Can They Come Together? A Computational Complexity Analysis of Conjunctive Possible Effects of Compound HTN Planning Tasks

Conny Olz[1] **and Pascal Bercher**[2]

[1] Ulm University
[2] School of Computing, The Australian National University
conny.olz@uni-ulm.de, pascal.bercher@anu.edu.au

Abstract

Recently, inferred effects of compound (totally ordered) HTN planning tasks were introduced. Guaranteed effects are those which hold true after all executable refinements of such a task, whereas possible effects are only required to hold after some of them. It is known that we can decide in P whether *a single* fact is a precondition-relaxed possible effect. For this relaxation, it was not clear whether groups of effects could be determined in P as well. We show that the problem turns NP-complete for conjunctive possible effects of arbitrary size. A more positive result is that this problem is fixed-parameter tractable, i.e., for *any* fixed number of possible effects, we can verify (and compute) them in P. As a side product of our investigations, we obtain novel results for total-order HTN planning problems with goal description: When ignoring action preconditions, plan existence is NP-complete and remains NP-hard even when the problem is additionally acyclic, regular, and delete-relaxed.

Introduction

Hierarchical Task Network (HTN) planning is a field of planning where plans are built by transferring abstract plans into more concrete ones until a solution plan is found that consists of executable actions (Erol, Hendler, and Nau 1996; Ghallab, Nau, and Traverso 2004; Bercher, Alford, and Höller 2019). So-called compound tasks are key in this process as they define the rules of refinement. More precisely, a compound task gets realized by different sets of other primitive or compound tasks, where the former are simply actions known from classical planning. While there are formalisms of HTN planning where domain modelers state *explicit* preconditions and effects of compound tasks (Bercher et al. 2016), in the currently most commonly used formalism compound tasks are not considered to have any. This is also the case in HDDL (Höller et al. 2020), the official domain modeling language of the last and upcoming International Planning Competition (IPC). Olz, Biundo, and Bercher (2021), however, introduced *inferred* preconditions and effects of compound tasks based on their possible refinements in total-order (t.o.) HTN planning in order to make their potential state transitioning visible. Their benefits are manifold: They can draw a domain modeler's attention to flaws in the process of modeling, they can increase the comprehension of given domains (Olz et al. 2021), they can be exploited to detect dead-ends and for finding solution plans on an abstract level (Marthi, Russell, and Wolfe 2007; de Silva, Sardina, and Padgham 2009) – also known as *Upward* and *Downward* properties (Yang 1990). Lastly, they provide additional information that can be exploited by heuristics and planners (Nau et al. 2003; Waisbrot, Kuter, and Könik 2008; de Silva, Sardina, and Padgham 2009; Bit-Monnot, Smith, and Do 2016; Goldman and Kuter 2019; Magnaguagno, Meneguzzi, and de Silva 2021; Schreiber 2021).

Olz, Biundo, and Bercher (2021) defined two types of effects. First, *guaranteed* effects, which are facts that are added or deleted by every executable refinement of a compound task, respectively. Second, *possible* effects, which are facts that are added or deleted by at least one (but not necessarily all) executable refinement. For the latter, however, we do not have information about which of the effects may actually occur together, i.e., were added or deleted by actions within the same refinement. Thus, the information given by this kind of effect is rather vague and overpredicting, especially if a compound task admits many refinements containing different kinds of actions. We advance this situation by introducing a generalized version of possible size-k effects that are sets of facts (of size k). They state which facts can be produced together by a single refinement, i.e., they can be viewed as a kind of conjunctive possible effects. Consider the situation (e.g., in the context of dead-end detection or finding solutions on abstract levels) in which we want to know whether a certain set of facts holds after the execution of a compound task to make some later task executable. The size-k effects allow a much preciser answer on whether the compound task can satisfy that demand than the ones by Olz, Biundo, and Bercher do since the latter only list possible effects independently of each other.

Generally, the inferred effects are computationally hard to compute. In order to make them accessible in practice, Olz, Biundo, and Bercher considered them under precondition-relaxation so that one can decide in P whether one fact is such a possible effect. Now, we show that the problem turns NP-complete when considering possible size-k effects for arbitrary k, also for an even stronger relaxation: precondition- and delete-relaxation. On the positive side, we

k	Hierarchy		Theorem
	acyclic & regular	arbitrary	
1	P	P	Thm. 6*
fixed	P	P	Thm. 6
arbitrary	NP-c	NP-c	Thm. 1 & Cor. 2

Table 1: Complexity for deciding $F_k \in k\text{-}\mathit{eff}^{+/-}_{\mathcal{PD}}(c)$ and $F_k \in k\text{-}\mathit{eff}^{+/-}_{\mathcal{P}}(c)$. *by Olz, Biundo, and Bercher (2021)

also prove that the problem is fixed-parameter tractable (for both relaxations). For fixed k we can verify a size-k effect in polynomial time. We state the respective algorithm, which allows the efficient calculation and exploitation for small k. A summary of the results with pointers to the respective theorems can be found in Table 1.

Since deciding if a set of facts is a possible size-k effect is very similar to the plan existence problem with goal description we get two new results as a side product: We show that the total-order plan existence problem under precondition-relaxation is in NP and already NP-hard for acyclic and regular precondition- and delete-relaxed problems, i.e. both problems and the intermediate ones are NP-complete.

Related work Several authors inferred preconditions and effects of abstract tasks in different settings like temporal partially ordered, acyclic HTN models (Clement, Durfee, and Barrett 2007), Belief-Desire-Intention (BDI) agent-oriented programming (Thangarajah and Padgham 2011; de Silva, Sardina, and Padgham 2016; de Silva, Meneguzzi, and Logan 2020) and non-deterministic action models (Yao, de Silva, and Logan 2016). Magnaguagno, Meneguzzi, and de Silva (2021) compute preconditions similar to Olz, Biundo, and Bercher's executability-relaxed preconditions but in a lifted manner. Tsuneto, Hendler, and Nau (1998) extract some sort of method preconditions from conditions predefined by the domain modeler. However, those of the mentioned authors who consider possible effects at all, do so in the sense like Olz, Biundo, and Bercher do, i.e., they do not know which of the effects occur together.

Marthi, Russell, and Wolfe (2007) define *exact reachable sets (of states)* of abstract tasks and *compact approximations* of them that can be computed more efficiently to exploit them by a top-down search algorithm. However, they do not state how to come up with or verify these sets. Our conjunctive effects might serve as a basis for this purpose.

Theoretical Background

In this chapter we formally introduce t.o. HTN planning and (inferred) preconditions and effects of compound tasks.

Total-order HTN Planning Formalism

We use a formalism for totally ordered (t.o.) HTN planning, which is based on the ones by Geier and Bercher (2011) and Behnke, Höller, and Biundo (2018). A t.o. planning domain $\mathcal{D} = (F, A, C, M)$ consists of a finite set of facts F, *primitive tasks* (or actions) A, *compound tasks* C (also called *abstract tasks*), and *decomposition methods* $M \subseteq C \times T^*$, where $T = (A \cup C)$ is the set of all tasks and T^* refers to the set of all sequences of tasks including the empty one (like for regular expressions). Primitive tasks $a = (\mathit{prec}, \mathit{add}, \mathit{del}) \in A$ are like actions in classical planning described by their *preconditions* $\mathit{prec}(a) \subseteq F$, *add* and *delete effects* $\mathit{add}(a), \mathit{del}(a) \subseteq F$. An action $a \in A$ is *applicable* in a state $s \in 2^F$ if $\mathit{prec}(a) \subseteq s$. If applicable the application of a in s results in the successor state $\delta(s, a) = (s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$. For sequences of actions $\bar{a} = \langle a_0 \dots a_n \rangle$ with $a_i \in A$ it holds $\bar{a}$ is applicable in a state s_0 if a_0 is applicable in s_0 and for all $1 \leq i \leq n$, a_i is applicable in $s_i = \delta(s_{i-1}, a_{i-1})$. A compound task $c \in C$ represents sequences of primitive and/or compound tasks $\bar{t} \in T^*$, which is specified by their methods $m = (c, \bar{t}) \in M$. Such (possibly empty) sequences of tasks are called *task networks*. Then given a method $m = (c, \bar{t})$ we can *decompose* a compound task c within a task network $tn_1 = \langle \bar{t}_1\, c\, \bar{t}_2 \rangle \in T^*$ into a task network $tn_2 = \langle \bar{t}_1\, \bar{t}\, \bar{t}_2 \rangle$, denoted $tn_1 \rightarrow_{c,m} tn_2$. If we can transform tn into tn' by a (possible empty) sequence of decompositions, we denote it $tn \rightarrow tn'$ and call tn' a refinement of tn. A *t.o. HTN planning problem* $\Pi = (\mathcal{D}, s_I, tn_I, g)$ contains a domain $\mathcal{D}$, an *initial state* $s_I \in 2^F$, an *initial task network* $tn_I \in T^*$, and a goal description $g \subseteq F$. A solution to Π is a sequence of actions $tn = \langle a_0 \dots a_n \rangle \in A^*$ if and only if $tn_I \rightarrow tn$, tn is applicable in s_I, and results in a goal state $s \supseteq g$.

