Language Classification of Hierarchical Planning Problems

Daniel Höller and **Gregor Behnke** and **Pascal Bercher** and **Susanne Biundo**[1]

Abstract. Theoretical results on HTN planning are mostly related to the plan existence problem. In this paper, we study the structure of the generated plans in terms of the language they produce. We show that such languages are always context-sensitive. Furthermore we identify certain subclasses of HTN planning problems which generate either regular or context-free languages. Most importantly we have discovered that HTN planning problems, where preconditions and effects are omitted, constitute a new class of languages that lies strictly between the context-free and context-sensitive languages.

1 Introduction

Hierarchical Task Network (HTN) planning [6] is an approach for solving planning problems that relies on abstract – *compound* – tasks and the refinement (decomposition) thereof into task networks containing both compound and *primitive* tasks. Primitive tasks correspond to actions in standard classical planning; a solution to an HTN planning problem is a task network consisting of primitive tasks that is executable and was generated from the *initial* task network; hence, the problem is to find a suitable sequence of decompositions in order to generate such a solution. That problem is, without further restrictions, known to be semi-decidable [6].

The representation of such problems (given: a set of primitive and compound tasks, a set of *decomposition methods* mapping compound tasks to task networks, and an initial task network) shows major similarities to formal grammars (given: a set of terminal- and non-terminal symbols, a set of rules, and a non-terminal start symbol). It is already known that context-free grammars can be encoded within the HTN planning framework. In fact, its encoding is used for Erol et al.'s proof for the semi-decidability of HTN planning [6, 8]. However, "the other way round" is not clear; i.e., whether HTN planning problems can express languages that are not context-free. Thus, in this paper, we study different kinds of HTN planning problems and their relation to formal languages.

Our results are summarized in Tab. 1. The most interesting result is that HTN planning problems including only actions without preconditions and effects, $HTN^{0\ pre}_{0\ eff}$, lie strictly between the context-free and the context-sensitive languages. Thus, $HTN^{0\ pre}_{0\ eff}$ can be regarded as new class of languages and grammars. This seems to be an important insight, as context-free languages are considered structurally relatively simple, whilst context-sensitive languages are highly complex, as they can express **PSPACE-complete** problems.

[1] Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany, {*daniel.hoeller, gregor.behnke, pascal.bercher, susanne.biundo*}@uni-ulm.de

Table 1. Summary of our results: languages generated by different versions of hierarchical planning and their relation to the Chomsky Hierarchy.

Loops	Partial Order	Preconditions and Effects	Task Insertion	Relation to Chomsky Class	
yes	yes	yes	yes	$TIHTN \subseteq REG$	Thm. 5
yes	yes	no	yes	$TIHTN^{0\ pre}_{0\ eff} \subsetneq REG$	Cor. 1
no	yes	yes	no	$HTN_{acyc} \subsetneq REG$	Thm. 3
yes	no	yes	no	$HTN_{ord} = CF$	Thm. 6
yes	yes	no	no	$CF \subsetneq HTN^{0\ pre}_{0\ eff} \subsetneq CS$	Thm. 8, 9
yes	yes	yes	no	$HTN \subseteq CS$	Thm. 7

2 Hierarchical Planning

In this section we first describe the HTN planning formalization we base our results upon. We use the one given in earlier work [8], where Geier and Bercher identified a relaxation of "pure" HTN planning that lowers the complexity of the plan existence problem from semi-decidable to **EXPSPACE** membership. That relaxation is the capability to insert actions into task networks without them being introduced via the decomposition of a compound task. They refer to the standard setting as HTN *planning*, and to the setting where inserting actions is allowed as HTN *planning with Task Insertion* (TIHTN).

Now, we first define these two problem classes and then extend the work of Geier and Bercher [8] by proposing a normal form for these problems and prove their existence for any HTN or TIHTN problem.

2.1 Problem Formalization

We start by describing task networks, which are partially ordered sets of tasks. A *task* is a unique identifier. Each task is mapped to a so-called *task name*. Task names, on the other hand are the "names", which in turn map to the actual operators that (finally) show the preconditions and effects.

Definition 1 (Task Network) *A* task network $\mathrm{tn} = (T, \prec, \alpha)$ *over a set of* task names N *is a 3-tuple, where*

- T *is a finite and non-empty set of tasks*
- $\prec\ \subseteq T \times T$ *is a strict partial order on* T *(irreflexive, asymmetric, and transitive)*
- $\alpha : T \to N$ *labels every task with a task name, its inverse is given by* $\alpha^{-1} : N \to 2^T$ *with* $\alpha^{-1} : n \mapsto \{t \mid \alpha(t) = n\}$

TN_N *denotes the set of all task networks over the task names* N*. By abuse of notation, we write* $T(\mathrm{tn}) = T$ *for* $\mathrm{tn} = (T, \prec, \alpha)$*.*

Having this definition at hand we can now formally define HTN and TIHTN planning problems. In earlier work, Geier and

Bercher [8] showed that the two problem classes of HTN and TIHTN differ solely in their solution criterion, while an identical syntactical representation can be used. Hence, we will skip the terms "TIHTN" and "HTN", when it comes to the (syntactical) problem description.

