Compiling Planning Problems with Non-deterministic Events into FOND Planning

Lukáš Chrpa[1,2], Martin Pilát[1], and Jakub Gemrot[1]

[1] Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
[2] Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Abstract. Automated Planning seeks to find a sequence of actions, a plan, transforming the environment from its initial state to some goal state. In real-world environments, however, exogenous events might occur and might modify the environment without agent's consent. Besides disrupting agent's plan, events might hinder agent's pursuit towards its goals and even cause damage (e.g. destroying the robot).
In this paper, we present how planning problems with non-deterministic events can be translated into Fully-Observable Non-Deterministic (FOND) planning problems and hence we can exploit FOND planning engines to solve them (i.e., find strong cyclic plans). Specifically, we will consider two cases in a single agent scenario – at most one event can occur after agent's action or a set of independent events can occur after agent's action. We compare the FOND-based approach with a traditional planning/execution/replanning one to highlight the performance gap as well as success rate of the latter approach.

1 Introduction

Planning and acting in real-world scenarios [8] encounters numerous challenges. For example, non-deterministic events (e.g. failure to establish communication, or a blocked passage) that might (or might not) happen under specific circumstances can change the environment without the consent of the actor.

The concept of events in planing is not new [5] and was used in some systems such as Circa [12]. These systems, however, reason with a very small state space. MDP-based approaches consider events [9] as well as Monte Carlo Tree Search based approaches [14]. They, however, focus on selecting the most promising action in a current state. Fully-Observable Non-Deterministic (FOND) planning considers non-deterministic action effects [3] that we can leverage for problems with non-deterministic events.

In this paper, we focus on a single-agent planning in fully observable environment with deterministic action effects and non-deterministic exogenous events where, roughly speaking, the task is to find policies such that the agent eventually reaches its goal (i.e., strong cyclic plans) under the fairness assumption that each event when applicable has a chance to occur. We consider two assumptions such that in the first one after agent's action at most one event can occur, while in the second one after agent's action a set of independent events can occur. We show how planning problems with events under both assumptions can be compiled to FOND planning that focuses on a different challenge – non-deterministic action effects (note that, for example, problems with partially observable environment can also be compiled to FOND [11]). Solutions of the compiled FOND problems (strong cyclic plans) can be translated to solutions of problems with non-deterministic events. Such an approach guarantees completeness under the fairness assumption, in other words, solutions of problems with events are "safe" to execute and the agent eventually reaches its goal. Consequently, solutions are "robust" even for problems with dead-ends which, on the other hand, are problematic for the planning/execution/replanning (PER) approach (e.g. FF-Replan [17]) that interleaves planning and execution phases while replanning if the next action becomes inapplicable, or events change the environment. We experimentally compare the FOND-based approach with a traditional planning/execution/replanning one to highlight the performance gap as well as success rate of the latter approach.

2 Preliminaries

Classical planning, in particular, assumes a static, deterministic and fully observable environment; a solution plan amounts to a sequence of actions. Technically, a ***classical planning domain model*** is a tuple $\mathcal{D} = (L, A)$, where L is the set of propositional atoms used to describe the ***state*** of the environment (set of propositions from L that are true), and A is the set of actions over L. An ***action*** is a tuple $a = (pre(a), del(a), add(a))$, where $pre(a)$, $del(a)$ and $add(a)$ are sets of atoms from L representing a's precondition, delete, and add effects, respectively. We assume that $del(a) \cap add(a) = \emptyset$. An action a is *applicable* (or executable) in a state s if and only if $pre(a) \subseteq s$. If possible, application (or execution) of a in s, denoted as $\gamma(s, a)$, yields the successor state of the environment $(s \setminus del(a)) \cup add(a)$, otherwise $\gamma(s, a)$ is undefined. The notion of applicability can be extended to sequences of actions, i.e., $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\ldots \gamma(s, a_1) \ldots, a_n)$.

A ***classical planning problem*** is a tuple $\mathcal{P} = (\mathcal{D}, I, G)$, where $\mathcal{D}$ is a planning domain model, I is the initial state of the environment, and G is the goal, generally in the form of a set of propositions. A ***solution plan*** (for a planning problem) is a sequence of actions such that their consecutive application starting in the initial state results in a state satisfying the goal (i.e., a goal state).

