Hierarchical Planning: Relating Task and Goal Decomposition with Task Sharing

Ron Alford,1 Vikas Shivashankar,2 Mark Roberts,3 Jeremy Frank,4 David W. Aha,5
1MITRE; McLean, VA | ralford@mitre.org
2Knexus Research Corporation; National Harbor, MD | vikas.shivashankar@knexusresearch.com
3NRC Postdoctoral Fellow; Naval Research Laboratory; Washington, DC | mark.roberts.ctr@nrl.navy.mil
4NASA Ames Research Center; Moffett Field, CA | jeremy.d.frank@nasa.gov
5Navy Center for Applied Research in AI;
Naval Research Laboratory; Washington, DC | david.aha@nrl.navy.mil

Abstract

Considerable work has focused on enhancing the semantics of Hierarchical Task Networks (HTNs) in order to advance the state-of-the-art in hierarchical planning. For instance, the Hierarchical Goal Network (HGN) formalism operates over a hierarchy of goals to facilitate tighter integration of compositional planning with classical planning. Another example is the Action Notation Markup Language (ANML) which adds aspects of generative planning and task-sharing to the standard HTN semantics.

The aim of this work is to formally analyze the effects of these modifications to HTN semantics on the computational complexity and expressivity of HTN planning. To facilitate analysis, we unify goal and task planning into Goal-Task Network (GTN) planning. GTN models use HTN and HGN constructs, but have a solution-preserving mapping back to HTN planning. We then show theoretical results that provide new insights into both the expressivity as well as computational complexity of GTN planning under a number of different semantics. Our work lays a firm footing to clarify exact semantics for recent planners based on ANML, HGNs, and similar hierarchical languages.

1 Introduction

Hierarchical Task Network (HTN) planning [Erol et al., 1994] is a task planning formalism that is widely used in real-world applications [Nau et al., 2005]. HTN planners use methods as recipes to recursively decompose tasks into primitive action sequences. We focus on two hierarchical formalisms, the Action Notation Markup Language (ANML) [Smith et al., 2008] and Hierarchical Goal Network (HGN) planning [Shivashankar et al., 2012], and their relationship with the more widely known HTN planning. All these formalisms now have systems that implement subsets of their features, yet little research examines the differences between them, either in terms of their computational complexity or their semantics.

HGN planning bridges classical planning and totally ordered task networks by operating over sequences of goals with methods that decompose goals with further subgoals. By attaching goals to methods, HGN planning adds explicit semantics to methods, making it easy to adapt classical heuristics to HGN planning [Shivashankar et al., 2016].

The Action Notation Markup Language (ANML) is a temporal planning framework with a hierarchical component [Smith et al., 2008], designed to address common challenges faced when modeling problems that require sophisticated temporal or resource reasoning within a single language that combines generative and hierarchical task planning [Smith and Cushing, 2008]; an essential feature of ANML is its ability to embed complex procedures as tasks. A translation from ANML to a variant of PDDL [Bonasso et al., 2009] and the planning system FAPE [Dvorak et al., 2014] have implemented subsets of ANML.

One of the unique features of ANML is that it allows subtasks to be shared between multiple tasks. For example, let us assume an ANML planner is considering two tasks $t_1$ and $t_2$ that are unordered with respect to one another which de-
compose into the same setup task; so, \( t_1 \) decomposes into \( \langle \text{setup}, a_1 \rangle \) (read "do setup, then do \( a_1 \)"), while \( t_2 \) decomposes into \( \langle \text{setup}, a_2 \rangle \). According to standard HTN planning semantics, the only solution would involve two setup tasks. ANML, however, also allows for sharing the same setup task across both task-decompositions, and can therefore generate a solution with a single setup task.

Traditional HTN planners strictly follow the method hierarchy when composing a plan. This rigidity is a liability in dynamic environments when actions may fail and the plan needs to be repaired [Ayan et al., 2007]. Task Insertion semantics modifies HTN semantics to allow planners to insert actions whenever they are useful, regardless of methods [Geier and Bercher, 2011]. Task insertion is also used in the HTN literature to capture plan repair and partial method knowledge. These semantics are also implemented as part of HGNs [Shivashankar et al., 2013] and are explicitly supported in ANML. It is not known how task insertion interacts with task sharing.