Definition 2 (Planning Problem) *A* planning problem *is a 6-tuple* $\mathcal{P} = (L, C, O, M, c_I, s_I)$*, where*

- L*, a finite set of* proposition symbols
- C*, a finite set of* compound task names
- O*, a finite set of* primitive task names *with* $C \cap O = \emptyset$
- $M \subseteq C \times TN_{C \cup O}$*, a finite set of* (decomposition) methods
- $c_I \in C$*, the* initial task name
- $s_I \in 2^L$*, the* initial state

For each primitive task name $o \in O$*, its* operator *(or* action*) is given by* $(prec_o, add_o, del_o) \in 2^L \times 2^L \times 2^L$ *and it consists of a precondition, an add-, and a delete list; the latter two constitute the effects. We denote by* $\mathrm{tn}_I := (\{t_I\}, \emptyset, \{(t_I, c_I)\})$ *the initial task network.*

Before we can state how to solve planning problems (and how solutions look like), we need to introduce the concepts of *decomposition* and *task insertion*. For the former, we first define restrictions on relations and functions: Let $R \subseteq D \times D$ be a relation. Its restriction to a set X is given by $R|_X := \{(r_1, r_2) \in R \mid r_1, r_2 \in X\}$. The restriction on a function $f : D \to V$ is defined as $f|_X := \{(d, v) \in f \mid d \in X\}$. That restriction operator can also be applied to task networks given by $\mathrm{tn}|_{T'} := (T \cap T', \prec|_{T'}, \alpha|_{T'})$.

Two task networks $\mathrm{tn} = (T, \prec, \alpha)$ and $\widetilde{\mathrm{tn}} = (\widetilde{T}, \widetilde{\prec}, \widetilde{\alpha})$ are called *isomorphic*, written $\mathrm{tn} \cong \widetilde{\mathrm{tn}}$, if and only if there exists a bijection $\sigma : T \to \widetilde{T}$, such that for all tasks $t, t' \in T$ it holds that $(t, t') \in \prec$ if and only if $(\sigma(t), \sigma(t')) \in \widetilde{\prec}$ and $\alpha(t) = \widetilde{\alpha}(\sigma(t))$.

Definition 3 (Decomposition) *A method* $m = (c, \mathrm{tn}_m) \in M$ *decomposes a task network* $\mathrm{tn} = (T, \prec, \alpha)$ *into a task network* tn' *by replacing task* t*, written* $\mathrm{tn} \xrightarrow[t,m]{} \mathrm{tn}'$*, if and only if* $t \in T$*,* $\alpha(t) = c$*, and there exists a task network* $\widetilde{\mathrm{tn}}_m = (\widetilde{T}_m, \widetilde{\prec}_m, \widetilde{\alpha}_m)$ *with* $\mathrm{tn}_m \cong \widetilde{\mathrm{tn}}_m$ *and* $T \cap \widetilde{T}_m = \emptyset$*, where*

$$\begin{aligned}
\mathrm{tn}' &:= (T', \prec \cup \widetilde{\prec}_m \cup \prec_{dec\text{-}t}, \alpha \cup \widetilde{\alpha}_m)|_{T'} \text{ with} \\
T' &:= (T \setminus \{t\}) \cup \widetilde{T}_m \\
\prec_{dec\text{-}t} &:= \{(t_1, t_2) \in T \times \widetilde{T}_m \mid (t_1, t) \in \prec\} \cup \\
&\qquad \{(t_1, t_2) \in \widetilde{T}_m \times T \mid (t, t_2) \in \prec\}
\end{aligned}$$

We write $\mathrm{tn} \to_D^* \mathrm{tn}'$*, if* tn *can be decomposed into* tn' *using an arbitrary number of decompositions.*

While in the (pure) HTN problem setting, changing task networks is only possible via decomposition of compound tasks, the TIHTN setting also allows the alteration of the same via *task insertion*.

Definition 4 (Task Insertion) *Let* $\mathrm{tn} = (T, \prec, \alpha)$ *be a task network.*

Let $o \in O$ *be a primitive task name; then, a task network* tn' *can be obtained from* tn *by insertion of* o*, if and only if for some* $t \notin T$*,* $\mathrm{tn}' = (T \cup \{t\}, \prec, \alpha \cup \{(t, o)\})$.

Let (t_1, t_2) *be an ordering constraint; then, a task network* $(T, \prec', \alpha)$ *can be obtained from* tn *by insertion of* (t_1, t_2)*, if and only if* $t_1, t_2 \in T$ *and* $\prec'$ *is the transitive closure of* $\prec \cup \{(t_1, t_2)\}$*. We write* $\mathrm{tn} \to_I^* \mathrm{tn}'$*, if* tn' *can be obtained from* tn *by adding an arbitrary number of primitive task names and ordering constraints.*

We proceed by defining a task network as being executable if there *exists* a linearization of its tasks that is executable in the standard way. We thereby follow our previous definition [8] and the one given by Erol et al. [6]. Note that one could also define it in such a way that *every* linearization needs to be executable, as it is required in *Hybrid planning* – an approach that fuses HTN planning with POCL planning [3, 5]. Note that this difference may influence the complexity of the plan existence problem [9, Thm. 14, 15],[2, Thm. 1],[6, Thm. 8].

Definition 5 (Executable Task Network) *A task network* $(T, \prec, \alpha)$ *is* executable in a state $s \in 2^L$*, if and only if it is primitive, i.e., for all* $t \in T$ *holds* $\alpha(t) \in O$ *and there exists a linearization of its tasks* $t_1, \ldots, t_n$ *that is compatible with* $\prec$ *and a sequence of states* $s_0, \ldots s_n$ *such that* $s_0 = s$*,* $prec_{\alpha(t_i)} \subseteq s_{i-1}$*, and for all* $1 \leq i \leq n$ *holds* $s_i = (s_{i-1} \setminus del_{\alpha(t_i)}) \cup add_{\alpha(t_i)}$.

Definition 6 (Solution) *A task network* tn_S *is a solution to a planning problem* $\mathcal{P}$*, if and only if*

(1) tn_S *is executable in* s_I *and*
(2) $\mathrm{tn}_I \to_D^* \mathrm{tn}_S$ *for* tn_S *being a* HTN *solution to* $\mathcal{P}$ *or*
(2') *there exists a task network* tn *such that* $\mathrm{tn}_I \to_D^* \mathrm{tn} \to_I^* \mathrm{tn}_S$ *for* tn_S *being a* TIHTN *solution to* $\mathcal{P}$.