2.1 FOND Planning

Fully Observable Non-Deterministic (FOND) planning assumes fully observable and static environment but in contrast to classical planning actions have non-deterministic effects [3, 6]. In particular, the result of application might be one of the sets of effects. Formally a ***non-deterministic action*** is a tuple $a = (pre(a), del^1(a),\ add^1(a), \ldots, del^k(a), add^k(a))$, where $pre(a)$ its precondition, and $del^1(a), add^1(a), \ldots, del^k(a), add^k(a)$ its delete and add effects respectively. Action applicability is the same as for classical planning. The result of applying a in s (if possible) is one of the states from $\{s' \mid s' = (s \setminus del^i(a)) \cup add^i(a), 1 \leq i \leq k\}$. The FOND planning problem definition is analogous to the classical planning problem definition. The task in FOND planning is to find a ***strong (a)cyclic plan***, which forms an (a)cyclic graph such that its nodes are states (including the initial state) and directed edges go from each state where a (single) non-deterministic action can be applied to all possibly resulting states, and from all states there is a path to a goal state. In plain words, strong cyclic plans guarantee that under the fairness assumption, i.e., each action outcome has a chance to occur, the agent will eventually achieve its goal [3].

2.2 Events

Similarly to the definition of an action, an ***event*** is a tuple $e = (pre(e), del(e), add(e))$, where $pre(e)$, $del(e)$ and $add(e)$ are sets of atoms representing e's precondition, delete, and add effects, respectively. We assume that $del(e) \cap add(e) = \emptyset$. Applicability of an event in a state as well as the result of an application (or execution) of an event is defined in the same way as for actions. In contrast to actions that are executed by agents, events can occur regardless of agent's consent. Technically, an event can (but does not necessarily have to) occur in a state where event's precondition is met modifying the state of the environment according to event's effects. A ***planning domain model***, in this case, is a triple $\mathcal{D} = (L, A, E)$, where L is the set of propositions, A and E is the set of actions and events over L, respectively. A planning problem, $\mathcal{P} = (\mathcal{D}, I, G)$, is defined analogously to the classical planning case.

We define a ***noop*** action/event that represents that no action has been taken by agent, or event has occurred. Formally, $pre(noop) = del(noop) = add(noop) = \emptyset$.

We say that events e_i, e_j are ***independent*** if and only if $del(e_i) \cap (pre(e_j) \cup add(e_j)) = \emptyset$ and $del(e_j) \cap (pre(e_i) \cup add(e_i)) = \emptyset$.

The following assumption simplifies the reasoning by considering a single-agent scenario such that actions of the agent and events of the environment alter like in a two-players game. The process hence follows the pattern in which the agent can apply an action (not necessarily has to), then the environment can trigger (apply) an event (not necessarily has to), and so on. Formally:

Assumption 1. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model. In a current state $s \in 2^L$, the agent can apply an action $a \in A \cup \{noop\}$ such that a is applicable in s. After that, a (arbitrarily selected) event $e \in E \cup \{noop\}$, applicable in $\gamma(s, a)$, is applied resulting in a state $s' = \gamma(\gamma(s, a), e)$ which will become a new current state. We say that s' is a* **successor** *state of s, a.*

The following assumption extends the previous assumption by considering the pattern in which the agent can apply an action (not necessarily has to), then the environment can trigger (apply) a set of independent events (not necessarily has to), and so on. Formally:

Assumption 2. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model. In a current state $s \in 2^L$, the agent can apply an action $a \in A \cup \{noop\}$ such that a is applicable in s. After that, a (arbitrarily selected) set of independent events $E^i \subseteq E \cup \{noop\}$, applicable in $\gamma(s, a)$, is applied resulting in a state $s' = \gamma(\gamma(s, a), E_i)$ which will become a new current state. We say that s' is a* **successor** *state of s, a.*

Let $\mathcal{D} = (L, A, E)$ be a planning domain model. We say that a state $s' \in 2^L$ is ***reachable*** from a state $s \in 2^L$ with respect to $\mathcal{D}$ and Assumption 1 (Assumption 2) if and only if there exists a sequence of actions and events (sets of independent events), including noops, such that its consecutive application in s results in s'. Otherwise, we say that s' is ***unreachable*** from s.

Let $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. Atoms p, q for which it is the case that there does not exist a state s reachable from I such that $p, q \in s$ are ***mutually exclusive***, or shortly ***mutex***.

Solutions in non-deterministic planning are in the form of policies, mappings from states to actions (similarly to probabilistic or FOND planning, for example). Policies, in plain words, provide information about which action the agent has to apply in a current state of the environment. As we assume events might be triggered between actions the agent executes, the policy execution interleaves agent's actions (including "noop") and environment's events (including "noop").

