

HDDL 2.1: Towards Defining a Formalism and a Semantics for Temporal HTN Planning

Damien Pellier,¹ Alexandre Albore,² Humbert Fiorino,¹ Rafael Bailon-Ruiz²

¹University of Grenoble Alpes, LIG, Grenoble, France

{damien.pellier, humbert.fiorino}@imag.fr

²ONERA/DTIS, University of Toulouse, France

{alexandre.albore, rafael.bailon_ruiz}@onera.fr

Abstract

Real world applications as in industry and robotics need modelling rich and diverse automated planning problems. Their resolution usually requires coordinated and concurrent action execution. In several cases, these problems are naturally decomposed in a hierarchical way and expressed by a Hierarchical Task Network (HTN) formalism. HDDL, a hierarchical extension of the Planning Domain Definition Language (PDDL), unlike PDDL 2.1 does not allow to represent planning problems with numerical and temporal constraints, which are essential for real world applications. We propose to fill the gap between HDDL and these operational needs and to extend HDDL by taking inspiration from PDDL 2.1 in order to express numerical and temporal expressions. This paper opens discussions on the semantics and the syntax needed for a future HDDL 2.1 extension.

1 Introduction

Real world applications of Automated Planning, like in industry and robotics, require modelling rich and diverse scenarios. Such planning problems are often naturally decomposed in a hierarchical way, with compound tasks that refine in different ways their execution model. These real world applications of planning use both numerical and temporal constraints to define the agents synchronisation on collaborative tasks, and sub-task decomposition. In fact, concurrency between actions, their duration, and agents coordination in HTN problems are needed to find solutions for nontrivial tasks in complex scenarios and require to make explicit the representation of time (Ghallab, Nau, and Traverso 2016).

The Hierarchical Task Network (HTN) formalism (Erol, Hendler, and Nau 1994) is used to express a wide variety of planning problems in real-world applications, e.g., in task allocation for robot fleets (Milot et al. 2021), video games (Menif, Jacopin, and Cazenave 2014) or industrial contexts such as software deployment (Georgievski et al. 2017). Over the last years, much progress has been made in the field of hierarchical planning (Bercher, Alford, and Höller 2019). Novel systems based on the traditional, search-based techniques have been introduced (Bit-Monnnot, Smith, and Do 2016; Ramoul et al. 2017; Shivashankar, Alford, and Aha 2017; Bercher et al. 2017; Höller et al. 2019, 2020; Höller

and Bercher 2021), but also new techniques like the translation to STRIPS/ADL (Alford, Kuter, and Nau 2009; Alford et al. 2016; Behnke et al. 2022), or revisited approaches like the translation to propositional logic (Behnke, Höller, and Biundo 2018, 2019; Schreiber et al. 2019; Schreiber 2021; Behnke 2021). Despite these advances, not all planning systems use the same formalism to represent hierarchical task decomposition, making it difficult to compare approaches and promote HTN planning techniques.

An extension of PDDL (Planning Domain Description Language) (Mcdermott et al. 1998), called HDDL (Hierarchical Planning Domain Description Language) (Höller et al. 2020), has been proposed to address this issue. HDDL is based on PDDL 2.1 (Fox and Long 2003) and is the result of several discussions within the planning community (Behnke et al. 2019) to fill the need of a standard language for the first Hierarchical Planning track of International Planning Competitions (IPC) in 2020. However, it was decided that the first version of HDDL would not include any of the temporal or numerical features of PDDL due to efforts to develop the language and related tools. In this paper, we illustrate the challenge of defining the semantics for a temporal extension of HDDL to meet the needs of the planning community and planning applications.

Our motivation is grounded on the compelling need to devise applications involving autonomous systems. We propose to extend HDDL, by including elements of PDDL 2.1 and ANML (*Action Notation Modeling Language*) (Smith, Frank, and Cushing 2008), to express temporal and numerical constraints. This is intended to initiate discussions within the HTN community on establishing a standard – HDDL 2.1 – aimed at filling the gaps between existing hierarchical-temporal planning approaches. To that end, we make this preliminary extension of HDDL an open source project with a public repository, where we propose a full syntax as well as a set of benchmarks based on this extension¹ and a parser for it, as part of the PDDL4J² library (Pellier and Fiorino 2018).

