Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Multi-Tier Automated Planning for Adaptive Behavior*

Daniel Ciolek, Nicolás D'Ippolito
Departamento de Computación
Universidad Nacional de Buenos Aires
Argentina

Alberto Pozanco
Departamento de Informática
Universidad Carlos III de Madrid
Spain

Sebastian Sardiña
School of Science
RMIT University
Australia

Abstract

A planning domain, as any model, is never "complete" and inevitably makes assumptions on the environment's dynamic. By allowing the specification of just one domain model, the knowledge engineer is only able to make one set of assumptions, and to specify a single objective-goal. Borrowing from work in Software Engineering, we propose a *multi-tier* framework for planning that allows the specification of different sets of assumptions, and of different corresponding objectives. The framework aims to support the synthesis of *adaptive* behavior so as to mitigate the intrinsic risk in any planning modeling task. After defining the multi-tier planning task and its solution concept, we show how to solve problem instances by a succinct compilation to a form of nondeterministic planning. In doing so, our technique justifies the applicability of planning with *both* fair and unfair actions, and the need for more efforts in developing planning systems supporting dual fairness assumptions.

Introduction

In AI planning (Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013), a plan is synthesized against *a model* of the environment—a *planning domain*—to achieve a given goal from an initial state of the environment. Such model describes how actions change the world, via the specification of their preconditions and effects. As any model, planning domains are never "complete" and they inevitable make assumptions on the dynamics of the environment. A limitation of standard planning formalism is that they do not account for deviations from such assumptions, and hence are not well prepared for integration within an execution framework. A common approach to handle discrepancies between what the planner expected and what happened at run-time is to simply perform *re-planning* or *plan-repair* (Fox et al. 2006). But, why would the system keep reasoning about the *same* model of the world that has been proven "wrong"?

As an actor's view on planning becomes more prominent in the field (Ghallab, Nau, and Traverso 2014), and inspired by work in Software Engineering (D'Ippolito et al. 2014), we propose a "generalized" planning framework that aims to better account for the uncertainties at design/engineering time. Concretely, rather than fixing the level of risk and objectives, we envision the specification of various assumption levels, as well as different goals for each assumption level. This is achieved by allowing the knowledge engineer to specify a family of planning domains, each carrying a set of assumptions. For example, in an fully idealized model of the blocks world, a robotic arm always successfully grabs blocks, whereas in less idealized models the griper may fail when picking, maybe missing or even breaking it. Depending on the assumptions imposed on the gripper operation, one may aim for different types of block towers.

The aim of the framework is to synthesize not one but a *collection* of inter-related policies embedding, together, *adaptive* behavior. So, as the environment violates assumptions, the agent should gracefully "degrade" to less refined planning models. Since such models carry less assumptions on the environment, less functionalities can generally be offered. If the gripper may break a block while picking it, building a complete block may just not be achievable. So, with model degradation, comes goal degradation, usually to a less ambitious (and often less demanding) one.

We call the above framework *multi-tier adaptive planning* and is the topic this paper. Let us start with a simple example to motivate the work and upcoming technical development.

Running Example

Consider a robot moving in a $1 \times n$ grid similar to the dust-cleaning robot example in (Bonet and Geffner 2015). The robot can *walk* one cell at a time or it can *run* covering multiple cells in one shot. Unfortunately, the physical shape of the corridor and the physical guarantees of the robot's actuators are not fully known to the designer. Because of that, in some scenarios, some cells may be impossible to navigate and the robot may get damaged or even broken. So, the knowledge engineer considers various possible assumption levels on the environment's behavior, together with corresponding adequate objectives.

In the most idealized model $\mathcal{D}_3$, the designer assumes that both walking and running actions succeed with no negative side-effects. The goal there is for the robot to reach a desti-

*Authors are listed in alphabetical order.

```
(:action walk
  :parameters
    (?o - Cell ?d - Cell)
  :precondition (and (at ?o)
    (adj ?o ?d) (not (broken)))
  :effect (and
    (not (at ?o)) (at ?d)) )

(:action run
  :precondition
    (and (at c2) (not (broken)))
  :effect (and
    (not (at c2)) (at c0)) )

(:goal (and (at c0)
            (not (scratch))
            (not (broken)) ))
```

(a) In model $\mathcal{D}_3$, any running and walking always succeeds.

```
(:action walk
 :parameters
   (?o - Cell ?d - Cell)
 :precondition (and (at ?o)
   (adj ?o ?d) (not (broken)))
 :effect (oneof
   (and (not (at ?o)) (at ?d))
   (and (not (at ?o)) (at ?d)
   (scratch))) )

(:action run
  :precondition
    (and (at c2) (not (broken)))
  :effect (oneof
    (and (not (at c2)) (at c0))
    (and (not (at c2)) (at c0)
    (scratch)) ))

(:goal (and (at c0) (not
(broken))) )
```

(b) In model $\mathcal{D}_2$, agent may move successfully but suffer minor scratch damage.

```
(:action walk
  :parameters (?o - Cell ?d - Cell)
  :precondition (and (at ?o)
    (adj ?o ?d) (not (broken)))
  :effect (oneof
    (and (not (at ?o)) (at ?d))
    (and (not (at ?o)) (at ?d)
    (scratch))
    (scratch) ))

(:action run
  :precondition
    (and (at c2) (not (broken)))
  :effect (oneof
    (and (not (at c2)) (at c0))
    (and (not (at c2)) (at c0)
    (scratch))
    (broken) ))

(:goal (and (at c2) (not broken)) )
```

(c) In model $\mathcal{D}_1$, movements may actually fail and may even leave the robot broken.

Listing 1: Actions walk left and run left in the three models.

nation cell c0 and intact. In a less idealized $\mathcal{D}_2$, both running and walking actions still cause the robot to advance, but no assumption can be made on their side effects and movement may cause minor damages. The robot should then just aim to reach the target destination c0. Finally, in the least idealized $\mathcal{D}_1$, a walking action may sometimes cause the robot to get minor damages without even advancing, and even worse, a running action may get the robot *broken* and render it unusable. Under those weaker assumptions, the robot should return to base location c2 for servicing.

Under the above multi-tier specification, the robot tries its best, but adapts its behavior as it discovers some assumptions may not hold. To do so, the robot initially assumes the most idealized world model and thus works for the most ambitious goal: reach destination undamaged. But, upon observing an inconsistency with the assumptions, it must *adapt* both the model of the environment as well as the objective being pursued. For example, if the robot succeeds in advancing when moving but gets a minor damage, it should *degrade* to $\mathcal{D}_2$. If it actually fails to move at all, it should *degrade* to $\mathcal{D}_1$ to operate under such level weaker assumptions.

A solution to this scenario must, on the one hand, strive for the best possible solution and, on the other hand, be open to a potential future degradation. Concretely, the robot should never attempt to perform an action that may prevent graceful degradation. In our example, while, in principle, running would be the most efficient way to reach the destination, it may cause a catastrophic failure in tier level 1, precluding the goals of every tier. Thus, the robot must be conservative and should cautiously move by walking.

