A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations

Pascal Bercher[1] , **Ron Alford**[2] and **Daniel Höller**[1]
[1]Institute of Artificial Intelligence, Ulm University, Germany
[2]MITRE, McLean, Virginia, USA
pascal.bercher@uni-ulm.de, ralford@mitre.org, daniel.hoeller@uni-ulm.de

Abstract

Hierarchical planning has attracted renewed interest in the last couple of years, which led to numerous novel formalisms, problem classes, and theoretical investigations. Yet it is important to differentiate between the various formalisms and problem classes, since they show – sometimes fundamental – differences with regard to their expressivity and computational complexity: Some of them can be regarded equivalent to non-hierarchical formalisms while others are clearly more expressive. We survey the most important hierarchical problem classes and explain their differences and similarities. We furthermore give pointers to some of the best-known planning systems capable of solving the respective problem classes.

1 Introduction

AI Planning is concerned with generating a course of action that achieves the goals of an agent. In its most basic form – *classical planning* – the world is fully known and described in terms of a predicate or propositional logic, stating all properties that are currently true. Actions describe in which world states they are executable and how they change the respective state in terms of deterministic effects. *Hierarchical planning* extends classical planning in terms of a task hierarchy supporting two kinds of tasks: *primitive* and *compound* (or *abstract*) ones. Primitive tasks are the before-mentioned actions. Compound tasks on the other hand are abstractions of sets of other primitive and compound tasks and can therefore demand further restrictions that cannot (easily) be captured by the preconditions and effects of actions. They can be regarded control rules: The goals of the agent are not given in terms of a desired state description, but in terms of compound tasks, which should be performed. That is, adhering the predefined mappings from compound tasks to sets of other primitive and compound tasks, an executable sequence of actions should be obtained. Due to the underlying task hierarchy and its rule-based nature, the approach allows to naturally express problems with procedural knowledge.

Hierarchical planning approaches are applied in many practical contexts such as robotics [Beetz *et al.*, 2012; Stock *et al.*, 2015; González *et al.*, 2017], e.g., for the Mars Exploration Rovers [Bresina *et al.*, 2005]; web service composition [Sirin *et al.*, 2004; Sohrabi *et al.*, 2009; Georgievski and Aiello, 2015]; generating narratives [Winer and Young, 2016]; or for personal assistants that are based on a step-wise presentation of instructions [Bercher *et al.*, 2014a; Behnke *et al.*, 2019b]. Hierarchical structures further occur naturally in workflows, where approaches related to hierarchical planning can be applied [Barták and Dvorák, 2016]. Similarly, hierarchical task models are also applied in scheduling [Barták and Vlk, 2016]. For earlier overviews of practical applications see the work by Ghallab *et al.* [2004, Chapter 11.9] and Nau *et al.* [2005].

The motivation for introducing a hierarchy is manifold: It ranges from introducing expert knowledge to speed-up search [Nau, 2007]; having multiple abstraction levels to communicate with human users, e.g., for the automated generation of explanations exploiting abstraction [Seegebarth *et al.*, 2012; Bercher *et al.*, 2014a] or for conveying the steps themselves on higher levels of abstraction [de Silva *et al.*, 2019; Behnke *et al.*, 2019b]; the direct user integration into the plan generation process [Behnke *et al.*, 2016]; exploiting similarities between hierarchical models and human planning behavior [Fox, 1997; Marthi *et al.*, 2008], e.g., by modeling assumed human planning behavior in terms of a (hierarchical) plan library for plan recognition [Geib, 2004; Höller *et al.*, 2018a]; to exploiting control rules to describe desired solutions [Höller *et al.*, 2014; Höller *et al.*, 2016].

As of yet there is no established standard for the specification of hierarchical planning problems [Höller *et al.*, 2019a] – despite early attempts [McDermott, 2000]. In fact, there is a large set of different hierarchical formalisms. To the best of our knowledge, only one work provided a general overview [Fox, 1997], but it is quite dated by now and many theoretical results for hierarchical planning were published more recently. A comprehensive survey was published more recently [Georgievski and Aiello, 2015], but it focuses on *Hierarchical Task Network (HTN) planning* – i.e., one specific hierarchical planning framework – in particular and some of its most influential (and thus older) HTN planning systems. In contrast to both, we provide a survey on different hierarchical planning *problem classes* and discuss their similarities and differences. We further give pointers to some of the best-known and more recent planning systems capable of solving the respective classes.

2 HTN Planning – The Most Basic Hierarchical Planning Formalism

Before we introduce hierarchical planning, we start with introducing classical planning, as most – if not all – hierarchical planning formalisms can be considered extensions thereof.

2.1 Introduction to Classical Problems

Classical planning problems are usually defined in the STRIPS formalism, given as 4-tuples (F, A, s_I, g), where F is a finite set of *facts* (also called propositional *state variables*), A is a finite set of actions defined over F, $s_I \in 2^F$ is the initial state and $g \subseteq F$ is the goal description. The facts F are propositional encodings of world properties. Any subset of F is considered a state. Given a state $s \in 2^F$, facts $f \in s$ are considered true in s and all others are not (this is called the *closed world assumption*). Actions are 3-tuples $\langle pre, add, del \rangle \in 2^F \times 2^F \times 2^F$ consisting of a precondition pre and its add and delete effects add and del. An action a is executable in a state s if its precondition holds in s, $pre \subseteq s$. If executable in s, its result is the successor state $s' = (s \setminus del) \cup add$, i.e., delete effects get removed and add effects get added. Solutions are action sequences executable in the initial state s_I that lead to a state s'' that satisfies all goals, i.e., $s'' \supseteq g$. Any such state s'' is called a goal state.

Classical planning problems essentially compactly describe deterministic state transition systems (or: deterministic finite automata, DFAs) in terms of its initial state, goal description, and actions – without listing all intermediate states and transitions. Instead, we only specify that implicitly given DFA in terms of the planning problem (F, A, s_I, g) and a planning system automatically computes relevant parts of the state transition system to find a goal-leading sequence of actions from the initial state to a goal state.