The rest of the paper is organised as follows. In Section 2 we define the basic concepts of the proposed extension. In Sections 3 and 4 we set down the semantics for Temporal

¹<https://github.com/pellierd/HDDL2.1>

²<https://github.com/pellierd/pddl4j>

HTN planning. We conclude on the central aspects of this planning paradigm, and on future work.

2 Lifted Temporal HTN planning

Throughout this section, we will use common notations from first-order logic, which we assume to be known. In the lifted formalism of HDDL 2.1, we assume for the sake of simplicity that all logical formulas are over a *function-free* first-order logic language $\mathcal{L} = (V, C, P)$. \mathcal{L} consists of sufficiently many *constant* $c \in C$ representing the *objects* in the real world, *variables* $x \in V$ and *predicates* $p \in P$. Predicates have parameters that are either variables or constants. The predicate arity is the number of predicate parameters. For instance, $p(x, c)$ is a 2-arity predicate. We can now define *formulas* in a function-free first-order logic: (i) a predicate is a formula ; (ii) if ϕ and ψ are formulas, then $\neg\phi$, $\phi \vee \psi$ and $\phi \wedge \psi$ are formulas ; (iii) if ϕ is a formula and x is a variable, then $\forall x\phi$ is a formula. We define $\exists x\phi$ as $\neg\forall x\neg\phi$, and $\phi \rightarrow \psi$ as $\neg\phi \vee \psi$. \forall and \exists are respectively the universal and the existential quantifier. Conceptually, grounding a formula consists in generating a set of variable-free i.e. *ground* formulas (Helmert 2009) as follows: a variable x in a quantifier-free formula ϕ is eliminated by replacing ϕ with $|C|$ copies, one for each $c \in C$, where x is substituted with c in the respective copy. This substitution is denoted by $\phi[x/c]$. Regarding quantified formulas, $\forall x\phi$ is replaced by $\bigwedge_{c \in C} \phi[x/c]$ and $\exists x\phi$ by $\bigvee_{c \in C} \phi[x/c]$. We refer the reader to the work of Behnke et al. (2020); Ramoul et al. (2017) for further details on grounding implementation. Note that it is always possible to transform a formula in function-free first-order logic into a finite set of ground formulas in propositional logic.

A *state* s is a set of ground predicates. For the sake of conciseness, we will also consider s as a Herbrand interpretation that assigns *true* to all ground predicates in s , and *false* to all ground predicates not in s . From this, a truth value can be computed for every *ground* formula from \mathcal{L} by using the usual rules for logical composition. Without loss of generality, a formula (not necessarily ground) ϕ is true in s if and only if grounding ϕ generates at least one ground formula true in s . We will use the notation $s \models \phi$ to mean that the formula ϕ is true in s .

A key concept in HTN planning and a fortiori in temporal HTN planning is the concept of *task*. Each task is given by a name and a list of parameters. We distinguish two kinds of tasks: the primitive tasks (also called actions), and the abstract tasks (or compound tasks). Primitive tasks are carried out by durative actions in the sense of classical temporal planning (Fox and Long 2003), while abstract tasks can be refined by applying methods that define the decomposition of the task into subtasks. The purpose of abstract tasks is not to induce a state transition. Instead, they refer to a predefined mapping to one or more tasks that can refine the abstract task. For instance, in the task of serving a dinner, *deliver-dinner(?food-style, ?place)* is the compound task consisting in performing first the task of serving the starters, then the main course, etc. In that sense, *deliver-dinner(?food-style, ?place)* can be refined in: *(serve-starters(?food-style, ?place), serve-main-course(?food-style, ?place), etc.)* This

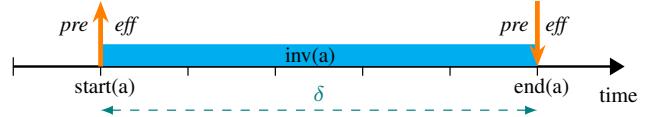


Figure 1: Timeline of a durative action a application.

mapping between tasks is achieved by a set of decomposition methods, namely the methods (Def. 5) and the Temporal Task Networks (Def. 6).

We first define the planning domain and problem for Temporal HTN Planning.