$Sol_{\text{HTN}}(\mathcal{P})$ *and* $Sol_{\text{TIHTN}}(\mathcal{P})$ *denote the sets of all* HTN *and* TIHTN *solutions of* $\mathcal{P}$*, respectively.*

2.2 A Normal Form for Hierarchical Planning

In this section, we propose a normal form for HTN and TIHTN planning problems and prove that any such problem can be transformed into it. It is used in the proof of Thm. 7.

Definition 7 (Normal Form for Planning Problems) *A planning problem* $\mathcal{P} = (L, C, O, M, c_I, s_I)$ *is in 1-free-Normal-Form (*$\text{NF}_{\neq 1}$*) if and only if for all methods* $(c, (T, \prec, \alpha)) \in M$ *it holds:*

$$(c \neq c_I \Rightarrow |T| \neq 1) \text{ and } \nexists t \in T \text{ with } \alpha(t) = c_I$$

and it is in 2-Normal-Form ($\text{NF}_{\geq 2}$*) if and only if for all methods* $(c, (T, \prec, \alpha)) \in M$ *it holds:*

$$(c \neq c_I \Rightarrow |T| \geq 2) \text{ and } \nexists t \in T \text{ with } \alpha(t) = c_I$$

The following two theorems state that every planning problem can be transformed into these normal forms without changing its set of solutions. The proof for the $\text{NF}_{\geq 2}$ will utilize the existence of a $\text{NF}_{\neq 1}$ for a given planning problem. Both proofs are constructive, thus the normal forms can be obtained in practice.

Theorem 1 *For every planning problem* $\mathcal{P} = (L, C, O, M, c_I, s_I)$ *there exists a planning problem* $\mathcal{P}'$ *in* $\text{NF}_{\neq 1}$*, such that:*

$$Sol_{\text{HTN}}(\mathcal{P}) = Sol_{\text{HTN}}(\mathcal{P}') \text{ and } Sol_{\text{TIHTN}}(\mathcal{P}) = Sol_{\text{TIHTN}}(\mathcal{P}')$$

Proof: We obtain that there is no method $(c, (T, \prec, \alpha))$ with $t \in T$ and $\alpha(t) = c_I$ by introducing a new initial task name c_I^* with a single method mapping it to the task network containing exactly c_I. Concerning the other criterion, we obtain $c \neq c_I \Rightarrow |T| \neq 1$ by induction on the number n of compound task names $c \neq c_I$, with $(c, (T, \prec, \alpha)) \in M$ and $|T| = 1$.

Case $n = 0$: Proved. Case $n > 0$: Let $c \in C$ be a task that violates the criterion and $M_{=1}(c) := \{(c, (T, \prec, \alpha)) \in M \mid |T| = 1\}$ the set of violating methods and $N_{=1}(c) := \{\alpha(t) \mid (c, (\{t\}, \emptyset, \alpha)) \in$

$M_{=1}(c)\} \setminus \{c\}$ the set of task names they lead to. Let $M^-(c) := \{(c', (T, \prec, \alpha)) \in M \mid t \in T, \alpha(t) = c\}$ be the set of all methods that generate c. We now generate a new planning problem $\mathcal{P} = (L, C, O, M', c_I, s_I)$, where all methods from $M_{=1}(c)$ are removed and their "effects" are propagated upwards to all methods in $M^-(c)$. Thus, $M' := (M \setminus M_{=1}(c)) \cup M_{subst}$ with:

$$M_{subst} := \{(c', (T, \prec, \alpha|_{T \setminus T'} \cup \alpha')) \mid (c', (T, \prec, \alpha)) \in M^-(c), \\ T' \subseteq \alpha^{-1}(c), \alpha' \subseteq T' \times N_{=1}(c), \text{ s.t. } \alpha' \text{ is a function}\}$$

Obviously, n is reduced by 1. It is clear that the proposed transformation does not change the set of solutions – independently of the two solution criteria. □

The set M_{subst} may in general contain exponentially many decomposition methods, as for any subset $T' \subseteq \alpha^{-1}(c)$, i.e., any subset of occurrences of c in a method, a new method is added to the original set of methods. Fig. 1 gives an example where it is necessary to replace all possible subsets of occurrences of T_2 in the method of T_1.

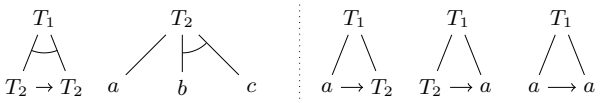


Figure 1. The left hand side gives a planning problem as AND/OR graph that depicts the necessity to replace all possible subsets in order to propagate $\text{NF}_{\neq 1}$ violations. It contains two compound (T_1 and T_2) and three primitive tasks (a, b and c). T_1 can be decomposed into two ordered tasks (both named T_2) that can either be decomposed into a single a or into both b and c. The methods on the right will be added to obtain $\text{NF}_{\neq 1}$, while the method that decomposes T_2 into a single a will be removed.

Theorem 2 *For every planning problem $\mathcal{P} = (L, C, O, M, c_I, s_I)$ there exists a planning problem $\mathcal{P}'$ in $\text{NF}_{\geq 2}$, such that:*

$$Sol_{\text{HTN}}(\mathcal{P}) = Sol_{\text{HTN}}(\mathcal{P}') \text{ and } Sol_{\text{TiHTN}}(\mathcal{P}) = Sol_{\text{TiHTN}}(\mathcal{P}')$$

Proof: We prove the claim by induction on the number n of compound task names with "empty" methods, i.e., task names $c \neq c_I$ with $(c, (\emptyset, \emptyset, \emptyset)) \in M$.

