# Reversibility and Reachability in HTN Planning: Formalization and Computational Complexities in the Totally-Ordered Setting

**Jakub Med**[1,2], **Mohammad Yousefi**[3], **Lukáš Chrpa**[1], **Pascal Bercher**[3]

[1]Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague
[2]Faculty of Electrical Engineering, Czech Technical University in Prague
[3]School of Computing, The Australian National University
{jakub.med, chrpaluk}@cvut.cz, {mohammad.yousefi, pascal.bercher}@anu.edu.au

## Abstract

Action reversibility, that is, whether it is possible to undo effects of an action by other actions, has been studied in classical planning and, recently, in non-deterministic planning. In this paper, we formalize the notions of method and primitive task reversibility in the context of hierarchical task network (HTN) planning and provide complexity results. On top of that, we introduce various notions of reachability in the HTN setting, for which the reachability is still an unexplored area (in contrast to classical planning, in which reachability is well studied) We divide the reachability into two classes based on two perspectives, one restricting the allowed progression rules (i.e., reachability using executions of primitive tasks, or reachability using decompositions of compound tasks), and the other focusing on the desired target (i.e., a state, independently on a task network; a task network, independently of a state; or both at once). We show that the complexity of these problems varies significantly, ranging from EXPTIME-complete to constant-time. We also show that the introduced reversibility problems exhibit theoretical properties and complexity results analogous to the broader reachability classes.

## Introduction

Hierarchical task network (HTN) planning is a framework for deliberating about a goal at multiple levels of abstraction by means of breaking down tasks into smaller subtasks. This hierarchical structure allows one to specify "how" to achieve a goal (i.e., a procedural form of reasoning). In contrast, classical planning only allows specifying "what" needs to be achieved (i.e., a declarative form of reasoning) (Bercher, Alford, and Höller 2019; Ghallab, Nau, and Traverso 2016; Erol, Hendler, and Nau 1994). The introduction of the hierarchy, while increasing the expressivity (Höller et al. 2014), makes the problem undecidable in the general case (Erol, Hendler, and Nau 1994). This undecidability result has prompted researchers to identify decidable subclasses that preserve expressive power, and one well-studied example is the total-order restriction (Yousefi et al. 2025; Schreiber 2021; Barták et al. 2021; Behnke, Höller, and Biundo 2018; Erol, Hendler, and Nau 1994).

In the literature, the notion of action reversibility refers to whether the effects of an action can be "undone" or "reverted" by the application of (other) actions. Early work by Eiter, Erdem, and Faber (2007, 2008) formally defined action reversals in a logic-based framework, analyzing complexity, and proposing reverse plan libraries for efficient recovery. Inspired by the former works, Weber et al. (2012, 2013) modeled a cloud management system as a planning domain and applied similar reasoning to cloud management tools to ensure safety by detecting reversible and irreversible commands. Later, a more general notion of (non-uniform) reversibility has been tackled by compiling the problem into contingent planning (Daum et al. 2016). In recent years, the work of Morak et al. (2020) unified the different notions of reversibility into one framework, called $S$-*reversibility*. It proposed notions of *uniform* and *universal* action reversibility, which were later researched in more detail by several works in the deterministic, classical planning setting (Chrpa, Faber, and Morak 2021; Faber, Morak, and Chrpa 2022; Faber and Morak 2025) and also generalized and investigated under Fully Observable Non-Deterministic (FOND) setting (Med et al. 2024, 2025).

Apart from the verification of safe operations (Weber et al. 2012, 2013), reversibility can classify action risks and generate recovery plans for cyber attacks (Boddy et al. 2005). Reversibility is particularly valuable for online planning (Cserna et al. 2018), environments with exogenous events (Chrpa, Pilát, and Med 2021; Chrpa and Karpas 2024), or in non-deterministic planning by verifying that agents can recover from undesirable effects (Camacho, Muise, and McIlraith 2016) and can determine the safety of deterministic replanning in unknown states (Yoon, Fern, and Givan 2007). Furthermore, reversibility analysis supports post-planning optimization by detecting redundant action cycles or inverse actions (Chrpa, McCluskey, and Osborne 2012a; Med and Chrpa 2022).

In this paper, we introduce the notions of hierarchical reversibility and reachability, in conjunction with the standard state-based reachability—a notion well-studied in classical planning, but absent in hierarchical task network (HTN) formalisms up to date. We motivate the questions investigated by the following, among others: 1) Goal-reachability is a special class of general reachability. Thus, one can ask questions like "can this one particular task be refined?" or "can we undo an internal state of the planner?" Such questions are answered in our formalization. 2) Reversibility might be more important in HTN planning than classical plan-

ning as the cost of replanning and plan repair is significantly higher in these settings (Zaidins et al. 2025). This stems from the commitment to refining a task, which cannot be easily thrown away without breaking the hierarchy. Thus, a preprocessing step to find reversible tasks might allow for better performance at runtime. 3) Learning on the fly is a popular approach to solving HTN problems (Patra et al. 2021). A reversibility analysis allows avoiding dead-ends in this approach.

The paper integrates previously investigated action reversibility definitions into the totally-ordered HTN (TOHTN) context, specifically introducing two variants: method reversibility and primitive task reversibility. Furthermore, we divide reversibility and reachability into two classes based on two perspectives, one restricting the allowed progression rules (i.e., reachability using executions of primitive tasks application only, or reachability using decompositions of compound tasks only, or under regular progression), and the other focusing on the desired target (i.e., a state, independently of a task network; a task network, independently of a state; or both at once) allowing us to capture broader spectrum of forms of state-hierarchical reversibility and reachability.

Then, we investigate how hard it is to decide on the proposed notions. We show that the computational complexity of defined notions of reversibility under progression follows the pattern known from non-hierarchical settings where (action) reversibility has been studied in the past, that is "as hard as planning". This is done by reducing the solvability of the TOHTN planning problem, which is known to be EXPTIME-complete (Erol, Hendler, and Nau 1994), to the action reversibility, which yields its EXPTIME-hardness. We follow with the compilation of reachability problems to the TOHTN planning problem solvability, yielding the blueprint on how to solve a reachability with existing tools and showing that reachability is "not harder than planning". Later, we (formally) discuss the link between reversibility and reachability that has already been discussed earlier in more depth, which allows us to infer the EXPTIME membership for reversibility questions and EXPTIME-hardness for reachability. Along with the investigation of the intractable problems, we prove that some variants are trivially or polynomially decidable. With such results, we complete the tightly-bounded complexity landscape.

## Preliminaries

In this work, we use standard definitions such as *string*, *word*, *alphabet*, and *language*. We denote $X^*$ for the Kleene closure of an alphabet $X$, $w_1w_2$ for the concatenation of two words $w_1$ and $w_2$ (Hopcroft, Motwani, and Ullman 2007).

The following definitions are based on the formalization provided by Yousefi et al. (2025). For a simple example of HTN planning we refer the reader to the work of Bercher, Alford, and Höller (2019). Now, we start with the definition of TOHTN planning domain.

**Definition 1.** *Let $A$ and $C$ be sets, and $A \cup C$ be the alphabet. A **(TOHTN planning) domain** is a tuple*[1] $\mathcal{D} = \langle F, A, C, M \rangle$ *where:*

- *$F$ is a finite set of facts,*
- *$C$, $A$; $C \cap A = \emptyset$, are disjoint finite sets of compound and primitive task names, respectively,*
- *$M \subseteq C \times (A \cup C)^*$ is a finite set of totally ordered decomposition methods.*

*The **set of all task names** of a planning domain $\mathcal{D}$, $A \cup C$, is denoted as $\boldsymbol{T(\mathcal{D})}$.*

For a domain $\mathcal{D}$, whenever it is clear to which domain we refer, we write $T$ instead of $T(\mathcal{D})$.