Definition 1. A planning domain \mathcal{D} is a tuple $(\mathcal{L}, \mathcal{T}, \mathcal{J}, \alpha, \mathcal{A}, \mathcal{M})$, where \mathcal{L} is the first-order logic language, \mathcal{T} is the set of tasks, \mathcal{J} is the set of task identifiers³, $\alpha : \mathcal{J} \rightarrow \mathcal{T}$ is the function that maps task identifiers to tasks, \mathcal{A} is a set of actions constituted by snap actions (Def. 3) and durative actions (Def. 4), \mathcal{M} is the set of methods (Def. 5).

The domain implicitly defines the set of all states S defined over all subsets of all ground predicates in \mathcal{L} .

Definition 2. A planning problem \mathcal{P} is a tuple $(\mathcal{D}, s_0, w_0, g)$, where \mathcal{D} is a planning domain, $s_0 \in S$ is the initial state, w_0 is the initial temporal task network (not necessary ground), and g is a formula (not necessary ground) describing the goal.

Let us start by defining the concepts of *snap* and *durative actions*, which are the primitive tasks, based on the definitions from Abdulaziz and Koller (2022). A snap action is an action whose execution is instantaneous in the sense of classical planning, meaning that it has a null duration between checking the preconditions and applying the effects.

Definition 3 (Snap Action). A snap action a is a tuple $(name(a), precond(a), effect(a))$, where $name(a)$ is the name of a , the precondition $precond(a)$ is a first-order formula, and the effects $effect(a) = effect^+(a) \cup effect^-(a)$ ($effect^+(a) \cap effect^-(a) = \emptyset$), $effect^+(a)$ and $effect^-(a)$ are conjunctions of predicates.

Definition 4 (Durative Action). A durative action a is a tuple $(name(a), start(a), end(a), inv(a), \delta)$: $name(a)$ is the name of a ; $start(a)$ and $end(a)$ are snap actions; $inv(a)$ is a first-order formula that must hold in all the states after the execution of $start(a)$ and until the execution of $end(a)$, and δ is the duration of a .

Actions do change the state of the world. Durative actions also change the time of the model, shifting it by a quantity δ , as shown in Figure 1. State transitions will be formally defined in Section 3.

Unlike a primitive task, a *compound task* does not directly change the world state. A compound task is identified by a name and defines the way other –possibly ordered– tasks (either primitive or compound) must be achieved with respect

³Task identifiers are arbitrary symbols, which serve as place holders for the actual tasks they represent. Identifiers are needed because tasks can occur multiple times within the same task network, as we will see below.

to some constraints in order to refine it. Like primitive tasks, compound tasks have also a start event and an end event, which will be associated to dates in Section 3. In this sense, methods allow to refer tasks to temporal task networks.

Definition 5 (Method). A method m is a tuple $(name(m), task(m), tn(m))$, where $name(m)$ is the name of the method, $task(m)$ is the task refined by the method, and $tn(m)$ is the temporal task network decomposing $task(m)$.

Definition 6 (Temporal Task Network). A temporal task network $w = (\mathcal{I}, \mathcal{C})$ is given by:

- $\mathcal{I} \subseteq \mathcal{J}$ is a set (possibly empty) set of tasks identifiers;
- \mathcal{C} is the set of constraints, with $\mathcal{C} = < \mathcal{C}_o, \mathcal{C}_v, \mathcal{C}_d, \mathcal{C}_t >$:
 - \mathcal{C}_o is a set of temporal qualitative ordering constraints over the start or the end events of the tasks in \mathcal{I} . The possible qualitative temporal ordering are those from the classical point algebra (Broxvall and Jonsson 2003): $<, \leq, >, \geq, =$ and \neq ;
 - \mathcal{C}_v is a set of parameter constraints. Each constraint can bind two variables to be equal or non-equal, or similarly bind a variable to a constant;
 - \mathcal{C}_d is a set of durative constraints over the duration of the tasks in \mathcal{I} ;
 - \mathcal{C}_t is a set of temporal decomposition constraints of the form $(at e \phi)$, expressing that some properties defined by the formula ϕ must hold in the state at date e .

The temporal task network w implicitly defines a temporal ordered multi-set of tasks $\mathcal{T}' = \{\alpha(i) \mid i \in \mathcal{I}\}$.

A temporal task network explicits the decomposition of abstract tasks into subtasks. Note that a temporal task network is ground if all its variables are bound to constants, and primitive if all its tasks \mathcal{T}' are primitive.