Definition 1. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model and $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. We say that $\Pi : 2^L \rightarrow A \cup \{noop\}$ is a* **policy** *for $\mathcal{P}$ such that for each $s \in 2^L$ reachable from I, $\Pi(s)$ is applicable in s.*

In analogy to FOND planning, we can define a *strong cyclic plan* that represents a policy that, roughly speaking, has to guarantee that the agent will eventually achieve its goal under the fairness assumption that each applicable event/set of independent events (including "noop") in each (reachable) state has a chance to be applied. The strict variant, i.e., a *strong acyclic plan*, guarantees that the agent always reaches a goal state in a finite number of steps (no state can be visited more than once).

Definition 2. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model and $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. Let π be a policy for $\mathcal{P}$. We construct a directed graph $\mathcal{G}$ such that its nodes are states $S \subseteq 2^L$, $I \in S$, for each $s \in S$ and each its successor state s' of $s, \pi(s)$ there is an edge from s to s'.*

A **strong cyclic plan** *for $\mathcal{P}$ is a policy π (for $\mathcal{P}$) such that for each state s in $\mathcal{G}$ there is a path to a goal state.*

A **strong acyclic plan** *for $\mathcal{P}$ is a policy π (for $\mathcal{P}$) such that for each state s in G there is a path to a goal state and $\mathcal{G}$ is acyclic.*

2.3 Relations between Actions and Events

Actions as well as events influence each other by achieving atoms that are required by other actions/events, or, in contrast, delete atoms required by other actions/events. An early work of Chapman [1] studies relations between actions such as being an "achiever" (i.e., one action creates an atom for another actions) or being a "cloberrer" (i.e., an action deletes an atom required by another action). Inspired by this work, we define notions of an *enabler* and a *disabler* that denote whether an action or event achieves, or deletes, an atom for an event.

Definition 3. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model. We say that an action/event $x \in A \cup E$ is an* **enabler** *for an event $e \in E$ if $add(x) \cap pre(e) \neq \emptyset$. We also say that x is a* **sole enabler** *for e if $add(x) \supseteq pre(e)$.*

Definition 4. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model. We say that an action/event $x \in A \cup E$ is a* **disabler** *for an event $e \in E$ if $del(x) \cap pre(e) \neq \emptyset$.*

2.4 Conditional Effects

Standard action (and event) definition can be extended by *Conditional Effects* that capture possible state changes if some extra condition is met in the current state. Formally, a **conditional effect** of an action (or event) x is specified as $ceff(x) = (cond(x), cdel(x), cadd(x))$ where $cond(x)$ is a set of atoms representing a condition and $cdel(x)$, $cadd(x)$ are sets of atoms representing conditional delete and add effects respectively. In plain words, if a condition in a current state is met, then the effects take place in the resulting state after action application. Action (or event) definition can be extended as follows $x = (pre(x), del(x), add(x), ceff^1(x), \ldots, ceff^k(x))$, where $ceff^1(x), \ldots, ceff^k(x)$ are conditional effects of x. Applicability of x in a state s is still determined as whether $pre(x) \subseteq s$. The result of application of x in s is

$$s \setminus (del(x) \cup \bigcup_{cond^i(x) \subseteq s} cdel^i(x)) \cup add(x) \cup \bigcup_{cond^i(x) \subseteq s} cadd^i(x)$$

In a nutshell, conditional effects do not influence action (or event) applicability but modify the resulting state if the conditions are met in the current state. Conditional effects can be compiled away, i.e., an equivalent classical representation can be obtained, however, the representation either grows exponentially or the length of solution plans grows polynomially [13].

For the sake of clarity, we will use conditional effects in our compilations of planning problems with events into FOND planning. Conditional effects are now supported by many state-of-the-art classical planners [16] as well as the FOND planner PRP [10].

3 Case Studies

To illustrate our concepts we introduce three case studies.

3.1 “Sticky” and “Slippery” BlocksWorld

We introduce a variant of the well known 4-ops BlocksWorld domain [7]. In this variant, we have two types of robotic hands that can move blocks. One robotic hand, called “slippery”, is not able to hold blocks firmly, so they might fall down to the table while held by this hand. The other robotic hand, called “sticky”, is able to hold blocks firmly, however, blocks will get “sticky” and might eventually become unmovable, i.e., get sticked to the table or to another block, so no robotic hand can unstack them, or pick them up.