Multi-Tier Planning

We propose a *multi-tier* automated planning framework in which the knowledge engineer can specify a ranked set of assumptions about the environment and a corresponding set of objective goals. A solution to such framework will display *adaptive* behavior at execution time, by aligning the model and objectives w.r.t. run-time observations. Before doing so, though, we first go over the standard technical machinery for *non-deterministic* planning.

A ***fully observable non-deterministic (FOND) planning domain*** (Rintanen 2008; Gerevini, Bonet, and Givan 2006) is a pair $\mathcal{D} = \langle V, O \rangle$ consisting of a set of Boolean state variables V and an operator set O. A ***state*** $s \in 2^V$ is the set of variables that are true in the state. We use S to denote the set of all states and $\bar{l}$ to denote the complement of literal l.

An ***operator*** is a tuple $\langle o, Pre_o, Eff_o \rangle$, where o is a unique name, Pre_o is a Boolean condition over V describing the *preconditions* of operator o, and $Eff_o = e_1 \mid \cdots \mid e_n$, with $n \geq 1$, is the (non-deterministic) *effect* of o where each e_i is a (set of) conditional effects $C \Rightarrow E$ with C being a Boolean condition over V and E a set (conjunction) of literals. The intended meaning is that *one* of the e_i effects ensues non-deterministically, by the environment's choice.

A ***policy controller*** is a function $\pi : S \mapsto 2^O$ that maps state $s \in S$ to a set of (executable) actions $\pi(s)$. A policy $\mathcal{C}$ executed from state $s \in S$ on domain $\mathcal{D}$ defines a set of ***possible executions*** $Ex_\pi(\mathcal{D}, s)$ of the form $\lambda = s_0 o_0 s_1 \cdots s_i o_i s_{i+1} \cdots$, where $s_0 = s$, $o_i \in \pi(s_i)$, $s_i \models Pre_{o_i}$, and s_{i+1} is a possible successor state when o_i is executed in state s_i w.r.t. domain $\mathcal{D}$, for all $i \geq 0$. We use $last(\lambda)$ to denote the last state in (finite) execution λ and $Ex(\mathcal{D}, s)$ to the set of all possible executions in $\mathcal{D}$ from state

$s \in S$ (i.e., $Ex(\mathcal{D}, s) = Ex_{\pi^*}(\mathcal{D}, s)$, where $\pi^*(s) = O$).

Finally, a ***FOND planning problem*** $\mathcal{P} = \langle \mathcal{D}, s_I, \phi_{\text{goal}} \rangle$ consists of a FOND domain $\mathcal{D}$, an initial state s_I, and a goal ϕ_{goal} as a conjunction of literals from V. There has been several solution concepts for FOND planning depending on the fairness of non-deterministic actions. Roughly speaking, a ***fair action*** is one in which all effects occur infinitively often when the action is executed infinitively many times in the same state (Geffner and Bonet 2013; Sardina and D'Ippolito 2015). When all actions are assumed fair, a ***strong-cyclic*** plan guarantees that the agent, by "retrying," eventually achieves the goal (Cimatti et al. 2003). In turn, when no fairness can be assumed, a plan with acyclic executions that reaches the goal in a bounded number of steps—a ***strong*** policy—is required. The ***Dual FOND*** (or FOND+) hybrid variation has recently been introduced to deal with domains that have both fair and unfair actions/-effects (Camacho and McIlraith 2016; Geffner and Geffner 2018). In that setting, a solution amounts to a policy whose "fair" executions w.r.t. the actions/effects assumed to be fair (not necessarily all) are goal reaching. Lastly, we note that while planning under non-determinism is EXPTIME-complete (Rintanen 2004), effective optimized techniques and solvers have been developed, and is an area of significant active work (e.g., (Muise, McIlraith, and Beck 2012; Kuter et al. 2008; Kissmann and Edelkamp 2009; Muise, Belle, and McIlraith 2014; Geffner and Geffner 2018)).

With the technical machinery on FOND planning at hand, we are ready to formally present our framework for multi-tier adaptive planning. Following (D'Ippolito et al. 2014), we aim for the knowledge engineering to be able to specify a variety of models carrying different assumptions.

Definition 1. A ***multi-tier planning domain*** (MTD) is a tuple $\langle \Omega, \leq \rangle$ such that:

1. Ω is a set of FOND planning domains over the same variables V and operator signatures, and every operator has the same preconditions across all domains in Ω;
2. $\leq$ is a partial-order relation over Ω such that $\mathcal{D}_1 \leq \mathcal{D}_2$ implies $Ex(\mathcal{D}_2, s) \subseteq Ex(\mathcal{D}_1, s)$ for all states $s \in S$; and
3. $\leq$ has a greatest element in Ω, denoted $\hat{\mathcal{D}}$, as well as a minimum element.

The first condition states that an MTD is just a collection of planning domains over the same vocabulary, with actions having the same names and preconditions but with possibly different effects across domains. The differences in operators' effects will reflect *different assumptions on the environment* and should reflect model "refinements." Specifically, the second condition specifies that domains in lower tiers of the hierarchy ($\mathcal{D}_1$) must produce the same behaviors as higher models ($\mathcal{D}_2$), and possibly more. The intuition is that higher-level models are "refinements" of lower-level models, posing possibly more assumptions on the environment (e.g., by actions having fewer non-deterministic effects), hence permitting fewer execution runs.

As in standard planning, a problem instance task adds a specific initial situation and a (set of) objectives.

Definition 2. A ***multi-tier planning problem*** (MTP) is a tuple $\mathcal{M} = \langle \langle \Omega, \leq \rangle, s_I, \mathcal{G} \rangle$ where $\langle \Omega, \leq \rangle$ is an MTD, s_I is $\mathcal{M}$'s initial state, and $\mathcal{G}$ is a function mapping each domain $\mathcal{D}$ in Ω to a goal $\mathcal{G}(\mathcal{D})$ (or just $\mathcal{G}_{\mathcal{D}}$). ■

Observe that unlike standard planning approaches, we allow the designer to specify various goals, depending on the risk imposed by the assumptions on the environment. Often, the weaker the assumptions, the lower level of functionality that may be guaranteed (D'Ippolito et al. 2014).

Finally, we define a structure that associates a specific policy to each domain in an MTD, prescribing what behavior should ensue from the executor under the different models.