3 Temporal HTN Planning Semantics

The solution of a temporal HTN planning problem is an executable temporal task network that is obtained from the problem initial task network by applying method decomposition and constraint satisfaction.

Lifted problems are just a compact representation of their ground instances. Variable constraints are satisfied by the grounding, so there is no need to use them with ground instances. Therefore, for simplicity, this section defines the semantics of a lifted problem in terms of its ground instances. For details on the grounding process, the reader is referred to (Behnke et al. 2020; Ramoul et al. 2017).

Let us start by defining a temporal sequence of tasks.

Definition 7 (Temporal Sequence of Tasks). A temporal sequence of tasks π over a planning domain \mathcal{D} , is a sequence of tuples $\langle (t_0, e_0, \delta_0), \dots, (t_n, e_n, \delta_n) \rangle$ where t_0, \dots, t_n are ground tasks defined over \mathcal{T} , and for $0 \leq i \leq n$, the natural numbers⁴ $e_i \in \mathbb{N}_{\geq 0}$ and $\delta_i \in \mathbb{N}_{\geq 0}$ are the starting date and the duration of the task t_i , respectively. For a temporal sequence of tasks π , the set of dates $\{e \mid (t, e, \delta) \in \pi\} \cup \{e + \delta \mid (t, e, \delta) \in \pi\}$ induces a sorted sequence $< e_0, \dots, e_n >$ of happening events of π . A temporal sequence of tasks π is primitive if and only if for every task $(t, e, \delta) \in \pi$, t is primitive, i.e. t is carried out by an action (either snap or durative).

The duration of a task t is generally unbounded, as the bound would be the sum of the durations of the tasks of which t is compounded of. Only when a task t is primitive, then $duration(t)$ is given by the duration δ of the action that achieves t .

In order to guarantee the executability of concurrent plans, in the sense of the “required concurrency” as described by Cushing, Kambhampati, and Weld (2007), a central notion is *non-interference*, i.e. when preconditions and effects of snap actions do not overlap. We consider that two snap actions a and b are *not interfering* if and only if (i) $precond(a) \cap (effect^+(b) \cup effect^-(b)) = \emptyset$, (ii) $precond(b) \cap (effect^+(a) \cup effect^-(a)) = \emptyset$, (iii) $effect^+(a) \cap effect^-(b) = \emptyset$ and $effect^+(b) \cap effect^-(a) = \emptyset$.

Two snap actions or more can be executed at the same time if they are pairwise non-interfering. The execution semantics of snap actions are similar to the semantics of \forall -step parallel plans (Rintanen, Heljanko, and Niemelä 2006), and used in PDDL 2.1 (Fox and Long 2003). With this notion in mind, we define an executable temporal sequence of tasks.

Definition 8 (Executable Temporal Sequence of Tasks). A temporal sequence of tasks $\pi = \langle (t_0, e_0, \delta_0), \dots, (t_n, e_n, \delta_n) \rangle$ is executable in a state s_0 if and only if for every $(t, e, \delta) \in \pi$: (i) t is primitive; (ii) e is a happening event of π ; (iii) state s_i at date e_i transitions to a new state s_{i+1} s.t.: given the set B_{e_i} of snap actions being executed at e_i : $B_{e_i} = \{start(a_j) \mid (a_j, e_j, \delta_j) \in \pi \text{ and } e_j = e_i\} \cup \{end(a_k) \mid (a_k, e_k, \delta_k) \in \pi \text{ and } e_i + \delta_k = e_k\}$ and given the set I_{e_i} of the invariants holding at e_i : $I_{e_i} = \{inv(a_j) \mid (a_j, e_j, \delta_j) \in \pi \wedge e_j < e_i < e_j + \delta_j\}$ the transition of s_i to s_{i+1} , given that all the snap actions in B_{e_i} are pairwise non-interfering, is defined as $s_i \models precond(a)$ for every $a \in B_{e_i}$, $s_i \models inv(a)$ for every $inv(a) \in I_{e_i}$ and $s_{i+1} = (s_i - \cup_{a \in B_{e_i}} effect^-(a)) \cup_{a \in B_{e_i}} effect^+(a)$.

Definition 8 can be extended to a temporal task network.