In contrast to traditional, classical planning, HTN planning induces additional constraints on the implicitly encoded state transition system. The additional constraints are enforced by the fact that the planner no longer has the freedom to execute every applicable action, as (for example) is the case in classical planning. Instead, we are provided an initial task network that we need to "perform".

A *task network* is a finite string of compound and primitive task names. In TOHTN, the task network is a totally-ordered sequence of arbitrary tasks that are waiting to be decomposed (if composed) or executed (if primitive).

**Definition 2.** *A **totally ordered task network** $tn$ of a domain $\mathcal{D}$ is a word (or a string) over the alphabet $T(\mathcal{D})$, formally, $tn = (t_i)_{i=1}^{|tn|} \in T(\mathcal{D})^*$. For $k \in \mathbb{N}_0$, $1 \leq k \leq |tn|$, we denote **$k$-th symbol** $t_k$ of the task network $tn = (t_i)_{i=1}^{|tn|}$ as $\boldsymbol{tn_{[k]}}$, i.e. $tn_{[k]} = t_k$. Furthermore, we write $\boldsymbol{t \in tn}$ to denote that a task name $t$ is **contained** in a task network $tn$; formally, $t \in tn$ if and only if $t = tn_{[k]}$ for some $k \in \mathbb{N}_0$, $1 \leq k \leq |tn|$. We write $t \notin tn$ otherwise.*

In this work, whenever clear, we refer to "task name" by "task" only. Also, the notation $tn$ in this work refers to a single "task network", not to a concatenation of two words $t$ and $n$. Lastly, note that if there are multiple occurrences of $t$ in $tn$, then there are multiple $k$ for which $t = tn_{[k]}$. In other words, the notation $t \in tn$ does not refer to a specific occurrence, but rather states that there is at least one $t$ in $tn$.

Given a domain $\mathcal{D} = \langle F, A, C, M \rangle$, for each primitive task name $a \in A$, the corresponding *action* is a tuple $\langle pre(a), add(a), del(a) \rangle$ where $pre(a) \subseteq F$ denotes the preconditions that must hold to be able to execute the action, and $add(a) \subseteq F, del(a) \subseteq F$, with $add(a) \cap del(a) = \emptyset$, denote the *add* and *delete* effects of $a$, respectively.[1] An action $a \in A$ is *executable* in state $s \in 2^F$ if and only if $pre(a) \subseteq s$. If a primitive action $p \in A$ is executable in a state $s \in 2^F$, the resulting state is defined as $\gamma(s, a) = (s \setminus del(a)) \cup add(a)$ and is undefined otherwise.

As already stated, in HTN planning, the planner can only "perform" tasks that are ordered before any other task. Under total order HTN such a task is the first task of the network.

Compound tasks induce the task hierarchy. As the name suggests, they are tasks consisting of potentially (several) additional tasks. To "process" a compound task $c$ away, we need to substitute it using some method $(c, \omega) \in M$ that

[1] For brevity, we have omitted the explicit mapping function $\delta$ (which maps primitive task names to the preconditions and effects) from the domain $\mathcal{D}$ known from other works (Höller et al. 2020a).

describes one of the substitution rules for the task $c$ in $\mathcal{D}$. However, as we need to follow left-to-right progression, to do such a substitution, the task needs to be ordered first.

**Definition 3.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain, $tn \in T(\mathcal{D})^*$ be a non-empty task network, and $m = (c, \omega) \in M$ be a method. We say that the method $m$ is **decomposable** in $tn$ if $c = tn_{[1]}$.*

If one decomposes a compound task $c$ with a method $m = (c, \omega)$ in a task network where $c$ is decomposable, the (first) occurrence of $c$ is (simply) substituted with $\omega$.

**Definition 4.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a domain, $m = (c, \omega) \in M$ be a method, with $c \in T(\mathcal{D})$, and $tn_1 \in T(\mathcal{D})^*$ be a task network such that $tn_1 = cw$ for $w \in T(\mathcal{D})^*$. The result of a **decomposition** of $c$ with $\omega$ by $m$, denoted as $\boldsymbol{tn_1 \rightarrow^c_m tn_2}$, is the task network $tn_2 = \omega w$.*

The decomposability and decomposition of a method in a state-task network pair $\langle s, tn\rangle$ is defined analogously.

The action execution works analogously. The primitive task is applicable if and only if it is executable and the task is ordered first.

**Definition 5.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain, $s \in 2^F$ be a state, $tn \in (A \cup C)^*$ be a task network, and $p \in A$ be a primitive action. We say that the primitive action $p$ is **applicable** in $\langle s, tn\rangle$ if and only if $p$ is executable in $s$ and $p = tn_{[1]}$.*

**Definition 6.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain, $s \in 2^F$ be a state, $tn$ be a task network and $p \in A$ be a primitive task which is applicable in $\langle s, tn\rangle$. The result of an **application** of $p$ in $\langle s, tn\rangle$, denoted as $\boldsymbol{\langle s, tn\rangle \mapsto_p \langle s', tn'\rangle}$, is a state-task network pair $\langle s', tn'\rangle = \langle \gamma(s, p), (tn_{[i]})^{|tn|}_{i=2}\rangle$.*

Note that the execution of a primitive task is equivalent to the action application in the classical planning setting.

As the task network can "begin" with a primitive or compound task, and as we strictly need to "process" the tasks in the left-to-right manner, given the definitions of primitive task applicability and compound task decomposition, we can now define the progressions of a task network (Höller et al. 2020b) which captures all possible cases.

**Definition 7.** *Let $s \in 2^F$ be a state, $tn$ be a task network. The pair $\langle s, tn\rangle$ is said to be **progressed** into $\langle s', tn'\rangle$, denoted as $\langle s, tn\rangle \rightsquigarrow \langle s', tn'\rangle$, if and only if:*

- *$tn_{[1]}$ is a compound task $c \in C$ and there exist a method $m = (c, \omega) \in M$ such that $\langle s, tn\rangle \rightarrow^c_m \langle s, tn'\rangle$, or*
- *$tn_{[1]}$ is a primitive task $p \in A$ such that $\langle s, tn\rangle \mapsto_p \langle s', tn'\rangle$.*

Effectively, progression always "processes" away the first task. It either changes the state and pops the task from the task network, or it keeps the state untouched and substitutes the compound task with another task network.

The progression provides us with a way to achieve one state-task network pair from another. Similar to non-hierarchical settings, this is referred to as reachability.

**Definition 8.** *We say that $\langle s', tn'\rangle$ is **reachable** from $\langle s, tn\rangle$, denoted as $\boldsymbol{\langle s, tn\rangle \rightsquigarrow^* \langle s', tn'\rangle}$, if and only if $\langle s', tn'\rangle$ is in the reflexive and transitive closure of $\rightsquigarrow$ starting in state-task network pair $\langle s, tn\rangle$.*

In this paper, we use a standard definition of TOHTN planning problem with goal conditions. Note that in most of state of the art, the goal conditions are not present. Yet, it is well known that the two are interchangeable (Geier and Bercher 2011).