Technically speaking, the operators unstack, stack, pickup and putdown are amended as follows. Unstack and pickup are split into “slippery” and “sticky” variants, where each variant requires the corresponding robotic hand as a precondition. Precondition of both variants of unstack and pickup contains a (movable ?b) predicate representing whether a block ?b can be moved by either robotic hand. The add effects of the “sticky” variant of unstack and pickup is extended by a (sticky ?b) predicate representing that a block ?b became “sticky”.

We define two events such that an event slip causes the block held by the “slippery” hand to fall onto the table and an event stick makes the “sticky” block unmovable (by removing the (movable ?b) predicate).

A strong acyclic plan unstacks blocks from their initial position with the “slippery” hand and puts them down on the table (if they do not fall down by themselves) and then stack them on their goal positions with the “sticky” hand (if they are in the goal positions they do not have to be moved). A strong cyclic plan, for instance, can afford to use only the “slippery” hand, since blocks that fell down can be picked up again (and eventually be placed into their goal positions).

3.2 AUV Sampling

We introduce a simplified variant of task planning for AUV, inspired by the recent work of Chrpa et al. [2]. It simulates the situation where an AUV has to perform sampling of some objects of interest while there might be ships passing by that might endanger the AUV. We have a 4-grid environment, an AUV, a ship and several resources. Resources can be found on given cells. Each cell is either free, has the AUV on it, or the ship on it (presence of a resource does not interfere with any cell status). The AUV can move to an adjacent cell, if the cell is free. The AUV can sample a resource if it is at the same cell. The task for the AUV is to sample the resources and return back to the place of origin.

Ships, however, are not controlled by the agent, i.e., ships are controlled by the environment. Ships can move only on some cells of the grid or might not be present in the area. Each ship can enter the area at its entry cells, can move to adjacent cells according to its route, and leave the area at its exit cells. A ship can appear in its entry cell, if the ship is not already in the area. A ship can leave the area, if it is in its exit cell. Two “move” events are considered, move-ship-to-free and move-ship-to-auv. Both require that the ship can move to the destination cell. The effect of both events is that the ship moves to the destination cell. If the ship moves to a free cell, then besides the cell becomes not free for a moment, nothing else happens. However, if the ship moves to the cell with the AUV, then the AUV is destroyed (and can no longer perform any action).

A strong cyclic plan has to avoid situations in which the AUV is, after applying its action, next to any ship, or in any ship's entry point (if the ship is not yet in the area). Strong acyclic plans might not always exist, since ships can (if being active adversaries) prevent the AUV to get to some resources.

3.3 The "Perestroika" domain

The Perestroika domain we introduce here is inspired by the well known Perestroika game (also known as a Toppler game). In our domain, an agent has to navigate through a 4-grid of solid and shrinking platforms and collect all resources that can be placed on solid platforms. Solid platforms remain stable, i.e., they do not change its size nor disappear. In contrary, the shrinking platforms can have *large*, *medium* or *small* shape, or disappear completely.

The agent can perform two types of actions. It can move to a neighbouring platform (if it has not disappeared) and/or collect a resource if the resource is on the same platform as the agent. Shrinking platforms are affected by five events each. Two events change the shape of the platform from *large* to *medium* and from *medium* to *small*, respectively. Two events make the platform disappear if it has a small shape – the difference is whether the platform is empty or the agent is on it. In the former case, the platform just disappears whereas in the latter case it kills the agent. The last event allow the platform to reappear in a *large* shape.

A strong cyclic plan has to avoid situations in which the agent moves to or stays on small platforms. Also, the agent must always have an escaping way to a nearby solid platform to avoid being "trapped" on a shrinking platform which might disappear at some point and kill the agent. Similarly to the AUV domain, strong acyclic plans might not always exist as the agent might have to wait until the shrinking platforms are large enough to be safely crossed.

4 Translating Planning with Events into FOND Planning

In FOND planning, the agent does not have control of action outcome but the outcome is determined immediately after action execution finishes. In contrast, an event might be applicable even without "enabling" it by agent's actions (e.g., the event preconditions are met in the initial state), or when "enabled" by agent's action event's applicability does not cease until some other action or event "disables" it.