**Contributions.** The central aim of this paper is to unify these various families of hierarchical planning and their semantics as summarized in Figure 1. In particular,

- **A Unified Formalism:** We define Goal-Task Network (GTN) planning that models partially ordered HTN planning and extends HGN planning with partially-ordered sets of subgoals. We present standard semantics for GTN that extends HTN semantics with goals. We then extend these semantics with the addition of task insertion semantics, denoted GTN\textsubscript{I}, with the addition of task sharing GTN\textsubscript{S}, and finally, with the combined semantics of GTN\textsubscript{IS}.
- **Expressivity Results:** We show that every GTN problem with any combination of the above semantics can be translated in a solution preserving manner to HTN planning with standard semantics, which is in general semi-decidable.
- **Membership Results:** For certain classes of GTN planning, we improve the semi-decidable upper bound by proving that HGN\textsubscript{I} is in PSPACE, while GTN\textsubscript{IS} and GTN\textsubscript{I} planning are in NEXPTIME.

2 A Unified Hierarchical Planning Formalism

To precisely describe the relationship between HTN and HGN planning, we model both in a single framework called Goal-Task Network (GTN) planning\(^1\). GTN planning has already been extremely useful in formalizing goal reasoning [Roberts et al., 2016]. We augment the set-theoretic notation of Geier and Bercher [2011] with goal decomposition from HGN planning [Shivashankar et al., 2012] and with SHOP2-style method preconditions [Nau et al., 2003]. After formalizing GTNs, we clarify their operations before discussing GTN problems and their solutions.

---

\(^1\)GTN was partly inspired by conversations with Ghallab et al. following their paper [Ghallab et al., 2014] and upcoming book [Ghallab et al., in press] wherein they argued for use of a hybrid task-planning formalism for Planning and Acting.

2.1 Preliminaries

Let \( \mathcal{L} \) be a propositional language and \( \mathcal{T} \) be a set of task names represented as propositional symbols not appearing in \( \mathcal{L} \) with \( \mathcal{L} \cap \mathcal{T} = \emptyset \), and let \( O \) and \( C \) be a partition of \( \mathcal{T} \) \((O \cup C = \mathcal{T}, O \cap C = \emptyset)\). \( O \) will correspond to primitive tasks, or actions that can be executed directly, while \( C \) represents Compound or non-primitive tasks which need to be recursively decomposed into primitive tasks before they can be executed.

Then HTN planning is over partially-ordered multisets of task names in \( \mathcal{T} \) representing activities to accomplish, while HGN planning is over totally-ordered subgoals in \( \mathcal{L} \) which must be met in sequence. GTNs elegantly models both, and in addition extends the scope of HGN planning to allow for partially-ordered sets of subgoals.

2.2 Goal-Task Networks (GTNs)

A goal-task network is a tuple \((I, \prec, \alpha)\) where \( I \) is a set of instance symbols which are placeholders for task names and goals, \( \prec \subseteq I \times I \) is a partial order on \( I \), and \( \alpha : I \rightarrow \mathcal{L} \cup \mathcal{T} \) maps each instance symbol to a task or task name.

Some symbol instances are special because they occur first or last in a network. An instance symbol \( i \) is unconstrained if no symbols are constrained before it \((\forall i', i'' : i' \neq i'' \Rightarrow i') \) or last if it is constrained after all other symbols \((\exists i' : i' \neq i) \). We refer to \( i \) as a task if \( \alpha(i) \in \mathcal{T} \) and as a subgoal if \( \alpha(i) \in \mathcal{L} \); recall that \( \mathcal{L} \) and \( \mathcal{T} \) are disjoint.

Two goal-task networks, \((I_1, \prec_1, \alpha_1)\) and \((I_2, \prec_2, \alpha_2)\), are isomorphic if there exists a bijection \( f : I_1 \rightarrow I_2 \) such that: (1) \((i, i') \in \prec_1 \iff (f(i), f(i')) \in \prec_2\) and (2) \(\alpha_1(i) = \alpha_2(f(i))\) for all \(i \in I_1\).

**Methods.** We distinguish the methods for GTNs by the kind of symbol they decompose. A task method \( m_i \) is a tuple \((n, \chi, gtn)\) where \( n \subseteq C \) is a non-primitive task name, \( \chi \in \mathcal{L} \) is \( m_i \)’s precondition, and \( gtn \) is a goal-task network over \( \mathcal{L} \) and \( \mathcal{T} \). A task method is relevant to a task \( i \) in a goal-task network \((I, \prec, \alpha)\) if \( n = \alpha(i) \). The method \( m_i \) is a specific decomposition of a task \( n \) into a partially-ordered set of subtasks and subgoals, and there may be many such methods.