**Definition 9.** *A **(TOHTN planning) problem** is a tuple $\mathcal{P} = \langle\langle F, A, C, M\rangle, s_I, tn_I, G\rangle$ where:*

- *$\langle F, A, C, M\rangle$ is a (TOHTN planning) domain $\mathcal{D}$,*
- *$s_I \in 2^F$ be an initial state,*
- *$tn_I \in T(\mathcal{D})^*$ is an initial task network,*
- *$G \subseteq F$ are the goal conditions.*

Typically, the aim in HTN planning is to refine the initial compound task to the empty task network (i.e., perform all tasks). However, in our setting with goal conditions, we also require that the resulting state satisfies given goal conditions.

**Definition 10.** *Let $\mathcal{P} = \langle\langle F, A, C, M\rangle, s_I, tn_I, G\rangle$ be a planning problem. We say that the problem is **solvable** if and only if there exists a state $s' \in 2^F$ and a sequence of progressions $\rightsquigarrow^*$ such that $G \subseteq s'$ and $\langle s_I, tn_I\rangle \rightsquigarrow^* \langle s', \epsilon\rangle$, where $\epsilon$ is an empty task network.*

## Reachability

In this section, we define two classes of reachability restrictions: first, we restrict the allowed progression options, then we restrict what exactly we want to reach.

We start with restricting possible refinement operators. Obviously, not restricting anything here makes the most sense since this is exactly what progression planners will have available. As for allowing only some of the refinement options: We do not have any practical motivation for them yet, but believe it's important to analyse the whole matrix of possibilities, just for understanding their interplay and identifying what contributes to hardness. Besides, hard special cases might prove useful for reductions.

Throughout this work, whenever we use the symbol $\triangleright$, we refer to one of the following types of progression rules (unless stated otherwise): substitution $\rightarrow$, application $\mapsto$ or (unrestricted) progression $\rightsquigarrow$. Conditions (ii) and (iii) are formulated this way for convenience, as Definition 12 clarifies.

**Definition 11.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain, $\langle s, tn\rangle \in 2^F \times T(\mathcal{D})^*$ and $\langle s', tn'\rangle \in 2^F \times T(\mathcal{D})^*$ be state-task network pairs. Let $\triangleright \in \{\rightsquigarrow, \mapsto, \rightarrow\}$ be a progression rule. Then, we say that $\langle s', tn'\rangle$ is **$\triangleright$-reachable** from $\langle s, tn\rangle$ if and only if there exists a state-task network pair $\langle s'', tn''\rangle \in 2^F \times T(\mathcal{D})^*$ such that:*

*(i) $\langle s'', tn''\rangle$ is in the reflexive and transitive closure of $\triangleright$ starting in $\langle s, tn\rangle$, also denoted $\boldsymbol{\langle s, tn\rangle \triangleright^* \langle s'', tn''\rangle}$,*
*(ii) $s'' = s'$, and*
*(iii) $tn'' = tn'$.*

The second class of restrictions defines what search node elements we actually care about. Ignoring task networks and just checking whether a certain state can be reached has obvious applications in heuristic search (Höller et al. 2020b) and pruning (Olz and Bercher 2023): if we know that a certain action occurs somewhere in the current task network, or will be introduced later (e.g., because it was identified

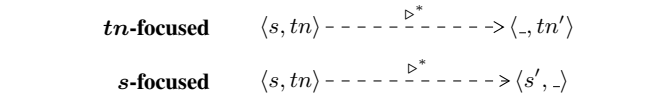


Figure 1: Visualisation of reachability variants. The dashed arrow $\triangleright^*$ denotes any sequence of a progression type, while $\langle \_ \rangle$ indicates components not in focus.

as landmark (Höller and Bercher 2021; Putrich, Meneguzzi, and Pereira 2025)), checking for reachability of its precondition will be useful. As for reaching a certain task network, disregarding the state, one application that comes to mind is plan recognition (Höller et al. 2018), where we have a hypothesis of which plan an agent might want to pursue. Also, modeling support and model repair (Bercher, Sreedharan, and Vallati 2025) would be a candidate, where the modeler might want to check whether some desired task network could be reached.

**Definition 12.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain, and $\langle s, tn\rangle \in 2^F \times T(\mathcal{D})^*$ and $\langle s', tn'\rangle \in 2^F \times T(\mathcal{D})^*$ be state-task network pairs. Let $\triangleright \in \{\rightsquigarrow, \mapsto, \rightarrow\}$ be a progression rule. Then:*

*I. $\langle s', tn'\rangle$ is **state-focused** (or **$s$-focused**) **$\triangleright$-reachable** in $\mathcal{D}$ if and only if all conditions except the condition (iii) for $\triangleright$-reachability of $\langle s', tn'\rangle$ from $\langle s, tn\rangle$ in $\mathcal{D}$ hold;*

*II. $\langle s', tn'\rangle$ is **task-network-focused** (or **$tn$-focused**) **$\triangleright$-reachable** in $\mathcal{D}$ if and only if all conditions except the condition (ii) for $\triangleright$-reachability of $\langle s', tn'\rangle$ from $\langle s, tn\rangle$ in $\mathcal{D}$ hold.*

The different notions of focused reachability (as well as $\triangleright$-reachability) are visualized in Figure 1.

In HTN planning, the hierarchy introduces an additional "dimension". The nodes of the underlying implicitly encoded graph encoding the search space do not consist only of the states, as in non-hierarchical forms of planning. In HTN planning, the nodes consist of pairs of states and task networks, effectively changing the semantics of the search.

## Reversibility

Following the reachability landscape, we define the corresponding reversibilities. The definition of $ST^*$-reversibility can be seen as a generalization of $S$-reversibility (Morak et al. 2020) to TOHTN planning. The intention and motivation of S-reversibility in the literature is to be able to state that the action is reversible in any state of the set. This allows one to investigate analogous notions of universality and uniformity (whose importance is discussed in the related work) of ST*-reversibility in future work.

**Definition 13.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain and $ST^* \subseteq 2^F \times T(\mathcal{D})^*$ be a set of state-task network pairs. Let $\triangleright \in \{\rightsquigarrow, \mapsto, \rightarrow\}$ be a progression rule. Then:*

- *a primitive **task $p \in A$** is called **$ST^*$-reversible under the progression type** $\triangleright$ in $\mathcal{D}$ if and only if for each pair $\langle s, tn\rangle \in ST^*$ in which $p$ is applicable there exists a sequence of progressions $\triangleright^*$ such that: (i) $\langle s, tn\rangle \mapsto_p \langle s', tn'\rangle \triangleright^* \langle s'', tn''\rangle$, (ii) $s'' = s$, and (iii) $tn'' = tn$;*
- *a **method $m \in M$** is called **$ST^*$-reversible under the progression type** $\triangleright$ in $\mathcal{D}$ if and only if for each pair $\langle s, tn\rangle \in ST^*$ in which $m$ is substitutable there exists a sequence of progressions $\triangleright^*$ such that: (i) $\langle s, tn\rangle \rightarrow_m \langle s, tn'\rangle \triangleright^* \langle s'', tn''\rangle$, (ii) $s'' = s$, and (iii) $tn'' = tn$.*

**Definition 14.** *Let $\mathcal{D} = \langle F, A, C, M\rangle$ be a TOHTN domain, $s_i \in 2^F$ be a state, $tn \in T(\mathcal{D})^*$ be a task network. Let $\triangleright \in \{\rightsquigarrow, \mapsto, \rightarrow\}$ be a progression rule. Let $ST^* \subseteq 2^F \times T(\mathcal{D})^*$ be a set of state-task network pairs. Then:*

*I. A **primitive task $p \in A$** or a **method $m \in M$** is called **state-focused** (or $s$-focused) **$ST^*$-reversible under the progression type** $\triangleright$ in $\mathcal{D}$ if and only if all conditions except condition (iii) for $ST^*$-reversibility of $p$ under $\triangleright$ in $\mathcal{D}$ hold.*

*II. A **primitive task $p \in A$** or a **method $m \in M$** is called **task-network-focused** (or $tn$-focused) **$ST^*$-reversible under the progression type** $\triangleright$ in $\mathcal{D}$ if and only if all conditions except condition (ii) for $ST^*$-reversibility of $m$ under $\triangleright$ in $\mathcal{D}$ hold.*

See Figure 2 for a visualization of reversibility variants.

