Laying the Foundations for Solving FOND HTN Problems: Grounding, Search, Heuristics (and Benchmark Problems)

Mohammad Yousefi , **Pascal Bercher**
School of Computing, The Australian National University
mohammad.yousefi@anu.edu.au, pascal.bercher@anu.edu.au

Abstract

Building upon recent advancements in formalising Fully Observable Non-Deterministic (FOND) Hierarchical Task Network (HTN) planning, we present the first approach to find strong solutions for HTN problems with uncertainty in action outcomes. We present a search algorithm, along with a compilation that relaxes a FOND HTN problem to a deterministic one. This allows the utilisation of existing grounders and heuristics from the deterministic HTN planning literature.

1 Introduction

Hierarchical Task Network (HTN) planning allows the decomposition of complex tasks into smaller sub-tasks using predefined methods [Ghallab *et al.*, 2016; Bercher *et al.*, 2019]. The hierarchy imposed by the decompositions restricts which tasks can be *included* in the solutions, and more interestingly, *excluded* from them. This fine-grain control over the solutions makes HTN planning provably more expressive than the classical goal-satisfying approach [Erol *et al.*, 1996; Geier and Bercher, 2011; Alford *et al.*, 2015].

One of the prevailing assumptions in hierarchical planning systems is that the operating domain will react deterministically to the execution of actions [Erol *et al.*, 1994; Nau *et al.*, 2003; Bercher *et al.*, 2019; Höller *et al.*, 2021]; this is, however, rarely the case when dealing with real-world scenarios. While many planning systems in classical setting can deal with non-determinism [Cimatti *et al.*, 2003; Kissmann and Edelkamp, 2009; Mattmüller *et al.*, 2010; Fu *et al.*, 2011; Pereira *et al.*, 2022], to the best of our knowledge, there is no system to tackle HTN problems in the context of Fully Observable Non-Deterministic (FOND) environments. While hierarchical planners such as YoYo [Kuter *et al.*, 2005; 2009], and ND-SHOP2 [Kuter and Nau, 2004] permit hierarchy as advice on how to solve the underlying classical FOND planning problems, they do not leverage the inherent expressivity of HTN planning. In other words, plans following their policies are usually not refined according to the task hierarchy. Other treatments of uncertainty in hierarchical planning such as the UPOM planner [Patra *et al.*, 2020; 2021] which extends the Reactive Acting Engine (RAE) [Ghallab *et al.*, 2016] to include learning strategies for online planning in dynamically changing environments, emphasize "acting" [Ghallab *et al.*, 2014]. As a consequence, they use a non-standard operational model to achieve hierarchy [Patra *et al.*, 2019]. In this work, we aim to bridge this gap by building upon a theoretical framework proposed for solving FOND HTN problems [Chen and Bercher, 2022]. To achieve this goal we lay the entire groundwork necessary:

- A compilation from FOND HTN domains to their all-outcome-determinized relaxation in order to:
  - ground lifted FOND HTN domains using existing deterministic HTN planning grounders, and
  - derive heuristics from the deterministic planning literature (present and future),
- A heuristic search algorithm to solve (propositional) acyclic FOND HTN planning problems.

As a minor (non-scientific) contribution, we also extend the Hierarchical Domain Definition Language (HDDL) [Höller *et al.*, 2020a] to allow the expression of FOND HTN problems, and provide the first set of benchmark problems denoted in this language for our (or future) empirical evaluations.

Overall, we introduce (to the best of our knowledge) the first planning system for HTN planning problems capable of handling some level of uncertainty while strictly adhering to the hierarchy and retaining important features such as the partial ordering of tasks. We would like to note that the AO* algorithm requires acyclic search spaces, which is why we restrict our evaluation to acyclic problems. However, none of our other contributions (i.e., the all-outcome-determinization, the grounder, and heuristics) requires this restriction. We call our system $\mathcal{K}oala$, and it is accessible at `https://github.com/koala-planner/main`.

2 Problem Statement

Currently, two formalizations for FOND HTN (under standard HTN semantics) exist [Chen and Bercher, 2021; 2022]. The biggest difference between the solutions put forward in 2021 and those in 2022 is that the former only considers primitive task networks as solutions, i.e., a complete and finite refinement of the initial task network. Its underlying policy just tells when each primitive task is to be executed, based on the observed outcome. This has been called outcome-dependent *fixed-method policy* [Chen and Bercher, 2021]. The latter version, called *method-based policy*, differs significantly from this since the choice of methods is postponed until primitive tasks have been executed [Chen and Bercher, 2022]. As a consequence, solutions are complex policies, not fixed task networks. Any primitive action sequence executed is still a refinement of the initial task network, but which plan that is will be decided during runtime after observing non-deterministic action outcomes. For this flexible kind of solution, two variants exist: *Strong* and *strong cyclic policies*. Strong solutions guarantee finite execution time since they prohibit cycling. Strong cyclic solutions are allowed to exploit cyclic method definitions. As a consequence, while these plans still guarantee to end up in a primitive and executable refinement eventually, it could take arbitrarily long due to unfortunate action outcomes. The definitions resemble those found in FOND classical planning [Goldman and Boddy, 1996; Pryor and Collins, 1996; Cimatti *et al.*, 2003]. Our work aims at finding strong method-based policies for FOND HTN problems. As such, in this section, we provide the formalization introduced by Chen and Bercher [2022].

A FOND HTN domain contains all objects, relations, and valid operations on them.

Definition 1 (Planning Domain)**.** *A FOND HTN planning domain D is a tuple $D = \langle F, N_p, N_c, \delta, M\rangle$ where:*

- *F is a finite set of facts, and we define the set of states to be $S := 2^F$,*
- *$N := N_p \cup N_c$ is a finite set of primitive, N_p, and compound, N_c, task names where $N_p \cap N_c = \emptyset$,*
- *$\delta\colon N_p \to A$, with $A \subseteq S \times 2^{2^F \times 2^F}$ a finite set of non-deterministic actions, is the action mapping, and*
- *$M \subseteq N_c \times TN$ is a finite set of decomposition methods, where TN denotes the set of all possible task networks.*

We will sometimes use the term *action name* and *action* synonymously (as it is often clear from the context which one is meant), although the former first has to be mapped to the latter via the action mapping δ. Also, note that the terms *action* and *primitive task* are used synonymously.