A goal method \( m_g \), similarly, is a tuple \((g, \chi, gtn)\) where \( g, \chi \in \mathcal{L} \) are the goal and precondition of \( m_g \) and \( gtn \) is a goal-task network. A goal method is relevant to a subgoal \( i \) in a goal-task network \((I, \prec, \alpha)\) if at least one literal in the negation-normal form (NNF) of \( g \) matches a literal in the NNF of \( \alpha(i) \) (e.g., accomplishing \( g \) ensures that at least part of \( \alpha(i) \) is true). By convention, the goal-task network \( gtn = ((I, \prec, \alpha) \text{ has a last instance symbol } i \in I \text{ with } \alpha(i) = g \text{ to ensure that } m_g \text{ accomplishes its own goal.})\)

Task and goal methods are respectively identical to methods in HTN and HGN planning.

**Operators.** An operator \( o \) is a tuple \((n, \chi, e)\) where \( n \subseteq O \) is a primitive task name (assumed unique to \( o \), \( \chi \) is a propositional formula in \( \mathcal{L} \) called \( o \)’s precondition (or \( \text{prec}(o) \)), and \( e \) is a set of literals from \( \mathcal{L} \) called \( o \)’s effects. We refer to the set of positive literals in \( e \) as \( \text{add}(o) \) and the negated literals as \( \text{del}(o) \). An operator is relevant to primitive task \( i_0 \) if \( n = \alpha(i_0) \) and to a subgoal \( i_g \) if the effects of \( o \) contain a matching literal from the NNF of \( \alpha(i_g) \).

---
States. A state \( s \in \mathcal{L}_{\text{sym}} \) is any subset of \( \mathcal{L}_{\text{sym}} \), the propositional symbols of \( \mathcal{C} \), the set of all states is \( 2^{\mathcal{L}_{\text{sym}}} \). A set of operators \( \mathcal{O} \) forms a transition (partial) function \( \gamma : 2^{\mathcal{L}_{\text{sym}}} \times \mathcal{O} \rightarrow 2^{\mathcal{L}_{\text{sym}}} \) as follows: \( \gamma (s, o) \) is defined iff \( s \models \text{prec}(o) \) (the precondition of \( o \) holds in \( s \)), and \( \gamma (s, o) = (s \setminus \text{del}(o)) \cup \text{add}(o) \).

2.3 GTN Nodes and Node Operations

A node is a (state, goal-task network) pair \( N = (s, gtn) \). GTN planning allows four operations on nodes, described below.

We use the term progression as an umbrella term for the various GTN node operations. We write \( N \rightarrow N' \) if any of the operations can make that transition between nodes. For progression sequences from \( N \) to \( N'' \), we write \( N \rightarrow N' \rightarrow N'' \).

Operator application of an operator \( o \) to a node \( (s, gtn) \), with \( gtn = (I, \prec, \alpha) \), written as \( (s, gtn) \xrightarrow{\gamma} (s', gtn') \), is defined if \( s \models \text{prec}(o) \) and \( o \) is relevant to an unconstrained instance symbol \( i \) in \( gtn \). If \( i \) is a primitive task with task name \( m \), then this corresponds to primitive task application in HTNs; the result of application is given by \( s' = \gamma (s, o) \) and \( gtn' = (I \setminus \{i\}, \{(i_1, i_2) \in < | i_1 \neq i \}, \{(i', n) \in \alpha | i' \neq i \}) \), transitioning the state and removing the instance symbol from the network. If \( i \) is a goal task relevant task instead, this corresponds to primitive task application in HGNs; in this case, \( gtn' = gtn \), and the subgoal remains while the state changes.

Goal release for an unconstrained subgoal \( i \), written \( (s, gtn) \xrightarrow{G} (s, gtn') \), is defined whenever \( s \models \alpha(i_\gamma) \). Goal release can remove a subgoal whenever it is satisfied by \( s \), and \( gtn' \) is given by \( (I \setminus \{i\}, \{(i_1, i_2) \in < | i_1 \neq i \}, \{(i', n) \in \alpha | i' \neq i \}) \), just as in operator application.

Task decomposition for an unconstrained task \( i \) by a relevant task method \( m = (c, \chi, gtn_m) \), written \( (s, gtn) \xrightarrow{D} (s, gtn') \), is defined whenever \( s \models \chi \). Task decomposition expands \( i \) in \( gtn \), replacing it with the network \( gtn_m \). Let \( (I, \prec, \alpha) = gtn_m \) and \( (I_m, \prec_m, \alpha_m) = gtn_m \), assuming without loss of generality that \( I \cap I_m = \emptyset \). Then the result of task decomposition, \( gtn' = (I', \prec', \alpha') \) is given by:

\[
I' := (I \setminus \{i\}) \cup I_m
\]

\[
\prec' := \{(i, i') \in < | i \neq i' \} \cup \prec_m
\]

\[
\cup \{(i_1, i_2) \in I_m \times I | (i, i_2) \in < \}
\]

\[
\alpha' := \{(i, n) \in \alpha | i \in I' \} \cup \alpha_m
\]