We note that although this work does not address it, the granularity of our definitions for primitive task and method reversibility establishes a robust foundation for formally deriving the *reversibility of compound tasks* or of (any) *tasks* in general.

## Computational Complexity

In this section, we investigate the computational complexity of the following decision problems:

REACH($\triangleright$)
**Instance:** A TOHTN domain $\mathcal{D} = \langle F, A, C, M\rangle$, state-task network pairs $\langle s, tn\rangle \in 2^F \times T(\mathcal{D})^*$ and $\langle s', tn'\rangle \in 2^F \times T(\mathcal{D})^*$.
**Question:** Is $\langle s', tn'\rangle$ $\triangleright$-reachable from $\langle s, tn\rangle$ in $\mathcal{D}$?

$ST^*$-REVERSE($\triangle$, $\triangleright$)
**Instance:** An HTN domain $\mathcal{D} = \langle F, A, C, M\rangle$, either a primitive task or a method $\triangle \in A \cup M$, and a set of state-task network pairs $ST^* \subseteq 2^F \times T(\mathcal{D})^*$.
**Question:** Is the primitive task or the method $\triangle$ $ST^*$-reversible under the derivation rule $\triangleright$ in $\mathcal{D}$?

The decision problems for state- or task-network-focused reversibility or reachability are defined analogously.

Whenever we use the symbol $\triangle$, we refer either to a primitive task or a method $\triangle \in A \cup M$, unless stated otherwise.

### Reversibility

**Under Progression as Hard as Planning** Starting with one of the most general variants of defined reversibilities, for either primitive task or method, namely (full) $ST^*$-reversibility under the (full) progression, we show that the hardness results of both classical and FOND planning investigated by Morak et al. (2020); Med et al. (2025) translate also to TOHTN planning.

**Theorem 1.** *$ST^*$-REVERSE($\triangle$, $\rightsquigarrow$) is* EXPTIME*-hard.*

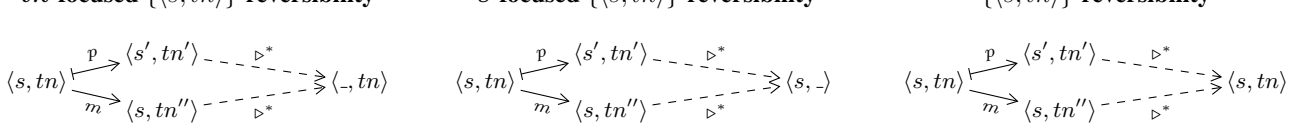


Figure 2: Intuition behind variants of reversibility for TOHTNs arranged horizontally. Each diagram illustrates the branching by a primitive task $p$ (using $\mapsto$) or method $m$ (using $\rightarrow$), followed by a derivation sequence $\triangleright^*$ (dashed) restoring the specific components required by the definition. We use $\langle \_ \rangle$ to denote components that are not in focus (i.e., arbitrary).

*Proof.* To show the hardness, we reduce TOHTN plan existence, a known EXPTIME-complete problem (Erol, Hendler, and Nau 1994; Alford, Bercher, and Aha 2015), to $ST^*$-REVERSE($p$, $\rightsquigarrow$), with $p \in A$.

Given a $\mathcal{P} = \langle\langle F, A, C, M\rangle, s_I, tn_I, G\rangle$, we construct $\mathcal{D}$ by extending the domain $\langle F, C, A, M\rangle$ as follows. The set of facts is extended by a "fresh" fact $done$. Then, we add two additional primitive tasks $p_S$ and $p_G$ (*starting* and *goal* task). The *starting* primitive task requires $done$, deletes it and adds the facts of $s_I$. The *goal* primitive task requires $G$, deletes all facts $F$ and adds the fact $done$. Finally, we add a new compound task $R$ (*reset* task), and the method $(R, p_S tn_I p_G R)$.

We show that $\mathcal{P}$ is solvable if and only if the primitive task $p_S$ is $\{\langle\{done\}, p_S tn_I p_G R\rangle\}$-reversible.

If $\mathcal{P}$ is solvable, then $\langle s_I, tn_I\rangle \rightsquigarrow^* \langle s_G, \epsilon\rangle$, where $s_G \supseteq G$. Then $\langle\{done\}, p_S tn_I p_G R\rangle \mapsto_{p_S} \langle s_I, tn_I p_G R\rangle \rightsquigarrow^* \langle s_G, p_G R\rangle \mapsto_{p_G} \langle\{done\}, R\rangle \rightarrow \langle\{done\}, p_S tn_I p_G R\rangle$, meaning $p_S$ is $\{\langle\{done\}, p_S tn_I p_G R\rangle\}$-reversible.

If $p_S$ is $\langle\{done\}, p_S tn_I p_G R\rangle$-reversible, then there is a way to "regain" $p_S$ in the task network. Since the only task that produces $p_S$ is $R$, $R$ must have been decomposed. As $R$ cannot be produced in $tn_I$, the "initial" occurrence of $R$ had to be decomposed. Any decomposition of $R$ increases the number of occurrences of $p_G$ in the current task network by one, meaning it necessarily increases. Since $p_G$ is also in the initial task network, the first occurrence needs to be progressed away. That is possible only in a state $s_G$ such that $s_G \supseteq G$ and when $tn_I$ is consumed. In other words, there must be a sequence of progressions $\rightsquigarrow_G^*$ such that $\langle\{done\}, p_S tn_I p_G R\rangle \mapsto_{p_S} \langle s_I, tn_I p_G R\rangle \rightsquigarrow_G^* \langle s_G, p_G R\rangle$. Clearly, $\langle s_I, tn_I\rangle \rightsquigarrow_G^* \langle s_G, \epsilon\rangle$, proving $\mathcal{P}$ is solvable.

To prove EXPTIME-hardness of a $ST^*$-REVERSE($m$, $\rightsquigarrow$), with $m \in M$, we follow the same construction where $(R, p_S tn_I p_G R)$ is replaced by methods $(T_S, p_s)$ and $(R, T_S tn_I p_G R)$. We then reduce the planning problem to $\{\langle s_I, T_S tn_I p_G R\rangle\}$-reversibility of the method $(T_s, p_s)$ in the same way. □

Notice that the same reformulation can be used to prove the hardness of both state- and task-network-focused variants for both primitive task and method, except for the variant of method state-focused reversibility that is trivially reversible (see the latter Observation 1). We formally argue the claim in the following Theorems and their proof sketches.