Definition 9 (Executable Temporal Task Network). A temporal task network $w = (\mathcal{I}, < \mathcal{C}_o, \mathcal{C}_v, \mathcal{C}_d, \mathcal{C}_t >)$ is executable in a state s_0 if and only if there exists an executable temporal sequence of tasks $\pi = \langle (\alpha(i_0), e_0, \delta_0), \dots, (\alpha(i_n), e_n, \delta_n) \rangle$ where i_0, \dots, i_n are task identifiers in \mathcal{I} that matches the following conditions: (i) π matches the temporal constraints \mathcal{C}_o , (ii) π matches the duration constraints \mathcal{C}_d , and (iii) the sequence of states and their associated dates $\langle (s_0, e_0), \dots, (s_n, e_n) \rangle$ resulting from the execution of π matches the constraints \mathcal{C}_t .

It remains to define how to transform a temporal task network into another one by using method decomposition in order to obtain an executable task network, and what represents a temporal task network solution.

⁴Rational numbers are used in the definition from Fox and Long (2003). However, integers should be used for dates, because using rationals without adding further conditions can yield to an undecidable planning problem (Barringer, Kuiper, and Pnueli 1986).

Definition 10 (Decomposition). A task network $w_1 = (\mathcal{I}^1, \mathcal{C}^1)$ is decomposed into a new task network $w_2 = (\mathcal{I}^2, \mathcal{C}^2)$ by refinement by a method $m = (name(m), task(m), (\mathcal{I}^m, \mathcal{C}^m))$ if and only if there exists $i \in \mathcal{I}^1$ such that $\alpha(i) = task(m)$ and $\mathcal{I}^2 = (\mathcal{I}^1 - \{i\}) \cup \mathcal{I}^m$, and

$$\begin{aligned}\mathcal{C}_o^2 &= \mathcal{C}_o^1 \cup \mathcal{C}_o^m \\ &\cup \{start(t) \leq start(t') \mid t = \min(e_{start(\alpha(i))}, e_{start(\alpha(j))}), \\ &t' = \max(e_{start(\alpha(i))}, e_{start(\alpha(j))}), j \in \mathcal{I}^m\} \\ &\cup \{end(t) \geq end(t') \mid t = \max(e_{start(\alpha(i))}, e_{start(\alpha(j))}), \\ &t' = \min(e_{start(\alpha(i))}, e_{start(\alpha(j))}), j \in \mathcal{I}^m\}\end{aligned}$$

with $e_{start(\alpha(k))}$ the start date of $\alpha(k)$ for $k \in \mathcal{I}^1 \cup \mathcal{I}^2$.

$$\mathcal{C}_v^2 = \mathcal{C}_v^1 \cup \mathcal{C}_v^m, \mathcal{C}_d^2 = \mathcal{C}_d^1 \cup \mathcal{C}_d^m, \mathcal{C}_t^2 = \mathcal{C}_t^1 \cup \mathcal{C}_t^m$$

Definition 11 (Temporal Task Network Solution). Let $\mathcal{P} = (\mathcal{D}, s_0, w_0, g)$ be a planning problem with $\mathcal{D} = (\mathcal{L}, \mathcal{T}, \mathcal{A}, \mathcal{M})$. A task network $w_s = (\mathcal{I}, \mathcal{C})$ is solution to a temporal HTN planning problem \mathcal{P} if and only if:

- There exists a sequence of decompositions from w_o to w_s resulting from the application of the methods \mathcal{M} of \mathcal{D} ;
- w_s is executable and the temporal sequence of states resulting from its execution starts with s_0 , and achieves a state $s \models g$.