Goal decomposition for an unconstrained subgoal \( i \) by a relevant goal method \( m = (g_m, \chi, gtn_m) \), also written \( (s, gtn) \xrightarrow{D} (s, gtn') \), is defined whenever \( s \models \chi \). Goal decomposition prepends \( i \) with \( gtn_m \), leaving the subgoal in place, so that \( gtn' \) is given by

\[
(I \cup I_m, \prec \cup \prec_m \cup \{(i_m, i) \in I_m \times \{i\} \}, \alpha \cup \alpha_m)
\]

3 GTN Planning Problems and Solutions

A problem is a tuple \( P = (\mathcal{L}, \mathcal{O}, \mathcal{M}, N_0) \) where \( \mathcal{L} \) is propositional language for defining the set of operators (\( \mathcal{O} \)), the set of goal and task methods (\( \mathcal{M} \)), and \( N_0 \) is the initial node consisting of the initial goal-task network \( gtn_0 \), and the initial state \( s_0 \). \( O \) and \( C \) are implicitly defined by \( \mathcal{O} \) and \( \mathcal{M} \). Nodes are isomorphic if their goal-task networks are isomorphic and their states are identical.

We say \( P \) is solvable under GTN semantics iff there is a progression \( N_0 \rightarrow^*_P N_k \) where \( N_k = (s_k, gtn_k) \), \( s_k \) is any state, \( gtn_k \) is the empty network. The subsequence of operator applications of that progression is a plan for \( P \).

Whenever all networks (method and initial) in a problem \( P \) contain only tasks, we call it an HTN problem since goal release and goal decomposition are irrelevant, and so the semantics are identical to HTN planning under progression [Alford et al., 2012]. Whenever all networks contain only goals, we call it an HGN problem, since the semantics are identical to HGN planning with a straightforward extension to partially-ordered networks [Shivashankar et al., 2012].

3.1 Task-Insertion Semantics

Formally, task insertion of an operator \( o \) is defined whenever \( o \)’s precondition is supported by the state, and is written \( (s, gtn) \xrightarrow{\theta_1} (s', gtn) \), where \( s' = \gamma (s, o) \). Note that this combines two operations, insertion of \( o \) and operator application of \( o \).

If a node \( (s', gtn') \) is reachable from another node \( (s, gtn) \) via any sequence of progressions and insertions, we write \( (s, gtn) \rightarrow^*_P (s', gtn') \). We say that a problem \( P \) is solvable under GTN semantics with task insertion (GTN1 semantics) iff \( (s_0, gtn_0) \rightarrow^*_P (s, gtn) \) where \( s \) is any state and \( gtn_0 \) is the empty goal-task network. HTN1 planning is \( \text{NEXPSPACE} \)-complete [Alford et al., 2015b].

3.2 Task-Sharing Semantics

We formalize task sharing here using a task merging operation that takes two task symbols in the goal-task network that map to the same task name and merges them into one task symbol. Formally, task merging of two unconstrained tasks \( i_1, i_2 \) in a network \( gtn = (I, \prec, \alpha) \) is applicable whenever the tasks share a task name \( (\alpha (i_1) = \alpha (i_2)) \). The operation, written \( (s, gtn) \xrightarrow{i_1, i_2} (s, gtn') \), is given by

\[
\prec' := \{(i, i') \in < | i \neq i' \} \cup \{(i_1, i_2) \in I \times I | (i, i_2) \in < \}
\]

\[
\alpha' := \{(i, n) \in \alpha | i \neq i' \}
\]

We say a problem \( P \) with initial node \( (s, gtn) \) is solvable under GTN semantics with task sharing (GTN5) if there is any sequence of progression and merging to a node \( (s', gtn') \), written \( (s, gtn) \rightarrow^*_P (s', gtn') \). Solvability under both task insertion and sharing (GTN9 semantics) is defined similarly, with the sequence denoted as \( (s, gtn) \rightarrow^*_P (s', gtn') \).