We defined the HTN planning problem with goal description since we will use it throughout this paper. There are, however, also formalisms and benchmark problems without a goal description. Nevertheless, one can simulate a goal description by introducing a primitive task $a_g = (g, \emptyset, \emptyset)$ and adding it as last task into tn_I.

Effects of Compound Tasks

We furthermore recapitulate some of the definitions by Olz, Biundo, and Bercher (2021) concerning effects of compound tasks as they form a basis on which we build upon.

The set of *executability-enabling states* of a compound task $c \in C$ is $E(c) = \{s \in 2^F \mid \exists\, \bar{a} \in A^* : c \rightarrow \bar{a}$ and $\bar{a}$ is applicable in $s\}$ and the set of all states into which the execution of c in a state $s \in 2^F$ can result is given by $R_s(c) = \{s' \in 2^F \mid \exists\, \bar{a} \in A^* : c \rightarrow \bar{a}, \bar{a}$ is applicable in s and results in $s'\}$.

Facts that were added or deleted, resp., by every executable refinement of a compound task c – independent of the initial state – are called *state-independent positive and negative effects*:

- $\mathit{eff}^+_*(c) := (\bigcap_{s \in E(c)} \bigcap_{s' \in R_s(c)} s') \setminus \bigcap_{s \in E(c)} s$
- $\mathit{eff}^-_*(c) := \bigcap_{s \in E(c)} (F \setminus \bigcup_{s' \in R_s(c)} s')$

if $E(c) \neq \emptyset$, otherwise $\mathit{eff}^{+/-}_*(c) := \mathit{undef}$.

On the other hand, facts that are only added or deleted, resp., by *some* but not necessarily all executable refinements are called *possible state-independent effects*:

- $\mathit{poss\text{-}eff}^+_*(c) := \bigcup_{s \in E(c)} (\bigcup_{s' \in R_s(c)} s' \setminus s)$
- $\mathit{poss\text{-}eff}^-_*(c) := \bigcup_{s \in E(c)} ((\bigcup_{s' \in R_s(c)} (F \setminus s')) \cap s)$

if $E(c) \neq \emptyset$ and $\mathit{poss\text{-}eff}^{+/-}_*(c) := \mathit{undef}$ otherwise.

Generalized Effects of Compound Tasks

Generally, the set of guaranteed effects defined by Olz, Biundo, and Bercher (2021) is an underapproximation whereas the possible effects are an overapproximation of the facts that one concrete refinement of a compound task c produces. More precisely, a refinement of c will add all guaranteed and some but likely not all of the possible effects. In order to get more fine-grained information on the interplay we would like to know which of the possible effects occur together, i.e., were produced by the same refinement in contrast to sets of facts that can never be added together. This leads us to the following definition:

Definition 1. *Let $\mathcal{D} = (F, A, C, M)$ be a planning domain, $c \in C$ a compound task, and $F_k \subseteq F$ a set of facts of size $k \in \mathbb{N}$. Then F_k is a*

- positive-size-k *effect of c iff* $\exists\, s \in E(c) \wedge \exists\, s' \in R_s(c) : F_k \subseteq s' \wedge s \cap F_k = \emptyset$.
- negative-size-k *effect of c iff* $\exists\, s \in E(c) \wedge \exists\, s' \in R_s(c) : F_k \cap s' = \emptyset \wedge F_k \subseteq s$.
- positive-size-k *postcondition of c iff* $\exists\, s \in E(c) \wedge \exists\, s' \in R_s(c) : F_k \subseteq s'$.
- negative-size-k *postcondition of c iff* $\exists\, s \in E(c) \wedge \exists\, s' \in R_s(c) : F_k \cap s' = \emptyset$.

One can differentiate between effects and postconditions. The latter additionally contain facts that (don't) hold already prior to execution and (don't) hold afterwards as well. This paper focuses on effects. For that, we denote the set of all positive-size-k and negative-size-k effects $k\text{-}\mathit{eff}^{+}(c)$ and $k\text{-}\mathit{eff}^{-}(c)$, respectively. Sometimes we write $k\text{-}\mathit{eff}^{+/-}(c)$ as shorthand if we refer to both but still consider them as separate sets. Note that $1\text{-}\mathit{eff}^{+/-} = \cup_{f \in \mathit{poss}\text{-}\mathit{eff}_*^{+/-}(c)} \{f\}$.

Possible vs. Guaranteed Semantically a size-k effect is a possible but not necessarily a guaranteed effect because we only know that it can be produced by *some* (but not necessarily all) refinements of a compound task. We did not reflect this in its symbol ($k\text{-}\mathit{eff}$ vs. $\mathit{poss}\text{-}k\text{-}\mathit{eff}$) because we do not need to define guaranteed size-k effects explicitly as they are already captured by the state-independent effects: Since every guaranteed effect $f \in \mathit{eff}_*^{+}(c)$ is added by every executable refinement of c we know that all facts in $\mathit{eff}_*^{+}(c)$ were added together by every executable refinement and there is no other set of facts $F' \subseteq F$ with $F' \not\subseteq \mathit{eff}_*^{+}(c)$ that is added simultaneously by all refinements of c. Similar arguments hold for negative effects. Thus, when talking about size-k effects we always refer to *possible* effects.

Preconditions As a counterpart to the effects Olz, Biundo, and Bercher also introduced inferred *preconditions*, i.e. facts that are required by every refinement of a compound task to be executable. Semantically they resemble *guaranteed* effects since always all of them must hold. So far there is no version of "possible preconditions". Informally, they could be defined as a set of facts that are demanded by one refinement so that it is executable. So all of them of a compound task together form some sort of disjunctive landmark. However, since we focus on effects here, we regard the formal definition and analysis to be future work.

Complexity Olz, Biundo, and Bercher showed that determining whether a fact is a *possible* state-independent precondition or effect is as hard as solving a plan existence problem (with matching upper and lower bounds), i.e., ranging from **PSPACE**- to **EXPTIME**-complete, depending on the hierarchy structure of the decomposition methods. Our case now is very similar because deciding whether a set of facts F_k is a size-k effect of some compound task c within domain $\mathcal{D} = (F, A, C, M)$ contains the question of whether there exists an executable refinement (hardness). Moreover, it is also not harder than solving a planning problem since F_k is a positive size-k effect if and only if the problem $(\mathcal{D}, F \setminus F_k, \langle c \rangle, F_k)$ has a solution, which shows membership. The same can be shown for negative size-k effects by introducing new facts for every such effect.

Relaxations For many applications like planning algorithms the above-mentioned computation of effects of compound tasks is too costly. Therefore Olz, Biundo, and Bercher introduced a version based on precondition-relaxation that can be computed in polynomial time. While they approximate the actual preconditions and effects they still maintain properties so that they can be exploited in practice. In this paper, we investigate size-k effects under the same relaxation and also under an even stronger relaxation: precondition- and delete-relaxation.

Let $\mathcal{D} = (F, A, C, M)$ be a domain. The *precondition-relaxation* ($\mathcal{P}$-relaxation) of $\mathcal{D}$ is the domain $\mathcal{D}' = (F, A', C, M)$ with $A' = \{(\emptyset, \mathit{add}, \mathit{del}) \mid (\mathit{prec}, \mathit{add}, \mathit{del}) \in A\}$. The *precondition- and delete-relaxation* ($\mathcal{PD}$-relaxation) of $\mathcal{D}$ is the domain $\mathcal{D}'' = (F, A'', C, M)$ with $A'' = \{(\emptyset, \mathit{add}, \emptyset) \mid (\mathit{prec}, \mathit{add}, \mathit{del}) \in A\}$.