**Theorem 2.** TN-FOCUSED-$ST^*$-REVERSE($\triangle$, $\rightsquigarrow$) *is* EXPTIME*-hard.*

*Proof (Sketch).* Consider the same $\mathcal{P}$ and construction as in the proof of Theorem 1. We show that $\mathcal{P}$ is solvable if and only if the primitive task $p_S$ is task-network-focused $\{\langle\{done\}, p_S tn_I p_G R\rangle\}$-reversible. For that, one uses the same argumentation as in the proof of Theorem 1. For the hardness of TN-FOCUSED-$ST^*$-REVERSE($m$, $\rightsquigarrow$), $m \in M$, the same modeling trick is used. □

For the state-focused variant, there is a difference in the proof. To prove that $\mathcal{P}$ is solvable once the primitive task $p_S$ is reversible is done by arguing that we need to reachieve $done$, thus apply $S_G$.

**Theorem 3.** S-FOCUSED-$ST^*$-REVERSE($p$, $\rightsquigarrow$), *with* $p \in A$*, is* EXPTIME*-hard.*

*Proof (Sketch).* Consider the same $\mathcal{P}$ and the same construction as in the proof of Theorem 1. To show that $\mathcal{P}$ is solvable if and only if the primitive task $p_S$ is state-focused $\{\langle\{done\}, p_S tn_I p_G R\rangle\}$-reversible, we replace the argumentation of the "if" part of the proof while keeping the "only if" argumentation the same.

If $p_S$ is $\langle\{done\}, p_S tn_I p_G R\rangle$-reversible, then there is a way to reachieve $done$. As $p_G$ is the only achiever, $p_G$ had to be applied. As it requires $G$, the state reached before the application of $p_G$, that is, a state resulting from progression $tn_I$ and of any task it produced, and is denoted as $p_G$, is a superset of $G$. In other words, there has to be a sequence of progressions $\rightsquigarrow_G^*$ such that $\langle\{done\}, p_S tn_I p_G R\rangle \mapsto_{p_S} \langle s_I, tn_I p_G R\rangle \rightsquigarrow_G^* \langle s_G, p_G R\rangle$. Clearly, $\langle s_I, tn_I\rangle \rightsquigarrow_G^* \langle s_G, \epsilon\rangle$, proving $\mathcal{P}$ is solvable. □

**Trivial Cases** Notice that the proof fails when we consider method reversibility, as, when decomposing, there is no reason to reachieve $done$, since the state does not change. This gives us the first trivial observation.

**Observation 1.** S-FOCUSED-$ST^*$-REVERSE($m$, $\triangleright$), *with* $m \in M$*, is trivially reversible and decidable in* $\mathcal{O}(1)$.

In the reversibility landscape, there are three additional trivial observations similar to the result of Observation 1. When one restricts the progression type allowed to the application only, by the definition of the application in TOHTN, the task network shrinks with each application, and without progression there is no way to grow it. Consequently, when one asks about the reversibility of a primitive task under the application, when the task network is in focus, since the application of the primitive action itself shrinks the task network, it cannot be reversed by applications only.

**Observation 2.** *The problems* TN-FOCUSED-$ST^*$-REVERSE$(p, \mapsto)$ *and* $ST^*$-REVERSE$(p, \mapsto)$*, with* $p \in A$*, are trivially irreversible and decidable in* $\mathcal{O}(1)$.

The last trivial observation concerns primitive task reversibility and is relevant to Observation 1. If the progression is restricted to decomposition only, there is no way to change the state. If the primitive task in question changes the state and the state is in focus, it is not reversible. On the other hand, if it has no influence, then we are trivially done.

**Observation 3.** S-FOCUSED-$ST^*$-REVERSE$(p, \rightarrow)$*, with* $p \in A$*, degenerates to whether* $s = \gamma(s, p)$ *and is decidable in* PTIME.

**Tractable Variants** When we keep the focus on the state and do not care about the task network and restrict progression to application of other primitive tasks only, one can imagine that the influence of the primitive task in question on the state can be remedied by the application of the other (strictly) following, primitive tasks. Such a check can be done in polynomial time, and its generalized version (for actions of classical planning) was described by Med and Chrpa (2022) and named *action cycle*. It is a generalization of a pair of inverse actions (Chrpa, McCluskey, and Osborne 2012b) and can be understood as an (local, plan-bounded) occurrence of a reversible action that is (not necessarily consecutively) followed by its reverse plan.

**Proposition 1.** S-FOCUSED-$ST^*$-REVERSE$(p, \mapsto)$*, with* $p \in A$*, is in* PTIME.

*Proof.* Considering a primitive action $p$, the procedure goes through all (linearly many, with respect to the input) state-task network pairs $\langle s, tn\rangle \in ST^*$ and it checks the presence of a strictly consecutive sequence of primitive task prefixing the task network $tn$ forming an action cycle (Med and Chrpa 2022). The check can be done in linear time. The primitive task $p$ is HTN-state $ST^*$-reversible under application in $\mathcal{D}$ if and only if there exists such a prefix. □

On the other hand, when we ask about method reversibility and we focus on task networks only, a similar question arises. If the decomposition produces the same compound task in the last position, to achieve task-network-focused reversibility, we need to apply away anything that precedes it.

**Proposition 2.** TN-FOCUSED-$ST^*$-REVERSE$(m, \mapsto)$*, with* $m \in M$*, is in* PTIME.

*Proof.* Given $m = (c, \omega)$, for any possible $\langle s, tn\rangle \in ST^*$, the result of substitution of $c$ (if $c$ is substitutable in $tn$ by $m$) is $\langle s, tn'\rangle$. Formally, $\langle s, tn\rangle \rightarrow_m^c \langle s, tn'\rangle$. If $c \notin \omega$, then we prove the method is not reversible, as no sequence of applications can generate a new symbol. Also, if $\omega$ contains another non-primitive task name or any task name $t$ follows $c$, then we prove the method is not reversible, as $\mapsto^*$ cannot remove it from $tn'$. Therefore, for the method to be reversible under application, $\omega$ may contain only primitive task names as a prefix and has to end with $c$. In such a case the method is task-network-focused $\{\langle s, tn\rangle\}$-reversible if and only if the sequence of the primitive tasks $\pi = \langle \omega_{[i]} \rangle_{i=1}^{|\omega|-1}$ is applicable in $s$, which can be checked in PTIME. □

If we focus on the state as well, we restrict ourselves from "hiding" any applicable primitive task prefix into it.

**Proposition 3.** $ST^*$-REVERSE$(m, \mapsto)$*, with* $m \in M$*, is in* PTIME.

*Proof (Sketch).* The proof combines the proofs from Propositions 1 and 2. Here, the method is (generally) $ST^*$-reversible under application if and only if the method adds a prefix that forms a consecutive action cycle in $s$. □

In the end, we complete the reversibility hardness landscape with the first tractable, yet non-linear set of problems. All these problems have in common that, to prove the reversibility, they need to show that a certain task network can be restored. These essentially solve task-network-focused reachability, i.e., whether there exists $\rightarrow^*$ such that $tn \rightarrow tn' \rightarrow^* tn$ (up to constraints caused by the state). To decide the issue, as the hierarchy of TOHTN is essentially equivalent to context-free grammars (CFGs) (Hopcroft, Motwani, and Ullman 2007), one can invoke the CYK algorithm (Younger 1967) known from Formal Language Theory.

