leq_r c'$ for every $c, c' \in N_C$). □

This extends the result by Alford et al. (2012) that such problems cannot be solved by progression. Furthermore:

Corollary 12. *The plan existence problem for mixed-recursive HTN planning problems is undecidable, even if tn_I consists of exactly one compound task.*

Proof. If $tn \prec C \prec p$ is a decomposition method in the almost-tail-recursive case where C is a compound task with $c \leq_r \alpha(C) \leq_r c$ and p is primitive, we can replace this with $tn \prec C'$ for a new compound task C' which only decomposes into $C \prec p$. This is mixed-recursive as we can create a new stratification $\leq_{r'}$ generated by $\leq_r$ and the relations $C \leq_{r'} C' \leq_{r'} C$ for any compound task $C \in N_C$. □

6 Conclusions

We have explored a wide variety of modifications to two common restrictions on HTN planning: regularity, and its generalisation, tail-recursiveness. In particular, we have discovered some heavily restricted classes of HTN problems for which plan existence is undecidable, which we believe is a significant contribution as our new problem classes are small variations of actively studied problem classes (unlike those produced by the standard undecidability reduction). While we haven't addressed specific applications to planners, the reductions we have found could be useful for identifying compilations from more complex restrictions of HTN planning to easier problems. (For example, Behnke et al. (2022) show that bounds on progression can produce compilations from tail-recursive problems to STRIPS.) Conversely, the undecidability results we have found inform us that compilations cannot possibly exist from their respective problem classes, so for example when modelling problems as HTN planning problems we should avoid modelling them in a way which allows for the full generality of mixed-recursive problems.

One line of investigation for future study which further generalises the linearity condition would be to allow task networks to have multiple compound tasks, so long as every compound task is ordered with respect to every other task in the network. It is clear to us that plan existence for the resultant problem class lies in EXPSPACE and is PSPACE-hard, but its exact complexity is unclear.

Acknowledgements

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. W. 2015. Tight Bounds for HTN Planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*, 7–15. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN Problem Spaces: Structure, Algorithms, Termination. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012*, 2–9. AAAI Press.

Behnke, G.; Pollitt, F.; Höller, D.; Bercher, P.; and Alford, R. 2022. Making Translations to Classical Planning Competitive with Other HTN Planners. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*, 9687–9697. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning - One Abstract Idea, Many Concrete

Realizations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, 6267–6275. ijcai.org.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.*, 69(1-2): 165–204.

Dekker, M.; and Behnke, G. 2024. Barely Decidable Fragments of Planning. In *ECAI 2024 – 27th European Conference on Artificial Intelligence*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, 4198–4206. IOS Press.

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. *Artif. Intell.*, 2(3/4): 189–208.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 1955–1961. AAAI.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. *Introduction to automata theory, languages, and computation, 2nd Edition*. Addison-Wesley series in computer science. Addison-Wesley-Longman. ISBN 978-0-201-44124-6.

Zhang, Y.; and Bercher, P. 2025. Computational Complexity of Planning for Recursive Primitive Task Networks: Selective Action Nullification with State Preservation. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025)*. IJCAI.