Now we can define relaxed size-k effects:

Definition 2. *Let $\mathcal{D} = (F, A, C, M)$ be a domain and $c \in C$ a compound task. Then, $k\text{-}\mathit{eff}_{\mathcal{P}}^{+/-}(c)$ and $k\text{-}\mathit{eff}_{\mathcal{PD}}^{+/-}(c)$ are* $\mathcal{P}$-relaxed *and* $\mathcal{PD}$-relaxed size-k effects *of c and are defined just like the original ones but based on the $\mathcal{P}$-relaxed and $\mathcal{PD}$-relaxed variant of $\mathcal{D}$, respectively.*

The set $k\text{-}\mathit{eff}_{\mathcal{PD}}^{-}(c)$ is actually empty for all compound tasks because actions do not have delete effects anymore under $\mathcal{PD}$-relaxation. Semantically, an effect $F_k \in k\text{-}\mathit{eff}_{\mathcal{PD}}^{+}(c)$ means that there is a refinement of c such that for every fact $f \in F_k$ there is an action adding it but we ignore whether this refinement is executable or whether facts get deleted again. The same consideration could be done for delete effects, i.e., guaranteeing that there is a refinement with actions deleting all facts while ignoring preconditions and add effects, which would lead to a precondition- and add-relaxation. Since we do not see much potential for exploiting them, we will not define or consider them explicitly but rather implicitly with $k\text{-}\mathit{eff}_{\mathcal{PD}}^{-}(c)$. The arguments to come for $k\text{-}\mathit{eff}_{\mathcal{PD}}^{+}$ apply quite similarly to such $k\text{-}\mathit{eff}_{\mathcal{PD}}^{-}$.

For the purpose of exploitation, it is important that the relaxed versions do not produce "wrong" candidates. Fortunately, Olz, Biundo, and Bercher could show that subset properties are preserved, which is also true for the general effects:

Proposition 1. *It holds* $k\text{-}eff^{+/-}(c) \subseteq k\text{-}eff^{+/-}_{\mathcal{PD}}(c)$, $k\text{-}eff^{+/-}(c) \subseteq k\text{-}eff^{+/-}_{\mathcal{P}}(c)$ *and* $k\text{-}eff^{+}_{\mathcal{P}}(c) \subseteq k\text{-}eff^{+}_{\mathcal{PD}}(c)$.

Proof. Let $F_k \in k\text{-}eff^{+}(c)$. Then there exists a state $s \in E(c)$ and $s' \in R_s(c)$ such that $F_k \subseteq s'$. The refinement of c that transforms s to s' is still executable under $\mathcal{P}$ and $\mathcal{PD}$-relaxation since all preconditions are gone. The resulting state is the same under $\mathcal{P}$ relaxation since effects did not change. For $\mathcal{PD}$ relaxation we may end up with a state $s'' \supseteq s'$ since facts were not deleted anymore but it still holds $F_k \subseteq s''$. Thus, $F_k \in k\text{-}eff^{+}_{\mathcal{P}}(c)$ and $F_k \in k\text{-}eff^{+}_{\mathcal{PD}}(c)$. Similar arguments apply to negative effects. □

Hierarchy Classes The main purpose of this paper is to investigate how hard it is to compute size-k effects under different relaxations and to identify tractable cases. Generally in HTN planning, the structure of the hierarchy influences the complexity of the plan existence problem. Four kinds of such classes are considered in the literature so far, which we will also take into account in our analysis. We now introduce them briefly together with the complexity of the respective total-order plan existence problem given in brackets. Planning problems are called *acyclic* if the number of possible decompositions is finite, i.e., there are no recursions within the methods (**PSPACE**-complete (Alford, Bercher, and Aha 2015)). A problem is *regular* if in the initial task network and in all methods' task networks only one task is compound and if so it must be the last one (**PSPACE**-complete (Erol, Hendler, and Nau 1996)). A generalization of those are *tail-recursive* problems (**PSPACE**-complete (Alford, Bercher, and Aha 2015)). Since they are not explicitly considered in this paper, we refrain from the exact (and rather complicated) definition here. Lastly and for completeness, one considers arbitrary problems without any restrictions (**EXPTIME**-complete (Alford, Bercher, and Aha 2015)).

Effects of Methods Alongside effects of compound tasks one can also define effects of decomposition methods as already mentioned by Olz, Biundo, and Bercher. This is not only possible for the former effects defined by them but also for our generalized version, the size-k effects. For that, introduce a new compound task c', "copy" the task network of a method $m = (c, tn)$, and build a new method $m' = (c', tn)$. Now the effects of m are equal to the ones of c', which has only one method, namely m'.

Complexity of $\mathcal{PD}$-relaxed Effects

We start with considering precondition- and delete-relaxed HTN planning domains and problems, i.e., primitive actions have only add effects. Such problems are simpler than just precondition-relaxations.

Olz, Biundo, and Bercher (2021) showed that computing precondition-relaxed effects of compound tasks ($1\text{-}eff^{+/-}_{\mathcal{P}}(c)$) is possible in polynomial time. By additionally relaxing the delete effects the problem becomes easier and one can still use their proposed polynomial algorithm for computation. Thus, we can conclude:

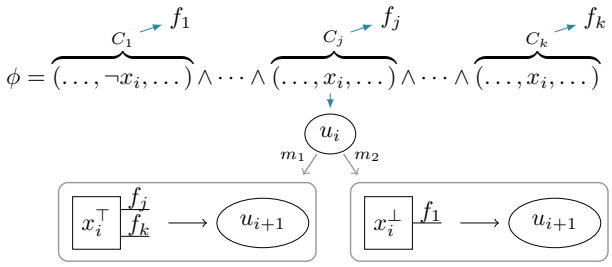

Figure 1: Illustration of the proof idea behind Theorem 1.

Corollary 1. *Deciding whether* $f \in 1\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ *for some compound task* c *and fact* f *can be done in polynomial time for arbitrary recursion.*

We go on to the more general effects. It turns out that deciding whether a set of facts of arbitrary size is a $\mathcal{PD}$-relaxed effect of a compound task is already NP-hard even for the simplest class, acyclic and regular hierarchies:

Theorem 1. *Let* $\mathcal{D} = (F, A, C, M)$ *be an acyclic and regular domain,* $c \in C$ *a compound task, and* $F_k \subseteq F$ *a set of facts. Deciding whether* $F_k \in k\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ *is NP-hard.*

Proof. To prove hardness we reduce from 3SAT. The idea is illustrated in Fig. 1. So let $C_1 \dots C_k$ be the clauses of a 3SAT formula ϕ in conjunctive normal form consisting of the propositional variables $x_1 \dots x_n$. We construct the following planning domain: For every clause C_j ($1 \leq j \leq k$) there is a fact f_j. For every variable x_i ($1 \leq i \leq n$) there is a compound task u_i and two primitive tasks $x_i^\top$, $x_i^\perp$ with add effects $add(x_i^\top) = \{f_l \mid x_i \in C_l\}$ and $add(x_i^\perp) = \{f_l \mid \neg x_i \in C_l\}$, respectively. The preconditions and delete effects are empty. Each compound task u_i ($1 \leq i \leq n-1$) has two decomposition methods $m_i^1 = (u_i, \langle x_i^\top u_{i+1} \rangle)$ and $m_i^2 = (u_i, \langle x_i^\perp u_{i+1} \rangle)$; u_n has two methods $m_n^1 = (u_n, \langle x_i^\top \rangle)$ and $m_n^2 = (u_i, \langle x_i^\perp \rangle)$. At last there is one compound task c^* with one method $m^* = (c^*, \langle u_1 \rangle)$. The constructed domain is clearly acyclic since every compound task will be added exactly once. Moreover, every method contains at most one compound task and the task is ordered last. Thus, the domain is also regular.