4 Decomposition Constraint Semantics

Decomposition constraints are conditions that must be satisfied by all the states visited while executing a solution task network. They are expressed through temporal modal operators over first-order formulas involving state predicates, as in PDDL. The semantics of the decomposition constraints can be formally specified similarly to the approach taken by Gerevini and Long (2005). Let $w = (\mathcal{I}, \mathcal{C})$ be a ground task network, a state s_0 and a primitive temporal sequence of tasks $\pi = \langle (t_0, e_0, \delta_0), \dots, (t_n, e_n, \delta_n) \rangle$ with $t_0 = \alpha(i_0), \dots, t_n = \alpha(i_n)$ resulting from the decomposition of w . We denote by $\tau = \langle (s_0, e_0), \dots, (s_n, e_n) \rangle$ the temporal sequence of states produced by the execution of π in s_0 , ordered according to its happening events, with $i \leq 0 \leq n$. Decomposition w satisfies a constraint $(at\ e\ \phi) \in \mathcal{C}_t$ iff $\exists (s_k, e_k) \in \tau$ such that $s_k \models \phi$. Note that it is required that every temporal constraint of the form $(at\ e_i\ \phi) \in \mathcal{C}_t$ to be defined for $0 \leq i \leq n$ so as to avoid defining constraints that are out of the scope of the temporal task network w . Each decomposition constraint defined in HDDL 2.1 can be rewritten in terms of constraints of the form $(at\ e\ \phi)$. The proposed HDDL extension distinguishes two types of decomposition constraints: (1) the *temporal decomposition constraints* that define the constraints that must hold at specific happening events whose semantics is based on the plan trajectory constraints from PDDL 3.0 (Gerevini and Long 2005) and (2) the *classical decomposition constraints* (*before*, *after*, *between*) used in HTN planning (Erol, Hendler, and Nau 1994) to represent constraints between tasks.

For the temporal decomposition constraints, we suggest to keep the same syntax and semantics as introduced in PDDL 3.0 to maintain a language consistency be-

tween different versions. For classical decomposition constraints, it can be shown that they can be expressed in terms of the former, as are method precondition semantics (*at start*, *at end*, *overall*).

5 Discussion and Conclusion

We have introduced the main features of a HDDL version for hierarchical temporal planning tasks. This aims at bridging the gap between HTN planning and real world applications, where temporal features like concurrent actions, coordination, and hierarchical distribution of tasks, are prominent. In our view, the absence of a unified language for temporal and numerical constraints in PDDL’s evolution is an obstacle that needs addressing. Although the community has developed various approaches to model complex planning problems, including temporal features and hierarchical task decomposition, the variety of language solutions hinders the development of common tools and solvers.

Action Notation Modeling Language (ANML) (Smith, Frank, and Cushing 2008) has an expressivity close to what we seek here. In ANML effects that happen at time intervals during an action duration can be specified. This can also be represented in HDDL 2.1 by using constraint semantics and dividing actions with intermediate effects into separate durative actions.

Some planners propose benchmarks where ordering constraints are delayed, e.g., FAPE (Dvorák et al. 2014). For instance in ANML it is possible to specify that an action must happen at least some amount of time after the end of a previous action. With the proposed syntax and semantics, such expressivity can be reached by using auxiliary tasks that decompose in a durative primitive task (of the desired duration) with no effects.

Time sampling represents another open question for this extension of HDDL with time. Basically, two approaches exist. Sampling can be either constant —when time is divided into regular-spaced discrete steps— or with variable time steps instantiated when effects and preconditions are applied. The latter can benefit from the Simple Temporal Problem formalism to model the temporal features of the plan and to include timed initial effects, and Interval Temporal Logic can be used to define truth of formulas relative to time intervals, rather than time points (Bresolin et al. 2014).

In order to be fully compatible with PDDL 3.0 features, the language HDDL 2.1 needs to include axioms and preferences, besides the associated parsing and validation tools. For this reason, the present work has to be seen as a baseline for the planning community to build upon. In fact, many applications require features that have been (on purpose) omitted in this paper. Most importantly, we did not explicitly report a syntax, and we did not allow to define delays with point algebra for ordering constraints. Other features are simply not detailed here, e.g., continuous effects. Such language elements are left as natural extension of this paper for future work.