For each primitive task name $p \in N_p$, δ defines an action $\delta(p) = (\mathit{pre}, \mathit{eff})$ where $\mathit{pre} \subseteq F$ is the set of preconditions that must hold in order to execute the action and $\mathit{eff} \subseteq 2^{2^F \times 2^F}$ consists of a set of effect pairs of the form $(\mathit{add}, \mathit{del})$, with add effects $\mathit{add} \subseteq F$ and delete effects $\mathit{del} \subseteq F$. For convenience, we also write $\mathit{pre}(p)$ and $\mathit{eff}(p)$ to refer to the preconditions and effects of an action $p \in N_p$. An action $p \in N_p$ is *executable* in state $s \in S$ (denoted by $\tau(p,s) = \top$) if and only if $\mathit{pre}(p) \subseteq s$.

The execution of an action $p \in N_p$ in a state $s \in S$ results in a non-deterministic state transition, defined as follows.

$$\gamma(p,s) = \begin{cases} \{(s \setminus \mathit{del}) \cup \mathit{add} \mid & \tau(p,s) = \top \\ \quad (\mathit{add}, \mathit{del}) \in \mathit{eff}(p)\} & \\ \mathit{undefined} & \tau(p,s) = \bot \end{cases}$$

The tasks that need to be accomplished and their desired ordering are organized in *task networks*. The definition is identical to deterministic HTN planning, as it simply organizes the tasks in a partially ordered fashion, which is independent of whether actions are deterministic or not.

Definition 2 (Task Network)**.** *A task network is a tuple $tn = \langle T, \prec, \alpha\rangle$ where T is a finite set of task IDs, $\prec$ is a partial order over T, and $\alpha\colon T \to N$.*

Task IDs are required to differentiate between multiple occurrences of the same tasks. Therefore, the ordering is defined on these IDs, and then α maps those IDs back to the actual tasks. Therefore, it does not matter which concrete identifiers are used for the IDs. This is captured by the definition of isomorphism (i.e., equivalence) of task networks:

Definition 3 (Isomorphism of Task Networks)**.** *Let $tn_1 = \langle T_1, \prec_1, \alpha_1\rangle$ and $tn_2 = \langle T_2, \prec_2, \alpha_2\rangle$ be two task networks. tn_1 and tn_2 are isomorphic (denoted $tn_1 \cong tn_2$) if there is a bijection $\sigma\colon T_1 \to T_2$ such that for all $t, t' \in T_1$ we have $\alpha_1(t) = \alpha_2(\sigma(t'))$, and further it holds $(t, t') \in \prec_1$ iff $(\sigma(t), \sigma(t')) \in \prec_2$.*

Note that our definition does not incorporate so-called method preconditions [Nau *et al.*, 2003], which provide state properties that have to hold in order to be allowed/capable of applying a method in a certain state during progression search. This is for two main reasons: (1) We base on the unaltered formalism by Chen and Bercher [2022] for which a comprehensive complexity investigation has been conducted, and (2) for the sake of simplicity because under a common interpretation of method preconditions, they can easily be compiled away by introducing a new artificial primitive task as the first task into the respective task network with the method's precondition as its precondition [Höller *et al.*, 2020a]. However, despite being considered standard, this interpretation is semantically doubtful in a partial order setting (while being perfectly fine in total order), as discussed by Höller *et al.* [2020a]. Existing grounders, such as the one presented by Behnke *et al.*, which has been employed in this paper, eliminate method preconditions.

Definition 4 (Decomposition Method)**.** *A (decomposition) method is a tuple $m = \langle c, tn\rangle$ where $c \in N_c$ and tn is a task network.*

We now define task decomposition in the context of progression search [Höller *et al.*, 2018; 2020b], where only compound tasks without predecessor (in their ordering constraints) can be decomposed.

Definition 5 (Progression Task Decomposition)**.** *A method $m = \langle c, tn\rangle$ decomposes a task network $tn_1 = \langle T_1, \prec_1, \alpha_1\rangle$ into $tn_2 = \langle T_2, \prec_2, \alpha_2\rangle$ if there exists an unconstrained task $t \in T_1$ (i.e., there is no $x \in T_1$ where $(x,t) \in \prec$) such that $\alpha_1(t) = c$ and there is a task network $tn' = \langle T', \prec', \alpha'\rangle$ with $tn' \cong tn$ where $T_1 \cap T' = \emptyset$. The task network tn_2 is defined as $tn_2 = \langle (T_1 \setminus \{t\}) \cup T', \prec' \cup \prec_D, ((\alpha_1 \setminus \{(t,c)\}) \cup \alpha')\rangle$ where $\prec_D$ is defined as follows.*

$$\begin{aligned} \prec_D = {} & \{(t_1,t_2) \mid (t,t_2) \in \prec_1, t_1 \in T'\} \cup \\ & \{(t_1,t_2) \mid (t_1,t_2) \in \prec_1, t_1 \neq t \wedge t_2 \neq t\} \end{aligned}$$

FOND HTN planning problems are defined as follows.

Definition 6 (FOND HTN Problem)**.** *A FOND HTN planning problem P is a tuple $\langle D, tn_I, s_I\rangle$ where D is its planning domain and $s_I \in 2^F$ and tn_I are its initial state and task network, respectively.*

We still need to define the set of solutions for a planning problem. Whereas in deterministic HTN planning [Geier and Bercher, 2011; Bercher *et al.*, 2019] (or even in FOND HTN planning with fixed-methods [Chen and Bercher, 2021]), a solution is simply a fixed task network, we – for the more flexible method-based FOND solutions – require a more complex data structure, called *method-based policy* [Chen and Bercher, 2022]. This is a mapping from a current task network plus its state to either a primitive task (the next task to execute) or a compound task and an appropriate method (how to decompose the next compound task). After formally defining this (partial) function, we provide the criteria *which* of these policies are actually solutions.

Definition 7 (Method-Based Policy). *A (method-based) policy π is a partial function $\pi : TN \times S \to T \times M'$ where T is the union of the sets of (corresponding) tasks in the task networks of TN, and $M' = M \cup \{\epsilon\}$ where ϵ is the reserved symbol for execution of primitive tasks.*

For a policy π to be well-defined, we have to impose that for all (tn_1, s_1) and (tn_2, s_2) in the domain of π, if $s_1 = s_2$ and $tn_1 \cong tn_2$ then $tn_1 = tn_2$.

Following a policy we get an execution trace, i.e., a sequence of task networks and states – the search nodes processed by an HTN progression planner. This is formally captured by what was called *execution structure* [Chen and Bercher, 2022]. In the following definition, for a task network $tn = \langle T, \prec, \alpha\rangle$ we use the shorthand $tn \setminus t$ to denote $\langle T \setminus \{t\}, \{(x, y) \mid (x, y) \in \prec \text{ and } x, y \neq t\}, \alpha \setminus \{(t, \alpha(t)\}\rangle$.