Definition 3. A ***multi-tier controller*** (MTC) for an MTD $\langle \Omega, \leq \rangle$ is a function $\mathcal{C} : \Omega \mapsto (S \mapsto 2^O)$ mapping each domain $\mathcal{D} \in \Omega$ to a specific policy $\mathcal{C}(\mathcal{D})$ (or just $\mathcal{C}_{\mathcal{D}}$). ■

The challenge now is to formally capture when an MTC amounts to a "solution" strategy for a multi-tier planning problem. To do so, it is important to first understand how the MTC structure is meant to be deployed in the environment. Intuitively, at any point in time, the executor is operating relative on some planning domain (i.e., model) of the world $\mathcal{D}$ from the ones available in Ω, by carrying out its corresponding policy $\mathcal{C}(\mathcal{D})$ so as to bring about the level's goal $\mathcal{G}(\mathcal{D})$. Initially, the executor deploys policy $\mathcal{C}(\hat{\mathcal{D}})$ from the initial problem state s_I on the most idealized domain $\hat{\mathcal{D}}$, aiming at achieving the most ambitious goal $\mathcal{G}(\hat{\mathcal{D}})$. However, if at any point during execution, an inconsistency with the current model $\mathcal{D}_i$ is observed, the executor ought to switch to an alternative domain $\mathcal{D}_j \in \Omega$ such that $\mathcal{D}_j \leq \mathcal{D}_i$. Technically, an inconsistency amounts to observing an actual state s that cannot be explained with planning domain $\mathcal{D}_i$. Of course, once the executor switches downwards—referred as *degradation*—the model it operates on to a more permissive one (i.e., one with weaker assumptions), the objective sought, and hence the strategy, must be changed too. A smart executor, though, aims to degrade gracefully, that is, as little as possible, switching to a planning domain that retains as many assumptions as possible about the environment (and the most ambitious goal).

Let us now develop the solution concept for MTPs. We define the set of triggering states for a domain in an MTD as those states in which the executor, when deployed in a given multi-tier controller as per the above operational scheme, may need to start operating under such domain. As expected, the initial state s_I is the triggering state for the highest level, most idealized, domain $\hat{\mathcal{D}}$. For other domains, a triggering state amounts to a degradation step.

Definition 4. Let $\mathcal{C}$ be an MTC for a MTD $\langle \Omega, \leq \rangle$, and let s be a state (over variables V_Ω). We inductively define the set of ***triggering initial states*** for each planning domain $\mathcal{D} \in \Omega$ under $\mathcal{C}$, denoted $\mathit{Init}(\mathcal{D}, \mathcal{C})$, as follows:

1. $\mathit{Init}(\hat{\mathcal{D}}, \mathcal{C}) = \{s_I\}$;

2. if $\mathcal{D}$ is not the maximum in Ω (i.e., $\mathcal{D} \neq \hat{\mathcal{D}}$), then

$$\begin{aligned} &\mathit{Init}(\mathcal{D}, \mathcal{C}) = \\ &\quad \{s \mid \mathcal{D} < \mathcal{D}', s' \in \mathit{Init}(\mathcal{D}'), \\ &\qquad \lambda \in \mathit{Ex}_{\mathcal{C}_{\mathcal{D}'}}(\mathcal{D}', s'), o = \mathcal{C}_{\mathcal{D}'}(\mathit{last}(\lambda)), \\ &\qquad \lambda o s \in \mathit{Ex}(\mathcal{D}, s') \setminus \bigcup_{\mathcal{D}'' : \mathcal{D} < \mathcal{D}''} \mathit{Ex}(\mathcal{D}'', s').\} \end{aligned}$$

■

Let us explain the second case. Suppose the executor has so far been carrying out policy $\mathcal{C}_{\mathcal{D}'}$ on a domain model $\mathcal{D}'$, from some (triggering) state s'. Suppose this has yielded execution run λ (consistent with $\mathcal{D}'$). However, when executing the next prescribed operator o (as per the corresponding policy $\mathcal{C}_{\mathcal{D}'}$ for $\mathcal{D}'$), the resulting evolution to a state s yields an execution $\lambda o s$ that can be explained by (i.e., its a legal execution in) domain $\mathcal{D}$ but not by any model higher than $\mathcal{D}$ (including $\mathcal{D}'$). When this happens, state s is a triggering state for $\mathcal{D}$, that is state where the executor may have to start operating under domain model $\mathcal{D}$ when using policy $\mathcal{C}_{\mathcal{D}}$.

Next, for a controller $\mathcal{C}$ to be a solution for an MTP $\mathcal{M}$, it must achieve the associated goal of a domain in $\mathcal{M}$ *from all the triggering states of the domain in question*.

Definition 5. An MTC $\mathcal{C}$ is a ***solution controller*** for an MTP $\mathcal{M} = \langle \langle \Omega, \leq \rangle, s_I, \mathcal{G} \rangle$ *iff* for every domain $\mathcal{D} \in \Omega$, the projected policy $\mathcal{C}_{\mathcal{D}}$ is a solution plan for planning problem $\langle \mathcal{D}, s, \mathcal{G}_{\mathcal{D}} \rangle$, for every state $s \in \mathit{Init}(\mathcal{D}, \mathcal{C})$. ■

Note that, unlike standard planning, this definition requires each policy to work from more than one initial state. However, it is not the case that all policies in $\mathcal{C}$ need to work from the initial state s_I. Such a requirement would be too demanding for capturing the intended operational framework as described above, this is because most policies, if not all but $\mathcal{C}_{\hat{\mathcal{D}}}$, will ever be used at state s_I (unless the system comes back to such state after some degradation).

Solving Multi-Tier Planning Problems

Informally, an MTP is a collection of similar planning problems and a solution amounts to solution policies for each problem that can be "connected", if necessary, at degradation time. A naive approach thus would repetitively compute solution policies for each planning problem, making sure they "connect." We show here we can solve the whole problem in a principled manner and in one shot. Concretely, we build a single Dual FOND planning task $\mathcal{P}_{\mathcal{M}}$ from a given MTP $\mathcal{M}$ such that a strong-cyclic solution for $\mathcal{P}_{\mathcal{M}}$ amount to an MTC solution for $\mathcal{M}$. To argue for technique's generality, we first identify a meaningful fragment of MTDs.

Definition 6. A planning domain $\mathcal{D}_2 = \langle V_2, O_2 \rangle$ is an ***oneof-refinement*** of a domain $\mathcal{D}_1 = \langle V_1, O_1 \rangle$ iff $V_1 = V_2$ and for every $\langle o, \mathit{Pre}^2_o, \mathit{Eff}^2_o \rangle \in O_2$, there is a $\mathcal{D}_1$-operator $\langle o, \mathit{Pre}^1_o, \mathit{Eff}^1_o \rangle \in O_1$ such that $\mathit{Pre}^1_o = \mathit{Pre}^2_o$ and $\mathit{Eff}^2_o \subseteq \mathit{Eff}^1_o$; ■

That is, $\mathcal{D}_2$ is like $\mathcal{D}_1$ but may contain fewer nondeterministic effects on some operators. It turns out that, in the context of Dual FOND planning, oneof-refinements capture all possible refinements in a multi-tier planning task—any MTD is equivalent to a oneof-refinement type.