References

- Abdulaziz, M.; and Koller, L. 2022. Formal Semantics and Formally Verified Validation for Temporal Planning. In *AAAI*, 9635–9643.
- Alford, R.; Kuter, U.; and Nau, D. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *IJCAI*, 1629–1634.
- Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. 2016. Hierarchical Planning: Relating Task and Goal Decomposition with Task Sharing. In *IJCAI*, 3022–3029.
- Barringer, H.; Kuiper, R.; and Pnueli, A. 1986. A really abstract concurrent model and its temporal logic. In *ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 173–183.
- Behnke, G. 2021. Block Compression and Invariant Pruning for SAT-based Totally-Ordered HTN Planning. In *ICAPS*, 25–35.
- Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; Pellier, D.; Fiorino, H.; and Alford, R. 2019. Hierarchical planning in the IPC. In *ICAPS Workshop on the International Planning Competition*.
- Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT-Totally-ordered hierarchical planning through SAT. In *AAAI*, volume 32.
- Behnke, G.; Höller, D.; and Biundo, S. 2019. Bringing order to chaos—A compact representation of partial order in SAT-based HTN planning. In *AAAI*, volume 33, 7520–7529.
- Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On Succinct Groundings of HTN Planning Problems. In *AAAI*, 9775–9784.
- Behnke, G.; Pollitt, F.; Höller, D.; Bercher, P.; and Alford, R. 2022. Making Translations to Classical Planning Competitive with Other HTN Planners. In *AAAI*, 9687–9697.
- Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning - One Abstract Idea, Many Concrete Realizations. In *IJCAI*, 6267–6275.
- Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An Admissible HTN Planning Heuristic. In *IJCAI*, 480–488.
- Bit-Monnot, A.; Smith, D.; and Do, M. 2016. Delete-Free Reachability Analysis for Temporal and Hierarchical Planning. In *ECAI*, volume 285, 1698–1699.
- Bresolin, D.; Della Monica, D.; Montanari, A.; Sala, P.; and Sciavicco, G. 2014. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theoretical Computer Science*, 560: 269–291.
- Broxvall, M.; and Jonsson, P. 2003. Point algebras for temporal reasoning: Algorithms and complexity. *Artif. Intell.*, 149(2): 179–220.
- Cushing, W.; Kambhampati, S.; and Weld, D. S. 2007. When is temporal planning really temporal? In *IJCAI*, 1852–1859.
- Dvorák, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and Acting with Temporal and Hierarchical Decomposition Models. In *ICTAI*, 115–121.
- Erol, K.; Hendler, J.; and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In *AAAI*, 1123–1128.
- Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Int. J. of Artif. Intell. Res.*, 20: 61–124.
- Georgievski, I.; Nizamic, F.; Lazovik, A.; and Aiello, M. 2017. Cloud Ready Applications Composed via HTN Planning. In *IEEE Conference on Service-Oriented Computing and Applications*, 81–89.
- Gerevini, A.; and Long, D. 2005. Plan constraints and preferences in PDDL3 - the language of the fifth International Planning Competition. Technical Report 2005-08-07, Department of Electronics for Automation (Imperial College).
- Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press. Chap. 4.
- Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5–6): 503–535.
- Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *AAAI*, 9883–9891.
- Höller, D.; and Bercher, P. 2021. Landmark Generation in HTN Planning. In *AAAI*, 11826–11834.
- Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2019. On Guiding Search in HTN Planning with Classical Planning Heuristics. In *IJCAI*, 6171–6175.
- McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
- Menif, A.; Jacopin, E.; and Cazenave, T. 2014. SHPE: HTN Planning for Video Games. In *Workshop on Computer Games*, 119–132.
- Milot, A.; Chauveau, E.; Lacroix, S.; and Lesire, C. 2021. Solving Hierarchical Auctions with HTN Planning. In *ICAPS Workshop on Hierarchical Planning*.
- Pellier, D.; and Fiorino, H. 2018. PDDL4J: a planning domain description library for Java. *J. Exp. Theor. Artif. Intell.*, 30(1): 143–176.
- Ramoul, A.; Pellier, D.; Fiorino, H.; and Pesty, S. 2017. Grounding of HTN Planning Domain. *Int. J. of Artif. Intell. Tools*, 26(5).
- Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artif. Intell.*, 170(12–13): 1031–1080.
- Schreiber, D. 2021. Lilotane: A Lifted SAT-based Approach to Hierarchical Planning. *J. of Artif. Intell. Res.*, 70: 1117–1181.
- Schreiber, D.; Pellier, D.; Fiorino, H.; and Balyo, T. 2019. Tree-REX: SAT-Based Tree Exploration for Efficient and High-Quality HTN Planning. In *ICAPS*, 382–390.
- Shivashankar, V.; Alford, R.; and Aha, D. 2017. Incorporating Domain-Independent Planning Heuristics in Hierarchical Planning. In *AAAI*, 3658–3664.
- Smith, D.; Frank, J.; and Cushing, W. 2008. The ANML Language. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling*.