**Proposition 4.** $ST^*$-REVERSE$(\triangle, \rightarrow)$ *and* TN-FOCUSED-$ST^*$-REVERSE$(\triangle, \rightarrow)$ *are in* PTIME.

*Proof.* For any possible $\langle s, tn\rangle \in ST^*$, the result of the progression of $\triangle$ in question, we get to $\langle s', tn'\rangle$, formally $\langle s, tn\rangle \rightsquigarrow \langle s', tn'\rangle$. To prove task-network-focused $\{\langle s, tn\rangle\}$-reversibility of $\triangle$ under $\rightarrow$, we need to show that $tn$ is reachable from $tn'$ (no matter what the state is). As methods of TOHTN are equivalent to CFGs, this can be decided by the CYK algorithm deciding word reachability of a given CFG. CYK is known to run in polynomial time, thus we prove PTIME membership. For (general) $\{\langle s, tn\rangle\}$-reversibility of $\triangle$ under $\rightarrow$, we need to achieve also the same state. But, (i) in the case when $\triangle \in M$, $s = s'$, therefore the check degenerates to reachability of $tn$ from $tn'$ only, and (ii) in case when $\triangle \in A$, if $s \neq s'$, we can never reach $s$, thus it is not reversible, and if $s = s'$ ($\triangle$ has no effects on $s$), it degenerates to CYK. Hence, we prove PTIME membership for (general) $\{\langle s, tn\rangle\}$-reversibility of $\triangle$ under $\rightarrow$. □

An overview of the proven complexity results for TOHTN reversibility is provided in Table 1.

## Reachability

Despite the fact that the (action) reversibility has been widely researched in the past, to our best knowledge, few works explicitly emphasize the natural link between the reversibility and reachability. We attribute this to the fact that the reversibility has been defined mostly as the existence of a *solution* that, when applied, results in a desired ("goal") state. We deliberately use an ambiguous notion of goal here, as we want to point out that usually the to-be-reversed state has been used as a new "goal" and then the methods relied on (traditional) goal plan existence. However, in HTN setting, this reasoning is no longer applicable, as plan existence and reachability are not straightforwardly interchangeable (due to the solution criterion of standard HTN formulation).

| $\triangleright^*$ | $\triangle$ | $s$-focused | $tn$-focused | $s$-$tn$-focused |
|---|---|---|---|---|
| $\rightsquigarrow^*$ | $p \in A$ | EXPTIME-complete [Thm. 3, Cor. 1] | EXPTIME-complete [Thm. 2, Cor. 1] | EXPTIME-complete [Thm. 1, Cor. 1] |
| | $m \in M$ | $\mathcal{O}(1)$, triv. rev. [Obs. 1] | EXPTIME-complete [Thm. 2, Cor. 1] | EXPTIME-complete [Thm. 1, Cor. 1] |
| $\mapsto^*$ | $p \in A$ | PTIME [Pro. 1] | $\mathcal{O}(1)$, triv. irre. [Obs. 2] | $\mathcal{O}(1)$, triv. irre. [Obs. 2] |
| | $m \in M$ | $\mathcal{O}(1)$, triv. rev. [Obs. 1] | PTIME [Pro. 2] | PTIME [Pro. 3] |
| $\rightarrow^*$ | $p \in A$ | PTIME, irre. iff $s \neq \gamma(s,p)$ [Obs. 3] | PTIME [Pro. 4] | PTIME, tr. irre. if $s \neq \gamma(s,p)$ [Pro. 4] |
| | $m \in M$ | $\mathcal{O}(1)$, triv. rev. [Obs. 1] | PTIME [Pro. 4] | PTIME [Pro. 4] |

Table 1: Complexity landscape for reversibility problems. Rows indicate the progression type (unrestricted $\rightsquigarrow^*$, application-only $\mapsto^*$, decomposition-only $\rightarrow^*$) subdivided by the task type (primitive task or method), and columns indicate the focus (state, task network, or both).

Clearly, reversibility inherently contains reachability. Consequently, as one can use reachability to decide the reversibility, any hardness of reversibility applies to reachability, and any membership of reachability implies the membership of reversibility. Intuitively, to decide reversibility, one needs to show that the configuration before the application of the primitive task or method (or of an action, in the case of non-hierarchical planning) can be reached.

**Theorem 4.** *If a decision problem of an arbitrary-focused $\triangleright$-reachability is in $\mathcal{C} \supseteq$ PTIME that is closed under polynomial-time Turing reductions, then the decision problem of the corresponding identically-focused $ST^*$-reversibility of any $\triangle \in A \cup M$ under $\triangleright$ is in $\mathcal{C}$.*

*Proof (Sketch).* In order to decide reversibility, we need to check that we have reached the state-task network pair in which the method or the primitive task has been progressed. To do that, for each state-task network pair, we invoke a procedure to solve the reachability. The primitive task or the method is reversible if and only if for all pairs, the starting state-task network pair is reachable, meaning we $|ST^*|$-times call the $C$-time or $C$-space procedure (in the case of a space class, we reuse the space). □

We note that the closeness of C is not overly restrictive, as standard space and deterministic time classes satisfy it.

**Theorem 5.** *Let $\mathcal{C}$ be an arbitrary complexity class and $ST^* \subseteq 2^F \times T(\mathcal{D})^*$ be a set of state-task network pairs such that $|ST^*| = 1$. If a decision problem of an arbitrary-focused $ST^*$-reversibility of any $\triangle \in A \cup M$ under progression class $\triangleright$ is $\mathcal{C}$-hard, the decision problem of the corresponding identically-focused $\triangleright$-reachability is $\mathcal{C}$-hard.*

*Proof (Sketch).* As $|ST^*| = 1$, we reduce the reversibility to reachability, as the primitive task or method is reversible if and only if the starting $\langle s, tn \rangle$ pair is reachable. □

**Membership** On the other hand, proving the membership of reachability in HTN is not that straightforward. In classical planning (or similarly in FOND planning), reachability and goal existence are essentially the same notion by the definition, as a plan to a (specific) goal exists if and only if a goal is reachable. The introduction of an additional dimension causes a discrepancy in the definitions of problem solvability (which typically cares about the emptiness of the task network independently of the state) and (full node) reachability (which should intuitively ask "is a particular state-task network configuration reachable?"). Observe that the standard HTN plan existence has no explicit mechanism to "ask" that a particular state is reached, and since the solvability criterion requires a progression toward an empty task network, nor has an explicit mechanism to require a specific task network other than an empty one.

All that said, while HTN planning can be understood as a restriction or filter on plans of classical planning, the hierarchy allows us to use many "modeling/encoding" tricks, as the formalism is strictly more expressive than classical planning. To prove EXPTIME membership of state-task-network-focused progression-reachability, we put forward a reduction to TOHTN plan existence.

The idea behind the proof is to construct a TOHTN planning problem having a solution if and only if $\langle s', tn' \rangle$ can be achieved from $\langle s, tn \rangle$. This is done by forcing any solution of the constructed problem to reach $s'$ and verifying that exactly $tn'$ is "available". The first is ensured by the introduction of a new "blocking" primitive task name $p^{s'}$ and of corresponding action $a^{s'}$, which requires (exactly and only) the facts of $s'$ in preconditions and has no effects. The latter is done by making all former primitive tasks compound in the new domain, allowing to choose whether one wants to check their presence in the task network or to use their actual (former) effects. If the presence of all required tasks of $tn'$ is checked in the order and without any interfering primitive action applications and no other tasks remain (except for the second blocking action checking the presence of the last task of $tn'$), we have verified that $\langle s', tn' \rangle$ can be achieved by $\langle s, tn \rangle$ in $\mathcal{D}$.