In practice, problems are modeled relying on a first-order predicate logic. Here, actions' preconditions and effects are conjunctions of literals. Let's consider a classical domain that models the assembly of a home entertainment system [Bercher *et al.*, 2014a]. In this scenario various hifi devices can be connected by a range of cables. This can be modeled by a single *plugIn* action. It uses four parameter variables. They represent the current cable $?c$, its port $?cp$ to be used (each cable has two: one for each end), the current device $?d$, and its port to be used $?dp$. That action's precondition is a conjunction of the (positive) literals *free*$(?cp)$, *free*$(?dp)$, and *compatible*$(?cp, ?dp)$ meaning that it is applicable in a state in which the involved ports are not occupied and compatible to each other. The action's add effect is *connected*$(?cp, ?dp)$ and its delete effects are *free*$(?cp)$ and *free*$(?dp)$ describing that the used ports are not free anymore and instead connected with each other. Every such action using variables can be transformed into a propositional one via *grounding*, i.e., by replacing every variable by a suitable constant (which are given in the problem description). For the sake of simplicity we present all formalisms in a ground/propositional fashion (i.e., in the above-introduced STRIPS formalism) as mostly done in the literature. The goal of the classical problem is given as a partial state description, for instance $g = \{$*hasSignal*(TV)$\}$, encoding that the TV has a video signal. The *plugIn* actions need to propagate these *hasSignal* facts, which can be quite complicated to model [Bercher *et al.*, 2014a]. We ignored this part in the example for the sake of simplicity and because we will model the signal flow by means of a task hierarchy instead.

2.2 Introduction to Hierarchical Problems

When modeling a system as a planning domain, there will certainly be more than one possible model that is sufficient to generate plans that work in the real system. An interesting decision is the amount of *advice* that is introduced. A common philosophy when modeling *classical* planning problems is to model *physics, not advice* [McDermott, 2000, p. 37], not because no advice shall be used at all, but to make it as explicit as possible. In the beginning, hierarchical structures have been seen as a means of re-introducing advice for the planners [McDermott, 2000, p. 37]. The hierarchy can be used to restrict the search space severely, resulting in quite efficient planning systems. However, the systems rely on the additional advice; they are *configured* for the domain and are therefore sometimes called domain-*configurable* (in contrast to domain-*independent*) systems [Nau, 2007]. Even though a hierarchy among the tasks can be used as advice for a planner, it also adds a new means of modeling that makes the resulting formalism more powerful than non-hierarchical planning. This can be seen from various studies about the computational complexity of the formalism [Erol *et al.*, 1996; Geier and Bercher, 2011; Alford *et al.*, 2015a; Alford *et al.*, 2015b; Bercher *et al.*, 2016; Alford *et al.*, 2016b] and from the studies of their theoretical expressivity [Höller *et al.*, 2014; Höller *et al.*, 2016].

As a running example let us revisit the previous problem and extend it to a task hierarchy [Bercher *et al.*, 2014a; Höller *et al.*, 2018b]. In a classical model, the states (i.e., preconditions and effects) describe *all* relevant information. This includes established connections as well as which cable and device possesses which signal. Also, the goal is given in a state-based form as our example showed. When extending the classical model by a task hierarchy, the information about the signal flow can be modeled *into the hierarchy* rather than into the state: When a source device $?d_{so}$ shall be connected with a sink device $?d_{si}$, this can be done (1) directly or (2) via an intermediate device $?d_{in}$ as illustrated in Fig. 1. We can now define the goal in terms of an initial compound task *connect(Blu-ray, TV)*, as we know that based on these two

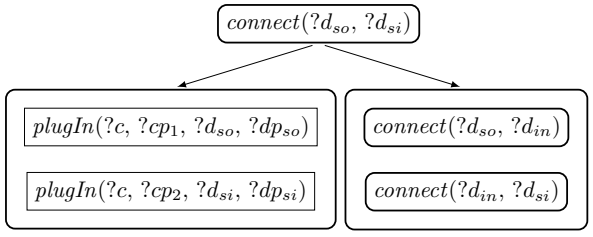

Figure 1: Depiction of the compound *connect* task and two of its decomposition methods (which do not use ordering constraints). Compound tasks and methods are depicted with rounded corners, primitive tasks (i.e., actions) are depicted angled.

rules (called *decomposition methods*), the latter being recursive, one does not have to model the parts of the states representing signal flow since we know that the signals are propagated as required if the devices are connected as demanded by the hierarchy. In such a model, the hierarchy models physics, not advice – so the hierarchy adds a new means of modeling. It can, just like states, be used to model physics *or* advice. Resulting plans must satisfy the constraints introduced by the hierarchy as well as the ones introduced by the state.

Usually, the goal in hierarchical planning is to find a primitive refinement of the initially given compound tasks. What *exactly* is allowed in order to find solutions differs significantly among the variety of present formalisms, which also influences the respective formalism's expressivity and computational complexity. We start with explaining the most commonly known representative of hierarchical planning: *Hierarchical Task Network (HTN) planning* [Ghallab *et al.*, 2004, Chapter 11] and then extend or alter it to capture and discuss the related hierarchical planning formalisms.

2.3 HTN Planning

There are many formalizations for hierarchical planning in the literature. One of the most influential formalizations, called HTN planning, is the one by Erol *et al.* [1994; 1996] since they were the first to prove many foundational properties of HTN planning and to describe a provably sound and complete HTN planning system: UMCP [Erol *et al.*, 1994].

The formalization we choose in this survey to illustrate core elements of the formalism bases upon the one by Geier and Bercher [2011]. It is a minimalistic formalism that still captures the core ideas behind HTN planning. It shows many similarities to the one introduced by Ghallab *et al.* [2004, Chapter 11.2] that they refer to as *Simple Task Network (STN) planning*. For *practical* purposes, other formalizations, like the one by Erol *et al.*, might be more appropriate due to richer language features that allow for defining more constraints such as prevail conditions, demanding that a certain state feature holds directly before or after some task or for the complete sequence of states between two tasks. In contrast to Erol *et al.*'s formalism, which is based upon a first-order predicate logic, the one presented here is fully propositional. For most formalisms, including the ones we present here, lifted variants exist as well.