Case $n = 0$: To eliminate $\text{NF}_{\neq 1}$ violations, we use the (constructive) proof of Thm. 1. Since that proof does not introduce further "empty" methods we obtain $\text{NF}_{\geq 2}$.

Case $n > 0$: Due to Thm. 1 there is a planning problem $\mathcal{P}' = (L, C, O, M', c_I, s_I)$ in $\text{NF}_{\neq 1}$ that is equivalent to $\mathcal{P}$. Both have the same number of "empty" methods. There is at least one compound task name $c \neq c_I$ with an "empty" method $m_\varepsilon = (c, (\emptyset, \emptyset, \emptyset)) \in M'$. Let $M^-(c) = \{(c', (T, \prec, \alpha)) \in M' \mid t \in T, \alpha(t) = c\}$ be the set of methods leading to (at least) one c. We can replace m_ε by several new methods given by $M'' := (M' \setminus \{m_\varepsilon\}) \cup M_{subst}$ where

$$M_{subst} := \{(c', \text{tn}|_{T(\text{tn}) \setminus T'}) \mid (c', \text{tn}) \in M^-(c), T' \subseteq \alpha^{-1}(c)\}$$

The solution sets remain equal. Instead of using m_ε to delete c, a method from M_{subst} is used that does not produce c in the first place. The modification may introduce new "empty" methods, at most one for each task name c'. This happens if and only if c' can be decomposed solely into multiple instances of c. If this is the case, the newly added method can be propagated upwards itself. But the propagation must be done at most once for each task name. The second propagation would not add new decomposition methods. Thus the number of necessary propagations is limited. □

3 Language Classification

It is widely known that hierarchical planning can be used to encode context-free grammars. Erol et al. [6] used that fact to prove that (unrestricted) hierarchical planning is semi-decidable. Geier and Bercher [8] showed that their proof can still be applied in the simplified planning formalization used in this paper. However, we are unaware of any formal studies on which types of languages can be expressed by hierarchical planning problems or loosened variants thereof. We will begin by defining several such classes and then proceed by investigating *structural restricted variants* (such as acyclic problems) and the influence of the solution criterion (i.e. HTN vs. TIHTN). Then, we investigate *restrictions to preconditions and effects* by considering the case of no-operations.

We will refer to the class of regular languages as REG, to the (class of) context-free languages as CF, and to the context-sensitive languages as CS [4]. Each of the three classes has a corresponding type of grammar that generates it: the regular, context-free and context-sensitive grammars, respectively. In addition to these standard grammars, there are also several other grammars, such as ID/LP grammars [11, 10, 7] used in language recognition.

ID/LP grammars seem to be close to our hierarchical setting and in particular to $HTN^{0\ pre}_{0\ \mathit{eff}}$, the HTN variant without preconditions and effects. Both allow a *partial order* on their grammar rules. However, Nederhof et al. [10] showed that ID/LP grammars are equivalent to context-free grammars, as after the application of a partially ordered rule, a total order on the newly generated word must be chosen. Hence, ID/LP grammars are only a more compact representation of context-free grammars. In HTN planning, the ordering decision may be postponed. We will show that this enables the expression of more languages and that $HTN^{0\ pre}_{0\ \mathit{eff}}$ is a strict superset of CF.

Definition 8 (Hierarchical Language Classes) *Let $\mathcal{P}$ be a planning problem. We define the language of a planning problem $\mathcal{P}$ using the solution criterion* $\text{S} \in \{\text{HTN}, \text{TIHTN}\}$ *by*

$$L_{\text{S}}(\mathcal{P}) := \{\omega \mid \omega \textit{ is an executable linearization of } \text{tn} \in Sol_{\text{S}}(\mathcal{P})\}$$

Now, we can define the following classes *of languages:*

- $HTN := \{L_{\text{HTN}}(\mathcal{P}) \mid \mathcal{P}$ *is a planning problem*$\}$
- $HTN_{acyc} := \{L_{\text{HTN}}(\mathcal{P}) \mid \mathcal{P}$ *is an acyclic*[2] *planning problem*$\}$
- $HTN_{ord} := \{L_{\text{HTN}}(\mathcal{P}) \mid \mathcal{P}$ *is a planning problem, where each decomposition method $m \in M$ is totally ordered*[3]$\}$
- $HTN^{0\ pre}_{0\ \mathit{eff}} := \{L_{\text{HTN}}(\mathcal{P}) \mid \mathcal{P}$ *is a planning problem, where for each $o \in O, prec_o = add_o = del_o = \emptyset$*$\}$
- $TIHTN := \{L_{\text{TIHTN}}(\mathcal{P}) \mid \mathcal{P}$ *is a planning problem*$\}$
- $TIHTN^{0\ pre}_{0\ \mathit{eff}} := \{L_{\text{TIHTN}}(\mathcal{P}) \mid \mathcal{P}$ *is a planning problem, where for all $o \in O, prec_o = add_o = del_o = \emptyset$*$\}$
- $EXE := \{L_{\text{TIHTN}}(\mathcal{P}) \mid \mathcal{P}$ *is a planning problem without hierarchy, i.e., $M = \{(c_I, (\emptyset, \emptyset, \emptyset))\}$*$\}$

Note that our definition of the language of a planning problem, $L_{\text{S}}(\mathcal{P})$, uses only the executable linearizations of solution task networks. If we use all linearizations induced by a solution, some of them might not be executable (cf. Def. 5). This means that they would not be able to transfer the initial state into a goal state.

[2] A planning problem is acyclic if the size of possible decomposition trees [8, Def. 7, 8] is limited by some constant.

[3] A method $m = (c, (T, \prec, \alpha))$ is totally ordered if $\prec$ is a total order.

3.1 Unrestricted Preconditions and Effects

We start by classifying the languages of planning problems with restrictions on the decomposition methods.