4 Plan-preserving translations

We can study the computational complexity of GTN planning by reductions, which are transformations between problems and/or formalisms. In this section, we use a special kind of reduction we call plan-preserving translations, which preserve the solution set of the problem across translation. For example, we can trivially use a GTN planner to solve a GTN1.
problem $P$ by adding a new task to the initial node along with methods that decompose it into any sequence of operators. Since this transformation preserves the set of plans, (1) an optimal plan from the GTN planner is an optimal plan $P$, and (2) when creating GTN$_2$ planners, the transformation is a procedure for using any admissible GTN heuristic as a GTN$_1$ heuristic. Further, we know that GTN planning is at least as expressive as GTN$_1$ planning, since the (possibly infinite) set of solutions for any problem under GTN$_1$ semantics can be exactly expressed as the solutions of the translated problem under GTN semantics.

**Same-semantics translations from GTNs to HTNs**

In this section, we provide a plan preserving translation from GTN problems to HTN problems that preserves plans under like semantics (GTN to HTN, GTN$_2$ to HTN with task insertion, etc.). This establishes that GTN planning is no harder in the general case than HTN planning across all the semantic variants we discuss.

**Construction 4.1.** Let $P$ be a problem. We iteratively construct an HTN problem $P'$ as follows:

Let $g$ be a subgoal appearing in the initial network or any method network. Let $t_g$ be a fresh task name, and let $m_{og} = (t_g, g, \text{gtn}_0g)$, which will act as the goal release operation for $t_g$. Let $N'_0$ and $M'$ be the $P$’s initial node and set of methods where each instance of $g$ in a network is replaced by $t_g$. Let $\mathcal{M}_Og$ be a set of methods of the form $m_{og} = (t_g, \text{true}, \text{gtn}_0g')$ for each operator $o$ relevant to $g$, where $\text{gtn}_0g'$ contains the task $o$ preceding the task $t_g$. Finally, let $\mathcal{M}_Mg$ be a set of methods of the form $m_{og} = (t_g, \chi, \text{gtn}_0g)$ for each goal method $(g', \chi, \text{gtn}) \in M'$ relevant to $g$, where $\text{gtn}_0g$ is $\text{gtn}$ with a new task for $t_g$ constrained to come after all other instance symbols. Then $P' = (L, O, M' \cup \mathcal{M}_Og \cup \mathcal{M}_Mg \cup \{m_{og}\}, N'_0)$. Iterate until no goal appears in a network, and remove all goal methods.

**Theorem 4.2.** For a problem $P$ with initial node $N_0$, use Construction 4.1 to construct an HTN problem $P'$. Then, under GTN semantics, a sequence of actions is a plan for $P$ iff it is a plan for $P'$. The same result holds under GTN$_S$ semantics.

**Proof.** We prove this theorem by showing that for each iteration of Construction 4.1, after replacing $g$ with $t_g$, $P$ and $P'$ have the same set of plans.

(⇒) Let $N_0 \xrightarrow{t_g} P$. $N_k$ be a derivation of a plan $p$ for $P$, where $N_1, \ldots, N_{k-1}$ are the intermediate nodes. Construct $N'_0 \xrightarrow{t_g} N'_k$ as follows: $N'_0$ is isomorphic to $N_0$ with $g$ replaced with $t_g$. Suppose $N'_i$ is isomorphic to $N_i$ with $g$ replaced with $t_g$. The following cases ensure the same for $N'_{i+1}$ and $N_{i+1}$:

1. If $N_i \xrightarrow{i,a} N_{i+1}$ is an application of $o$ to goal task $g$, then decompose the corresponding $i$ in $N'_i$ with $m_{og}$ and immediately apply $o$ ($N'_i \xrightarrow{i,m_{og}} D N'_i \xrightarrow{i,o} A N'_{i+1}$). If $N_i \xrightarrow{i,o} N_{i+1}$, then $g$ is satisfied in $N_i$’s state (identical to the state of $N'_i$), so decompose the corresponding $i$ from $N'_i$ with $m_{og}$, removing $i$, to get the corresponding $N'_{i+1}$. If $N_i \xrightarrow{i,m} D N_{i+1}$ corresponds to goal-method decomposition, then $N'_i \xrightarrow{i,m_{og}^D} D N'_i \xrightarrow{i+1}$ is applicable and produces the corresponding $N'_{i+1}$. Otherwise, the operation does not involve $g$, so apply it directly to $N'_i$. Since this preserves operator applications in the derivation, $p$ is also a plan for $P'$.

(⇐) Let $N'_0 \xrightarrow{t_g} P$. $N'_k$ be a derivation of a plan $p$ for $P'$. Note that decomposition with any $m_{og}$ can be push back in the derivation until it is immediately before the application of $o$. Then the inverse of the above procedure can produce a derivation of $p$ for $P$.