Definition 8 (Execution Structure). *Let P be a FOND HTN problem. Let $L = \langle U, V\rangle$ be a tuple where $U \subseteq TN \times S$ and $V \subseteq (TN \times S) \times (T \times M \cup \{\epsilon\}) \times (TN \times S)$ be the minimal sets satisfying $(tn_I, s_I) \in U$ and:*

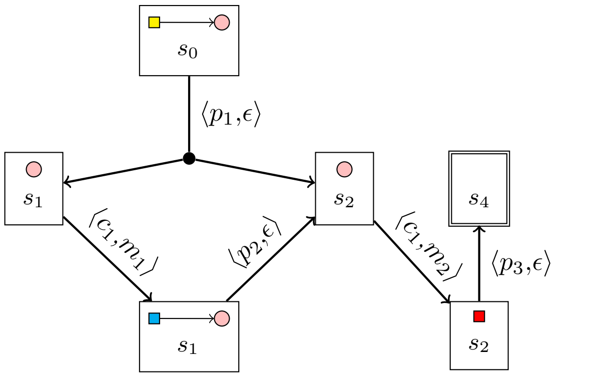
If $(tn, s) \in U$ and $\pi(tn, s) = (t, m)$, then

- *if t is primitive, then for all $s' \in \gamma(t, s)$ we have $(tn \setminus t, s') \in U$ and $((tn, s), (t, \epsilon), (tn \setminus t, s')) \in V$, and*
- *if t is compound, then we have $(tn', s) \in U$ and $((tn, s), (t, m), (tn', s)) \in V$, where tn' is the result of decomposing task t in tn with method m.*

The execution structure induced by a policy π is the tuple $[L] = \langle [U], [V]\rangle$ where $[U]$ is the set U quotient out by the relation $(tn, s) \sim (tn', s)$ iff tn and tn' are isomorphic and similarly for $[V]$ where $((tn_1, s), (t, m), (tn_2, s)) \sim ((tn_1', s), (t, m), (tn_2', s))$ iff tn_1 and tn_2 are isomorphic and tn_1' and tn_2' are isomorphic.

The execution structure $[L] = \langle [U], [V]\rangle$ induced by a policy π can be viewed as a directed graph where each node $u \in [U]$ is a FOND HTN (sub-)problem with its own task network and state, and each transition $v \in [V]$ is an instruction on which task (either primitive or compound) must be performed in order to step toward solving the problem – see Fig. 1 for an example. This simply corresponds to all task networks a progression planner would create while following all possible outcomes of a given policy.

Given an execution structure L, we refer to (tn_I, s_I) as the initial node, any node that has no outgoing edge as *terminal*, and terminal nodes (tn, s) where tn contains no task (also denoted by $tn_\emptyset$) as a *goal* node.



s_0
$\langle p_1,\epsilon\rangle$
s_1
s_2
$\langle c_1,m_1\rangle$
$\langle p_2,\epsilon\rangle$
$\langle c_1,m_2\rangle$
s_4
$\langle p_3,\epsilon\rangle$
s_1
s_2

Figure 1: An illustration of the execution structure of a (strong) policy π for problem $P = \langle D, p_1 \to c_1, s_0\rangle$, where D is the domain (not formally provided) and $p_1 \to c_1$ is shorthand for a totally ordered task network with a primitive task p_1 having two (non-deterministic) effects and a compound task c_1. Nodes and edges correspond to those in execution structure. Edge labels show the output of the policy for the respective parent node.

We can now define the solution criteria for FOND HTN problems. Note that we define, purely for the sake of completeness, all three standard kinds of solutions: weak, strong, and strong cyclic ones – but in this paper, we are only concerned with finding strong solutions.

Definition 9 (Weak, Strong, and Strong Cyclic Solutions). *Let P be a FOND HTN problem. A policy π with execution structure $L = \langle U, V\rangle$ is a weak solution to P if at least one terminal node of L is a goal node, a strong cyclic solution to P if every terminal node of L is a goal node, and a strong (acyclic) solution to P if L is acyclic, finite, and every terminal node of L is a goal node.*

For *weak solutions*, Chen and Bercher [2022] also demand finiteness. Although we are not concerned with weak solutions, we removed this constraint as it would exclude some policies from considered weak although they admit a solution given favorable outcomes.

3 All Outcome Determinization

In this section, we show how a FOND problem can be turned into a deterministic problem serving as problem relaxation, meaning that the set of solutions of the deterministic problem is a superset of the (traces of the) solutions of the non-deterministic one. We believe that this transformation has multiple practical applications in FOND HTN planning, two of which are proposed in this paper. There might also be further theoretical applications as evidenced by one result relying on it already. Chen and Bercher [2021] used it for the less flexible HTN formalization (of fixed-method policies) to prove their Lemma 4.3 which relates this problem relaxation to weak solutions. Since we now build on it in a more significant way we present it in more technical detail and state important theoretical properties.

A core idea behind the transformation is the so-called *all-outcome-determinization*, which is well known in non-hierarchical planning and has been used, among others, in FF-replan for probabilistic planning problems [Yoon *et al.*, 2007]. The transformation extends these ideas to make it work in the hierarchical setting.

FOND problems are often represented as an AND/OR graph (also known as a hypergraph) [Bonet and Geffner, 2000], which is a generalization of a canonical graph that allows multiple connections using a single edge.

Definition 10 (Hypergraph)**.** *A hypergraph is a tuple $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times 2^V$ is a set of connectors. A k-connector is a connector in a hypergraph G such that $(v, V') \in E$ where $|V'| = k$.*

Intuitively, the transformation converts all k-connectors in a search space to k 1-connectors, thus sacrificing the "AND" semantics and relaxing it into an "OR" semantics.

Formally, for any non-deterministic primitive task $p \in N_p$ (i.e., we require $|\mathit{eff}(p)| \geq 2$) we construct a set of deterministic actions $A_p := \{(\mathit{pre}(p), \mathit{add}, \mathit{del}) \mid (\mathit{add}, \mathit{del}) \in \mathit{eff}(p))\}$ and redefine p to now be a compound task with $|A_p|$ methods where each method decomposes the task into one of the actions in A_p. We use the notation N_{nd} to refer to non-deterministic action names in a domain. Formally, $N_{nd} := \{p \mid p \in N_p,\ |\mathit{eff}(p)| \geq 2\}$. Furthermore, without loss of generality, we assume the existence of an injective labelling function, $\beta\colon\ N_{nd} \times (2^F \times 2^F) \to \mathit{Sym}$, that, given a non-deterministic action name and one its effects, provides a unique symbol to be used as a name, where $\mathit{Sym} \cap N = \emptyset$. More precisely, if $p \in N_{nd}$ and $e \in \mathit{eff}(p)$, then $\beta(p, e) = s$ for some $s \in \mathit{Sym}$, acting as a new task name.