Theorem 1. *Let $\langle \Omega, \leq \rangle$ be an MTD and $\mathcal{D}_1, \mathcal{D}_2 \in \Omega$. Then, $\mathcal{D}_1 < \mathcal{D}_2$ (i.e., planning domain $\mathcal{D}_2$ is a refinement of domain $\mathcal{D}_1$)* **iff** *there exists a planning domain $\mathcal{D}'_2$ such that:*

1. *$\mathit{Ex}(\mathcal{D}_2, s) = \mathit{Ex}(\mathcal{D}'_2, s)$, for all $s \in S$ (that is, $\mathcal{D}'_2$ are equivalent planning domains); and*
2. *$\mathcal{D}_2$ is an oneof-refinement of $\mathcal{D}_1$.*

This states that the only meaningful difference between ordered domains in Ω comes, only, in the refined domain ($\mathcal{D}_2$) having *less* (in terms of set inclusion) nondeterministic effects in some operators.

Compilation to Dual FOND Planning

Let $\mathcal{M} = \langle \langle \Omega, \leq \rangle, s_I, \mathcal{G} \rangle$ be a a multi-tier planning problem such that $\mathcal{D} \leq \mathcal{D}'$ if and only if $\mathcal{D}'$ is a oneof-refinement of $\mathcal{D}$. Due to Theorem 1, restricting $\leq$ to a oneof-refinement relation does not affect generality. From now on, for technical legibility and without loss of generality, we assume domains in Ω are STRIP-like with no conditional effects.[1]

In this section, we shall construct a single dual-FOND planning problem $\mathcal{P}_{\mathcal{M}} = \langle \mathcal{D}_{\mathcal{M}}, s_{\mathcal{M}}, G_{\mathcal{M}} \rangle$ that will fully capture problem $\mathcal{M}$. For compactness, we use $\mathit{Eff}^{\mathcal{D}}_o$ to denote the effects of operator o in planning domain $\mathcal{D}$. We also abuse notation and treat non-deterministic effects as sets.

Let us start by explaining the general strategy being encoded into $\mathcal{P}_{\mathcal{M}}$. Roughly speaking, the planning problem $\mathcal{P}_{\mathcal{M}}$ will model a dynamic system running as per multi-tier specification $\mathcal{M}$. As such, at any time, the system is operating relative to some model $\mathcal{D}$ in Ω (initially, the most ambitious $\hat{\mathcal{D}}$), trying to achieve $\mathcal{D}$'s goal via an appropriate plan, and degrading to an adequate (lower) model when action outcomes' do not align with model $\mathcal{D}$. To achieve this, the encoding will model an *iterative two-phase* process in which an *acting* phase is, sometimes if necessary, followed by an *alignment & degradation* phase. A special variable *act* is used to distinguish both phases. As expected, during an *acting phase*, an operator representing some domain action is executed. This step involves the execution of a nondeterministic action with fair semantics, and the optional subsequent execution of an unfair version of the action. In the latter case, the system will then evolve to an *alignment phase*, in which the encoding verifies whether the outcomes seen correspond to the assumed current model $\mathcal{D}$; and if not, the behavior is "degraded" to an appropriate (lower-level) model that is able to explain the observed outcome.

It turns out that one of the key challenges is to encode a proper and scalable alignment phase in a planning domain (i.e., in PDDL): *how can we encode that a given effect could be explained by some model in Ω (but not by another model)?* In some sense, doing so would amount to reducing meta-level reasoning to the object (PDDL) level. We show that, via a clever encoding, that reduction is indeed possible. The technical difficulty is depicted in the following example.

[1] All results can be generalized to domains with conditional effects, but would result in a significantly more cumbersome presentation and notation without providing significant insights.

Example 1. Consider the case in which the robot is operating in the highest domain model $\mathcal{D}_3$ and decides to execute action `walk`. Upon execution, the robot senses variable `scratch` true—the robot is now damaged. In the standard (intuitive) configuration, in which the robot starts non-damaged, the robot should *degrade* its operational model to domain $\mathcal{D}_2$, as that is the highest model explaining the damage. However, if the robot happens to start damaged (i.e., scratched) already, then domain model $\mathcal{D}_3$ still explains the transition, and no degradation should occur. Here, `walk`'s effects under $\mathcal{D}_3$ and $\mathcal{D}_2$ are indistinguishable.

This example shows that just observing a proposition (`scratch`) in a transition that does not appear in an effect (`walk`'s effect under $\mathcal{D}_3$) does *not* directly imply the effect may not explain the transition. Can we then characterize, succinctly, under which conditions a set of observed propositions E is explained by some operator o in a domain model $\mathcal{D}$? It turns out we can.

Definition 7 (Effect Explicability). Let E be a set of literals (e.g., effects that have just seen to be true after an action execution). The conditions for operator o in domain $\mathcal{D}$ to explain E, denoted $\mathit{Explains}[o, \mathcal{D}, E]$, is defined as follows (recall Δ is the set symmetric difference operation):

$$\mathit{Explains}[o, \mathcal{D}, E] = \bigvee_{E' \in \mathit{Eff}_o^{\mathcal{D}}} \bigwedge_{l \in E \Delta E'} l. \qquad \blacksquare$$

That is, some effect E' of o in model $\mathcal{D}$ yields the same result as effect E, if all the literals that one effect has an the other does not *were already true* (at the outset of o's execution). In our Example 1, if we take E to be the second effect of `walk` in $\mathcal{D}_2$ (i.e., $E = \{$`(not (at ?o))`, `(at ?d)`, `(scratch)`$\}$) we have $\mathit{Explains}[$`walk`$, \mathcal{D}_3, E]$ = `scratch`, as the literal `scratch` is the only one in the effects' symmetric difference.

Observe that if E and E' are inconsistent, the formula will contain the conjunction of a proposition and its negation, thus reducing to false. In fact, the following result guarantees the intended meaning of the above definition.

Lemma 1. *Let $s, s' \in 2^{V \cup \overline{V}}$ be two domain states. Let $E \in 2^{V \cup \overline{V}} \supseteq s' \setminus s$ be a set of literals including at least all new literals in s' w.r.t. to s. Then, state s' is a possible successor when operator o is executed in state s under model $\mathcal{D}$ if and only if $s \models \mathit{Explains}[o, \mathcal{D}, E]$.*

We are now ready to provide the encoding of $\mathcal{M}$ into a dual-FOND planning problem $\mathcal{P}_{\mathcal{M}} = \langle \mathcal{D}_{\mathcal{M}}, s_{\mathcal{M}}, \mathcal{G}_{\mathcal{M}} \rangle$.