4.1 Considering Assumption 1

Let $\mathcal{D} = (L, A, E)$ be a domain model. We construct an equivalent FOND domain model $\mathcal{D}^F = (L^F, A^F)$ as follows. We introduce atoms (act-turn) and (ev-turn)

https://en.wikipedia.org/wiki/Perestroika_(video_game) to determine "action" and "event" turn respectively. For each event $e_i \in E$ we introduce atoms $\mathsf{(enab\text{-}ei)}$ and $\mathsf{(disab\text{-}ei)}$ representing whether e_i is applicable in a current state or not, and an atom $\mathsf{(selected\text{-}ei)}$ representing whether e_i has been selected. L^F is constructed by a union of the introduced atoms and L (without loss of generality we assume that none of the introduced atoms is in L). An action $a_j \in A$ is translated into $a_j^F \in A^F$ as follows:

$$\begin{array}{rcl}
pre(a_j^F) & = & pre(a_j) \cup \{\mathsf{(act\text{-}turn)}\} \\
del(a_j^F) & = & del(a_j) \cup \{\mathsf{(act\text{-}turn)}\} \cup \\
 & & \cup\{\mathsf{(enab\text{-}ei)} \mid a_j \text{ is a disabler for } e_i\} \cup \\
 & & \cup\{\mathsf{(disab\text{-}ei)} \mid a_j \text{ is a sole enabler for } e_i\} \\
add(a_j^F) & = & add(a_j) \cup \{\mathsf{(ev\text{-}turn)}\} \cup \\
 & & \cup\{\mathsf{(disab\text{-}ei)} \mid a_j \text{ is a disabler for } e_i\} \cup \\
 & & \cup\{\mathsf{(enab\text{-}ei)} \mid a_j \text{ is a sole enabler for } e_i\} \\
\multicolumn{3}{l}{\text{For each } \ e_i \in E \text{ such that } a_j \text{is a non-sole enabler for } e_i} \\
\mathit{ceff}^{e_i}(a_j^F) & = & (pre(e_i) \setminus add(a_j), \mathsf{(disab\text{-}ei)}, \mathsf{(enab\text{-}ei)})
\end{array}$$

On top of the actions defined in the domain model we have to explicitly model the "noop" action $a_{noop}^F \in A^F$ that only switches from the action to the event turn (as the agent decided to do nothing in its turn). In particular, $pre(a_{noop}^F) = del(a_{noop}^F) = \{\mathsf{(act\text{-}turn)}\}$ and $add(a_{noop}^F) = \{\mathsf{(ev\text{-}turn)}\}$.

The a_j^F actions can be executed in the "action turn" and since they correspond to the actions in A their effects are deterministic. After executing an action we move to the "event turn".

In the "event turn", we have to "transfer" non-deterministic events into non-deterministic action effects. We can do that in two steps. First, we select an event or "noop" by non-deterministic action effects. Then, if the selected event is enabled, the event is executed, otherwise (if the event is disabled) the case is considered as "noop". The "event selecting" action $a_{sel} \in A^F$ is encoded as follows, $del^0(a_{sel})$ and $add^0(a_{sel})$ represent the selection of "noop" (i.e., no event will be executed in the current "turn") while $del^i(a_{sel})$ and $add^i(a_{sel})$ represent the selection of an event $e_i \in E$ (all the events in E are considered):

$$\begin{array}{r}
pre(a_{sel}) = \{\mathsf{(ev\text{-}turn)}\} \\
del^0(a_{sel}) = \{\mathsf{(ev\text{-}turn)}\} \\
add^0(a_{sel}) = \{\mathsf{(act\text{-}turn)}\} \\
del^i(a_{sel}) = \{\mathsf{(ev\text{-}turn)}\} \\
add^i(a_{sel}) = \{\mathsf{(selected\text{-}ei)}\}
\end{array}$$

Then, for each event $e_j \in E$ we construct two actions $a_{e_j}^e \in A^F$ and $a_{e_j}^d \in A^F$ representing an execution of e_j if enabled or resorting to the "noop" case if e_j

is disabled:

$$
\begin{array}{rcl}
pre(a^e_{e_j}) & = & \{\mathsf{(selected\text{-}ej)}, \mathsf{(enab\text{-}ej)}\} \\
del(a^e_{e_j}) & = & del(e_j) \cup \{\mathsf{(selected\text{-}ej)}\} \cup \\
 & & \cup\{\mathsf{(enab\text{-}ei)} \mid e_j \text{ is a disabler for } e_i\} \cup \\
 & & \cup\{\mathsf{(disab\text{-}ei)} \mid e_j \text{ is a sole enabler for } e_i\} \\
add(a^e_{e_j}) & = & add(e_j) \cup \{\mathsf{(act\text{-}turn)}\} \cup \\
 & & \cup\{\mathsf{(disab\text{-}ei)} \mid e_j \text{ is a disabler for } e_i\} \cup \\
 & & \cup\{\mathsf{(enab\text{-}ei)} \mid e_j \text{ is a sole enabler for } e_i\} \\
\multicolumn{3}{l}{\text{For each } \ e_i \in E \text{ such that } e_j \text{is a non-sole enabler for } e_i} \\
\mathit{ceff}^{e_i}(a^e_{e_j}) & = & (pre(e_i) \setminus add(e_j), \{\mathsf{(disab\text{-}ei)}\}, \{\mathsf{(enab\text{-}ei)}\}) \\
pre(a^d_{e_j}) & = & \{\mathsf{(selected\text{-}ej)}, \mathsf{(disab\text{-}ej)}\} \\
del(a^d_{e_j}) & = & \{\mathsf{(selected\text{-}ej)}\} \\
add(a^d_{e_j}) & = & \{\mathsf{(act\text{-}turn)}\}
\end{array}
$$

For a planning problem $\mathcal{P} = (\mathcal{D}, I, G)$, the corresponding FOND problem $\mathcal{P}^F = (\mathcal{D}^F, I^F, G)$ is constructed such that

$$I^F = I \cup \{\mathsf{(act\text{-}turn)}\} \cup \{\mathsf{(enab\text{-}ei)} \mid pre(e_i) \subseteq I\} \cup \{\mathsf{(disab\text{-}ei)} \mid pre(e_i) \not\subseteq I\}.$$

We can see that to solve the FOND planning problem we have to simulate "action" and "event" turns. The (act-turn), (ev-turn), (selected-ei) atoms ensure the applicability of an action, of the event selection action, of an event, respectively. In particular, in the action turn an action is (non-deterministically) selected. The event turn comprises an "event selecting" action and possibly an "event" action. The "event selecting" action represents event selection (or noop selection) in its non-deterministic effects. Noop selection skips the "event" action (as no event occurs). Otherwise, if an event e_i is selected ((selected-ei) becomes true), then either $a^e_{e_i}$ or $a^d_{e_i}$ can be applied depending whether e_i is applicable or not. Applying $a^e_{e_i}$ simulates e_i application while $a^d_{e_i}$ simulates noop as e_i is inapplicable. Also, we have to maintain information about applicability of particular events, which is done through the (enab-ei) and (disab-ei) atoms. If an action or event is a sole enabler for e_i, then it adds (enab-ei) and removes (disab-ei). Analogously, if an action or event is a disabler for e_i, then it adds (disab-ei) and removes (enab-ei).

Let π^F be a strong cyclic plan of $\mathcal{P}^F$, then a strong cyclic plan π for a corresponding planning problem $\mathcal{P}$ is constructed as follows. For each $s^F \in 2^{L^F}$ and $a^F_i \in A^F$ such that $\pi^F(s^F) = a^F_i$ we define $\pi(s^F \cap L) = a_i$. In plain words, we consider only pairs $\langle$state,action$\rangle$ from the strong cyclic plan associated with actions translated from actions defined in $\mathcal{P}$. As states of $\mathcal{P}^F$ contain additional atoms, these have to be removed to obtain states of $\mathcal{P}$. It can be observed that if a strong cyclic plan of $\mathcal{P}^F$ is found there is no "open state" which is not a goal state. As events occur implicitly (as in Assumption 1), the corresponding strong cyclic plan of $\mathcal{P}$ has to consider only "action turns". The above is summarised in the following theorem.

Theorem 1. *Let $\mathcal{P}$ be a planning problem (with events) and $\mathcal{P}^F$ be its translation into a FOND problem (as described above). Then, π^F is a strong cyclic plan of $\mathcal{P}^F$ if and only if π (constructed from π^F as above) is a strong cyclic plan of $\mathcal{P}$.*

Proof (Sketch). *To find a strong cyclic plan of $\mathcal{P}^F$ the "action" and "event" turns alternate. In particular, in the action turn a planner (non-deterministically) selects an action. The event turn comprises an "event selecting" action and possibly an "event" action so simulate all possibilities of event occurrence. The "event selecting" action represent possibilities of event selection (or noop selection) in its non-deterministic effects. Noop selection skips the "event" action (as no event occurs). Otherwise, if an event e_i is selected ((selected-ei) becomes true), then either $a^e_{e_i}$ or $a^d_{e_i}$ can be applied depending whether e_i is applicable or not. Applying $a^e_{e_i}$ simulates e_i application while $a^d_{e_i}$ simulates noop as e_i is inapplicable. It can be seen that (enab-ei) is true if and only if e_i is applicable while (disab-ei) is true if and only if e_i is inapplicable. Multiple noops caused by disabled events result in the same state as the regular noop. Enabled events result in (possibly) new states that have to be explored. If a strong cyclic plan of $\mathcal{P}^F$ is found there is no "open state" which is not a goal state. As events occur implicitly (as in Assumption 1), the corresponding strong cyclic plan of $\mathcal{P}$ has to consider only "action turns".*