**Theorem 6.** REACH($\rightsquigarrow$) *is in* EXPTIME.

*Proof.* We reduce it to TOHTN planning problem.

Let $k = |tn'|$. For $k = 0$, the reachability problem is the TOHTN plan existence problem $\langle \mathcal{D}', s, tn, s' \rangle$. For $k > 0$, a different planning problem is constructed. We add a set of fresh facts $\Phi$, one for each task $tn'_{[i]}$ of $tn'$, i.e. $\Phi = \{f_i \mid i \in \mathbb{N}, 1 \leq i \leq k\}$ such that $\Phi \cap F = \emptyset$ and $|\Phi| = k$; and a fresh phase-controlling fact $\psi$ (with $\psi \notin \Phi \cup F$). The set of compound task names is extended by the primitive task names from the domain $\mathcal{D}$. The set of primitive task names is a replaced with $\{p' \mid p \in A\} \cup \{p^g_i \mid i \in \mathbb{N}, 1 \leq$

| $\triangleright^*$ | $s$-focused | $tn$-focused | $s$-$tn$-focused |
|---|---|---|---|
| $\rightsquigarrow^*$ | EXPTIME-complete [Thms. 5, 3, 8] | EXPTIME-complete [Thms. 5, 2, 7] | in EXPTIME-complete [Thms. 5, 1, 6] |
| $\mapsto^*$ | PTIME [Obs. 4] | PTIME [Obs. 4] | PTIME [Obs. 4] |
| $\rightarrow^*$ | $\mathcal{O}(1)$, reach. iff $s = s'$ [Obs. 4] | PTIME [Obs. 4] | if $s \neq s'$ triv. unreach. else PTIME [Obs. 4] |

Table 2: Complexity landscape for reachability problems. Rows indicate the progression type (unrestricted $\rightsquigarrow^*$, application-only $\mapsto^*$, decomposition-only $\rightarrow^*$), and columns indicate the focus (state, task network, or both).

$i \leq k\} \cup \{p^G\}$, where $p'$ is a fresh task name ($p' \notin T(\mathcal{D})$ nor it coincides with any other fresh task name) with the same corresponding action $a$ except it also requires $\psi$ and deletes all facts of $\Phi$, and $p_i^g$ is a fresh task name with a corresponding action $a_i^g$ requiring $f_{i-1}$, deleting $\psi$ and $f_{i-1}$, and adding $f_i$, if $i > 1$; and requiring exactly (and only[2]) facts of $s'$ and adding $f_1$, if $i = 1$. Primitive task name $p^G$ is a fresh name with a corresponding action that has no effects and requires just the fact $f_k$. All former methods are kept. A set of methods $\{(p, p') \mid p \in A\}$ is added, allowing to change the newly compound tasks into former actions (with extended delete effects). The last set of methods allows the verification of the reachability of $\langle s', tn' \rangle$. For any task $tn'_{[i]}$ of $tn'$, we add a method $(tn'_{[i]}, p_i^g)$.

The planning problem $\langle \langle F \cup \Phi \cup \{\psi\}, \{p' \mid p \in A\} \cup \{p_i^g \mid i \in N, 1 \leq i \leq k\} \cup \{p^G\}, T(\mathcal{D}), M \cup \{(p, p') \mid p \in A\} \cup \{(tn'_{[i]}, p_i^g) \mid i \in N, 1 \leq i \leq k\} \rangle, s, tnp^G, s' \rangle$ has a solution if and only $\langle s', tn' \rangle$ is reachable from $\langle s, tn \rangle$ in $\mathcal{D}$.

If there is a solution to the planning problem, $p^G$ has to be applied. To apply $p^G$, $f_k$ had to hold. The only way to achieve the fact $f_k$ is to apply $p_k^g$ right before $p^G$ (no other action that does not delete $f_k$ is applicable; thus any interleaving task needs to decompose into an empty task network). To apply $p_k^g$, $f_{k-1}$ had to hold. Inductively, $s'$ had to hold and no other primitive task interleaved the application of the actions $p_i^g$. As $p_i^g \notin tn$ and as the only way how to obtain $p_i^g$ from $tn$ is to use method $(tn'_{[i]}, p_i^g)$, we prove that $\langle s', tn' \rangle$ has been reached from $\langle s, tn \rangle$.

If $\langle s', tn' \rangle$ is reachable from $\langle s, tn \rangle$ in $\mathcal{D}$, there exists a $\rightsquigarrow^*$ such that $\langle s, tn \rangle \rightsquigarrow^* \langle s', tn' \rangle$. That means $\langle s', tn' p^G \rangle$ is reachable in the constructed domain from $\langle s, tnp^G \rangle$. But then methods $(tn'_{[i]}, p_i^g)$ can be used to substitute each element $tn'_{[i]}$ with goal-achieving primitive task $p_i^g$ with no interleaving primitive tasks from the former domain, thus $tn' p^G$ can be progressed away without touching $s'$ and the constructed constructed planning problem is solvable.

Clearly, the construction is polynomial, concluding the proof of EXPTIME membership. □

Notice, that the compilation allows us to decide progression-reachability with (standard) TOHTN planners.

**Theorem 7.** TN-FOCUSED-REACH($\rightsquigarrow$) *is in* EXPTIME.

*Proof.* Given a domain $\mathcal{D} = \langle F, A, C, M \rangle$, there are at most exponentially many states $s \in 2^F$. Since general reachability is in EXPTIME (see Theorem 6), the task-network-focused reachability can be decided in EXPTIME. □

[2]This can be done by introducing *"not-"* facts in linear time.

**Theorem 8.** S-FOCUSED-REACH($\rightsquigarrow$) *is in* EXPTIME.

*Proof (Sketch).* A similar construction to the proof of Theorem 6 can be utilized. Instead of adding primitive task names with corresponding actions that verify the presence of $tn'$, we allow all tasks of the former domain to decompose into a "blocker" action requiring exactly[2] $s'$ and with no effects. Let $\mathcal{D}'$ be a constructed domain. Then, a planning task $\langle \mathcal{D}', s, tnp^G \rangle$ has a solution if and only if $\langle s, tn \rangle \rightsquigarrow^* \langle s', tn'' \rangle$ where $tn''$ is some task network over $T(\mathcal{D})$. If there is a solution to the planning problem, $s'$ has to be reached. If there is a pair $\langle s', tn'' \rangle$ such that $\langle s, tn \rangle \rightsquigarrow^* \langle s', tn'' \rangle$ where $tn''$ is some task network over $T(\mathcal{D})$, then, if $tn''$ contains some primitive task, there exists a different pair where the primitive task has not been decomposed into the primitive task. Thus, without loss of generality, assume $tn''$ contains compound tasks only. Then all of them can be decomposed into "blocker" variables that are executable in $s'$; thus, the planning task has a solution. □

We omit proofs of tractable variants of reachability, as the underlying logic was already demonstrated in the proofs for the tractable cases of reversibility.

**Observation 4.** *For decomposition- and application-reachability problems, the* PTIME *membership follows directly from the reversibility proofs (Propositions 1 to 4) and relevant observations.*

See Table 2 for a summarization of reachability results.

**Implication for Reversibility** As already argued and noted by Theorem 4 before, all the membership results for reachability translate to the corresponding reversibility problem as well (by the definition).