Domain variables. The *set of propositional variables* V^+ of $\mathcal{D}_{\mathcal{M}}$ is obtained by extending the set of variables V in $\mathcal{M}$'s domains with the following additional variables:

- $\varepsilon_{\mathcal{D}}$, for each domain $\mathcal{D} \in \Omega$, that will be used to signal that model $\mathcal{D}$ is a/the highest model explaining the effect of the last executed action;
- $\ell_{\mathcal{D}}$, for each domain $\mathcal{D} \in \Omega$, that will be used to track the most "ambitious" compatible model so far;
- *act*, use to denote the system is in the *acting phase* (otherwise, it is in the *alignment phase*);
- u_o, for each operator $o \in \hat{\mathcal{D}}$, that will be used to ensure the execution of a unfair action; and
- *end*, used to denote the goal achievement of the current model of execution.

Initial state & goal condition. The *initial state* of $\mathcal{P}_{\mathcal{M}}$ is:

$$s_{\mathcal{M}} = s_I \wedge [\ell_{\hat{\mathcal{D}}} \wedge \bigwedge_{\mathcal{D} \in \Omega^-} (\neg\varepsilon_{\mathcal{D}} \wedge \neg\ell_{\mathcal{D}}) \wedge \mathit{act} \wedge \neg \mathit{end}].$$

This encodes the initial state s_I of the MTP $\mathcal{M}$ and the fact that the system starts in the Ω's greatest, most ambitious, domain model $\hat{\mathcal{D}}$ (proposition $\ell_{\hat{\mathcal{D}}}$ and all other ℓ_x's are false) and in the action phase. In addition, all effect level signaling variables ε_x are initialized to false (no action has been executed), as well as the goal variable *end*.

Finally, the goal condition of $\mathcal{P}_{\mathcal{M}}$ is simply $G_{\mathcal{M}} = \mathit{end}$. We will see below which actions make variable *end* true.

Domain operators. The planning domain $\mathcal{D}_{\mathcal{M}}$ will include *two types* of operators, one for modeling the actual domain actions and one for implementing the alignment check (and potential degradation) process. Let us start with the former.

So, for each (domain) operator o in domain $\mathcal{D} \in \Omega$, we include a operator $\langle o_{\mathcal{D}}, \mathit{Pre}, \mathit{Eff} \rangle$ in $\mathcal{D}_{\mathcal{M}}$, where:

- $\mathit{Pre} = \mathit{Pre}_o^{\mathcal{D}} \wedge \ell_{\mathcal{D}} \wedge \mathit{act} \wedge \bigwedge_{o \in \hat{\mathcal{D}}} \neg u_o$, that is, action $o_{\mathcal{D}}$ is executable when o itself is executable in $\mathcal{D}$, and the system is currently operating under model $\mathcal{D}$ and is the fair-acting phase (*act* is true and all u_x are false); and
- $\mathit{Eff} = \mathit{Eff}_o^{\mathcal{D}} \cup \{u_o\}$, that is, when operator $o_{\mathcal{D}}$ is executed, either one of original effects of o in $\mathcal{D}$ ensues or a distinguished predicate o_u is made true.

When one of the effects of o in domain $\mathcal{D}$ happens, it just resembles the dynamics of domain $\mathcal{D}$. However, if the effect that ensues is u_o, the system evolves into a "unfair-acting" phase ($\mathit{act} \wedge u_o$), explained after the following example.

Example 2. The resulting walk action for the domain level $\mathcal{D}_2$ in the compilation would look as follows in PDDL:

```
(:action walk_d2
  :parameters (?o - cell ?d - cell)
  :precondition (and (at ?o) (adj ?o ?d) (not (broken))
        (d2) (act) (not (u_walk)) (not (u_run)))
  :effect (oneof
   (and (not (at ?o)) (at ?d))
   (and (not (at ?o)) (at ?d) (scratch))
   (u_walk) ))
```

Next, when the the system evolves to the unfair-acting phase (e.g., due to effect `u_walk` happening in the example above), the only executable action will be a second version of domain operator o, which in turn will include *all effects* of *o across all domains* in Ω, together with additional bookkeeping variables ε_x to support the next alignment, and potential degradation, reasoning phase. More concretely, for

each domain operator o in $\hat{\mathcal{D}}$, $\mathcal{D}_{\mathcal{M}}$ includes a rather powerful operator $\langle o_{unfair}, Pre, Eff \rangle$, where:

- $Pre = act \wedge u_o \wedge Pre_o$, that is, the system is in the unfair-acting phase for operator o; and
- Eff is a set of nondeterministic effects, each being a collection (i.e., conjunction) of conditional effects built as follows. For every effect E of operator o that is mentioned in a domain $\mathcal{D}$ but not in any lower one (i.e., $E \in Eff_o^{\mathcal{D}} \setminus \bigcup_{\mathcal{D}':\mathcal{D}'<\mathcal{D}} Eff_o^{\mathcal{D}'}$), Eff contains, as one of its non-deterministic effects, the following complex effect:

$$\bigwedge_{\mathcal{D}':\mathcal{D}'\geq\mathcal{D}} (C^E_{\mathcal{D}'} \Rightarrow E \wedge \neg act \wedge \neg u_o \wedge \varepsilon_{\mathcal{D}'}),$$

where $C^E_{\mathcal{D}'} = Explains[o, \mathcal{D}', E] \wedge \bigwedge_{\mathcal{D}'':\mathcal{D}''>\mathcal{D}'} \neg Explains[o, \mathcal{D}'', E]$.

Intuitively, the operator o_{unfair} contains each possible effects E of o (from any domain in $\mathcal{M}$) as a non-deterministic option. In turn, the set of conditional effects for a particular effect E will not only make the effect E itself ensue, but will also set a "marker" proposition $\varepsilon_{\mathcal{D}}$ signaling the highest domains explaining the effect in question. To realize that, condition $C^E_{\mathcal{D}'}$ above states that the original effect E is explained (as per Definition 7) by (some effect of) operator o at domain model $\mathcal{D}'$ but not by any other model higher than $\mathcal{D}'$. When that is the case, proposition $\varepsilon_{\mathcal{D}'}$ is set to true, recording the fact that $\mathcal{D}'$ is the *highest* model explaining such effect. Observe that by the way all conditions are designed, they ought to be mutually exclusive, so only one ε_x will be made true. In addition, *act* is set to false so as to force the reasoner into the *alignment* phase, to be explained shortly. (We note that the effects of level $\mathcal{D}$ itself are accounted when $\mathcal{D}' = \mathcal{D}$.)

Importantly, while $o_{\mathcal{D}}$ operator will be treated *fair*, action o_{unfair} will be treated as *unfair*—this is where dual FOND semantics (Geffner and Geffner 2018) come into play. Also, as the following example shows, significant syntactic simplifications can be achieved in o_{unfair} by analyzing conditions in conditional effects and precondition of the action.