Since the set of GTN problems includes all HTN problems and HTN planning is semi-decidable [Erol et al., 1996], by the above theorem, GTN planning is also semi-decidable:

**Corollary 4.3.** Deciding whether a solution exists for a problem under GTN semantics is semi-decidable.

We know by Alford et al. [2015b] that HTN planning with task insertion is NEXPTIME-complete, so by the above, the same result extends to GTN$_S$ planning:

**Corollary 4.4.** Deciding whether a solution exists for a problem under GTN$_1$ semantics is NEXPTIME-complete.

We can also show that Construction 4.1 preserves plan sets under GTN$_S$ semantics:

**Theorem 4.5.** For a problem $P$, let $P'$ be the resulting HTN problem from Construction 4.1. Then, under GTN$_S$ semantics, a sequence of actions is a plan for $P$ iff it is a plan for $P'$. The same result holds under GTN$_{S5}$ semantics.

**Proof.** Again, we show that for each iteration of Construction 4.1, after removing $g$, $P$ and $P'$ have the same set of plans.

(⇒) Task merging does not involve goals, so given a derivation $N_0 \xrightarrow{t_g} P$. $N_k$ of a plan $p$ for $P$, extend the (⇒) procedure of Theorem 4.2 with merging so that $N_j \xrightarrow{i_1,i_2} M N_{j+1}$ maps to $N'_j \xrightarrow{i_1,i_2} M N'_{j+1}$.

(⇐) Once $g$ is replaced by $t_g$, there are two threats to inverting the above procedure to find a derivation of a plan $p$ for $P$ from $p'$’s derivation from $P'$: (1) two tasks for $t_g$ may be merged, and (2) a primitive task for an operator $o$ may be merged with another task for $o$.

For (1), if there is a merge $N_j \xrightarrow{i_1,i_2} M N_{j+1}$ where $\alpha(i_1) = t_g$ in the derivation of $p$, then there must also be an $N_k (g > j)$ where the method corresponding to goal release is applied (meaning $g$ holds in the state of $N_k$). Then, since $i_1$ and $i_2$ are not ordered with respect to each other, another derivation of $p$ is to skip the merge of $i_2$, and decompose $i_2$ with $m_{og}$ immediately after $N_k$, since $g$ must hold in that node’s state.

For (2), assume WLOG that the task $i_2 (\alpha(i_2) = o)$ was inserted via decomposition with $m_{og}$ of a task $i_3$. The derivation of $p$ must include a decomposition of $i_3$ with $m_{og}$, which replaces $i_3$’s task for $t_g$ with the tasks $o$ preceding $t_g$. Later in the derivation, the task for $o$ is merged with another task, and later applied, all before any further decomposition of $t_g$. Then another derivation of $p$ is just to skip the first decomposition of $i_3$ and subsequent merge.

Since we can eliminate any merging of $t_g$ or its related primitive tasks, we can invert (⇒) to find a derivation of $p$ for
As a special case, consider that HGN problems have the
same plan set under GTNS semantics as under GTN seman-
tics, as there are no tasks to merge. So by the above theo-
rem, Construction 4.1 is a plan-preserving translation of HGN
problems to HTNS planning:
Corollary 4.6. For any HGN problem P, let P′ be the result-
ing HTN problem from Construction 4.1. Then a sequence of
actions is a plan for P under GTN semantics iff it is a plan
for P′ under GTN semantics. The same result holds under
GTNS semantics.

So HTN planning is neither harder nor more expressive
than HTN planning with sharing.

Translating from GTNS to GTN

In this section, we give a plan-preserving translation from
GTNS semantics to standard GTN semantics as part of our
proof that GTNS planning is semi-decidable. To start, we note
that, at least as far as solvability goes, GTNS semantics are
equivalent to a sharing semantics that only allows merging of
primitive operators.

Lemma 4.7. Let N0 \rightarrow_{p_M} N_k be the derivation of a plan p
for a problem P. Then there is another derivation N0 \rightarrow_{p_M} N_k
that any merging only happens for primitive tasks.

Proof. We can iteratively remove merging of non-primitive
tasks from N_0 \rightarrow_{p_M} N_k as follows: Let N_j \rightarrow_{i_1,i_2} \rightarrow_{M} N_{j+1}
be the first merge of any two non-primitive tasks i_1, i_2 in the
derivation p. Since i_1 is a non-primitive task, there is a subse-
cquent decomposition N_k \rightarrow_{i, m_D} N_{k+1}. Let N_i \rightarrow_{p_M} N_{k+1}
be the same sequence of progressions above with the initial
merge omitted (including the decomposition of i_1). Then de-
compose i_2 with the same method as i_1 and merge all its re-
sulting tasks with those from the decomposition of i_1. The
resulting node is isomorphic with N_{k+1}.