We start with the definition of task networks, which are generalizations of totally ordered action sequences.

Definition 1. A *task network* tn is a tuple $(T, \prec, \alpha)$, where
- T is a finite set of task id symbols,
- $\prec \subseteq T \times T$ is a strict partial order on T,
- $\alpha : T \to N$ maps every task id to a task name in N.

Because tasks can occur several times within the same task network, the partial order $\prec$ is not defined on them directly, but instead on task identifier symbols T, which are mapped to the tasks by α.

For consistency reasons, α does not map to primitive or compound tasks directly, but instead to their names (called *task name*). Some names (like *plugIn*($?c$, $?cp_1$, $?d_{so}$, $?dp_{so}$)) represent primitive tasks, i.e., actions, and others (like *connect*($?d_{so}$, $?d_{si}$)) represent compound ones. While a primitive task name maps to its unique action (given another mapping δ, which is given in the domain model), compound task names map to one or more task networks that can be used to refine them. This mapping is established by a set of so-called *decomposition methods* M, which are part of the domain model. Each method $m \in M$ is a 2-tuple (n, tn) mapping a compound task name n to a task network tn (see Fig. 1 for a graphical illustration of two methods for the task *connect*($?d_{so}$, $?d_{si}$) of our running example). Given that a task network tn_1 contains a compound task name n (i.e., there is a task id $t \in T_1$ with $\alpha_1(t) = n$), then a method (n, tn) can be *applied* to tn_1 creating a new task network tn_2, in which t is replaced by tn and ordering constraints pointing from or to t are inherited by the tasks in tn.

For the sake of completeness, we would like to note that many HTN planning models support so-called *method preconditions*. Initially introduced for the HTN planning system SHOP [Nau *et al.*, 2001], they are still being used for models that are solved by its successor systems SHOP2 [Nau *et al.*, 2003] and SHOP3 [Goldman and Kuter, 2019]. Method preconditions are state-based conditions specifying whether the respective method may be applied in a current state. Since their definition is tailored to progression-based search (cf. Sec. 2.4) and therefore intended for this kind of planning systems, we do not go into more detail here. A more in-depth discussion of this feature is given by Höller *et al.* [2019a].

Definition 2. An *HTN problem* $\mathcal{P}$ is a tuple $(\mathcal{D}, s_I, tn_I)$ consisting of the HTN domain $\mathcal{D}$, an initial state $s_I \in s^F$, and an initial task network tn_I. An *HTN domain* $\mathcal{D}$ is a tuple (F, N_P, N_C, δ, M) consisting of a finite set of facts F, finite sets of primitive and compound task names N_P and N_C, respectively, the action mapping $\delta : N_P \to 2^F \times 2^F \times 2^F$ assigning actions to the primitive task names, and a finite set of decomposition methods M.

One of the most important aspects in hierarchical planning is the definition of the solution criteria, because they alone define in which way the model's task hierarchy has to be respected. In HTN planning, there is an initial task network containing compound tasks that have to be refined relying on the model's methods. This is captured by the solution criteria. Ghallab *et al.* [2004, Chapter 11.1] phrase it as follows: *"HTN planners differ from classical planners in what they plan for and how they plan for it. In an HTN planner, the objective is not to achieve a set of goals but instead to perform some set of tasks."* More formally:

Definition 3. A task network tn_S is a *solution* to an HTN problem $\mathcal{P}$ if and only if tn_S can be obtained from the initial task network tn_I by a sequence of method applications, does not contain compound tasks anymore, and is executable, i.e., it possesses a linearization of its (primitive) tasks that is executable in the initial state s_I.

Restricting the set of solutions to those that can be obtained from the initial tasks via decomposition enables one to *exclude* executable action sequences from the set of desired solutions – simply because they cannot be produced by exploiting the task hierarchy. This is the main reason for the high expressive power of the HTN planning formalism. Erol *et al.* [1996] proved that HTN planning is expressive enough

to model undecidable problems, such as the language intersection problem of two context-free languages. Geier and Bercher [2011] reproduced that proof in the more simplistic formalism outlined here. It is thus more expressive than the classical STRIPS formalism (and many of its extensions) as it can only be used to model decidable problems. Relying on a comparison to the Chomsky hierarchy, Höller *et al.* [2014; 2016] conducted an in-depth analysis showing which problems (i.e., which intended solution sets) can be expressed by various hierarchical and non-hierarchical formalisms.

2.4 Techniques for Solving HTN Problems

At the time being, there are essentially two important techniques to solve HTN problems: Via standard search (like A^*) and via compilation to a different problem class. For the first, Alford *et al.* [2012] explain four different kinds of search algorithms and provide necessary and sufficient criteria when search terminates. Two of these search algorithms can be considered standard as many planning systems implement them[1]. We briefly outline them below. Afterwards, we briefly outline several compilation techniques to solve HTN problems.

Decomposition-based (or plan space) search. Search is done in the space of task networks starting with the initial task network given in the problem description. Given a search node, the set of its successors in the search space is determined by the set of all decomposition methods for the compound tasks present in the current search node. That is, each existing method gets applied to its compound task in the current task network thereby creating a new one. Adding ordering constraints is allowed as well in order to obtain a reasonable ordering within a task network to guarantee executability. Decomposition and ordering insertion is repeated until an executable primitive task network is created. Two of the more recent plan space-based planners are PANDA [Bercher *et al.*, 2014b; Bercher *et al.*, 2017] and FAPE [Dvořák *et al.*, 2014; Bit-Monnot *et al.*, 2016].