We claim that ϕ is satisfiable if and only if $\{f_1, \dots, f_k\}$ is a possible size-k effect of c^*. Consider a model of ϕ, i.e., a truth assignment of the variables x_i so that ϕ evaluates to true. Then, by choosing method m_i^1 if $x_i = \mathit{true}$ and m_i^2 if $x_i = \mathit{false}$ for all $i = 1 \dots n$ we get a refinement of c^* such that all $f_1 \dots f_k$ hold after execution since for every clause C_l there is at least one variable $x_{l'}$ that evaluates C_l to true and therefore the corresponding action $x_{l'}^\top$ or $x_{l'}^\perp$ (depending on whether $x_{l'}$ is true or false) adds f_l.

The other way around follows the same argumentation. Every fact f_l must be added by some action $x_{l'}^\top$ or $x_{l'}^\perp$, which corresponds to setting $x_{l'}$ to true or false in order to satisfy clause C_l. Because all facts $f_1 \dots f_k$ were added, all clauses are made true and therefore the whole formula ϕ as well. □

As previously mentioned, the problem of deciding whether $F_k \in k\text{-}eff^{+/-}(c)$ is quite related to the plan ex-

istence problem. Given a problem $(\mathcal{D}, s_I, tn_I, g)$ we can solve it by asking whether $g \in k\text{-}eff^+(c^*)$, where c^* is a new compound task with one method $(c^*, \langle a_I, tn_I \rangle)$ and $a_I = (\emptyset, s_I, \emptyset)$ is a new primitive task simulating the initial state. Thus, we determine the complexity of the plan existence problem under $\mathcal{PD}$-relaxation as a side product.

Corollary 2. *The plan existence problem of acyclic and regular $\mathcal{PD}$-relaxed HTN planning problems is NP-hard.*

Alford et al. (2014) showed NP-completeness of the HTN plan existence problem under delete-relaxation. It is interesting to see that the problem remains NP-hard even when additionally there are no preconditions or cycles in the domain, which seems to make the problem much simpler. Closely related, Höller, Bercher, and Behnke (2020) proved that the delete- and ordering-free HTN plan existence problem (in the context of partially ordered domains) is also NP-complete.

In order to show NP-completeness of $\mathcal{PD}$-relaxed problems it remains to prove that it is also in NP. It does not directly follow from the membership result by Alford et al. (2014) because, in contrast to our formalism, theirs does not allow for empty task networks (e.g., in the methods). Moreover, it is still open and should be investigated whether there is a *fixed* k for which deciding whether $F_k \in k\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ is NP-hard. We saw that it is polynomial for $k = 1$ and there might be a number k' so that deciding $F_{k'} \in k'\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ is polynomial but $F_{k'+1} \in (k'+1)\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ is NP-hard like it is the case for the prominent problems 2SAT and 3SAT. We will look at both problems in the next section, which focuses on precondition-relaxed domains. Since those results are to come, transfer to $\mathcal{PD}$-relaxation we do not redundantly state them here as well.

Complexity of $\mathcal{P}$-relaxed Effects

In this section, we consider only precondition-relaxation (no delete-relaxation). Since this is a less restricted relaxation than $\mathcal{PD}$-relaxation (which we considered in the last section), we can conclude that the plan existence problem and also deciding $k\text{-}eff^{+/-}_{\mathcal{P}}(c)$ must also be at least NP-hard:

Corollary 3. *Deciding whether $F_k \in k\text{-}eff^{+/-}_{\mathcal{P}}(c)$ and deciding the plan existence problem of acyclic and regular $\mathcal{P}$-relaxed domains are NP-hard.*

Deciding $F_k \in k\text{-}eff^{+/-}_{\mathcal{P}}(c)$ for arbitrary recursion seems much more complicated than $F_k \in k\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ for acyclic domains but it turns out that it is still in NP. The first part of this section is concerned with proving the NP membership. Afterwards, we show its fixed-parameter tractability.

NP Membership

We show that the plan existence problem of precondition-relaxed HTN planning problems is in NP, which is non-trivial since even acyclic problems can lead to exponential plans (e.g., $c_1 \to \langle c_2, c_2 \rangle$, $c_2 \to \langle c_3, c_3 \rangle$, etc.). Then the NP membership of deciding $F_k \in k\text{-}eff^{+/-}_{\mathcal{P}}(c)$ follows easily.

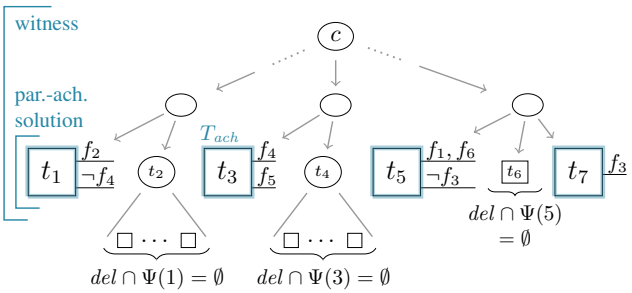

Figure 2: A witness that the given problem $(\mathcal{D}, \emptyset, \langle c \rangle, \{f_1 \dots f_6\})$ is solvable ($\mathcal{D}$ is only depicted partially) to illustrate the proof idea of Theorem 2. Ellipses embody compound tasks, rectangles are primitive ones.

We need to show that there is a witness of polynomial size that can be verified in polynomial time if the plan existence problem is solvable. For that, we generate a witness that is composed of two parts: a sequence of (potentially primitive and compound) tasks that can definitely be refined into a solution and a tree representing the decompositions leading to that partial plan. The idea is sketched in Fig. 2 with an exemplary partial plan. Our proof follows the idea by Alford et al. (2014) to show NP membership of solving delete-free HTN problems. We state the main adaptations later on.

Abstract Solutions We start with introducing the first part, a sequence of (potentially primitive and compound) tasks that represents an abstract solution plan. Solutions of even acyclic ($\mathcal{P}$-relaxed) HTN planning problems can be exponentially long. To represent such a solution in polynomial size our witness contains a task network that still contains compound tasks and it can be verified in polynomial time that this task network can be refined into a solution. Generally, primitive solutions $\langle a_1 \dots a_n \rangle \in A^*$ of $\mathcal{P}$-relaxed problems are of the following structure. For every fact $f \in g$ there is a primitive task a_i such that $f \in add(a_i)$ and none of the tasks $a_{i+1} \dots a_n$ deletes f. We call these facts and the corresponding tasks *achiever facts* and *tasks*, respectively. The tasks are later referred to by their indices stored in T_{ach}. This can be generalized to task networks, which contain also compound tasks. We call them partial-achiever solutions (of a $\mathcal{P}$-relaxed planning problem). Such a partial-achiever solution is a task network $tn \in T^*$ with the following properties. In parentheses, we state the constraint numbers of the subsequent formal definition. For every goal fact $f \in g$ there is exactly one primitive achiever task (1.+2.) that adds f (3.), none of the subsequently ordered primitive tasks deletes f (4.) and every compound task $t_c \in tn$ has a refinement such that none of its tasks deletes any of the previously achieved facts (5.). To define them formally we introduce two functions: ψ maps achiever tasks to the respective goal facts that they satisfy; Ψ is based on ψ and maps achiever tasks to the set of goal facts that they and their predecessor achiever tasks add. So Ψ is for convenience to state which facts are achieved up to the respective index. Note that according to our definition, the task network of a partial-achiever solution does not necessarily need to be a refinement of the initial

task network. We will only cover this property in the next subsection. To ease notation we assume w.l.o.g. that s_I is empty and asserted by an action right at the beginning of the task network.