4.2 Considering Assumption 2

Let $\mathcal{D} = (L, A, E)$ be a domain model. We construct an equivalent FOND domain model $\mathcal{D}^F = (L^F, A^F)$ as follows. We introduce atoms (act-turn), (enab-ei), (disab-ei) and (selected-ei) as in the previous case. On top of that, we introduce atoms (ev-turn) and (ev-turn2) representing two parts of the event turn and for each event $e_i \in E$, an atom notsel-ei representing that e_i has not been selected (in a given turn), and an atom (wenab-ei) representing that e_i will be enabled in the next turn. Similarly to the previous case, L^F is constructed by a union of the introduced atoms and L (without loss of generality we assume that none of the introduced atoms is in L). The translation of each action $a_j \in A$ into $a^F_j \in A^F$ as well as the noop action is the same as in the previous case.

The "event selecting" action $a_{sel} \in A^F$ has to reflect selection of sets of independent events (including noop). That involves calculating power sets for all sets of independent events. To reduce the number of these power sets we can exploit mutexes such that events whose preconditions are mutex are not present in any set together (despite being independent). In the AUV domain, for example, a single ship cannot move between different locations at the same time as it can be at at most one location, or outside at the same time.

The precondition of a_{sel} as well as the effects representing the noop event selection is the same as in the previous case. For a set of independent events

$\{e_{i_1}, \dots, e_{i_m}\} \subseteq E$ we define non-deterministic effects $del^i(a_{sel})$ and $add^i(a_{sel})$ in which all the sets of independent events from E are considered:

$$\begin{aligned} del^i(a_{sel}) &= \{(\textsf{ev-turn}), (\textsf{notsel-ei1}), \dots, (\textsf{notsel-eim})\} \\ add^i(a_{sel}) &= \{(\textsf{ev-turn2}), (\textsf{selected-ei1}), \dots, (\textsf{selected-eim})\} \end{aligned}$$

For the "event" actions $a^e_{e_j} \in A^F$ and $a^d_{e_j} \in A^F$ representing an execution of e_j (if enabled or resorting to the "noop" case if disabled), the main difference, in contrast to the previous case, is that after applying the "event" action, we are still in the "event" turn (hence we can apply all the events from the selected set). Also, enabling events must not influence applicability of selected events because the selected (independent) events have to be applied at one step. It, however, might be that case that one independent event is an enabler for another independent event (on the other hand, an independent event cannot be a disabler for another independent event) and thus it has to be indicated that an event will be enabled in the following step (by the wenab atom).

$$\begin{aligned} pre(a^e_{e_j}) &= \{(\textsf{selected-ej}), (\textsf{enab-ej})\} \\ del(a^e_{e_j}) &= del(e_j) \cup \{(\textsf{selected-ej})\} \cup \\ &\quad \cup \{(\textsf{enab-ei}), (\textsf{wenab-ei}) \mid e_j \text{ is a disabler for } e_i\} \\ add(a^e_{e_j}) &= add(e_j) \cup \{(\textsf{notsel-ej})\} \cup \\ &\quad \cup \{(\textsf{disab-ei}) \mid e_j \text{ is a disabler for } e_i\} \cup \\ &\quad \cup \{(\textsf{wenab-ei}) \mid e_j \text{ is a sole enabler for } e_i\} \\ &\text{For each } \ e_i \in E \text{ such that } e_j \text{is a non-sole enabler for } e_i \\ ceff^{e_i}(a^e_{e_j}) &= (pre(e_i) \setminus add(e_j), \{\}, \{(\textsf{wenab-ei})\}) \end{aligned}$$

$$\begin{aligned} pre(a^d_{e_j}) &= \{(\textsf{selected-ej}), (\textsf{disab-ej})\} \\ del(a^d_{e_j}) &= \{(\textsf{selected-ej})\} \\ add(a^d_{e_j}) &= \{(\textsf{notsel-ej})\} \end{aligned}$$