Progression-based search. Again, search nodes are task networks and search starts with the initial one. Whereas the decomposition-based search allows to work on any part of the given task network, the progression-based search works on it in a "left-to-right fashion". Among all tasks that have no predecessor in their ordering constraints, search branches over the choice which of these tasks to process next. If it is a compound task, every available decomposition method produces a new successor task network. If it is a primitive task and executable in the current state, it gets executed (and removed from the current task network) thereby progressing said state. A solution is found when the current task network has become empty since this means that all compound tasks have been refined by applying decomposition methods and all primitive tasks have been executed. A solution (i.e., a sequence of actions) can be extracted from the path of action executions from the initial task network to the empty one. The best-known systems implementing this technique are the SHOP systems [Nau *et al.*, 2001; Nau *et al.*, 2003; Goldman and Kuter, 2019]. Recently, we proposed an optimization of that algorithm that reduces branching and integrates heuristics [Höller *et al.*, 2018b]. For the latter, i.e, heuristic integration, we refer readers yet unfamiliar with HTN planning to the illustrative explanation by Höller *et al.* [2019b].

Compilation techniques. Since HTN planning is in general undecidable, there cannot be a *single* compilation to *any* other (decidable) problem class. Instead, one can use some suitable bound and perform a *series* of compilations of increasing size. Every solution in the compiled problem can be translated to a solution to the original one. If a compiled problem is proved unsolvable, a new compilation is performed with increased size. Due to the undecidability of HTN planning, this process may have to be repeated arbitrarily often – termination can only be guaranteed for some special cases where decidability is known. Alford *et al.* introduced such translations to classical planning by bounding the decomposition depth [Alford *et al.*, 2009] or the size of any task network that can be obtained when performing an HTN progression search [Alford *et al.*, 2016a]. Another approach compiles HTN problems into Answer Set Programs (ASPs) while bounding the length of the plans [Dix *et al.*, 2003]. Similarly, Mali and Kambhampati [1998] introduced the idea to compile HTN problems into a sequence of SAT problems in propositional logic. This approach was recently revived and improved [Behnke *et al.*, 2018; Behnke *et al.*, 2019a; Schreiber *et al.*, 2019]. Similar to the older encoding for HTN problems to classical ones [Alford *et al.*, 2009], this encoding bounds the decomposition depth, representing all possible ways on how to decompose the initial task network using n levels of decomposition.

3 Variants of Standard HTN Problems

There are many variants and extensions of standard HTN planning. Some of them were developed as a compensation to the strict way in which the task hierarchy dictates which solutions may be generated, others introduced extensions that make reasoning about the state-based consequences of compound tasks easier, while again others abandon the idea of *task* hierarchies altogether and introduce a hierarchy on the *facts* instead. We provide a brief overview of the most important classes, most of which were developed or formalized within the last ten years.

3.1 TIHTN Planning/Task Insertion

In HTN planning, the solution criteria enforce every single action in a plan to be introduced by decomposition. In terms of modeling effort, there are domains where this is an intended capability, since this control rule-like requirement may make the modeling process easier. Goldman [2009] raises the example that it might be known that a certain medical treatment (that can be modeled as a "recipe", i.e., in terms of methods) will lead to a good outcome, but that it might be difficult to model the causality behind it in terms of state transitions. In other domains, this strict adherence to the decomposition hierarchy might be a drawback, because it enforces the domain designer to ensure that all desired solu-

[1]For illustrative examples we refer to our (introductory) HTN planning tutorial (https://www.uni-ulm.de/in/ki/htn-tutorial-2018).

tions can be created via the task hierarchy. As an alternative, many researchers have fused task hierarchies with the capability to insert tasks directly, just as in classical planning. That is, every action that is part of the domain model might be added to any task network at any position. This idea has often been described in the context of planning systems and algorithms [Young *et al.*, 1994; Kambhampati *et al.*, 1998; Gerevini *et al.*, 2008; Dvořák *et al.*, 2014; Winer and Young, 2016]. Due to the capability to insert actions directly and arbitrarily the set of solutions might be different compared to the version of the same problem, in which task insertion is prohibited. Thus, we obtain a new problem class. Geier and Bercher [2011] formally defined the respective problem class, called *HTN planning with task insertion (TIHTN planning)*, by extending the solution criteria of HTN planning (cf. Def. 3) in the way that solution task networks can be obtained from the initial task network via a sequence of method applications *and task insertions*.

In our running example for an HTN problem (the home entertainment domain), the signal flow between some source and sink device was modeled by a recursive hierarchy, while the actions' preconditions and effects were used to ensure that the used devices/cables are actually compatible. If the same model/problem would be regarded a TIHTN problem, actions harmful to a given overall task (think about an *unplug* action) could be inserted arbitrarily thus invalidating the intended signal flow. Thus, TIHTN problems, while having the *additional* capability of task insertion, *lose expressivity* compared to HTN problems. Thus we learn that task insertion has to be treated with care in case the task hierarchy is used to model physics rather than advice. If the task hierarchy is introduced for advice on how to achieve something – i.e., if not adhering the task hierarchy does not violate the correctness of the intended solutions – then allowing task insertion might be the right choice.

The lower expressivity of TIHTN problems is also backed up by many theoretical investigations [Geier and Bercher, 2011; Alford *et al.*, 2014; Höller *et al.*, 2014; Alford *et al.*, 2015b; Höller *et al.*, 2016; Alford *et al.*, 2016b] making it – in contrast to HTN planning – decidable [Geier and Bercher, 2011]. We would like to mention the most important observation that is responsible for its limited expressiveness: In TIHTN planning, recursion in the task hierarchy does not contribute to the solutions that can be generated, i.e., it can be ignored. In other words: For every TIHTN problem with recursion there exists a non-recursive TIHTN problem with the same set of solutions [Geier and Bercher, 2011, Lem. 1 and 2]. All that recursion does is to introduce further compound and non-compound tasks – eventually resulting in some primitive task network. Instead of using recursion to add the required primitive tasks we can instead rely on task insertion making recursion redundant. Due to enforced acyclicity we can compute the maximal number of decompositions and thus the maximal task network size without task insertion. We now exploit task insertion to turn such a task network into an executable one. Between each two tasks we never need to insert more actions than states (because we would run into a state cycle otherwise), thereby showing that TIHTN problems can be decided. This lower computational complexity can be exploited for heuristics, since task insertion plus delete relaxation (ignoring all negative effects of primitive actions) lowers the computational complexity even to polynomial time [Alford *et al.*, 2014].