Given that we now deal with deterministic domains, the definition of action mappings is slightly altered. Now, an action mapping $\delta : N \to 2^F \times 2^F \times 2^F$ maps to a 3-tuple.

Definition 11 (All-Outcome-Determinized Relaxation)**.** *Let $D = \langle F, N_p, N_c, \delta, M\rangle$ be a FOND HTN domain and N_{nd} be its set of non-deterministic action names. Then, the all-outcome-determinized relaxation of D is $D' = \langle F, N'_p, N'_c, \delta', M'\rangle$, a deterministic HTN domain, where:*

- $N'_p = (N_p \setminus N_{nd}) \cup \{\beta(p, e) \mid p \in N_{nd}, e \in \mathit{eff}(p)\ \}$
- $N'_c = N_c\ \cup\ N_{nd}$,
- *For all $n \in N$ we define $\delta'(n) = (\mathit{pre}(n), \mathit{add}, \mathit{del})$, where $e = (\mathit{add}, \mathit{del})$ with $e \in \mathit{eff}(p)$, and*

  either $n \in N_p \setminus N_{nd}$ *(n is already deterministic),*
  or $p \in N_{nd}$, $\beta(p, e) = n$ *(new action per effect).*
- $M' = M \cup \{(p, \langle\{p\}, \emptyset, \{(p, \beta(p, e))\}\rangle) \mid p \in N_{nd}, e \in \mathit{eff}(p)\}$

The corresponding planning problem can be defined without further changes:

Definition 12 (All-Outcome-Determinized Problem)**.** *Let $\langle D, s_I, tn_I\rangle$ be a FOND HTN problem and D' be the determinization of D. Then, the all-outcome determinization of P is the (deterministic) problem $P' = \langle D', s_I, tn_I\rangle$.*

As mentioned before, the core idea behind this encoding was introduced before, where polynomial runtime was stated [Chen and Bercher, 2021, Lemma 4.3]. Since this useful problem encoding was a bit "hidden" in a (short) proof, we make aware of its formal properties more prominently, and provide tighter properties as well.

Proposition 1. *Let P be a FOND HTN planning problem and P' its deterministic version. Then, P' can be computed in time linear to the count of non-deterministic effects.*

Our definitions are provided, for the sake of simplicity, on a fully propositional level. However, we would like to stress that the transformation also works on a lifted level, and also requires only linear runtime to compute. This is because one can still turn a single non-deterministic action with n effects into n deterministic actions (with its accompanying n methods) – whether these actions are ground or lifted is irrelevant.

We now prove a novel result that will prove useful for several possible applications, such as its usage for heuristics and grounding: *The transformation is a problem relaxation.*

Theorem 1. *Let P be a FOND HTN problem and P' be its all-outcome-determinization. Let π be a strong or strong cyclic policy. Then, each execution trace $\overline{tr}_1 = \langle(tn_1, s_1), (t_1, m_1)\rangle, ..., \langle(tn_k, s_k), \emptyset\rangle$ with $(tn_1, s_1) = (tn_I, s_I)$ and $tn_k = tn_\emptyset$ following π corresponds to a sequence of progressions $\overline{tr}_2 = \langle(tn_1, s_1), (t'_1, m'_1)\rangle, ..., \langle(tn'_{k'}, s_{k'}), \emptyset\rangle$ with $|\overline{tr}_2| = k' \geq k = |\overline{tr}_1|$ for P'. More precisely: Let $\bar{t}_1 = p_1, \ldots, p_k$ be the primitive task (name) sequence induced by $\overline{tr}_1$. Then, the task name sequence $\bar{t}_2 = p'_1, \ldots, p'_k$ induced by $\overline{tr}_2$ is a (classical) solution to P' and it holds:*

- *If $p_i \in N_p \setminus N_{nd}$, then $p'_i = p_i$.*
- *If $p_i \in N_{nd}$ and its effect $e_{j_i} \in \mathit{eff}(p_i)$ produced the respective successor state, then $p'_i = \beta(p_i, e_{j_i})$.*

Proof. By construction, every method for a compound task in P exists in P' and is hence still applicable there. The same applies to primitive *deterministic* actions. The only case to look closely at is the application of a *non-deterministic* action. If however, the trace contains a progression from a non-deterministic action p to a new state s, then this is due to some specific non-deterministic effect $e \in \mathit{eff}(p)$. Then, by construction, the action $\beta(p, e)$ leads to s. $\square$

This implies that whenever there exists a weak, strong, or strong cyclic solution, the all-outcome-determinized problem also has a solution, which enables us to use it as a problem relaxation, as exploited by, e.g., heuristics and a grounder.

4 Grounding Procedure

While most planning systems – both in classical as well as in HTN planning – work on a propositional model, the domains are commonly expressed in a lifted representation with variables to abstract over concrete object instances, such as PDDL for classical models [Fox and Long, 2003] or HDDL for (deterministic) hierarchical ones [Höller *et al.*, 2020a]. The extension is conceptually very simple: every fact, task, and method now has a sequence of (typed) variables which may be instantiated by a (type-conforming) constant. An example, provided in the HDDL syntax, is provided in Fig. 2. Examples for valid groundings are provided later.

The process of turning a lifted model into a propositional one is called grounding. A ground domain can be obtained by instantiating all variables. However, in practice, naive instantiation would result in an exponential number of propositions.

Thus, most planners utilize a more sophisticated grounding procedure that exploit reachability analyses and in case of HTN planning further exploit the task hierarchy in a non-trivial manner [Behnke *et al.*, 2020].

In order to ground a FOND HTN problem, instead of creating an entirely new approach to deal with the additional challenges of dealing with non-determinism in the context of HTN planning, we propose to exploit the all-outcome-determinization and thereby, existing – and future – HTN grounding technology. Our algorithm resembles the approach by Scala and Vallati [2021] that infers a grounding for PDDL+ using classical grounders. The procedure is as follows: First, Compute the all-outcome-determinized relaxation as defined in Def. 11 on the *lifted* model, then use an existing oracle (i.e., any deterministic HTN problem grounder) to obtain a ground representation of the determinized problem. And finally, re-infer a valid (over-approximation) grounding of the original model based on the grounding of the determinized model. When we say "valid" grounding, then we mean any grounding that *(i)* is in line with all constraints (this is trivial, e.g., respecting type constraints of variables and constants) and, more importantly, *(ii)* no grounding is missing, i.e., that any ground instance (of any domain element, like action, method, etc.) that could be part of a solution is part of the grounding.