Theorem 3 $HTN_{acyc} \subsetneq REG$

Proof: The language of each acyclic planning problem is finite and thus regular. The two classes are not equal, the regular language Σ^* (with $\Sigma \neq \emptyset$) cannot be generated by an acyclic planning problem. □

Next we want to investigate the class EXE of executable sequences of primitive task names, which plays an important role in the proofs in this section.

Theorem 4 $EXE \subsetneq REG$

Proof: Let $\mathcal{P} = (L, C, O, \{(c_I, (\emptyset, \emptyset, \emptyset))\}, c_I, s_I)$ be a planning problem without hierarchy. $L_{\text{TiHTN}}(\mathcal{P})$ contains all executable sequences of primitive task names $o \in O$. We define an automaton $A = (\Sigma, S, s_0, \delta, F)$ with $\Sigma = O$, $S = 2^L \cup \{\dagger\}$, $s_0 = s_I$, $F = 2^L$,

$$\delta(s, o) := \begin{cases} \dagger & \text{if } s = \dagger \text{ or } \text{prec}_o \not\subseteq s \\ (s \setminus \text{del}_o) \cup \text{add}_o & \text{else} \end{cases}$$

That automaton A keeps track of the state generated by an input word w being a sequence of primitive task names. If the sequence is not executable the state † will be reached. This automaton accepts exactly $L_{\text{TiHTN}}(\mathcal{P})$ making EXE regular. For each language in EXE holds: if a word w is in the language, so is also each prefix of w. Thus, the regular language $\{ab\}$ can't be expressed by EXE. □

Having Thm. 4 at hand, we can start to examine the other language classes. For each planning problem $\mathcal{P} = (L, C, O, M, c_I, s_I)$ we define $\mathcal{P}_{\text{no-H}} = (L, C, O, \{(c_I, (\emptyset, \emptyset, \emptyset))\}, c_I, s_I)$ to be a relaxed version of $\mathcal{P}$ not showing hierarchy and $\mathcal{P}_{\text{no-PE}}$ to be the relaxed version of $\mathcal{P}$ not showing preconditions or effects for its operators. Note that for both solution criteria S $\in \{\text{HTN}, \text{TiHTN}\}$: $L_{\text{S}}(\mathcal{P}) = L_{\text{S}}(\mathcal{P}_{\text{no-PE}}) \cap L_{\text{TiHTN}}(\mathcal{P}_{\text{no-H}})$ and that $L_{\text{TiHTN}}(\mathcal{P}_{\text{no-H}})$ is regular due to Thm. 4. Hence, for the remaining theorems it often suffices to show the results we are interested in just for $L_{\text{S}}(\mathcal{P}_{\text{no-PE}})$.

Theorem 5 $TIHTN \subseteq REG$

Proof: Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem. Because $L_{\text{TiHTN}}(\mathcal{P}) = L_{\text{TiHTN}}(\mathcal{P}_{\text{no-PE}}) \cap L_{\text{TiHTN}}(\mathcal{P}_{\text{no-H}})$ it suffices to show that $L_{\text{TiHTN}}(\mathcal{P}_{\text{no-PE}})$ is regular, as the intersection of two regular languages is regular. Lem. 2 of the paper of Geier and Bercher [8] provides that every word $\omega \in L(\mathcal{P})$ contains a word ω' as a non-consecutive substring that can be obtained by decomposing c_I and having a size of at most $m^{|C|}$, where $m = \max_{(c,(T,\prec,\alpha)) \in M} |T|$. Since ω' can be interleaved with arbitrary inserted tasks, it is sufficient so check whether the input ω contains such an ω'. We construct a non-deterministic automaton A. Let $O_p = \{\omega \mid \text{tn}_I \to_D^* \text{tn}, \text{tn} \text{ is primitive}, |T(\text{tn})| \leq m^{|C|}, \text{and } \omega \text{ is a linearization of tn}\}$ be the set of all "small" words that can be obtained by decomposing c_I. Each word in $L_{\text{TiHTN}}(\mathcal{P}_{\text{no-PE}})$ contains at least one element $\omega \in O_p$ as a non-consecutive substring. An automaton is given by $A = (\Sigma, S, s_0, \delta, F)$ with $\Sigma = O$, $S = \bigcup_{n=0}^{m^{|C|}} \Sigma^n$, $s_0 = \varepsilon$, $F = O_p$ and $\delta(s, o) = \{s, s \circ o\} \cap S$, where $\circ$ denotes string concatenation. It accepts a word ω if and only if it contains $\omega' \in O_p$ as a non-consecutive substring. □

Further, one can see that $TIHTN^{0\ pre}_{0\ eff}$ cannot describe finite languages. Combining this fact with the last theorem, we can conclude:

Corollary 1 $TIHTN^{0\ pre}_{0\ eff} \subsetneq REG$

Next we consider problems where each method is totally ordered.

Theorem 6 $HTN_{ord} = CF$

Proof: $\underline{CF \subseteq HTN_{ord}}$: Let $G = (T, NT, R, S)$ be a context-free grammar. We construct a totally ordered planning problem $\mathcal{P}$ with $L(G) = L_{\text{HTN}}(\mathcal{P})$. It is similar to Erol et al.'s proof of HTN semi-decidability [6, Thm. 1]. It is $\mathcal{P} = (\emptyset, NT, T, M, S, \emptyset)$ with

$$M := \{(A, (\{t_1, \ldots, t_n\}, \{(t_i \prec t_j) \mid 1 \leq i < n \text{ and } i < j \leq n\}, \\ \{(t_i, \omega_i) \mid 1 \leq i \leq n\}) \mid A \to \omega_1 \omega_2 \ldots \omega_n \in R\}$$

$\underline{HTN_{ord} \subseteq CF}$: Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem in which all decomposition methods are totally ordered. Consider $L_{\text{HTN}}(\mathcal{P}) = L_{\text{HTN}}(\mathcal{P}_{\text{no-PE}}) \cap L_{\text{TiHTN}}(\mathcal{P}_{\text{no-H}})$.