**Corollary 1.** *By Theorems 4 and 6 to 8, all investigated notions of reversibility are in* EXPTIME.

## Conclusion

This paper studied *reversibility* and *reachability* in (TO)HTN planning. Although reachability is well studied in classical planning, to our best knowledge, this is the first study in (TO)HTN planning. Similarly, this paper is the first to introduce the concept of (primitive) task and method reversibility in TOHTN planning. We classified both concepts (of reversibility and reachability) into several classes depending on the allowed progression rules (applying primitive tasks or decomposing compound tasks) or the desired target (a state of the environment, a state of the network, or both). We provided the complexity results for all introduced classes of reachability and reversibility that range from constant time to EXPTIME-complete (see Tables 1 and 2).

## Acknowledgments

This research is supported by the Czech Science Foundation (project no. 24-13337L) and by the Grant Agency of Czech Technical University (project no. SGS24/140/OHK3/3T/13). Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

## References

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight Bounds for HTN Planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 7–15. AAAI Press.

Barták, R.; Ondrčková, S.; Behnke, G.; and Bercher, P. 2021. On the Verification of Totally-Ordered HTN Plans. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI 2021)*, 263–267. IEEE.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT - Totally-Ordered Hierarchical Planning Through SAT. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 6110–6118. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 6267–6275. IJCAI Organization.

Bercher, P.; Sreedharan, S.; and Vallati, M. 2025. A Survey on Model Repair in AI Planning. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2025)*, 10371–10380. IJCAI Organization.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 12–21. AAAI Press.

Camacho, A.; Muise, C.; and McIlraith, S. 2016. From FOND to Robust Probabilistic Planning: Computing Compact Policies that Bypass Avoidable Deadends. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 65–69. AAAI Press.

Chrpa, L.; Faber, W.; and Morak, M. 2021. Universal and Uniform Action Reversibility. In *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 651–654. IJCAI Organization.

Chrpa, L.; and Karpas, E. 2024. On Verifying Linear Execution Strategies in Planning Against Nature. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 86–94. AAAI Press.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012a. Determining Redundant Actions in Sequential Plans. In *Proceedings of the 2012 IEEE Twenty-Fourth International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, 484–491. IEEE.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012b. Optimizing Plans through Analysis of Action Dependencies and Independencies. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 338–342. AAAI Press.

Chrpa, L.; Pilát, M.; and Med, J. 2021. On Eventual Applicability of Plans in Dynamic Environments with Cyclic Phenomena. In *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 184–193. IJCAI Organization.

Cserna, B.; Doyle, W.; Ramsdell, J.; and Ruml, W. 2018. Avoiding Dead Ends in Real-Time Heuristic Search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 1306–1313. AAAI Press.

Daum, J.; Torralba, Á.; Hoffmann, J.; Haslum, P.; and Weber, I. 2016. Practical Undoability Checking via Contingent Planning. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 106–114. AAAI Press.

Eiter, T.; Erdem, E.; and Faber, W. 2007. On reversing actions: Algorithms and complexity. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 336–341. Morgan Kaufmann Publishers Inc.

Eiter, T.; Erdem, E.; and Faber, W. 2008. Undoing the effects of action sequences. *Journal of Applied Logic*, 6(3): 380–415.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)*, 1123–1128. AAAI Press / The MIT Press.

Faber, W.; and Morak, M. 2025. Encoding action reversibility in planning using quantified ASP and bule. In *Proceedings of the Nineteenth European Conference on Logics in Artificial Intelligence (JELIA 2025)*, Lecture notes in computer science, 327–342. Springer.

Faber, W.; Morak, M.; and Chrpa, L. 2022. Determining Action Reversibility in STRIPS Using Answer Set Programming with Quantifiers. In *Practical Aspects of Declarative Languages*, volume 13165, 42–56. Springer International Publishing. Series Title: Lecture Notes in Computer Science.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. New York, NY: Cambridge University Press. ISBN 978-1-107-03727-4.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2007. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley. ISBN 978-0-321-47617-3.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *The Proceedings of Twenty-First European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 447–452. IOS Press. Holler_Behnke_Bercher_Biundo_2014.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2018. Plan and Goal Recognition as HTN Planning. In *Proceedings of the Thirtieth IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2018)*, 466–473. IEEE.

Höller, D.; and Bercher, P. 2021. Landmark Generation in HTN Planning. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 13, 11826–11834. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020a. HTN Plan Repair via Model Transformation. In *KI 2020: Advances in Artificial Intelligence*, volume 12325, 88–101. Springer International Publishing. Series Title: Lecture Notes in Computer Science.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020b. HTN Planning as Heuristic Progression Search. *Journal of Artificial Intelligence Research*, 67: 835–880.

Med, J.; and Chrpa, L. 2022. On Speeding Up Methods for Identifying Redundant Actions in Plans. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 252–260. AAAI Press.

Med, J.; Chrpa, L.; Morak, M.; and Faber, W. 2024. Weak and Strong Reversibility of Non-deterministic Actions: Universality and Uniformity. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 369–377. AAAI Press.

Med, J.; Morak, M.; Chrpa, L.; and Faber, W. 2025. Nondeterministic Action Reversibility: Complexity Results. In *Proceedings of the Twenty-Second International Conference on Principles of Knowledge Representation and Reasoning (KR 2025)*, 462–466. IJCAI Organization.

Morak, M.; Chrpa, L.; Faber, W.; and Fiser, D. 2020. On the Reversibility of Actions in Planning. In *Proceedings of the Seventeenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)*, 652–661. IJCAI Organization.

Olz, C.; and Bercher, P. 2023. A Look-Ahead Technique for Search-Based HTN Planning: Reducing the Branching Factor by Identifying Inevitable Task Refinements. In *Proceedings of the Sixteenth International Symposium on Combinatorial Search (SoCS 2023)*, 65–73. AAAI Press.

Patra, S.; Mason, J.; Ghallab, M.; Nau, D.; and Traverso, P. 2021. Deliberative acting, planning and learning with hierarchical operational models. *Artificial Intelligence*, 299.

Putrich, V. S.; Meneguzzi, F.; and Pereira, A. G. 2025. Landmark Generation in HTN Planning Revisited. In *Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)*, 228–235. AAAI Press.

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

Weber, I.; Wada, H.; Fekete, A. D.; Liu, A.; and Bass, L. 2012. Automatic Undo for Cloud Management via AI Planning. In *Proceedings of the Eighth Workshop on Hot Topics in System Dependability (HotDep 2012)*. USENIX Association.

Weber, I.; Wada, H.; Fekete, A. D.; Liu, A.; and Bass, L. 2013. Supporting Undoability in Systems Operations. In *Proceedings of the Twenty-Seventh Large Installation System Administration Conference (LISA 2013)*, 75–88. USENIX Association.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 352–352. AAAI Press.

Younger, D. H. 1967. Recognition and parsing of context-free languages in time $n^3$. *Information and Control*, 10(2): 189–208.

Yousefi, M.; Schmautz, M.; Haslum, P.; and Bercher, P. 2025. How Good is Perfect? On the Incompleteness of A* for Total-Order HTN Planning. In *Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)*, 112–120. AAAI Press.

Zaidins, P.; Goldman, R. P.; Kuter, U.; Nau, D.; and Roberts, M. 2025. HTN Plan Repair Algorithms Compared: Strengths and Weaknesses of Different Methods. In *Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)*, 297–305. AAAI Press.