Before we provide our procedure in pseudo code, we explain it using an illustrative example that also semi-informally introduces the formalization as required for our purposes.

Consider the following non-deterministic action, `Lift`, expressed in our extension of HDDL (Fig. 2). The new "`oneof`" keyword allows us to define mutual exclusion on effects, mirroring the approach adopted in extending PDDL for non-deterministic effects [Bertoli *et al.*, 2003].

```
(:action Lift
 :parameters (?h - hoist ?c - crate)
 :precondition (reachable ?h ?c)
 :effect (oneof
                (lifting ?h ?c)
                (not (lifting ?h ?c))
         )
)
```

Figure 2: Non-deterministic action in extended HDDL

To go through the grounding process, assume that we have two hoists (`h1` and `h2`) and also two crates (`c1`, `c2`). So in total, four groundings exist in the worst-case, leading to the four ground actions `Lift[h1, c1]`, `Lift[h1, c2]`, `Lift[h2, c1]`, and `Lift[h2, c2]`. The all-outcome-determinization (run on the lifted model) will create two lifted actions `Lift-eff1`$[?h, ?c]$ and `Lift-eff2`$[?h, ?c]$, and so we will get at most eight ground actions for the determinized problem (four for each).

Suppose the black-box grounder (for deterministic planning) rules out two groundings for `Lift-eff1`$[?h, ?c]$ and even three for `Lift-eff2`$[?h, ?c]$, resulting in the following remaining ground instances:

- `Lift-eff1[h1,c2]`, `Lift-eff1[h2,c2]`
- `Lift-eff2[h2,c2]`
- we also have 2 ground compound tasks (`Lift[h1,c2]` and `Lift[h2,c2]`), and 3 ground methods (one for `Lift[h1,c2]` and two for `Lift[h2,c2]`, one for each reachable effect)

At this point we *could* stop and use the two identified groundings `[h1,c2]` and `[h2,c2]` for the ground non-deterministic action. However, we can do even better.

Notice that the grounding `[h1,c2]` only appears in one of the two effects of our non-deterministic action. However, according to the definition of strong (or strong cyclic) policies, every outcome must eventually lead to a solution. Due to Thm. 1 we know that the corresponding action would be used in some solution. Since we know that this is not the case (as the respective grounding wasn't computed), we can conclude that `Lift[h1,c2]` cannot be part of any strong (or strong cyclic) policy. We can hence discard this grounding and identified only a single (of four possible) one: `Lift[h2,c2]`. The procedure is summarized in Alg. 1.

Algorithm 1: FOND Grounding Procedure

```
Input: $P$, a lifted FOND HTN planning problem;
       $P'$, its (lifted) all-outcome-determinisation;
       $G'$, a valid grounding of $P'$
Output: $G$, a valid grounding of $P$
1 Let $G$ be an empty set of groundings.
2 Let $N_{nd}$ be the set of names of actions with more than
    one effect in the domain of $P$.
3 foreach $C[o_1, o_2, ..., o_k]$ in ground compound tasks of
    $G'$ do
4     if $C \in N_{nd}$ then
5         If the number of ground methods in $G'$ that
            decompose $C[o_1, o_2, ..., o_k]$ is equal to $|eff(n)|$
            in the original lifted domain, add $[o_1, o_2, ..., o_k]$
            to $G$ as a valid grounding for the non-determi-
            nistic action that corresponds to $C$.
6     else
          // a "normal" compound task
7         add $C[o_1, o_2, ..., o_k]$ and all of its methods to $G$.
8 Copy all remaining groundings in $G'$ (those not
    covered in the previous loop, i.e., actions) to $G$.
9 return $G$
```

From the arguments above, especially by exploiting Thm. 1, we can conclude:

Proposition 2. *The grounding procedure described in Alg. 1 is correct, i.e., no grounding of tasks, methods, etc. that could be part of any strong (or strong cyclic) policy will be missing in its output.*

5 Search Algorithm

While there are several algorithms to search for a solution in a hypergraph [Bonet and Geffner, 2005], we propose to use the AO* algorithm [Nilsson, 1982; Edelkamp and Schrödl, 2012] to find a strong solution to a FOND HTN problem (cf.

Alg. 2). This heuristic search algorithm explores an acyclic hypergraph gradually extending the best (strong) policy.

Algorithm 2: AO* Search

```
Input: an acyclic FOND HTN planning problem
       $P = \langle D, tn_I, s_I \rangle$.
Output: a solution to problem $P$ or "No Solution"
1  $V \leftarrow \{\langle tn_I, s_I \rangle\}, \quad E \leftarrow \emptyset, \quad G \leftarrow (V, E)$
2  $cost(\langle tn_I, s_I \rangle) \leftarrow 0,\ label(\langle tn_I, s_I \rangle) \leftarrow$ "Ongoing"
3  while $label(\langle tn_I, s_I \rangle) \notin \{$"Solved", "Failed"$\}$ do
4      $\pi_p \leftarrow G$
5      if there are markers in $G$ then
6          $\pi_p \leftarrow$ the marked subgraph of $G$
7      $n \leftarrow$ a random unexpanded node of $\pi_p$
8      $succ(n) \leftarrow expand(n)$
9      foreach $SN_i \in succ(n)$ do
10         foreach $\langle tn, s \rangle \in SN_i$ do
11             if $\langle tn', s \rangle \in V$ for some $tn' \cong tn$ then
12                 Change the pointer to $\langle tn, s \rangle$ in $SN_i$ to its
                     equivalent search node in $V$.
13             else
14                 $cost(\langle tn, s \rangle) \leftarrow h(\langle tn, s \rangle)$
15                 $V \leftarrow V \cup \{\langle tn, s \rangle\}$
16         $E \leftarrow E \cup \{(n, SN_i)\}$
17     $W \leftarrow \{n\}$
18     while $W \neq \emptyset$ do
19         $q \leftarrow$ a node $x \in W$ such that $x$ does not have any
             descendants in $W$
20         if isTerminal?(q) then
21             $label(q) =$ "Failed"
22             if $isGoal?(q)$ then $label(q) =$ "Solved"
23             $W \leftarrow W \cup predecessors(q)$
24         else
25             if all successors of $q$ are terminal then
26                 $label(q) \leftarrow$ "Failed"
27                 if there is a "Solved" successor then
28                     $label(q) \leftarrow$ "Solved"
29                 $W \leftarrow W \cup predecessors(q)$
30             $best \leftarrow \text{argmin}(\{cost(x) \mid x \in succ(q)\})$
31             $cost(q) \leftarrow cost(best)$
32             $mark(q, best)$
33 if $label(\langle tn_I, s_I \rangle) =$ "Solved" then
34     return $\pi_p$
35 else
36     return "No solution"
```