Then the proposition holds if $L_{\text{HTN}}(\mathcal{P}_{\text{no-PE}})$ is context-free. This can be obtained by constructing a grammar $G = (O, C, R, c_I)$, s.t. $L(G) = L_{\text{HTN}}(\mathcal{P}_{\text{no-PE}})$ with R given by

$$R := \{c \to \omega \mid (c, (T, \prec, \alpha)) \in M \text{ and } \omega = \alpha(t_1)\alpha(t_2)\ldots\alpha(t_n) \\ \text{where } t_1 t_2 \ldots t_n \text{ is the linearization of } T \text{ according to } \prec\}$$ □

Finally, we classify unrestricted HTN planning problems.

Theorem 7 $HTN \subseteq CS$

Proof: To prove the claim it suffices to show the existence of a linear bounded automaton (LBA) that solves the word problem for $L_{\text{HTN}}(\mathcal{P})$ [4, p. 331] for every HTN planning problem $\mathcal{P}$ (an *acceptor*). Using the partition of the HTN languages $L_{\text{HTN}}(\mathcal{P}) = L_{\text{HTN}}(\mathcal{P}_{\text{no-PE}}) \cap L_{\text{TiHTN}}(\mathcal{P}_{\text{no-H}})$ and knowing that CS is closed under intersection [4, p. 337] we only need to show that our claim holds for all $HTN^{0\ pre}_{0\ eff}$ languages. From Thm. 2 we know that a planning problem $\mathcal{P}' = (L', C', O', M', c_I', s_I')$ in $\text{NF}_{\geq 2}$ exists that is equivalent to $L_{\text{HTN}}(\mathcal{P}_{\text{no-PE}})$. The increased number of methods in M' does not matter here as the problem $\mathcal{P}'$ is not part of the LBA's input.

Alg. 1 gives a generic acceptor. It first non-deterministically generates a so-called *decomposition tree* [8, Def. 7, 8]. Then, each symbol of the input word ω is matched to a primitive task in that tree. After each matching, the ordering constraints of this last symbol to all other primitive tasks are checked. Note that in HTN planning an ordering constraint between two tasks a and b can solely be introduced by the method that decomposed some other task into two subtasks t_a and t_b, where t_a is ancestor of a (and not of b) and t_b is ancestor of b (not of a). The ordering has to be checked in exactly this decomposition.

The algorithm is now explained in detail. Starting from c_I a decomposition tree is generated in line 2–4. To represent the tree on the LBA's tape, an encoding is used that omits all ordering constraints. These are checked later on. For each method m_i an arbitrary but fixed total order of its subtasks is used to write its decomposition in the following form to the tape: $m_i(st_1 st_2 \ldots st_n)$, where m_i is a method identifier and the st_js are the names of its subtasks. Iteratively, the subtasks are again decomposed; $m_i(\ldots)$ thereby replaces the task it is applied to. A primitive task pt is represented by $pt()$. Consider the representation of the problem shown in Fig. 3(a): $m_1(m_2(a()b())m_3(d()c()))$, where each m_i decomposes the task T_i. As $\mathcal{P}'$ is in $\text{NF}_{\geq 2}$, the tree has at most $2|\omega|$ nodes. At most $|\omega|$ of

them can be inner nodes, as each inner node has at least two successors. The needed space is thus limited to $6|\omega|$, including braces.

For simplicity we describe the acceptor as a multi-tape LBA. This does not increase its expressivity. END is used to denote a symbol that limits the tape. The LBA has the following tapes: TAPE-1 of length $|\omega|$ holds the word to parse. TAPE-2 of length $6|\omega|$ is used to represent the (decomposition) tree; initially c_I is on the tape. TAPE-3 is used to simulate a push-down automaton that is needed to go one level up in the tree. Thus its size is bounded by the length of TAPE-2.

```
1  function LBA_{P_no-PE}(ω)
2      while ∃c ∈ C' on TAPE-2 do
3          Replace it non-deterministically by a decomposition
4          if reached END then return failure;
5      while HEAD-1 has not reached END do
6          Pick next symbol s from TAPE-1 and delete it
7          Non-det. choose an unmatched primitive task p in the tree
           g (on TAPE-2) whose name equals s
8          Mark p as matched
           // check ordering to other tasks
9          foreach primitive task o on TAPE-2 do
10             Set mark m_p to position of p and m_o to o
11             OK ← false
               // find method (using PDA)
12             while ¬OK ∧ ¬(m_p is root of tree) do
13                 Move m_p one layer up in the tree; Set m_o to o
14                 while ¬OK ∧ ¬(m_o is root of tree) do
15                     Move m_o one layer up in the tree
16                     if m_p and m_o mark the same method m then
17                         if (o ≺ p) is a valid ordering in m then
18                             OK ← true
19                         else return failure;
20     if TAPE-2 contains unmarked task then return failure;
21     return success
```

Algorithm 1. Generic linear bounded automaton that decides the word problem for $L_{\text{HTN}}(\mathcal{P}')$ languages.

In line 5, TAPE-2 contains a valid decomposition tree g. Now the symbols of ω are matched to primitive tasks in g and the ordering constraints are checked. The symbols of ω are iteratively deleted on TAPE-1 and a primitive task in g that has the same name as the symbol and has not been matched before is marked as matched (line 6–8).