Recently, Xiao *et al.* [2017] extended the TIHTN formalism to express *state constraints*, called TIHTNS problems. For this, they reintroduced the before-mentioned prevail conditions by Erol *et al.*'s formalism that had been removed by Geier and Bercher.

In addition to the planners mentioned before, the plan space-based HTN planners PANDA [Bercher *et al.*, 2014b; Bercher *et al.*, 2017] and FAPE [Dvořák *et al.*, 2014; Bit-Monnot *et al.*, 2016] both support task insertion.

3.2 Task Sharing

The hierarchy enforces courses of action to be in a plan, not because of their effects on the world, but "just because I say so" [Goldman, 2009]. Other than in classical planning, the planner is not free to reuse parts of the plan that have already been introduced, e.g., for different sub-goals. To allow this, some systems introduce *task sharing*. With task sharing, when there are two identical tasks t_1 and t_2 that are unordered with respect to each other, only one must be executed. I.e., the planner may eliminate one and make the other a *shared* sub-task of the parent tasks of both t_1 and t_2.

Consider our running example again. There might be a task where a DVD player and a Blu-ray player must be connected with a TV by using an adapter, since the TV has only a single input port. To connect the DVD player, it needs to be connected with the adapter, and the adapter with the TV. To connect the Blu-ray player, it needs to be connected with the adapter as well and, again, the adapter with the TV. Such tasks might be modeled using task sharing. Then, the system can merge the connection of the adapter and the TV automatically and no specific modeling of such cases is necessary.

Task sharing was first proposed in the context of the Action Notation Markup Language (ANML) [Smith *et al.*, 2008] and implemented, for example, in the plan space-based planners FAPE [Dvořák *et al.*, 2014; Bit-Monnot *et al.*, 2016][2] and CHIMP [Stock *et al.*, 2015].

The influence of task sharing is not as drastic as that of task insertion. In a theoretical investigation, both considering task sharing and task insertion, Alford *et al.* [2016b] showed that task sharing neither increases nor decreasing computational complexity of the respective problem. Thus, HTN planning with task sharing still remains undecidable.

3.3 Adding Preconditions and Effects to Compound Tasks

In the previously introduced formalisms, compound tasks do not possess preconditions or effects – instead of changing a state their purpose is to get refined by one of its decomposition methods into a pre-defined task network. Yet, there are

[2]FAPE's plan-space planning algorithm does not specifically mention task sharing. Instead, when applying decomposition methods, it inserts constraints that the respective tasks must be present instead of actually inserting them. A single (shared) task can then be inserted to achieve multiple instances of the same constraint.

several motivations for being able to specify preconditions and effects for compound tasks as well [Bercher *et al.*, 2016].

One motivation is to make it directly visible which state features are changed due to the execution of a compound task. In the *angelic hierarchical planning* approach by Marthi *et al.* [2007; 2008], the compound tasks' preconditions and effects describe state transitions under some refinement. This property is exploited to find high-level solutions (i.e., containing compound tasks) that are guaranteed to be refinable into a primitive solution. It is noteworthy, though, that the models for which this property can be guaranteed are assumed to be totally ordered (i.e., the initial task network and all task networks of the methods are totally ordered). At least in theory this is a rather severe restriction as totally ordered HTN problems cannot express undecidable problems [Erol *et al.*, 1996; Alford *et al.*, 2015a] and are hence less expressive than HTN problems in general. Others exploit the compound tasks' preconditions and effects to formalize and test criteria under which a given decomposition method is regarded an actual implementation for the respective compound task [Biundo and Schattenberg, 2001; Bercher *et al.*, 2016]. These criteria ensure that decomposition methods not respecting the modeler's intent can be identified and corrected.

Another main motivation for adding preconditions and effects to compound tasks is to be able to exploit them for search. They can be used to detect state-based problems much earlier, as the respective preconditions and effects will be noticeable early on – possibly many levels of abstraction before decomposition actually introduces the respective action(s). In case task insertion is enabled, these preconditions and effects further allow to insert compound tasks as well.

So far, we only mentioned two key motivations for extending compound tasks with preconditions and effects. We did, however, not yet specify their impact on the respective problem class. That is, we did not yet mention the solution criteria the respective problem classes rely on. These criteria do, however, have a major impact on the problems that can possibly be expressed with the respective formalism.

The most important decisions to be taken are whether task insertion is allowed and, if so, whether the goal is specified in terms of a state-based goal condition (just as in classical planning) or in terms of an initial task network that we need to turn into a solution via decomposition. Some approaches using preconditions and effects for compound tasks do rely on an initial task network that has to be turned into a solution, e.g., the angelic hierarchical planning approach [Marthi *et al.*, 2007; Marthi *et al.*, 2008] or hybrid planning [Biundo and Schattenberg, 2001; Bercher *et al.*, 2014a; Bercher *et al.*, 2014b; Bercher *et al.*, 2016] while others do not: For some real-world planning applications, it might not be necessarily required to restrict the set of solutions to those that can be obtained from the initial task network. Many planning approaches were created that rely on a hierarchical task model but without an initial task network and a state-based goal description instead (i.e., with task insertion). Those include approaches that do *not* allow preconditions and effects for compound tasks [de Silva *et al.*, 2019] and those that do [Young *et al.*, 1994; Kambhampati *et al.*, 1998; Winer and Young, 2016]. Fox [1997] refers to the respective planners of the latter kind as *operator decomposition planners*, so we refer to the respective problem class as *decompositional planning*.