The algorithm is a loop consisting of two steps: Lines 5–16 expand the frontier of the search, and Lines 17–32 do a backward cost revision. Note that, in Line 11 when checking for equivalence between a new search node $\langle tn, s \rangle \in SN_i$ and explored nodes V, in the case where states are equal, we have to check for isomorphism between task networks (cf. Def. 8). This is a costly operation (it is not yet known whether a poly-time procedure exists), and in order to avoid a complete search for bijection, we first use the over-approximation algorithm introduced by Höller and Behnke [2021] to quickly rule out task networks that cannot be isomorphic. The expansion step performed in Line 8 is presented in Alg. 3.

Algorithm 3: Node Expansion

```
Input: task network $tn$, and state $s$
Output: all possible progressions of $tn$
1  $(U_p, U_c) \leftarrow unconstrained(tn),\ SN \leftarrow \emptyset$
2  foreach $t \in U_c$ do
3      foreach $m \in t.methods$ do
4          $tn' \leftarrow tn.decompose(t, m)$
5          $SN \leftarrow SN \cup \{\{\langle tn', s \rangle\}\}$
6  foreach $t \in U_p$ do
7      foreach $a \in N_p$ do
8          if $\tau(a, s) = \top$ then
9              $\langle tn', S' \rangle \leftarrow tn.apply(a)$
10             $SN \leftarrow SN \cup \{\{\langle tn', s' \rangle \mid s' \in S'\}\}$
11 return $SN$
```

The AO* algorithm is known to terminate with an optimal solution (defined as the minimum sum of transition costs) under the following conditions [Nilsson, 1982]: *(i)* There is a solution graph from the initial node to a set of goal nodes, *(ii)* for all search nodes, $h(n) \leq h^*(n)$ where $h^*(n)$ is the actual distance of node n to its nearest goal node (admissibility), and *(iii)* for all search nodes, $h(n) \leq c + h(n_1) + ... + h(n_k)$ where $n_1, ..., n_k$ are the successors of n (in the same connector) and c is the connector cost (monotonicity).

For the first condition it is known for deterministic HTN planning that a goal node, if one exists, can be reached via a sequence of progressions [Alford *et al.*, 2012; Höller *et al.*, 2020b]. Alg. 3 is an adaptation of progression-based expansion to non-deterministic settings where line 10 forces the algorithm to solve for all states that result from executing a non-deterministic action. Criteria *(ii)* and *(iii)* would have to be ensured by the deployed heuristic.

6 A Generic Approach to Use Deterministic HTN Heuristics in FOND HTN Planning

Even in deterministic HTN planning, there are only three heuristics available as of now [Bercher *et al.*, 2017; Höller and Bercher, 2021; Höller *et al.*, 2020]. Although they are all specifically designed for HTN planning, the most successful approach is one that re-uses existing heuristics for (non-hierarchical) *classical planning* [Höller *et al.*, 2018; 2020b]. It relaxes a given HTN search node into a classical planning problem and then deploys an existing classical heuristic to it. Our proposal for obtaining FOND HTN heuristics bases on the very same idea.

Given a FOND HTN search node, we turn it into a deterministic one by using the all-outcome-determinization put forward in Section 3. Then, instead of developing novel heuristics for FOND HTN planning, we can use any from deterministic HTN planning. By transitivity (i.e., by using the encoding by Höller *et al.*, 2018; 2020b), we can even use *classical* heuristics to guide search in the FOND HTN setting.

Theorem 2. *The all-outcome-determinization preserves safety, goal-awareness, and admissibility of deterministic HTN heuristics.*

Proof. Let h be a deterministic HTN heuristic, P be a FOND HTN problem, and P' be its all-outcome-determinized relaxation. Safety means that if $h(n) = \infty$, then $h^*(n) = \infty$. This follows directly from Theorem 1, as it implies that P' is a problem relaxation. Goal awareness means that if n is a solution, we get $h(n) = 0$. If a FOND search node is a solution, it is by definition empty, in which case the encoding will also be empty. We thus get $h(n) = 0$ since h is goal-aware by assumption. Admissibility means that $h(n) \leq h^*(n)$ for all search nodes n. This also follows from Theorem 1 (combined with h being admissible by assumption). □

7 Experimental Results

We evaluated our system using the following configuration: We used our proposed grounding procedure (Alg. 1) with the deterministic HTN grounder by Behnke *et al.* [2020] as our oracle. For the required heuristic, we again used our proposed approach that makes deterministic hierarchical planning heuristics available to the FOND setting using a cascade of problem relaxations. More precisely, we use the RC^{Max}, RC^{FF}, and RC^{Add} heuristics [Höller *et al.*, 2018; 2020b] on the all-outcome-determinization, which in turn relax the respective deterministic HTN problem into a classical problem and then use the classical planning heuristics Max [Bonet and Geffner, 1999], FF [Hoffmann and Nebel, 2001], and Add [Bonet and Geffner, 2001], respectively. We configured our heuristic to use action costs of one for each action from the HTN model (including those encoding method preconditions) and zero for each of the actions compiled by the RC encoding. Heuristics thus estimate final plan length (including method preconditions). Our implementation is a new system, called $\mathcal{K}$*oala*, implemented in Rust (see Introduction for a link).