Let s be the last matched symbol. Since ω is proceeded in a left-to-right order, the ordering $(o \prec s)$ must be a valid ordering for all other tasks o that are already marked in the tree. Starting from s and o and going up in the tree, the method that generates both an ancestor for o (denoted t_o) and a different one for s (denoted t_s) is searched and the ordering is checked: $(o \prec s)$ is valid if and only if $(t_o \succ t_s)$ does not hold in that method (line 12–19). To go up in the tree, its representation has to be parsed using a push-down automaton simulated on TAPE-3. The process is illustrated in Fig. 2 for the HTN given in Fig. 3(a): in the beginning a is already marked, then c is newly matched. By checking the ordering constraints iteratively, the ordering relation of the primitive tasks does not have to be maintained on the LBA's tape (this would not be possible due to the limited space).

After checking the decomposition and the ordering constraints, it is checked whether all primitive tasks in the tree have been matched (line 20). If this is the case, the word is accepted.

The acceptor generates a valid decomposition tree g, checks if the symbols in ω can be matched to primitive tasks having the same name and whether the ordering present in ω is valid in g. Thus it will accept every word in $L_{\text{HTN}}(\mathcal{P}')$. Words that are not in the language will violate (at least) one of the given constraints, otherwise they would be in the language. Thus they will be rejected. The space is linear-bounded in the size of ω. □

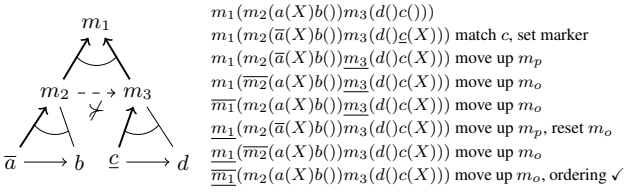


Figure 2. Example illustrating the check of the ordering relation of a newly matched c with an a that has been matched before.

One can even show $HTN \subsetneq CS$ using the context-sensitive language $\{a^p \mid p \text{ prime}\}$. To represent this infinite language, there must be a cycle in the HTN that leads to a non-prime number of as.

3.2 Restricted Preconditions and Effects

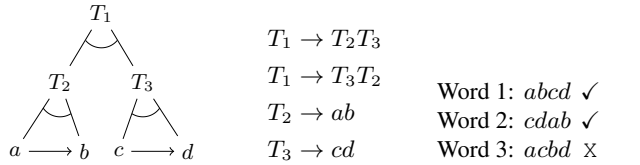
In this section, we classify the languages of planning problems with severe restrictions on the preconditions and effects: we only consider the case, where every primitive task name's operator is a no-operation, i.e., $(\emptyset, \emptyset, \emptyset)$.

The next corollaries follow from the proof of Thm. 7 and the first case of the proof given for Thm. 6 ($CF \subseteq HTN_{ord}$), respectively.

Corollary 2 $HTN^{0\ pre}_{0\ eff} \subseteq CS$

Corollary 3 $CF \subseteq HTN^{0\ pre}_{0\ eff}$

One might pose the question if we can stay context-free by transferring any partially ordered method of a precondition- and effect-free HTN problem into a set of totally ordered grammar rules. But though this approach provides the same ordering flexibility for a given decomposition, it introduces ordering constraints on the subtasks. Consider Fig. 3: using decomposition, all three words (and



(a) HTN definition (b) Context-free grammar translation and words

Figure 3. Planning problem whose language is not correctly transformed by the naive context-free translation.

even more) can be generated. When the decompositions are transferred to the four rules given in Fig. 3(b), only the words 1 and 2 can be generated by that grammar. Word 3 can not, though it is a member of the language induced by the planning problem. Thm. 8 states that it is not possible in general to transform an arbitrary precondition- and effect-free HTN planning problem into a context-free grammar.

Theorem 8 $CF \subsetneq HTN^{0\ pre}_{0\ \mathit{eff}}$

Proof: Consider the planning problem given in Fig. 4. An initial task T_1 is decomposed into two unordered tasks T_2 and T_3. The task T_2 can be decomposed into an arbitrary number of as followed by the same number of bs (cs and ds, for T_3). We will use the notation suggested by Nederhof et al. [10] to denote all interleavings of the words ω_1 and ω_2 via $\omega_1||\omega_2$. Then, $L_{\text{HTN}}(\mathcal{P}) = \bigcup_{n\in\mathbb{N}^+} \bigcup_{j\in\mathbb{N}^+} a^n b^n || c^j d^j$.

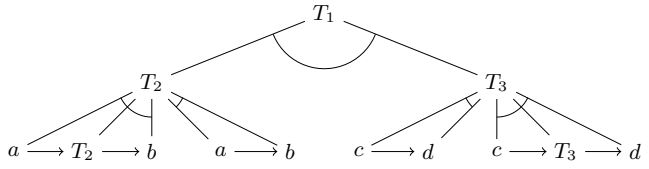


Figure 4. The planning problem $\mathcal{P}$ used in the proof of Thm. 8.

Suppose $CF = HTN^{0\ pre}_{0\ \mathit{eff}}$. Then, $L_{\text{HTN}}(\mathcal{P})$ is context-free. By the pumping lemma for context-free languages [4, p. 287], $\exists m \in \mathbb{N}$, so that $\forall z \in L_{\text{HTN}}(\mathcal{P})$ with $|z| \geq m$ holds: z can be written as $z = uvwxy$ with substrings u, v, w, x and y such that (i) $|vx| \geq 1$; (ii) $|vwx| \leq m$ and (iii) $uv^i wx^i y \in L, \forall i \in \mathbb{N}_0$ hold.

We choose $z = a^{m+1}c^{m+1}b^{m+1}d^{m+1}$ out of $L_{\text{HTN}}(\mathcal{P})$, obviously $|z| > m$. Let $z = uvwxy$ be an arbitrary partition of z fulfilling (i) and (ii). As each of the four segments of terminal symbols a, b, c, d has a length of $m+1$, vwx can contain at most two different terminal symbols. Thus, we can distinguish the following cases.