We can now reduce GTNS planning to GTN relying on op-
erators asserting in the state when they have been applied:

Construction 4.8. Let P = (L, O, M, M_0, N_0) be a GTN problem.
We iteratively construct a new GTN problem P′ as fol-
lowing: For each primitive task name o, we introduce a new
non-primitive task symbol t_o, as well as a new propositional
symbol f in_o. Let L′ = L \cup \{ f in_o \mid o \in O \}. Let O′ be the
set of operators of O modified so that each operator o asserts
f in_o and retracts f in_o for all o′ \neq o. Let N_0′ and M′ be
N_0 and M with all primitive tasks in their networks replaced
with the corresponding t_o. Finally, let M_0 contain two meth-
ods for each operator o: m_{0a} = (t_o, true, gt_n_o), where gt_n_o
contains only the task o, and m_{0m} = (t_o, f in_o, gt_n_o), which
can remove t_o from a network immediately after o is applied.
Then P′ = (L′, O′, M′ \cup M_0, N_0).

There are two important points to note in the above con-
struction. First, f in_o is set up in such a way that it is true
iff o was the last operator applied. Second, we use f in_o to
simulate task-sharing under GTN semantics by providing two
methods m_{0a} and m_{0m} for each o. These respectively intro-
duce and don’t introduce a new instance of o after one in-
stance of o has already been applied; m_{0m} thus simulates the

5 Encoding HTNs as HGNs

Although there are a wide array of complexity classes for
specific restrictions of HTN planning [Alford et al., 2015a],
HTN planning in general is semi-decidable [Erol et al., 1996].
Erol et al.’s proof of undecidability encodes the intersec-
tion of context free grammars (CFGs) as an HTN problem
with totally-ordered methods and two partially-ordered ini-
tial tasks. The two tasks function as producer and consumer-
one producing a sequence of symbols from one CFG, and the
other consuming them in the order of another. The reachable
tasks from both initial tasks are disjoint. Thus, any progres-
sion of the initial problem leads to a task network composed
of two totally-ordered sets of tasks. Under GTNS semantics,
no task within one of the sets can be merged (they are all or-
dered with respect to each other), and no task between sets
can be merged (the two sets have distinct task names). Thus,
Erol et al.’s construction has the same set of solutions under
GTNS semantics as GTNS semantics.

For HGNs, Shivashankar et al. [2012] gave a bidirectional
encoding of totally-ordered HGNs and HTNs, encoding task
names as new goals. This easily extends to encode Erol et
al.’s construction in partially-ordered HGN planning:

Theorem 5.1. Deciding whether a solution to an HGN
or GTNS problem exists under GTNS semantics is semi-
decidable.

These results do not easily extend to task insertion with
sharing. In particular, the proof that HTN planning is
NEXPTIME-hard encodes a Turing machine into the task network where an exponential number of partially-ordered tasks are synchronized through the state. In the following lemma and construction, we show that if there is a plan where two mergeable tasks were not merged, we can always merge the two tasks and replace the lost task’s primitive descendants with task insertion.

In a derivation \( N_0 \rightarrow_\ast N_k \), we say that an instance symbol \( i' \) in a node \( N_j \) is a child of a symbol \( i \) in node \( N_{j-1} \) iff (1) \( i = i' \) and was not otherwise involved in the progression or (2) \( i' \) was introduced in a decomposition \( N_{j-1} \rightarrow_D N_j \). A descendant symbol of \( i \) is any child in the subsequent node, or a descendant of those children in following nodes, while the reverse holds for ancestors.

**Lemma 5.2.** Let \( N_0 \rightarrow_\ast_{PIM} N_k \) be a derivation of a plan \( p \) for a problem \( P \) under GTN\(_{15}\) semantics, and let \( N_j \) be a node in that sequence with two mergeable tasks \( i_1 \) and \( i_2 \). Then there is a progression \( N_j \rightarrow_{PIM} N_k \) which preserves the remain sequence of operator application after merging \( i_1 \) and \( i_2 \).

**Proof.** Assume WLOG that \( i_1 \) constrains some instance symbol \( i'_1 \), and \( i'_1 \) is progressed before any other instance symbol constrained by \( i_1 \) or \( i_2 \) in the derivation \( N_j \rightarrow_{PIM} N_k \); or that \( i_1 \) and \( i_2 \) constrain no tasks. Modify the derivation as follows: Let \( N_j \rightarrow_{PIM} N'_j \). Drop any decomposition or goal release of descendant symbols of \( i_2 \). For any operator application to a descendant of \( i_2 \), replace with task insertion.