For the benchmarks, we have created five novel FOND domains with 15 instances each[1], all based on the hierarchical track of the International Planning Competition (IPC) 2020. All of them are totally ordered, but only two domains have exclusively totally ordered initial task networks, making the problems totally ordered (namely Childsnack and Depot), so the rest is partially ordered. All domains are acyclic. We use the Agile IPC Score[2] to assess our planner. We ran our experiments on a machine running Ubuntu 22.04 with one CPU core (Intel i7 13700), 8GB of RAM, and a 30 minute cut-off time. A summary of the result is provided in Table 1. All failed instances are a result of timeout. The characteristics (and modifications) of the domains are as follows. In our version of **Depots**, hoists are human-operated, and junior operators may fail to load/unload a truck. In case of failure, a supervisor must be called to assist in the operation. The main characteristic of this domain is that unfavorable outcomes are fixed in a few steps. In the **Satellite** domain, the phenomenons of interest may not be stationary. In the latter case, the motion trajectory must be calculated and compensated for by the satellite. This domain illustrates cases where the uncertainty can be eliminated by taking uncertainty-reducing actions such as observing the movement pattern at execution time. **Child-Snack** is modified so the tray may get dirty and require washing in the kitchen. In the new **Rover** domain, the camera may not be able to take an image. Thus, the rover must report this incident for new instructions. Finally, we have modified **Transport** such that the package recipient may not be at home, in which case the truck driver(s) cannot unload the package. Thus, unfavorable outcomes lead to the accumulation of packages in trucks, triggering capacity constraints.

[1]All of the problems, and their respective domains are available at `https://github.com/koala-planner/domains`.

[2]Given a maximum of T seconds to solve a problem, the score for a problem that is solved within 1 second is 1, within t seconds ($1 < t \leq T$) is $1 - \frac{\log t}{\log T}$, and 0 otherwise. The final score for a domain is the sum of scores for all of the problems in that domain.

Domain	Heuristic	IPC Score	Coverage	Avg. Execution Structure	
				# of Nodes	CP Length
Depots	RC^{Add}	6.92	10 of 15	150.80	127.20
	RC^{FF}	6.99	10 of 15	150.80	127.90
	RC^{Max}	7.26	10 of 15	149.70	126.20
Rover	RC^{Add}	5.16	6 of 15	28.00	21.83
	RC^{FF}	5.00	5 of 15	23.80	17.80
	RC^{Max}	5.00	5 of 15	23.00	17.00
ChildSnack	RC^{Add}	1.66	3 of 15	64.00	45.00
	RC^{FF}	1.59	3 of 15	64.00	45.00
	RC^{Max}	1.66	3 of 15	64.00	45.00
Satellite	RC^{Add}	5.92	7 of 15	33.86	16.86
	RC^{FF}	4.16	7 of 15	33.71	16.86
	RC^{Max}	5.92	7 of 15	32.57	15.86
Transport	RC^{Add}	3.48	5 of 15	400.60	45.40
	RC^{FF}	4.04	5 of 15	99.20	33.80
	RC^{Max}	3.32	4 of 15	94.50	31.75

Table 1: Evaluation results. CP is an abbreviation for Critical Path, which is the number of nodes of the policy's longest path minus one. This corresponds to the number of action and method applications (and includes the number of used methods with preconditions, since those are compiled into actions and thus part of the policy).

As for the first step, grounding was always possible within at most 8 seconds (including the time that PANDA's grounder needed for a determinized problem), also for the unsolved ones. As a matter of fact, most problems were grounded in less than a second. More interestingly, the overhead added by our grounding procedure on top of that of the black-box grounder was at most 1 second, which is thus negligible. This is hardly surprising because investigating Alg. 1 shows that its runtime is linear in the number of groundings produced by the deterministic base-grounder. So, in total, we showed that our procedure can successfully make use of deterministic grounders with almost zero overhead.

The next step, i.e., the search, has more interesting results. The performance of the algorithm varies on two axes: the intrinsic difficulty of a problem in a particular domain (i.e., branching factor and solution depth) and the choice of heuristic function for the relaxed problem. The intrinsic difficulty is captured by the IPC Score and the number of solved instances as reported in Table 1. The details of our experimental setup is available online [Yousefi and Bercher, 2024]. The choice of a heuristic function (in our case RC^{Max}, RC^{FF}, and RC^{Add}) has an impact on the performance of the algorithm. There were cases where the algorithm with one heuristic was able to solve a particular problem, while it could not with the others. This is noticeably reflected in the average solution structure in the transport domain where a particularly difficult instance was solved only with the Add heuristic. However, the total number of solved problems for each domain were almost the same. The time needed to compute the RC^{Add} and RC^{Max} heuristics are significantly less, allowing the algorithm to reach more depth in the search graph and explore more nodes. However, the cost of this property is usually decreased accuracy.

In conclusion, our investigation suggests that even though FOND HTN planning in theory seems intractable (more specifically of higher complexity classes than their deterministic counterparts [Chen and Bercher, 2021; 2022]), a great deal of problems can be solved within the practical constraints of the IPC.

8 Conclusion

We addressed a critical gap in the field of HTN planning by laying the foundational groundwork for tackling problems in non-deterministic domains. Since there was no approach available to solve FOND HTN problems, we had to provide the entire infrastructure, including a grounder, search procedure, and heuristics. Specifically, we provided a problem encoding that relaxes FOND HTN problems to deterministic ones that serves as the basis for using existing deterministic grounders in the FOND setting as well as existing HTN or classical heuristics. Due to non-existing competitors, we could not compare against existing approaches, but our evaluation showed that grounding is efficient and problems of realistic size can be solved within the constraints of the IPC.

Acknowledgements

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

[Alford *et al.*, 2012] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana S. Nau. HTN problem spaces: Structure, algorithms, termination. In *Proc. of the 5th SoCS*, pages 2–9. AAAI Press, 2012.

[Alford *et al.*, 2015] Ron Alford, Pascal Bercher, and David W. Aha. Tight bounds for HTN planning. In *Proc. of the 25th ICAPS*, pages 7–15. AAAI Press, 2015.

[Behnke *et al.*, 2020] Gregor Behnke, Daniel Höller, Alexander Schmid, Pascal Bercher, and Susanne Biundo. On succinct groundings of HTN planning problems. In *Proc. of the 34th AAAI*, pages 9775–9784. AAAI Press, 2020.

[Bercher *et al.*, 2017] Pascal Bercher, Gregor Behnke, Daniel Höller, and Susanne Biundo. An admissible HTN planning heuristic. In *Proc. of the 26th IJCAI*, pages 480–488. IJCAI, 2017.

[Bercher *et al.*, 2019] Pascal Bercher, Ron Alford, and Daniel Höller. A survey on hierarchical planning – one abstract idea, many concrete realizations. In *Proc. of the 28th IJCAI*, pages 6267–6275. IJCAI, 2019.

[Bertoli *et al.*, 2003] Piergiorgio Bertoli, Alessandro Cimatti, Ugo Dal Lago, and Marco Pistore. Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *ICAPS workshop on PDDL, informal proceedings*, pages 15–24, 2003.