Definition 3. *Let*

- $\Pi = (\mathcal{D}, s_I, tn_I, g)$ *be a* $\mathcal{P}$*-relaxed planning problem,*
- $tn = \langle t_1 \dots t_n \rangle \in T^*$ *a sequence of tasks,*
- $T_{ach} \subseteq \{1 \dots n\}$ *a subset of the indices of tasks in* tn*, called* achiever indices*, defining the* achiever tasks*, and*
- $\psi, \Psi : T_{ach} \to 2^g$ *with* $\Psi(i) = \bigcup_{j \in T_{ach} : j \leq i} \psi(j)$ *mappings from achiever indices to sets of goal facts.*

Then (tn, T_{ach}, ψ) *is called a* partial-achiever *solution iff*

1. $\forall\, f \in g\, \exists\, i \in T_{ach} : f \in \psi(i)$
2. $\forall\, i \neq j \in T_{ach} : \psi(i) \cap \psi(j) = \emptyset$
3. $\forall\, i \in T_{ach} : t_i \in A$ *and* $\psi(i) \subseteq add(t_i)$
4. $\forall\, i \in T_{ach}\, \forall\, j > i$ *with* $t_j \in A : del(t_j) \cap \Psi(i) = \emptyset$
5. $\forall i \in T_{ach}\, \forall\, j > i$ *with* $t_j \in C : \exists$ *a primitive refinement* $\bar{a} = \langle a_1 \dots a_m \rangle$ *of* t_j *s.t.* $\forall a \in \bar{a} : del(a) \cap \Psi(i) = \emptyset$

For the example in Fig. 2 it holds: $T_{ach} = \{1, 3, 5, 7\}$ and

- $\psi(1) = \{f_2\}$, $\Psi(1) = \{f_2\}$
- $\psi(3) = \{f_4, f_5\}$, $\Psi(3) = \{f_2, f_4, f_5\}$
- $\psi(5) = \{f_1, f_6\}$, $\Psi(5) = \{f_1, f_2, f_4, f_5, f_6\}$
- $\psi(7) = \{f_3\}$, $\Psi(7) = \{f_1, \dots, f_6\}$

A partial-achiever solution can be viewed as an abstract solution to a $\mathcal{P}$-relaxed planning problem since it can be refined into a primitive solution by construction (if it is a refinement of tn_I, which is what we will consider in the next subsection). Moreover, for every primitive solution tn_s of a $\mathcal{P}$-relaxed planning problem there is a T_{ach} and ψ such that (tn_s, T_{ach}, ψ) is a *partial-achiever* solution.

Corollary 4. *Let* $\Pi = (\mathcal{D}, s_I, tn_I, g)$ *be a* $\mathcal{P}$*-relaxed problem.* Π *is solvable if and only if there exists a refinement* tn *of* tn_I *and* T_{ach}, ψ *so that* (tn, T_{ach}, ψ) *is a* partial-achiever *solution.*

In order to use it as a witness we need to show that we can check the properties of a partial-achiever solution in polynomial time.

Proposition 2. *Given a* $\mathcal{P}$*-relaxed planning problem* Π *and a tuple* (tn, T_{ach}, ψ) *we can verify in polynomial time whether* (tn, T_{ach}, ψ) *is a partial-achiever solution.*

Proof. We only show how to verify the last condition (5.) since the other ones can clearly be checked in polynomial time. For that, let $t_i \in tn$ with $i \in T_{ach}$ be some achiever task and $\Psi(i) = \bigcup_{i' \in T_{ach} : i' \leq i} \psi(i')$. Let $t_j \in tn$ be some task with $j > i$, $t_j \in C$, and $\mathcal{D}' = (F, A', C, M')$ be the domain restricted to primitive tasks deleting some of the already achieved facts, i.e. $A' = \{a \in A \mid del(a) \cap \Psi(i) \neq \emptyset\}$ and M' is equal to M but restricted to the tasks in A' instead of A. If t_j in D' can be refined into an empty refinement, then t_j fulfills the desired property. To check whether this is the case one can start with methods having empty task networks. We mark the respective compound task with that property. Now, we might identify and mark further methods admitting empty refinements, i.e. those that contain only compound tasks and all of them admit an empty refinement. Iteratively, this marking can be propagated through the hierarchy until a fixpoint is reached, which takes at most $\mathcal{O}(|M|^2)$ iterations. We can check the other subsequent tasks of t_i with the same restricted and marked domain D'.

All in all, we need to do this for all $i \in T_{ach}$ but as we can check all of them independently we are still polynomial. □

Decomposition Trees We defined partial-achiever solutions and showed that they can be verified in polynomial time. Now we need to show that there is such a partial-achiever solution of polynomial size and that it is a refinement of the initial task network. Therefore, we use so-called *decomposition trees* introduced by Geier and Bercher (2011) to represent the chosen decompositions.

Definition 4 (Decomposition Tree)**.** *Given a planning problem* $\Pi = (\mathcal{D}, s_I, \langle c_I \rangle, g)$[1] *and a sequence of tasks* tn*, then a* decomposition tree $\mathcal{T} = (N, E, \beta)$ *for* Π *and* tn *is a tree with the following properties:*

- N *is a set of nodes,*
- $E : N \to N^*$ *is an edge function that maps every node to an ordered list of children* $\langle e_1, \dots, e_k \rangle$*,*
- $\beta : N \to T$ *is a function, which assigns a task to every node. The root node* n_r *is* $\beta(n_r) = c_I$*. For every inner node* n *it holds* $\beta(n) \in C$ *and for leaf nodes* $\beta(n) \in T$*,*
- $\forall$ *inner nodes* $n\, \exists\, (\beta(n), \langle t_1 \dots t_m \rangle) \in M$ *s.t.* $|E(n)| = m$ *and* $t_i = \beta(e_i)\, \forall e_i \in E(n) = \langle e_1, \dots, e_m \rangle$*,*
- *Let* $\langle n_1 \dots n_l \rangle$ *be the sequence of leaf nodes*[2]*, then it holds* $tn = \langle \beta(n_1) \dots \beta(n_l) \rangle$*.*

So, a decomposition tree embodies which decompositions need to be chosen in order to refine tn_I to tn. Thereby, conditions (2.+4.) ensure that each inner node is decomposed according to an applicable decomposition method. We call tn the *yield of a decomposition tree* $\text{yield}(\mathcal{T})$. Note that $\text{yield}(\mathcal{T})$ is *not* necessarily executable and it can still contain compound tasks. The last condition is only relevant if an explicit yield is given. Note that this definition can not express that a compound task is decomposed by a method with an empty task network. The respective compound task will just be a leaf node and thus in the yield. However, this does not cause an issue in what follows.

Combined Witness Now we combine both concepts. To form a witness we take a decomposition tree whose yield is a partial-achiever solution. Geier and Bercher (2011) already proved that given a planning problem $\Pi = (\mathcal{D}, s_I, tn_I, g)$ and a task network $tn \in T^*$, there exists a decomposition tree $\mathcal{T}$ with $\text{yield}(\mathcal{T}) = tn$, if and only if $tn_I \to tn$.[3]

Definition 5. *Let* $\Pi = (\mathcal{D}, s_I, tn_I, g)$ *be a* $\mathcal{P}$*-relaxed planning problem,* $\mathcal{T}$ *a decomposition tree of* Π *whose yield* tn *is part of a partial-achiever solution* $pas = (tn, T_{ach}, \psi)$*. Then the pair* $(\mathcal{T}, pas)$ *is a witness to the solvability of* Π*.*

[1] We assume that the initial task network contains only one task, otherwise let c_I be a new compound task with method (c_I, tn_I).

[2] We assume that the ordering is inherited from E.

[3] They relied on a formalism that excluded methods with empty task networks. The result can still be used here by assuming that the respective compound tasks are still in tn.

It remains to show that there is always such a witness verifiable in polynomial time. A decomposition tree can be verified in polynomial time in the size of the tree.[4]

So we have to show that there is always a decomposition tree of polynomial size. Its yield and thus the partial-achiever solution are then polynomial as well. We start with a witness that might have exponential size, which must exist if the planning problem is solvable according to Geier and Bercher (2011) and show by transforming it that there must also exist a smaller one of polynomial size.

A partial-achiever solution has at most $|T_{ach}| \leq |g|$ achiever tasks. By keeping the other tasks as abstract as possible we can represent exponential parts of a primitive solution in a polynomial number of compound tasks. To form such a partial-achiever solution we make use of so-called *saplings*, which were introduced by Alford et al. (2014):

Definition 6 (Saplings)**.** *Let $\mathcal{T} = (N, E, \beta)$ be a decomposition tree and $S \subseteq N$ a set of nodes. Moreover, let N' be the set of nodes along any path from a node in S to the root of $\mathcal{T}$ (inclusively) and the siblings of each and every node along the path, i.e., N' is the smallest subtree of N such that:*

- $S \subseteq N'$
- $\forall\, n, n' \in N : n' \in N' \wedge n' \in E(n) \Rightarrow n \in N'$
- $\forall\, n, n_1, n_2 \in N : n_1 \in N' \wedge n_1, n_2 \in E(n) \Rightarrow n_2 \in N'$

Then the S-sapling of $\mathcal{T}$ is $(N', E|_{N'}, \beta|_{N'})$[5]*.*

Alford et al. (2014) proved a useful property:

Proposition 3. *Let $\mathcal{T} = (N, E, \beta)$ be a decomposition tree (DT) and $S \subseteq N$. Then the S-sapling of $\mathcal{T}$ is also a valid DT.*

Given a witness $(\mathcal{T}, (tn, T_{ach}, \psi))$ we can create a smaller witness by building an S-sapling, where S is the set of nodes corresponding to the achiever tasks defined by T_{ach}. In the following definition, we abuse notation by using $\beta^{-1}(t_i)$, which actually does not exist because β is not injective, i.e., several nodes might map to the same task. However, it could instead be defined using a depth-first search over the tree stopping at the ith leaf node.