Example 3. Let us see complete Example 2 by showing the unfair version of the `walk` action. After syntactic simplification w.r.t. conditions and the action precondition, we obtain the simpler, more readable, equivalent action:

```
(:action walk_unfair
  :parameters (?o - cell ?d - cell)
  :precondition (and (act) (u_walk)
                     (at ?o) (adj ?o ?d) (not (broken)))
  :effect (and (not (act)) (not (u_walk))
    (oneof
      (when (true)
          (and (not (at ?o)) (at ?d) (e3)) )
      (and (when (not (scratch))
              (and (not (at ?o)) (at ?d) (scratch)
              (e2)) )
           (when (scratch)
              (and (not (at ?o)) (at ?d) (e3)) ))
      (when (true) (and (scratch) (e1) )) )))
```

As we discussed before, this unfair action contemplates the effects present in all the domain models. The intended meaning of this is that whenever an action executes, it may fail and we may observe effects of any domain level. However, we do not want the planner to rely on these possible failures, so we contemplate them as unfair actions.

Alignment & degradation operators. When the unfair version of a domain operator has been executed, an effect could ensue that might not be explained by the current domain under which the reasoner is operating under (encoded via propositions ℓ_x). If so, the system ought to gracefully degrade to a lower level model that is able to explain the last system evolution. We encode this reasoning, and potential degradation, in the so-called *alignment* phase (*act* is false).

In the best case, the state observed after the execution of an action corresponds to one of the expected ones w.r.t. the current planning domain model the executor is operating under. Technically, the reasoner continues operating under current model $\mathcal{D}$ (proposition $\ell_{\mathcal{D}}$ is true), provided domain $\mathcal{D}$ has been able to explain the evolution of the last executed action: proposition $\varepsilon_{\mathcal{D}'}$ has been set to true for some domain $\mathcal{D}'$ that is either $\mathcal{D}$ itself or a higher one in the hierarchy (recall effects in higher level domains are subsets of). So, in such case, the planner (and executor) is able to execute special action $\langle \text{CONTINUE}_{\mathcal{D}}, Pre, Eff \rangle$ to keep planning/executing under the current model and goal:

- $Pre = (\neg act \wedge \ell_{\mathcal{D}} \wedge \bigvee_{\mathcal{D}'\geq\mathcal{D}} \varepsilon_{\mathcal{D}'})$, that is, the action can be executed during the alignment phase when the current domain or one of its refinements accounts for the last effect outcome.
- $Eff = (act \wedge \bigwedge_{\mathcal{D}\in\Omega^-} \neg\varepsilon_{\mathcal{D}})$, that is, effect signals are all reset and the system goes back to the action phase.

If, on the other hand, the state observed does *not* conform to the current operating model (i.e., proposition $\varepsilon_{\mathcal{D}}$ is false), then the system must *degrade* to a lower tier where the environment model would fit the observation, and adjust the objective to the corresponding (often less ambitious) goal. Needless to say, we expect a "smart" reasoner/executor to degrade as little as possible, by retaining as many assumptions on the environment as possible and only dropping those that have been observed to be wrong. This will allow the agent to aim for the highest, most valuable, goal so far.

Technically, when $\mathcal{D}, \mathcal{D}' \in \Omega$ such that $\mathcal{D}' < \mathcal{D}$, we include an operator $\langle \text{DEGRADE}_{\mathcal{D}\mathcal{D}'}, Pre, Eff \rangle$ in $\mathcal{P}_{\mathcal{M}}$, where:

- $Pre = \neg act \wedge \ell_{\mathcal{D}} \wedge \bigvee_{\{\mathcal{D}^*:\mathcal{D}^*\geq\mathcal{D}', \neg(\mathcal{D}\geq\mathcal{D}^*\geq\mathcal{D}'), \mathcal{D}^*\not\geq\mathcal{D}\}} \varepsilon_{\mathcal{D}^*}$; and
- $Eff = \neg\ell_{\mathcal{D}} \wedge \ell_{\mathcal{D}'} \wedge \bigwedge_{x\in\Omega}(\neg\varepsilon_x \wedge act)$.

That is, the controller can degrade from current operating domain $\mathcal{D}$ to domain $\mathcal{D}'$ if the last effect seen was explained by lower domain $\mathcal{D}'$ or any other domain higher than $\mathcal{D}'$ that is unrelated to $\mathcal{D}$ (so as to handle MTPs with a non-linear structure). The effect results in the controller being degraded to level $\mathcal{D}'$ (proposition $\ell_{\mathcal{D}'}$ becomes true), all booking explicability effect prepositions ε_x being reset, and the reasoner progressing to the acting phase.

Note that, effectively, the dynamics of level variables ℓ_x are *outside the control of the reasoner*, as these depend only on which non-deterministic effects have occurred and how (i.e., how variables ε_x have been set).

Goal operators. The only part remaining is the overall goal of the multi-tier problem. Intuitively this should be "achieve the highest level goal", which under a conservative degradation process, it reduces to "achieve the goal of the current operating model." We therefore include goal actions $\langle \text{CHECKGOAL}_{\mathcal{D}}, (\mathcal{G}_{\mathcal{D}} \wedge \ell_{\mathcal{D}}), \mathit{end} \rangle$, one per domain $\mathcal{D} \in \Omega$.

This completes the encoding of a multi-tier planning problem $\mathcal{M}$ into a single non-deterministic planning domain $\mathcal{P}_{\mathcal{M}}$. We now prove its correctness w.r.t. Definition 5. First, any solution policy for the planning task amounts, as is, to a solution to the corresponding multi-tier planning problem.

Theorem 2. *If π is a strong-cyclic solution for $\mathcal{P}_{\mathcal{M}}$, then controller $\mathcal{C}^{\pi}(\mathcal{D})$ is an* MTC *solution for $\mathcal{M}$, where:*

$$\mathcal{C}^{\pi}_{\mathcal{D}}(s) = \pi(s \wedge \ell_{\mathcal{D}} \wedge \mathit{act} \wedge \bigwedge_{o \in \hat{\mathcal{D}}} \neg u_o), \text{ for all } s \in S.$$

Proof Sketch. Consider $s_i \in S$ and $\mathcal{D} \in \Omega$ such that $s_i \in \mathit{Init}(\mathcal{D}, \mathcal{C}^{\pi})$, and an infinite and fair execution $\lambda \in \mathit{Ex}_{\mathcal{C}^{\pi}}(\mathcal{D}, s_i)$. We show that goal $\mathcal{G}(\mathcal{D})$ holds true somewhere along λ as follows:

1. We transform λ into an execution $\hat{\lambda} \in \mathit{Ex}_{\mathcal{C}^{\pi}}(\mathcal{D}_{\mathcal{M}}, s_i^+)$, with $s_i^+ = s_i \cup \{\mathit{act}, \ell_{\mathcal{D}}\}$, by adding propositions *act* and $\ell_{\mathcal{D}}$ to every state in λ and replacing every domain operator o with its $o_{\mathcal{D}}$ version.
2. If $so_{\mathcal{D}}$ appears infinitively often in $\hat{\lambda}$, we replace every second appearance of the form $so_{\mathcal{D}}s'$ by two-action steps $so_{\mathcal{D}}(s \cup \{u_{o_{\mathcal{D}}}\})o_{\mathit{unfair}}(s' \cup \{\mathit{act}, \varepsilon_{\mathcal{D}'}\})$ such that $\mathcal{D}' \geq \mathcal{D}$ is the highest domain in Ω that contains the effect of o that supports the transition $so_{\mathcal{D}}s'$—we know there is one because $\hat{\lambda}$ is a legal execution in $\mathcal{D}_{\mathcal{M}}$ from state s_i^+. By doing this changes in $\hat{\lambda}$ we are guarantee that the execution is fair, while still preserving the fact that every domain action in it has the effects as per domain $\mathcal{D}$. So, execution $\hat{\lambda}$ mirrors the original λ for domain $\mathcal{D}$ but over the extended language of $\mathcal{D}_{\mathcal{M}}$.
3. Because $s_i \in \mathit{Init}(\mathcal{D}, \mathcal{C}^{\pi})$, there exists a finite execution $\lambda_i \in \mathit{Ex}_{\pi}(\mathcal{D}_{\mathcal{M}}, s_I)$ (i.e., execution in $\mathcal{P}_{\mathcal{M}}$ via policy π) that ends in state $s_i \cup \{\ell_{\mathcal{D}}, \mathit{act}\}$. This means that $\lambda_i\hat{\lambda} \in \mathit{Ex}_{\mathcal{C}^{\pi}}(\mathcal{D}_{\mathcal{M}}, s_I)$, and since $\lambda_i\hat{\lambda}$ is fair (w.r.t. the fair actions) and $\hat{\lambda}$ has $\lambda_{\mathcal{D}}$ always true, it follows that $\hat{\lambda}$ has to reach the $\mathcal{P}_{\mathcal{M}}$'s goal by executing operator $\text{CHECKGOAL}_{\mathcal{D}}$. Then, its precondition $\mathcal{G}_{\mathcal{D}}$ holds true at some point in $\hat{\lambda}$ and therefore in λ too. □

That is, the MTC controller $\mathcal{C}$ under domain $\mathcal{D}$ and in state s, what the strong-cyclic solution for $\mathcal{P}_{\mathcal{M}}$ prescribes when $\ell_{\mathcal{D}}$ is true and the reasoning cycle is in the acting phase.

In addition, any possible MTC solution will be represented by some strong-cyclic policy of $\mathcal{P}_{\mathcal{M}}$ (i.e., completeness).

Theorem 3. *If $\mathcal{C}$ is an* MTC *solution for $\mathcal{M}$, then there exists a strong-cyclic solution π for $\mathcal{P}_{\mathcal{M}}$ such that $\mathcal{C}^{\pi}(\mathcal{D}) = \mathcal{C}(\mathcal{D})$, for every domain $\mathcal{D}$ in $\mathcal{M}$ (where $\mathcal{C}^{\pi}(\mathcal{D})$ is as in Theorem 2).*

Proof Sketch. Policy π follows the domain actions prescribed by $\mathcal{C}_{\mathcal{M}}$ exactly, augmented with the booking auxiliary actions as needed. A similar argument, based on execution traces, as in Theorem 2 can be built. □

We close by noting that the size of $\mathcal{P}_{\mathcal{M}}$ is increased by a linear number of bookkeeping propositional variables, and a quadratic number (w.r.t. the number of domains in Ω) of extra actions. So, while the multi-tier planning framework appears to be more involved than the standard (non-deterministic) planning, it can be suitably reduced to the latter, with an encoding that is, arguably, fairly natural and comparable in size. Importantly, though, the solution proposed relies on the fact that we can specify *both* fair and unfair actions in the same planning model. This is a feature that will prove a challenge when actually realizing the technique in current planning technology, as we shall see next.

Validation and Discussion

In this section, we demonstrate that MTPs can indeed be solved *today* with existing planning technology, but argue that additional effort in Dual FOND is necessary. The first obstacle is the availability of FOND planning technology supporting both fair and unfair assumptions. To the best of our knowledge, the only off-the-shelf planner to do so is Geffner and Geffner (2018)'s FOND-SAT system. By leveraging on SAT solvers, their system yields an elegant declarative technique for FOND planning that features the possibility of combining fair and unfair actions. So, we report on using FOND-SAT over the encoding for our non-running example. Notwithstanding, the experiments reported are intended to demonstrate the existence of systems to solve MTPs and to provide a baseline for future work, rather than for providing a performance evaluation.

Listing 2 shows a fragment, in a readable plan-like format, of the outcome when FOND-SAT is ran on our encoding for the non-running example.[2] First of all, the plan cautiously avoids the run action altogether, as it may get the robot broken and precludes the achievement of all tier goals.

After performing the walk *fair*-version action in (line 2) corresponding to the highest model $\mathcal{D}_3$, the plan checks its effects (line 3). If proposition `u_walk` remains false (lines 4-9), the effect in model $\mathcal{D}_3$ has occurred—the robot has done a successful move. If another walk (line 4) succeeds as well (lines 5-7), the robot achieves the top level $\mathcal{D}_3$' goal (line 6). Note that, in such a run, no alignment action is included: the walk unfair version has never been performed and hence only effects of $\mathcal{D}_3$ has ensued.

If, instead, the first walking action (line 2) yields the special effect `u_walk`, the plan jumps to line 11. There, the

[2]The full plan as well as more experiments can be found in the extended version of this paper (Ciolek et al. 2020). The code to perform the compilation from a MTP to Dual-FOND task can be found at https://github.com/ssardina-planning/pypddl-translator.

```
1(:plan [
2  (walk_d3 c2 c1)
3  (if ((not (u_walk))) [
4    (walk_d3 c1 c0)
5    (if ((not (u_walk))) [
6      (check_goal_d3)
7    ])
8    ...
9  ]
10 (else) [
11   (walk_unfair)
12   (case (eff_e3_walk) [
13     (walk_e3_explained_by_d3)
14     (continue_d3)
15     ...
16   ]
17   (case (eff_e2_walk) [
18     (walk_e2_explained_by_d2)
19     (degrade_d3_d2)
20     ...
21   ]
22   (case (eff_e1_walk) [
23     (walk_e1_explained_by_d1)
24     (degrade_d3_d1)
25     ...
26   ]
27 ]  ])
28
```

Listing 2: A fragment of the policy found by FOND-SAT.

only action available is the *unfair* version of walking (line 11), which has *all* the effects, as non-deterministic options, of the walking action across all domains $\mathcal{D}_3$, $\mathcal{D}_2$, and $\mathcal{D}_1$. As FOND-SAT does not handle conditional effects, we simulate each conditional effect for the effect chosen by a set of `walk_eE_explained_by_dx` whose precondition is $C^E_{\mathcal{D}_x}$, together with the original precondition of the operator (`walk` in this case). Finally, if the effect chosen could be explained by the current operating domain (line 13, explained by domain $\mathcal{D}_3$), the system executes a continue operation at the current level, enabling the next domain action. On the other hand, when the effect is explained by a lower domain than the one operating under (lines 18 and 23), degradation to the corresponding domain is carried out (line 19 and 24). It is easy to see how this plan also represents a MTC, by taking only the fair versions of the operators (all other actions and propositions encode the controller's internal memory).