[Bonet and Geffner, 1999] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *Proc. of the 5th ECP*, pages 360–372. Springer, 1999.

[Bonet and Geffner, 2000] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. of the 5th AIPS*, pages 52–61. AAAI Press, 2000.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *AIJ*, 129:5–33, 2001.

[Bonet and Geffner, 2005] Blai Bonet and Héctor Geffner. An algorithm better than AO*? In *Proc. of the 20th AAAI*, pages 1343–1347. AAAI Press, 2005.

[Chen and Bercher, 2021] Dillon Chen and Pascal Bercher. Fully observable nondeterministic HTN planning – formalisation and complexity results. In *Proc. of the 31st ICAPS*, pages 74–84. AAAI Press, 2021.

[Chen and Bercher, 2022] Dillon Z. Chen and Pascal Bercher. Flexible FOND HTN planning: A complexity analysis. In *Proc. of the 32nd ICAPS*, pages 26–34. AAAI Press, 2022.

[Cimatti *et al.*, 2003] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *AIJ*, 147(1):35–84, 2003.

[Edelkamp and Schrödl, 2012] Stefan Edelkamp and Stefan Schrödl. *Heuristic Search: Theory and Applications.* Morgan Kaufmann, 2012.

[Erol *et al.*, 1994] Kutluhan Erol, James A. Hendler, and Dana Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. of the 2nd AIPS*, pages 249–254. AAAI Press, 1994.

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

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR*, 20:61–124, 2003.

[Fu *et al.*, 2011] Jicheng Fu, Vincent Ng, Farokh B. Bastani, and I-Ling Yen. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proc. of the 22nd IJCAI*, pages 1949–1954. AAAI Press, 2011.

[Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *Proc. of the 22nd IJCAI*, pages 1955–1961. AAAI Press, 2011.

[Ghallab *et al.*, 2014] Malik Ghallab, Dana Nau, and Paolo Traverso. The actor's view of automated planning and acting: A position paper. *AIJ*, 208:1–17, 2014.

[Ghallab *et al.*, 2016] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[Goldman and Boddy, 1996] Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In *Proc. of the 3rd AIPS*, pages 110–117. AAAI Press, 1996.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Berhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Höller and Bercher, 2021] Daniel Höller and Pascal Bercher. Landmark generation in HTN planning. In *Proc. of the 35th AAAI*, pages 11826–11834. AAAI Press, 2021.

[Höller *et al.*, 2020] Daniel Höller, Pascal Bercher, and Gregor Behnke. Delete- and ordering-relaxation heuristics for HTN planning. In *Proc. of the 29th IJCAI*, pages 4076–4083. IJCAI, 2020.

[Höller *et al.*, 2021] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. The PANDA framework for hierarchical planning. *KI*, 35:391–396, 2021.

[Höller and Behnke, 2021] Daniel Höller and Gregor Behnke. Loop detection in the PANDA planning system. In *Proc. of the 31st ICAPS*, pages 168–173. AAAI Press, 2021.

[Höller *et al.*, 2018] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. A generic method to guide HTN progression search with classical heuristics. In *Proc. of the 28th ICAPS*, pages 114–122. AAAI Press, 2018.

[Höller *et al.*, 2020a] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *Proc. of the 34th AAAI*, pages 9883–9891. AAAI Press, 2020.

[Höller *et al.*, 2020b] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. HTN planning as heuristic progression search. *JAIR*, 67:835–880, 2020.

[Kissmann and Edelkamp, 2009] Peter Kissmann and Stefan Edelkamp. Solving fully-observable non-deterministic planning problems via translation into a general game. In *Proc. of the 32nd KI*, pages 1–8. Springer, 2009.

[Kuter and Nau, 2004] Ugur Kuter and Dana Nau. Forward-chaining planning in nondeterministic domains. In *Proc. of the 19th AAAI*, pages 513–518. AAAI Press, 2004.

[Kuter *et al.*, 2005] Ugur Kuter, Dana S. Nau, Marco Pistore, and Paolo Traverso. A hierarchical task-network planner based on symbolic model checking. In *Proc. of the 15th ICAPS*, pages 300–309. AAAI Press, 2005.

[Kuter *et al.*, 2009] Ugur Kuter, Dana Nau, Marco Pistore, and Paolo Traverso. Task decomposition on abstract states, for planning under nondeterminism. *AIJ*, 173(5):669–695, 2009.

[Mattmüller *et al.*, 2010] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher. Pattern database heuristics for fully observable nondeterministic planning. In *Proc. of the 20th ICAPS*, pages 105–112. AAAI Press, 2010.

[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. *JAIR*, 20:379–404, 2003.

[Nilsson, 1982] Nils J Nilsson. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[Patra *et al.*, 2019] Sunandita Patra, Malik Ghallab, Dana Nau, and Paolo Traverso. Acting and planning using operational models. In *Proc. of the 19th AAAI*, pages 7691–7698. AAAI Press, 2019.

[Patra *et al.*, 2020] Sunandita Patra, James Mason, Amit Kumar, Malik Ghallab, Paolo Traverso, and Dana Nau. Integrating acting, planning, and learning in hierarchical operational models. In *Proc. of the 30th ICAPS*, pages 478–487. AAAI Press, 2020.

[Patra *et al.*, 2021] Sunandita Patra, James Mason, Malik Ghallab, Dana Nau, and Paolo Traverso. Deliberative acting, planning and learning with hierarchical operational models. *AIJ*, 299:103523, 2021.

[Pereira *et al.*, 2022] Ramon Pereira, André Pereira, Frederico Messa, and Giuseppe De Giacomo. Iterative depth-first search for FOND planning. In *Proc. of the 32nd ICAPS*, pages 90–99. AAAI Press, 2022.

[Pryor and Collins, 1996] Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *JAIR*, 4:287–339, 1996.

[Scala and Vallati, 2021] Enrico Scala and Mauro Vallati. Effective grounding for hybrid planning problems represented in PDDL. *The Knowledge Engineering Review*, 36:e9, 2021.

[Yoon *et al.*, 2007] Sungwook Yoon, Alan Fern, and Robert Givan. FF-Replan: A baseline for probabilistic planning. In *Proc. of the 17th ICAPS*, pages 352–359. AAAI Press, 2007.

[Yousefi and Bercher, 2024] Mohammad Yousefi and Pascal Bercher. Experimental setup for the IJCAI 2024 paper: "Laying the foundations for solving FOND HTN problems: Grounding, search, heuristics (and benchmark problems)". doi: 10.5281/zenodo.7704558, 2024.