Proposition 4. *Let $\Pi = (\mathcal{D}, s_I, tn_I, g)$ be a solvable $\mathcal{P}$-relaxed planning problem, $(\mathcal{T}, (tn = \langle t_1 \dots t_{|tn|} \rangle, T_{ach}, \psi))$ a witness of its solvability. Then the yield tn' of the S-sapling $\mathcal{T}'$ of $\mathcal{T}$, where $S = \{n \in N \mid \exists i \in T_{ach} : \beta^{-1}(t_i) = n\}$, also forms a partial-achiever solution $(tn', T_{ach}{}', \psi')$. Together, $(\mathcal{T}', (tn', T_{ach}{}', \psi'))$ is a witness of the same size or smaller.*

Proof. We first check the conditions of Def. 3: Since S is the set of nodes corresponding to tasks defined by T_{ach} we know that the achiever tasks are still in tn', which are now defined by $T_{ach}{}'$ (only indices might have changed). The mappings ψ' (and thus also Ψ') adapt to the potentially new indices but should map the tasks to the same facts as ψ does, which fulfills conditions (1.+2.+3.). Condition (4.) is also satisfied since only a subset of the primitive tasks of tn can be in tn' and the ordering has not changed. To check condition (5.), let $t_i \in tn'$ be some compound task and $n \in \mathcal{T}'$ be the node such that $n = \beta^{-1}(t_i)$. All of n's leaf nodes in $\mathcal{T}$ fulfill condition (5.) and are ordered between the same achiever tasks as n (which is relevant with regard to the considered Ψ). So we know that t_c must also satisfy (5.) because it can be decomposed to this sequence of leaf nodes according to $\mathcal{T}$. So, (4.+5.) hold for all $t \in tn'$. Thus, $(tn', T_{ach}{}', \psi')$ is also a partial-achiever solution. According to Prop. 3 $\mathcal{T}'$ is a valid decomposition tree, which contains less or equally many nodes. So, $(\mathcal{T}', (tn', T_{ach}{}', \psi'))$ is also a witness of the same size or smaller. □

The size of the resulting tree (and also its yield, i.e., the partial-achiever solution) is then bounded by $|T_{ach}| \cdot |path| \cdot |m_{max}|$, where $|path|$ is the length of the longest path from the root to an achiever task in $\mathcal{T}$ and $|m_{max}|$ is the number of tasks of the method with the most tasks. So the witness has polynomial size as long as $|path|$ is polynomial, which we have not shown so far.

To prove NP membership of delete-relaxed HTN planning Alford et al. (2014) used a similar witness, it was also composed of a decomposition tree whose yield had a certain structure but instead of the partial-achiever solution they considered task networks with properties suited for their case. The arguments that there is a decomposition tree of polynomial size can be adapted straightforwardly to our case with one exception: Alford et al. (2014) assumed – in contrast to us – that there are no empty methods. So we have to slightly adapt it but the main idea is still the same. This is why we would like to claim that there is always a valid tree with bounded height, i.e., polynomial $|path|$, here and prove it formally in the appendix.

Since we showed that there is always a poly-sized witness for the solvability of $\mathcal{P}$-relaxed problems verifiable in polynomial time we proved NP-membership and conclude:

Theorem 2. *The plan existence problem of $\mathcal{P}$-relaxed HTN planning problems is NP-complete.*

Because $\mathcal{PD}$-relaxed problems are a simpler special case of $\mathcal{P}$-relaxed ones and the decision problem of size-k effects can be reduced to a plan existence problem we can state the following corollary:

Corollary 5. *Deciding whether $F_k \in k\text{-}\mathit{eff}^{+/-}_{\mathcal{PD}}(c)$ and the plan existence problem of $\mathcal{PD}$-relaxed HTN planning problems are NP-complete for acyclic, regular, tail-recursive, and arbitrary hierarchies.*

$\mathcal{P}$-relaxed HTN planning problems without goal description can be solved in P since one only needs to find some primitive refinement. It is interesting to see that it turns NP-complete when there is a single action at the end having preconditions that need to be satisfied.

Fixed-Parameter Tractability

We showed that deciding $F_k \in k\text{-}\mathit{eff}^{+/-}_{\mathcal{P}}(c)$ is NP-complete for arbitrary k. Remember that it is decidable in polynomial time for $k = 1$. Now, it is interesting to know whether there is a concrete number for k so that the problem turns from polynomial to NP-completeness. However, it turned

[4] This is only true for t.o. domains since for partially ordered ones, graph isomorphisms need to be checked.

[5] $E|_{N'}$ and $\beta|_{N'}$ denotes E and β restricted to N', resp.

Algorithm 1: Fixed-Parameter Algorithm

Input: A planning domain $\mathcal{D} = (F, A, C, M)$, a compound task $c^* \in C$, and a set of facts $F_k = \{f_1 \dots f_k\} \subseteq F$
Output: True, if $F_k \in k\text{-}\mathit{eff}^+_{\mathcal{P}}(c^*)$, false otherwise

1: $\mathit{updatedTasks} = \emptyset$
2: Initialize $\Phi(t_p)$ acc. to $(\mathit{add}(t_p), \mathit{del}(t_p))\, \forall\, t_p \in A$
3: $\Phi(t_c) = \emptyset\, \forall\, t_c \in C$
4: **for all** $m = (c', \bar{t}) \in M$ **do**
5: | **if** $\forall t \in \bar{t} : t \in A$ **then**
6: | | EVALUATE(m)
7: | | $\mathit{updatedTasks} = \mathit{updatedTasks} \cup \{c'\}$
8: **while** $\mathit{updatedTasks} \neq \emptyset$ **do**
9: | Select and remove $c \in \mathit{updatedTasks}$
10: | **for all** $m = (c', \bar{t}) : c \in \bar{t}$ **do**
11: | | **if** $\forall t \in \bar{t} : \Phi(t) \neq \emptyset$ **then**
12: | | | **if** EVALUATE(m) **then**
13: | | | | $\mathit{updatedTasks} = \mathit{updatedTasks} \cup \{c'\}$
14: **if** $(1, \dots, 1) \in \Phi(c^*)$ **then**
15: | **return** *True*
16: **else**
17: | **return** *False*

18: **function** EVALUATE($m = (c', \langle t_1 \dots t_l \rangle)$)
19: | $\Phi^{old} = \Phi(t_l)$, $\Phi^{new} = \emptyset$
20: | **for** $j = l - 1 \dots 1$ **do**
21: | | **for all** $\phi^{old} \in \Phi^{old}$ **do**
22: | | | **for all** $\phi^j \in \Phi(t_j)$ **do** ▷ *Build new tuple* ϕ^{new}
23: | | | | **for** $i = 1 \dots k$ **do**
24: | | | | | **if** $\phi_i^{old} \neq 0$ **then**
25: | | | | | | $\phi_i^{new} = \phi_i^{old}$
26: | | | | | **else**
27: | | | | | | $\phi_i^{new} = \phi_i^j$
28: | | | | $\Phi^{new} = \Phi^{new} \cup \{\phi^{new}\}$
29: | | $\Phi^{old} = \Phi^{new}$
30: | | $\Phi^{new} = \emptyset$
31: | **if** $\Phi^{old} \setminus \Phi(c') \neq \emptyset$ **then**
32: | | $\Phi(c') = \Phi(c') \cup \Phi^{old}$
33: | | **return** *true*
34: | **return** *false*

out there is no such k, which means that deciding $F_k \in k\text{-}\mathit{eff}^{+/-}_{\mathcal{P}}(c)$ is fixed-parameter tractable, i.e., for fixed k it is polytime-decidable. To prove this we present an algorithm that is polynomial for fixed k. The respective pseudo code is presented in Algorithm 1, which we now explain.