Now, what would happen if the robot starts scratched, as discussed in Example 1? It turns out the problem becomes *unsolvable*. The reason is that any observed scratch after movement does *not* need to be explained by a different model than $\mathcal{D}_3$, as the scratch is explained already by being true originally. Thus, when walking always advances the agent, it would never degrade its behavior, remain operating in $\mathcal{D}_3$ without ever achieving $\mathcal{D}_3$'s goal. If, however, we drop the non-scratched requirement from $\mathcal{D}_3$, the problem would be solvable again, though with a slightly different policy. The robot would just try to achieve the top goal, degrading only to $\mathcal{D}_1$ if it does not move after a walk action. Since, as discussed, $\mathcal{D}_2$'s scratch effect would be explained by $\mathcal{D}_3$ itself, line 18 would become `walk_e2_explained_by_l3` and line 19 would become `continue_l3`.

While the above demonstrates the possibility to solve MTPs using existing planning technology, running our simple example takes around 600 seconds to produce a 29 states controller in an i7-4510 CPU with 8GB of RAM, when using the off-the-shelf version of the planner. This clearly indicates the need for more and better dual-FOND implementations or the development of specialized optimizations for MTPs. For example, as we are only allowing degradation and not upgrades, one can modify the SAT encoding to specify a number of controllers to be used per domain level, without allowing transitions from lower to upper levels. In preliminary tests we did we experienced a speed-up of approx 30%. Another optimization involves an estimation of the number of controllers required to solve the MTP, for example by running the top level domain which will provide a lower bound.

Conclusions

To some extent, this work aims to contribute to Ghallab, Nau, and Traverso (2014)'s call for an *actors' perspective* on planning, by proposing a planning-based framework that "integrates better planning and acting." Under such framework, the knowledge engineer has the opportunity to consider multiple levels of assumptions and goals. The problem amounts to synthesizing a meta-controller that is able to gracefully degrade its "level of service" when the assumptions on the environment are not met. We developed a compilation technique to construct such adaptive meta-controllers via dual-FOND planning. We note that plain FOND planning, under which every action is assumed fair, would *not* work: the agent could decide to keep trying an action to obtain one of its "failing" effects so as to force an un-intended degradation that will activate an "easier" lower-level goal (this artifact was already observed by Camacho and McIlraith (2016)).

Our work is related to works aiming to extend the classical planning formalism to accommodate more robust behavior, including Fault Tolerant and Robust Planning (FTP), e.g., (Domshlak 2013; Buffet and Aberdeen 2005), planning with richer "dynamic" goals, e.g., (Lago, Pistore, and Traverso 2002; De Giacomo et al. 2016; Shivashankar et al. 2012). It also keeps some relationship with different control architectures (Myers 1999; McGann et al. 2008), though those systems tend to *replan* when goals (or perceived state) happen to change, whereas we aim to generate policies that take into account all the scenarios *prior* to the system's execution (as per model). See (Ciolek et al. 2020) for more details.

As stated, our proposal was inspired by that of D'Ippolito et al. (2014) in Software Engineering. Their account, though, is limited to a linear hierarchy of models, so independent assumptions, as in our example, cannot be represented. Moreover, D'Ippolito et al.'s framework require solution (sub-)controllers of lower tiers to *simulate* those in

upper tiers, and thus it would not be able to handle our no-running example. Finally, being rooted in knowledge representation, we are able to exploit planning technology.

There are several limitations of our framework in its current form. The approach is based on lattice structures that do not guarantee full ordering of environment models. As a consequence, the executor may have to arbitrarily select and commit to one degradation option from multiple available ones, without being able to re-consider such choice later on upon further inconsistencies. Also, we have not provided a mechanism for *enhancement*, that is, for "upgrading" to more refined models, for example, when certain transient failure has been fixed. An option could be allowing the specification of known *repair* actions as part of the model (e.g., servicing an artifact or emptying a tank) that would "fix" a given unintended effect. It is not clear, however, how to enforce the execution of such repair actions *only* when they (re)enable higher-level goals.

Acknowledgements

Alberto Pozanco carried out this work during his visit to RMIT University supported by FEDER/Ministerio de Ciencia, Innovación y Universidades – Agencia Estatal de Investigación TIN2017-88476-C2-2-R and RTC-2016-5407-4.

References

Bonet, B., and Geffner, H. 2015. Policies that generalize: Solving many planning problems with the same policy. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2798–2804.

Buffet, O., and Aberdeen, D. 2005. Robust planning with (l)rtdp. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1214–1219.

Camacho, A., and McIlraith, S. A. 2016. Strong-cyclic planning when fairness is not a valid assumption.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Ciolek, D.; D'Ippolito, N.; Pozanco, A.; and Sardina, S. 2020. Multi-tier automated planning for adaptive behavior (extended version). In *arXiv e-prints, https://arxiv.org/abs/2002.12445*.

De Giacomo, G.; Gerevini, A.; Patrizi, F.; Saetti, A.; and Sardina, S. 2016. Agent planning programs. *Artificial Intelligence* 231:64–106.

D'Ippolito, N.; Braberman, V. A.; Kramer, J.; Magee, J.; Sykes, D.; and Uchitel, S. 2014. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. 688–699.

Domshlak, C. 2013. Fault tolerant planning: Complexity and compilation. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 193–202. AAAI Press.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geffner, T., and Geffner, H. 2018. Compact policies for fully observable non-deterministic planning as SAT. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, 88–96.

Gerevini, A.; Bonet, B.; and Givan, B., eds. 2006. *Booklet of 4th International Planning Competition*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2014. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence* 208:1–17.

Kissmann, P., and Edelkamp, S. 2009. Solving fully-observable non-deterministic planning problems via translation into a general game. In *Proceedings of the Annual German Conference on AI*, 1–8.

Kuter, U.; Nau, D., S.; Reisner, E.; and Goldman, R., P. 2008. Using classical planners to solve nondeterministic planning problems. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 190–197.

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 447–454.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. S. 2008. A deliberative architecture for AUV control. In *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA*, 1049–1054. IEEE.

Muise, C.; Belle, V.; and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 172–180.

Myers, K. L. 1999. CPEF: A continuous planning and execution framework. *AI Magazine* 20(4):63–69.

Rintanen, J. 2004. Complexity of planning with partial observability. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 345–354.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *Proc. of the European Conference in Artificial Intelligence (ECAI)*, 568–572.

Sardina, S., and D'Ippolito, N. 2015. Towards fully observable non-deterministic planning as assumption-based reactive synthesis. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 3200–3206.

Shivashankar, V.; Kuter, U.; Nau, D. S.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 981–988.