**Construction 5.3.** Let \( P \) be an HTN problem (by Construction 4.1 if necessary). Alford et al. [2015b] extends the work of Geier and Bercher [2011] to show that any solvable task-insertion problem can be solved with acyclic progression, which never decomposes a task’s ancestor of the same task name. We modify it for merging as follows: Where acyclic progression maintains a history of ancestral task names, instead associate each task with a counter, starting the counters in the initial node with \( |C| \) (the number of non-primitive task names). Decompose only tasks with a positive counter, and associate a decremented counter with the resulting children. When merging two tasks, the new task receives the greater of the two counters. Finally, after every decomposition or action application, all mergeable task names are merged.

So GTN\(_{15}\) planners can merge all mergeable tasks after every operation, leaving at most \( |C| \) unconstrained tasks, which is incompatible with HTN\(_1\)’s hardness proof. This leaves us with a looser bound for GTN\(_{15}\) planning:

**Theorem 5.4.** Deciding whether a problem is solvable under GTN\(_{15}\) semantics is PSPACE-hard and in NEXPTIME.

Meanwhile, HGN\(_1\) planning is trivially in PSPACE: with insertion, one only needs to guess a totally-ordered sequencing of the initial subgoals, and solve it as a series of sequential propositional planning problems [Bylander, 1994]:

**Theorem 5.5.** Deciding whether an HGN problem is solvable under HGN\(_1\) semantics is PSPACE-complete.

### 6 Conclusion and Future Work

A number of hierarchical planning formalisms modify the standard HTN semantics in different ways; HGN planning uses goals in lieu of tasks, while ANML uses both tasks and goals as well as task insertion and sharing. Despite increased interest in using these planners in practice, there is little insight on how these modifications change the theoretical properties of HTN planning, both in terms of expressivity or computational complexity. Our key contributions against this are:

- We formalized GTN planning, a hybrid formalism that supports both task and goal decomposition. We also formalized task insertion and task-sharing semantics under this formalism, capturing features of HGNs and ANML.
- We provided plan-preserving translations from GTN problems with any combination of task insertion and sharing semantics to standard HTN semantics, thus showing that they are no more expressive than HTN planning, which in general is semi-decidable.
- We showed that HGN planning is also semi-decidable. In fact, we showed that HGN, HTN, and HTN\(_5\) planning are all equivalent complexity-wise.
- We provided plan-preserving translations from HGN problems to HTN problems with task sharing semantics, thus showing that HGN is no more expressive than HTN\(_5\).
- Finally, we provided new upper bounds for subsets of GTN planning, showing that that HGN\(_5\) is in PSPACE, and that GTN\(_{15}\) and GTN planning are in NEXPTIME.

While we show that HGNs and GTNs (with and without task sharing and insertion) can be compiled to HTN planning, we do not claim that this is a superior method of planning. Goals, sharing, and insertion all provide explicit semantics that are obscured by the compilation process. This reduces both the ability to analyze the domain for correctness and the applicability of some heuristic search techniques [Shivashankar et al., 2016].

**Future Work.** There are several avenues to extend this work, some of which are:

- We showed only that GTN\(_{15}\) belong in NEXPTIME. We conjecture GTN\(_{15}\) planning may be PSPACE-complete.
- For HGN planning, we derived a semi-decidability result for partially ordered problems, which complements a known EXPTIME-completeness result [Shivashankar et al., 2012]. A more nuanced classification of HGN planning (similar to the classification of HTN planning [Alford et al., 2012]) would also extend to GTN planning.
- We provided plan preserving translations only in one direction, e.g. HGN \(\rightarrow\) HTN\(_5\), GTN\(_4\) \(\rightarrow\) GTN, and GTN \(\rightarrow\) HTN. We suspect some of the inverse plan-preserving translations are possible and others are not. Proving the non-existence of any of these translations would provide a firm separation in expressivity [Höller et al., 2016].
- Given the partially-ordered nature of GTN planning, it seems natural to extend it to planning with concurrent actions. Even PDDL 3 semantics [Gerevini and Long, 2005]
for trajectory constraints assume linear planning, which does not appear to increase complexity; it is not clear if partial order planning plus trajectory constraints increases complexity.

Acknowledgment  This work is sponsored in part by OSD ASD (R&E). Ron Alford performed part of this work as an ASEE postdoc at NRL; affiliation with MITRE is provided for identification purposes only. The information in this paper does not necessarily reflect the position or policy of the sponsors nor MITRE, and no official endorsement should be inferred.

References