Let $k \in \mathbb{N}$ be fixed, $\mathcal{D} = (F, A, C, M)$ a domain, $c^* \in C$ a compound task, and $F_k = \{f_1 \dots f_k\} \subseteq F$ a set of facts. We want to determine whether $F_k \in k\text{-}\mathit{eff}^{+/-}_{\mathcal{P}}(c^*)$. First, we restrict the domain to only those facts that are contained in F_k since all others are irrelevant (because there are no preconditions). For every primitive and compound task $t \in A \cup C$ we keep a set of information over facts $\Phi(t)$ that a task can add and delete simultaneously, resp. Therefore, the elements of Φ are k-tuples $\phi = (\phi_1, \dots, \phi_k) \in \{1, 0, -1\}^k$, where $\phi_i = -1$ represents that fact f_i is deleted, $\phi_i = 0$ means that it is neither added nor deleted, and $\phi_i = 1$ if f_i is added. For primitive tasks $t_p \in A$ we define $\Phi(t_p) = \{\phi^{t_p}\}$, where $\phi_i^{t_p} = 1$ if $f_i \in \mathit{add}(t_p)$, $\phi_i^{t_p} = -1$ if $f_i \in \mathit{del}(t_p)$ and $\phi_i^{t_p} = 0$ otherwise. For compound tasks $t_c \in C$ the sets $\Phi(t_p)$ are empty in the beginning and filled step-by-step until a fixpoint is reached. The overall idea is that we keep track of all possible outcomes of tasks with the aid of Φ, where different refinements were kept separately (the ϕs). This information gets built and propagated in an upward manner through the hierarchy.

We begin with methods $m = (c, \bar{t})$ that contain only primitive tasks and evaluate their outcome. We do so by considering one fact after another and checking whether it is added or deleted eventually or never touched at all when applying the methods' sequence of actions. The result is then stored in a new k-tuple ϕ according to the semantics above, which is added to $\Phi(c)$ if it is not already contained. We keep track of compound tasks that got new information, since those need to be propagated further through the hierarchy. As data structure, we use the set $\mathit{updatedTasks} \subseteq C$ that is empty initially. So if a new tuple ϕ is added to some $\Phi(c)$ then c is also added to $\mathit{updatedTasks}$ (if it is not already contained).

After all methods with only primitive tasks are evaluated the main propagation phase can start. As long as $\mathit{updatedTasks}$ is not empty a compound task $c \in \mathit{updatedTasks}$ is picked and removed from the set. Then all methods $m = (c', \bar{t})$ that contain c ($c \in \bar{t}$) and for which it holds $\Phi(t) \neq \emptyset$ for all $t \in \bar{t}$ are evaluated according to EVALUATE that simulates the application of the tasks in the method depending on the different refinement choices:

Let $\bar{t} = \langle t_1 \dots t_l \rangle$. We want the outcomes of m for every combination of task outcomes $(\phi^1, \dots, \phi^l)$, where $\phi^j \in \Phi(t_j)$. For that, compute a new tuple ϕ' as follows. Consider ϕ'_i, i.e. the ith entry of ϕ'. If $\phi_i^j = 0$ for all $j \in \{1 \dots l\}$ then $\phi'_i = 0$. Otherwise, let $j' = max(\{j \in \{1 \dots l\} \mid \phi_i^j \neq 0\})$, then $\phi'_i = \phi_i^{j'}$. Basically, for every fact we check whether it is added or deleted eventually or never touched at all. Now, if $\phi' \notin \Phi(c)$ then add ϕ' to $\Phi(c)$ and add c to the set $\mathit{updatedTasks}$ (if not already contained). However, simply considering and evaluating every such combination separately ends up in a run-time exponential in l, which is contradicting a polynomial algorithm. However, one can also calculate it differently as stated in EVALUATE: We can "apply" task by task instead of considering the whole sequence at once. We start with t_l and t_{l-1} since for all $i = 1 \dots k$ it is always the rightmost task with a non-zero entry that determines the entry of the new tuple. For that, we consider every combination of task outcomes in $\Phi(t_l)$ and $\Phi(t_{l-1})$ and calculate the resulting tuples ϕ^{new} (lines 23-27). We go on and combine those with the ones of $\Phi(t_{l-2})$ and so forth down to $\Phi(t_1)$. The resulting set then contains all outcomes of m.

When $\mathit{updatedTasks}$ is empty a fixpoint is reached and no more information can be propagated. Now, we can read off $\Phi(c^*)$: If and only if $(1, \dots, 1) \in \Phi(c^*)$ then $F_k \in k\text{-}\mathit{eff}^+_{\mathcal{P}}(c^*)$ and $F_k \in k\text{-}\mathit{eff}^-_{\mathcal{P}}(c^*)$ if and only if $(-1, \dots, -1) \in \Phi(c^*)$.

Theorem 3. *Algorithm 1 is sound and complete and runs in* $\mathcal{O}(|C| \cdot |M| \cdot |m_{max}| \cdot k \cdot 3^{3k})$.

Proof. Runtime: Consider the function EVALUATE$(m = (c', \langle t_1 \dots t_l \rangle))$ first. A set Φ can contain at most 3^k elements. Thus, EVALUATE has a runtime of $\mathcal{O}(l \cdot k \cdot 3^{2 \cdot k})$. In the overall algorithm, the lines 8 to 13 are computationally the most expensive ones. Since every compound task can be added to $updatedTasks$ at most 3^k times we end up with a runtime of roughly $\mathcal{O}(|C| \cdot 3^k \cdot |M| \cdot |m_{max}| \cdot k \cdot 3^{2k})$, where $|m_{max}|$ is the number of tasks in the longest method.

Completeness: Assume it holds $F_k \in k\text{-}eff^{+}_{\mathcal{P}}(c^*)$. Then there is a refinement $\bar{t}$ and a sequence of decompositions such that $\bar{t}$ adds all $f \in F_k$. Considering those methods and intermediate tasks in a bottom-up order within Algorithm 1 produces $(1, \dots, 1) \in \Phi(c^*)$.

Soundness: Since we track the outcome of all possible refinements of a compound task separately within the Φs and we propagate them according to the methods' tasks and order we do not produce wrong outcomes. Note that the propagation is only possible because we do not consider any preconditions, which, in a general planning problem, massively restrict the executability of refinements.

□

Corollary 6. *Deciding $F_k \in k\text{-}eff^{+/-}_{\mathcal{P}}(c)$ and $F_k \in k\text{-}eff^{+/-}_{\mathcal{PD}}(c)$ is in P for fixed k and thus fixed-parameter tractable.*

The algorithm computes actually more than $k\text{-}eff^{+/-}_{\mathcal{P}}(c^*)$: We get every combination of added, deleted, and not touched facts (under precondition-relaxation).

Conclusion

We have seen that deciding whether a set of facts is a conjunctive possible effect of a compound task and the plan existence problem (with goal description) under $\mathcal{P}$- and $\mathcal{PD}$-relaxation is NP-complete. Interestingly, this ranges from the simplest case – acyclic and regular $\mathcal{PD}$-relaxed domains – to arbitrary $\mathcal{P}$-relaxed ones even though the problems do not seem equally hard. Additionally, we presented a polynomial algorithm for fixed k to compute size-k effects. Thus, for small k they can be calculated efficiently with the hope of offering valuable information that can be exploited by heuristics, pruning, or reachability analyses.

Appendix

In the process of proving NP membership of $\mathcal{P}$-relaxed HTN planning problems, we concluded that there is a decomposition tree of size at most $|T_{ach}| \cdot |path| \cdot |m_{max}|$ and it remained to show that there is a tree such that $|path|$ is polynomial. Alford et al. (2014) showed that in a similar scenario for delete-relaxed HTN planning problems but they assume that methods do not have empty task networks. So we have to slightly adapt the proof but can still use the same idea.

The main argument rests on the fact that we can concatenate trees that are higher than $|path|$. For that, we use subtree substitutions introduced by Geier and Bercher (2011), which preserve the property of being a decomposition tree.