This class can be regarded a hybrid between TIHTN problems and classical problems, since tasks can be inserted and compound tasks have to be decomposed if present in a current search node. Computationally, it can be regarded equivalent to classical planning, since the task hierarchy does not pose any constraints. Although compound tasks need to be refined in case they get inserted into a task network there will never be the necessity to insert them in the first place.

As example, the HTN home entertainment model introduced before would not work anymore if interpreted as decompositional problem: Since we are never *forced* to add the *connect* tasks, the signal flow would not be accounted for correctly. Just as in TIHTN planning, the signal flow had to be modeled via actions' preconditions and effects to obtain a valid *decompositional* planning problem. So using this approach excludes using a hierarchy as constraints, but still allows using it for practical purposes as was done, for example, for the generation of narratives [Winer and Young, 2016].

3.4 HGN Planning

All problem classes discussed so far were based upon a hierarchy among *tasks*. In *Hierarchical Goal Network (HGN) planning* [Shivashankar *et al.*, 2012] there is no hierarchy among tasks, but instead it defines a hierarchy on structured *state variables*, which are referred to as *goals* in this context. As in all previous problem classes, the hierarchy is given by a set of decomposition methods. In contrast, the goal hierarchy does *not* induce two kinds of state variables. All variables are regarded of the same kind, no matter whether there exists a method for it. Methods map goals to partially ordered sets of goals, which are called goal networks.

Definition 4. A *goal network* gn is a tuple $(G, \prec, \alpha)$, where
- G is a finite set of goal ids,
- $\prec \subseteq G \times G$ is a strict partial order on G, and
- for every $g \in G$, $\alpha(g)$ is a goal, i.e., a DNF (disjunctive normal form) formula over literals.

Please note that in the original publication about HGN planning [Shivashankar *et al.*, 2012], goal networks were assumed to be totally ordered. It was extended to a partial order later on [Alford *et al.*, 2016b].

In analogy to the other hierarchical planning approaches, where a method maps a compound task to a task network, here a goal g (resticted to a *conjunction* of literals) maps to a goal network. In addition, methods as defined by Shivashankar *et al.* [2012] contain a precondition *prec*, leading to the triple $(g, prec, gn)$. Their formalization assumes that a progression planner (cf. Sec. 2.4) is used to solve the respective problems, thus *prec* describes in which states (generated by the action sequence applied so far) the respective method is applicable – just as for some HTN formalisms [Nau *et al.*, 2001; Nau *et al.*, 2003; Goldman and Kuter, 2019]. If applied to a goal id g in a goal network gn_1, the method *prepends* $g \in G_1$ with gn. The set of solutions is defined inductively:

Definition 5. Let $\mathcal{P}$ be an HGN problem with the initial goal network $gn = (G, \prec, \alpha)$ and initial state s_0.

1. If G is empty, then the empty plan is a solution.
2. Otherwise, let $g \in G$ be a goal id in gn without predecessors, and $\alpha(g)$ be its goal.
   (a) If $\alpha(g)$ is satisfied in s_0, $\mathcal{P}'$ is the problem that results from removing g from gn, and π is a solution to $\mathcal{P}'$, then π is a solution to $\mathcal{P}$.
   (b) Let a be an action that is relevant for $\alpha(g)$ (i.e., it fulfills one of its literals) and applicable in s_0. Let π be a solution for $\mathcal{P}'$, which differs from $\mathcal{P}$ in the updated initial state $s_0' = \gamma(s_0, a)$. Then $a \circ \pi$ is a solution to $\mathcal{P}$.
   (c) Let $m = (g_m, prec_m, gn_m)$ be a method that is applicable to s_0 and g_m be relevant to $\alpha(g)$. Then, the set of solutions for $\mathcal{P}$ includes the set of solutions for $\mathcal{P}'$, in which gn has been replaced by gn' resulting from the application of m to g.

One important subtlety in Def. 5 is that only those actions can be applied that are contributing towards some goal. This prevents arbitrary action applications similar to HTN planning. Interestingly HGN problems are computationally as hard as HTN problems because of this [Alford *et al.*, 2016b].

Shivashankar *et al.* [2013; 2016; 2017] developed planners that solve HGN problems based on progression search and informed heuristics.

3.5 GTN Planning

Goal-Task Network (GTN) planning fuses HTN with HGN planning [Alford *et al.*, 2016b]. Instead of featuring *either* task *or* goal networks, this formalism features the fusion of both, called *goal-task networks*. Consequently, the solution criteria are those of Def. 3 and Def. 5 combined. Non-surprisingly, GTN problems are also undecidable, just as HTN and HGN problems [Alford *et al.*, 2016b]. As of yet there does not exist a planner that is capable of solving the respective problems.

3.6 Extension of the State Model

So far, all problem classes that we surveyed rely on a simple state transition semantics, i.e., all actions have deterministic effects that make some properties instantaneously true. In particular in real-work applications, this might not be sufficient anymore. Then, extensions to this simple model might be required that incorporate, for instance, continuous effects, the use of functions, or time. One of the few attempts to formally describe the integration of time and resource consumption is done by the Action Notation Modeling Language (ANML) [Smith *et al.*, 2008]. ANML implicitly relies on acyclic task hierarchies. The planner FAPE is able to cope with most of the language features given in ANML. One of the more recent appraoches that integrates time into HTNs is described by Molineaux *et al.* [2010]. They extend HTN models expressed in the SHOP2 input language [Nau *et al.*, 2003] (and the planner) by nonlinear continuous effects known from PDDL+ [Fox and Long, 2006]. This includes, e.g., reasoning about fluents (describing continues values like the fuel-level of a vehicle), which may change over time.

4 Discussion & Conclusion

For each given problem at hand, there are always countless possibilities on how to model it. Modeling it as a *hierarchical* planning task may bring several advantages, from practical ones like being able to present plans on more abstract levels to more theoretical ones like being able to express more constraints on the set of desired solutions. It is of crucial importance to understand the – sometimes subtle-looking – differences between the formalisms as otherwise plans may be generated that are no solutions to the modeled problem or the desired solution(s) cannot be found. We provided a survey and discussion on the vast set of more recent hierarchical planning formalisms highlighting their differences and similarities from a theoretical point of view while also giving pointers to systems capable of solving them.