1. $vwx = k^i$ for some $k \in \{a, b, c, d\}$ with $0 < i \leq m$ by (ii). (i) ensures that vx contains at least one k. By (iii), $uv^2wx^2y \in L_{\text{HTN}}(\mathcal{P})$. The number of ks in uv^2wx^2y is strictly larger than in $uvwxy$, the number of all other terminals remain unchanged. By definition, the number of as must match the number of bs (and cs the number of ds), thus $uv^2wx^2y \notin L_{\text{HTN}}(\mathcal{P})$.
2. $vwx = k^i l^j$ for some $(k, l) \in \{(a, c), (c, b), (b, d)\}$ and $0 < i, j \leq m$ (ii). By (i) vx contains at least one k or one l. By (iii), $uv^2wx^2y \in L_{\text{HTN}}(\mathcal{P})$. uv^2wx^2y contains more ks and/or more ls than z, while the number of the other terminals remains unchanged. As k can not be the partner of l, there is a mismatch in the number of as and bs or cs and ds: $uv^2wx^2y \notin L_{\text{HTN}}(\mathcal{P})$.

In both cases we have obtained a contradiction, thus $L_{\text{HTN}}(\mathcal{P})$ can not be context-free. Using Cor. 3 we have $CF \subsetneq HTN^{0\ pre}_{0\ \mathit{eff}}$. □

The fact that there is a language that is not context-free, but contained in $HTN^{0\ pre}_{0\ \mathit{eff}}$ raises the question if all context-sensitive languages are members of $HTN^{0\ pre}_{0\ \mathit{eff}}$.

Proposition 1 *Let $\mathcal{P}$ be a planning problem where all actions have no preconditions and effects. Deciding $Sol_{\text{HTN}}(\mathcal{P}) = \emptyset$ is in **NP**.*

This proposition is a trivial corollary from a result by Alford et al. [1], who proved that delete-free HTN planning is **NP-complete**. Since precondition- and effect-free HTN planning is a further restriction, the proposition holds.

Theorem 9 $HTN^{0\ pre}_{0\ \mathit{eff}} \subsetneq CS$

Proof: From Cor. 2, we know that the $HTN^{0\ pre}_{0\ \mathit{eff}}$ languages are a subset of the context-sensitive languages. Suppose they are equal; then $L_{\text{HTN}}(\mathcal{P}) = \emptyset$ would not be decidable, as in the case of context-sensitive languages [4, p. 339]. However, Prop. 1 states that this question is in **NP** and thus decidable, which contradicts the assumption. $HTN^{0\ pre}_{0\ \mathit{eff}} \neq CS$ holds and consequently our claim. □

Finally we know that $HTN^{0\ pre}_{0\ \mathit{eff}}$ is a separate language class lying strictly between the context-free and the context-sensitive languages.

4 Conclusion

In this paper we have provided a way to view the set of solutions to HTN and TIHTN planning problems as a formal language. We have shown that the expressiveness of HTN planning is at most that of context-sensitive languages. In addition, several subclasses, like acyclic, totally ordered, or precondition- and effect-free problems have been classified w.r.t. their position in the Chomsky Hierarchy. Our results are summarized in Tab. 1. They help to gain more insights into the structure of hierarchical planning in general. One of our results is the discovery of a new language class, $HTN^{0\ pre}_{0\ \mathit{eff}}$, which lies strictly between the context-free and the context-sensitive languages. Such a class is of interest for future work, e.g., to find the position of the border between tractable and intractable word problems.

ACKNOWLEDGEMENTS

We want to acknowledge the insightful comments of the reviewers. This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

REFERENCES

[1] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau, 'On the feasibility of planning graph style heuristics for HTN planning', in *Proc. of the 24th Int. Conference on Automated Planning and Scheduling (ICAPS 2014)*, (2014).

[2] Pascal Bercher, Thomas Geier, Felix Richter, and Susanne Biundo, 'On delete relaxation in partial-order causal-link planning', in *Proc. of the 2013 IEEE 25th Int. Conference on Tools with Artificial Intelligence (ICTAI 2013)*, pp. 674–681, (2013).

[3] Susanne Biundo and Bernd Schattenberg, 'From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning)', in *Proc. of the 6th European Conference on Planning (ECP 2001)*, pp. 157–168, (2001).

[4] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker, *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, Academic Press, San Diego, CA, USA, 2nd edn., 1994.

[5] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo, 'Improving hierarchical planning performance by the use of landmarks', in *Proc. of the 26th National Conference on Artificial Intelligence (AAAI 2012)*, pp. 1763–1769, (2012).

[6] Kutluhan Erol, James A. Hendler, and Dana S. Nau, 'Complexity results for HTN planning', *Annals of Mathematics and Artificial Intelligence*, **18**(1), 69–93, (1996).

[7] Gerald Gazdar and Geoffrey K. Pullum, *Generalized phrase structure grammar: a theoretical synopsis*, volume 7 of *CSRP-Sussex*, Indiana University Linguistics Club, 1982.

[8] Thomas Geier and Pascal Bercher, 'On the decidability of HTN planning with task insertion', in *Proc. of the 22nd Int. Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 1955–1961, (2011).

[9] Bernhard Nebel and Christer Bäckström, 'On the computational complexity of temporal projection, planning, and plan validation', *Artificial Intelligence*, **66**(1), 125–160, (1994).

[10] Mark-Jan Nederhof, Giorgio Satta, and Stuart M. Shieber, 'Partially ordered multiset context-free grammars and ID/LP parsing', in *Proc. of the 8th Int. Workshop on Parsing Technologies*, pp. 171–182, (2003).

[11] Stuart M. Shieber, 'Direct parsing of ID/LP grammars', *Linguistics and Philosophy*, **7**(2), 135–154, (1984).