Let $\mathcal{T} = (N, E, \beta)$ be a decomposition tree and $n \in N$ a node. The subtree of $\mathcal{T}$ induced by n, written $\mathcal{T}[n]$, is $\mathcal{T}[n] := (N', E', \beta|_{N'})$, where (N', E') is the subtree in (N, E) that is rooted at n. The next definition states how subtrees can be substituted by other subtrees. Note that the given definition is only meaningful under certain circumstances that follow in the subsequent proposition by Geier and Bercher (2011). In the general case, one should replace $\mathcal{T}[n_i]$ with an isomorphic copy of $\mathcal{T}[n_j]$.

Definition 7 (Subtree Substitution)**.** *Let $\mathcal{T} = (N, E, \beta)$ be a decomposition tree and $n_i, n_j \in N$ be two nodes. If n_i is the root node of $\mathcal{T}$, we define the result of the subtree substitution on $\mathcal{T}$ that substitutes n_i by n_j, written $\mathcal{T}[n_i \leftarrow n_j]$, as $\mathcal{T}[n_i \leftarrow n_j] := \mathcal{T}[n_j]$; otherwise, $\mathcal{T}[n_i \leftarrow n_j] := (N', E', \beta|_{N'})$ with*

- $N' := (N \setminus N(\mathcal{T}[n_i])) \cup N(\mathcal{T}[n_j])$
- $E' := E|_{N'}$ *and n_i is replaced by n_j in $E(p)$, where p is the parent node of n_i*

Proposition 5. *Let $\mathcal{T} = (N, E, \beta)$ be a decomposition tree for a $\mathcal{P}$-relaxed planning problem $\Pi = (\mathcal{D}, s_I, tn_I, g)$ and $n_i \in N$, $n_j \in N(\mathcal{T}[n_i])$ two nodes with $\beta(n_i) = \beta(n_j)$. Then $\mathcal{T}[n_i \leftarrow n_j]$ is also a decomposition tree for Π.*

Now we can state and prove the main theorem that we can always find a witness of bounded height, which is similar to Theorem 5.10 by Alford et al. (2014).

Theorem 4. *Let $\Pi = (\mathcal{D}, s_I, tn_I, g)$ be a solvable $\mathcal{P}$-relaxed problem having a minimal solution of size l. Then there exists a witness $(\mathcal{T}, (tn, \psi))$ that Π has a solution of size $\leq l$ with $|N(\mathcal{T})| \leq |g| \cdot |C| \cdot |A| \cdot |m_{max}|$.*

Proof. Let Π be solvable with an optimal solution of size l. Then there exists a decomposition tree $\mathcal{T}_{pre} = (N_{pre}, E_{pre}, \beta_{pre})$ whose yield is a primitive task network $tn_{pre} = \langle t_1 \dots t_l \rangle$, which can form a partial-achiever solution $(tn_{pre}, T^{pre}_{ach}, \psi_{pre})$. Assume that $\mathcal{T}_{pre}$ is such a tree of minimal height and let $\mathcal{T}$ be the S-sapling of $\mathcal{T}_{pre}$ with $S = \{n \in N_{pre} \mid \exists i \in T^{pre}_{ach} : \beta^{-1}_{pre}(t_i) = n\}$. The yield tn of $\mathcal{T}$ also forms a partial-achiever solution (tn, T_{ach}, ψ), where T^{pre}_{ach}, T_{ach} and ψ_{pre}, ψ, resp., are the same but adapted to new indices. So the tuple $(\mathcal{T}, (tn, T_{ach}, \psi))$ is a witness for Π's solvability of size at most $|T_{ach}| \cdot |path| \cdot |m_{max}|$, where $|path|$ is the height of $\mathcal{T}$.

Assume that $\mathcal{T}$ has a height greater than $|C| \cdot |A|$. Then there must exist a path from the root to some node $n_s \in S$ of that length. Let $n_1, \dots, n_s$ be the nodes of that path. Since all inner nodes of the tree represent compound tasks and there are $|C|$ many of them we know that always after at most $|C|$ nodes on that path two nodes n_i and n_j map to the same compound task, i.e., $\beta(n_i) = \beta(n_j)$. Furthermore, the path $n_1, \dots, n_s$ is joined at most $|S| - 1 \leq |A|$ times by other paths from S to the root. Since $|path| > |C| \cdot |A|$ there must be some segment n_i and n_j with $\beta(n_i) = \beta(n_j)$ between joins such that none of the descendants of n_i that are not also descendant of n_j are in $S \setminus \{n_s\}$.

So we can concatenate $\mathcal{T}$ by substituting n_i by n_j, which gives us $\mathcal{T}' = \mathcal{T}[n_i \leftarrow n_j]$. Since we did not remove any of the achiever tasks the new tree $\mathcal{T}'$ also forms a witness with its yield tn', which has a smaller height and contains fewer nodes than $\mathcal{T}$. We can repeat this procedure as long as the height of the tree is greater than $|C| \cdot |A|$ until we end up with a tree of height at most $|C| \cdot |A|$.

□

References

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

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. 2014. On the Feasibility of Planning Graph Style Heuristics for HTN Planning. In *Proc. of ICAPS*, 2–10. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning through SAT. In *Proc. of AAAI*, 6110–6118. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proc. of IJCAI*, 6267–6275. IJCAI.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks. In *Proc. of ECAI*, 225–233. IOS Press.

Bit-Monnot, A.; Smith, D. E.; and Do, M. 2016. Delete-Free Reachability Analysis for Temporal and Hierarchical Planning. In *Proc. of ECAI*, 1698–1699. IOS Press.

Clement, B. J.; Durfee, E. H.; and Barrett, A. C. 2007. Abstract Reasoning for Planning and Coordination. *Journal of Artificial Intelligence Research (JAIR)*, 28: 453–515.

de Silva, L.; Meneguzzi, F. R.; and Logan, B. 2020. BDI Agent Architectures: A Survey. In *Proc. of IJCAI*, 4914–4921. IJCAI.

de Silva, L.; Sardina, S.; and Padgham, L. 2009. First Principles Planning in BDI Systems. In *Proc. of AAMAS*, 1105–1112. IFAAMAS.

de Silva, L.; Sardina, S.; and Padgham, L. 2016. Summary Information for Reasoning About Hierarchical Plans. In *Proc. of ECAI*, 1300–1308. IOS Press.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 18(1): 69–93.

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

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goldman, R. P.; and Kuter, U. 2019. Hierarchical Task Network Planning in Common Lisp: the case of SHOP3. In *Proc. of the 12th European Lisp Symposium (ELS 2019)*, 73–80. ACM.

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. AAAI Press.

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

Magnaguagno, M. C.; Meneguzzi, F. R.; and de Silva, L. 2021. HyperTensioN: A three-stage compiler for planning. In *Proc. of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*.

Marthi, B.; Russell, S. J.; and Wolfe, J. A. 2007. Angelic Semantics for High-Level Actions. In *Proc. of ICAPS*, 232–239. AAAI Press.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research (JAIR)*, 20: 379–404.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks – A Complexity Analysis. In *Proc. of AAAI*, 11903–11912. AAAI Press.

Olz, C.; Wierzba, E.; Bercher, P.; and Lindner, F. 2021. Towards Improving the Comprehension of HTN Planning Domains by Means of Preconditions and Effects of Compound Tasks. In *Proc. of the 10th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2021)*.

Schreiber, D. 2021. Lilotane: A Lifted SAT-Based Approach to Hierarchical Planning. *Journal of Artificial Intelligence Research (JAIR)*, 70: 1117–1181.

Thangarajah, J.; and Padgham, L. 2011. Computationally Effective Reasoning About Goal Interactions. *Journal of Automated Reasoning*, 47(1): 17–56.

Tsuneto, R.; Hendler, J.; and Nau, D. 1998. Analyzing External Conditions to Improve the Efficiency of HTN Planning. In *Proc. of AAAI*, 913–920. AAAI Press.

Waisbrot, N.; Kuter, U.; and Könik, T. 2008. Combining Heuristic Search with Hierarchical Task-Network Planning: A Preliminary Report. In *Proc. of FLAIRS*, 577–578. AAAI Press.

Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6(1): 12–24.

Yao, Y.; de Silva, L.; and Logan, B. 2016. Reasoning about the Executability of Goal-Plan Trees. In *Proc. of the 4th Int. Workshop on Engineering Multi-Agent Systems (EMAS 2016)*, 176–191. Springer.