References

[Alford *et al.*, 2009] Ron Alford, Ugur Kuter, and Dana S. Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *IJCAI*, pages 1629–1634. AAAI Press, 2009.

[Alford *et al.*, 2012] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau. HTN problem spaces: Structure, algorithms, termination. In *SoCS*, pages 2–9. AAAI Press, 2012.

[Alford *et al.*, 2014] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau. On the feasibility of planning graph style heuristics for HTN planning. In *ICAPS*, pages 2–10. AAAI Press, 2014.

[Alford *et al.*, 2015a] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning. In *ICAPS*, pages 7–15. AAAI Press, 2015.

[Alford *et al.*, 2015b] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning with task insertion. In *IJCAI*, pages 1502–1508. AAAI Press, 2015.

[Alford *et al.*, 2016a] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo, and David Aha. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *ICAPS*, pages 20–28. AAAI Press, 2016.

[Alford *et al.*, 2016b] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *IJCAI*, pages 3022–3029. AAAI Press, 2016.

[Barták and Dvorák, 2016] Roman Barták and Tomás Dvorák. On verification of workflow and planning domain models using attribute grammars. In *Proc. of the 15th Mexican Int. Conf. on AI (MICAI)*, pages 332–345. Springer, 2016.

[Barták and Vlk, 2016] Roman Barták and Marek Vlk. Hierarchical task model for resource failure recovery in production scheduling. In *Proc. of the 15th Mexican Int. Conf. on AI (MICAI)*, pages 362–378. Springer, 2016.

[Beetz *et al.*, 2012] Michael Beetz, Dominik Jain, Lorenz Mösenlechner, Moritz Tenorth, Lars Kunze, Nico Blodow, and Dejan Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proc. of the IEEE*, 100(8):2454–2471, 2012.

[Behnke *et al.*, 2016] Gregor Behnke, Daniel Höller, Pascal Bercher, and Susanne Biundo. Change the plan – How hard can that be? In *ICAPS*, pages 38–46. AAAI Press, 2016.

[Behnke *et al.*, 2018] Gregor Behnke, Daniel Höller, and Susanne Biundo. totSAT – Totally-ordered hierarchical planning through SAT. In *AAAI*, pages 6110–6118. AAAI Press, 2018.

[Behnke *et al.*, 2019a] Gregor Behnke, Daniel Höller, and Susanne Biundo. Finding optimal solutions in HTN planning – A SAT-based approach. In *IJCAI*. IJCAI, 2019.

[Behnke *et al.*, 2019b] Gregor Behnke, Marvin Schiller, Matthias Kraus, Pascal Bercher, Mario Schmautz, Michael Dorna, Michael Dambier, Wolfgang Minker, Birte Glimm, and Susanne Biundo. Alice in DIY wonderland or: Instructing novice users on how to use tools in DIY projects. *AI Communications*, 32:31–57, 2019.

[Bercher *et al.*, 2014a] Pascal Bercher, Susanne Biundo, Thomas Geier, Thilo Hörnle, Florian Nothdurft, Felix Richter, and Bernd Schattenberg. Plan, repair, execute, explain – How planning helps to assemble your home theater. In *ICAPS*, pages 386–394. AAAI Press, 2014.

[Bercher *et al.*, 2014b] Pascal Bercher, Shawn Keen, and Susanne Biundo. Hybrid planning heuristics based on task decomposition graphs. In *SoCS*, pages 35–43. AAAI Press, 2014.

[Bercher *et al.*, 2016] Pascal Bercher, Daniel Höller, Gregor Behnke, and Susanne Biundo. More than a name? On implications of preconditions and effects of compound HTN planning tasks. In *ECAI*, pages 225–233. IOS Press, 2016.

[Bercher *et al.*, 2017] Pascal Bercher, Gregor Behnke, Daniel Höller, and Susanne Biundo. An admissible HTN planning heuristic. In *IJCAI*, pages 480–488. IJCAI, 2017.

[Bit-Monnot *et al.*, 2016] Arthur Bit-Monnot, David E. Smith, and Minh Do. Delete-free reachability analysis for temporal and hierarchical planning. In *ECAI*, pages 1698–1699. IOS Press, 2016.

[Biundo and Schattenberg, 2001] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief – A preliminary report on combining state abstraction and HTN planning. In *ECP*, pages 157–168. AAAI Press, 2001.

[Bresina *et al.*, 2005] John L. Bresina, Ari K. Jónsson, Paul H. Morris, and Kanna Rajan. Activity planning for the mars exploration rovers. In *ICAPS*, pages 40–49. AAAI Press, 2005.

[de Silva *et al.*, 2019] Lavindra de Silva, Lin Padgham, and Sebastian Sardina. HTN-like solutions for classical planning problems: An application to BDI agent systems. *Theoretical Computer Science*, 763:12–37, 2019.

[Dix *et al.*, 2003] Jürgen Dix, Ugur Kuter, and Dana Nau. Planning in answer set programming using ordered task decomposition. In *Proc. of the 26th German Conf. on AI (KI)*, pages 490–504. Springer, 2003.

[Dvořák *et al.*, 2014] Filip Dvořák, Roman Barták, Arthur Bit-Monnot, Félix Ingrand, and Malik Ghallab. Planning and acting with temporal and hierarchical decomposition models. In *ICTAI*, pages 115–121. IEEE, 2014.

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254. AAAI Press, 1994.

[Erol *et al.*, 1996] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and AI (AMAI)*, 18(1):69–93, 1996.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *Journal of AI Research (JAIR)*, 27(1):235–297, 2006.

[Fox, 1997] Maria Fox. Natural hierarchical planning using operator decomposition. In *ECP*, pages 195–207. Springer, 1997.

[Geib, 2004] Christopher W. Geib. Assessing the complexity of plan recognition. In *AAAI*, pages 507–512. AAAI Press, 2004.

[Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *IJCAI*, pages 1955–1961. AAAI Press, 2011.

[Georgievski and Aiello, 2015] Ilche Georgievski and Marco Aiello. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222(0):124–156, 2015.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining domain-independent planning and HTN planning: The Duet planner. In *ECAI*, pages 573–577. IOS Press, 2008.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Goldman and Kuter, 2019] Robert P. Goldman and Ugur Kuter. Hierarchical task network planning in common Lisp: The case of SHOP3. In *Proc. of the 12th Europ. Lisp Symp. (ELS)*, pages 73–80. ACM, 2019.

[Goldman, 2009] Robert Goldman. A semantics for HTN methods. In *ICAPS*, pages 146–153. AAAI Press, 2009.

[González *et al.*, 2017] José Carlos González, José Carlos Pulido, and Fernando Fernández. A three-layer planning architecture for the autonomous control of rehabilitation therapies based on social robots. *Cognitive Systems Research*, 43(Supplement C):232–249, 2017.

[Höller *et al.*, 2014] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of

hierarchical planning problems. In *ECAI*, pages 447–452. IOS Press, 2014.

[Höller *et al.*, 2016] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *ICAPS*, pages 158–165. AAAI Press, 2016.

[Höller *et al.*, 2018a] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Plan and goal recognition as HTN planning. In *ICTAI*, pages 466–473. IEEE, 2018.

[Höller *et al.*, 2018b] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. A generic method to guide HTN progression search with classical heuristics. In *ICAPS*, pages 114–122. AAAI Press, 2018.

[Höller *et al.*, 2019a] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. HDDL – A language to describe hierarchical planning problems. In *Proc. of the 2nd ICAPS Workshop on Hierarchical Planning*, pages 6–14, 2019.

[Höller *et al.*, 2019b] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. On guiding search in HTN planning with classical planning heuristics. In *IJCAI*. IJCAI, 2019.

[Kambhampati *et al.*, 1998] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *AAAI*, pages 882–888. AAAI Press, 1998.

[Mali and Kambhampati, 1998] Amol D. Mali and Subbarao Kambhampati. Encoding HTN planning in propositional logic. In *AIPS*, pages 190–198. AAAI Press, 1998.

[Marthi *et al.*, 2007] Bhaskara Marthi, Stuart J. Russell, and Jason Wolfe. Angelic semantics for high-level actions. In *ICAPS 2007*, pages 232–239. AAAI Press, 2007.

[Marthi *et al.*, 2008] Bhaskara Marthi, Stuart Russell, and Jason Wolfe. Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS*, pages 222–231. AAAI Press, 2008.

[McDermott, 2000] Drew V. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[Molineaux *et al.*, 2010] Matthew Molineaux, Matthew Klenk, and David Aha. Planning in dynamic environments: Extending HTNs with nonlinear continuous effects. In *AAAI*, pages 1115–1120. AAAI Press, 2010.

[Nau *et al.*, 2001] Dana S. Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. The SHOP planning system. *AI Magazine*, 22(3):91–94, 2001.

[Nau *et al.*, 2003] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of AI Research (JAIR)*, 20:379–404, 2003.

[Nau *et al.*, 2005] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Dan Wu, Fusun Yaman, Héctor Muñoz-Avila, and J. William Murdock. Applications of SHOP and SHOP2. *Intelligent Systems, IEEE*, 20:34–41, 2005.

[Nau, 2007] Dana S. Nau. Current trends in automated planning. *AI Magazine*, 28(4):43–58, 2007.

[Schreiber *et al.*, 2019] Dominik Schreiber, Tomáš Balyo, Damien Pellier, and Humbert Fiorino. Tree-REX: SAT-based tree exploration for efficient and high-quality HTN planning. In *ICAPS*. AAAI Press, 2019.

[Seegebarth *et al.*, 2012] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users – A formal approach for generating sound explanations. In *ICAPS*, pages 225–233. AAAI Press, 2012.

[Shivashankar *et al.*, 2012] Vikas Shivashankar, Ugur Kuter, Dana Nau, and Ron Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *AAMAS*, pages 981–988. IFAAMAS, 2012.

[Shivashankar *et al.*, 2013] Vikas Shivashankar, Ron Alford, Ugur Kuter, and Dana Nau. The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *IJCAI*, pages 2380–2386. AAAI Press, 2013.

[Shivashankar *et al.*, 2016] Vikas Shivashankar, Ron Alford, Mark Roberts, and David W. Aha. Cost-optimal algorithms for planning with procedural control knowledge. In *ECAI*, pages 1702–1703. IOS Press, 2016.

[Shivashankar *et al.*, 2017] Vikas Shivashankar, Ron Alford, and David Aha. Incorporating domain-independent planning heuristics in hierarchical planning. In *AAAI*, pages 3658–3664. AAAI Press, 2017.

[Sirin *et al.*, 2004] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[Smith *et al.*, 2008] David E. Smith, Jeremy Frank, and William Cushing. The ANML language. In *KEPS*, 2008.

[Sohrabi *et al.*, 2009] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. HTN planning with preferences. In *IJCAI*, pages 1790–1797. AAAI Press, 2009.

[Stock *et al.*, 2015] Sebastian Stock, Masoumeh Mansouri, Federico Pecora, and Joachim Hertzberg. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *IROS*, pages 6459–6464. IEEE, 2015.

[Winer and Young, 2016] David R. Winer and R. Michael Young. Discourse-driven narrative generation with bipartite planning. In *INLG*, pages 11–20. ACL, 2016.

[Xiao *et al.*, 2017] Zhanhao Xiao, Andreas Herzig, Laurent Perrussel, Hai Wan, and Xiaoheng Su. Hierarchical task network planning with task insertion and state constraints. In *IJCAI*, pages 4463–4469. IJCAI, 2017.

[Young *et al.*, 1994] R. Michael Young, Martha E. Pollack, and Johanna D. Moore. Decomposition and causality in partial-order planning. In *AIPS*, pages 188–193. AAAI Press, 1994.