Resorting back to the "action" turn can be done after all the selected events (respectively their corresponding "event" actions) were applied. In other words, if no event is selected, then we can resort to the "action" turn while enabling events that were marked as "will be enabled". The "resorting" action $a_{res} \in A^F$ is defined as follows:

$$\begin{aligned} pre(a_{res}) &= \{(\textsf{ev-turn2}), (\textsf{notsel-e1}), \dots, (\textsf{notsel-en})\} \\ del(a_{res}) &= \{(\textsf{ev-turn2})\} \\ add(a_{res}) &= \{(\textsf{act-turn})\} \\ &\text{For each } \ e_i \in E : \\ ceff^i(a_{res}) &= (\{(\textsf{wenab-ei})\}, \{(\textsf{wenab-ei}),(\textsf{disab-ei})\}, \{(\textsf{enab-ei})\}) \end{aligned}$$

For a planning problem $\mathcal{P} = (\mathcal{D}, I, G)$, the corresponding FOND problem $\mathcal{P}^F = (\mathcal{D}^F, I^F, G)$ is constructed in such a way that $I^F = I \cup \{(\textsf{act-turn})\} \cup \{(\textsf{enab-ei}) \mid pre(e_i) \subseteq I\} \cup \{(\textsf{disab-ei}) \mid pre(e_i) \not\subseteq I\} \cup \{(\textsf{notsel-ei}) \mid e_i \in E\}$.

Similarly to the previous case, to solve the FOND planning problem we simulate action and event turns. The main difference is that in this case more (independent) events can be selected in one turn. Therefore, the event turn is divided into three stages, event selection, event application and resorting to the action turn. On top of that, as events might enable other events, even one independent event can be an enabler of another independent event, to correctly simulate the event turn we have to postpone a possible enabling of events until the end of the turn. Technically speaking, the "event selecting" action selects a set of independent events that are all sequentially processed by "event" actions. The effects of a selected event take place if the event is enabled, otherwise the event is skipped. However, as the event effects might enable other events (even independent ones) it is important to postpone the information about newly enabled events until all the selected events were processed. The information about newly enabled events has to be kept (the wenab atoms) and when all the selected events are processed, the "resorting" action uses the information to determine enabled events for the next turn (the enab atoms).

Let π^F be a strong cyclic plan of $\mathcal{P}^F$, then a strong cyclic plan π for a corresponding planning problem $\mathcal{P}$ is constructed as follows. Analogously to the previous case, for each $s^F \in 2^{L^F}$ and $a_i^F \in A^F$ such that $\pi^F(s^F) = a_i^F$ we define $\pi(s^F \cap L) = a_i$. In plain words, we consider only pairs $\langle$state,action$\rangle$ from the strong cyclic plan associated with actions translated from actions defined in $\mathcal{P}$. Also the additional atoms defined in $\mathcal{P}^F$ have to be removed to obtain states of $\mathcal{P}$. It can be observed that if a strong cyclic plan of $\mathcal{P}^F$ is found there is no "open state" which is not a goal state. As events occur implicitly (as in Assumption 2), the corresponding strong cyclic plan of $\mathcal{P}$ has to consider only "action turns". The above is summarised in the following theorem.

Theorem 2. *Let $\mathcal{P}$ be a planning problem (with events) and $\mathcal{P}^F$ be its translation into a FOND problem (as described above). Then, π^F is a strong cyclic plan of $\mathcal{P}^F$ if and only if π (constructed from π^F as above) is a strong cyclic plan of $\mathcal{P}$.*

Proof (Sketch). *Analogously to the proof sketch of Theorem 1, the "action" and "event" turns alternate and we keep track of which events are enabled or disabled. The difference is that a set of independent events can occur at once in a single event turn. An "event selecting" action therefore selects a set of independent events. Selected events are processed in a sequence one by one (by "event" actions) such that event effects take place if the event is enabled, otherwise the event is considered as "noop". However, as "event" actions are processed in sequence, effects of one might enable an event that is going to be processed later. That would have deviated from Assumption 2 as such an event is inapplicable in the state before the set of independent events can occur. To prevent enabling events that are disabled in a given turn, we use the* wenab *atoms that conditionally enable events. Conditional enabling if persists becomes "real" at the end of the event turn. The "resorting" action, which switches from event to action turn, can be applied only if all selected events have been processed (i.e., all the events*

	Assumption 1					Assumption 2				
Problem	RA-S	RA-T	RE-S	RE-T	FOND	RA-S	RA-T	RE-S	RE-T	FOND
AUV-p1	71	0.28	75	1.05	203.26	74	0.16	70	1.03	69.92
AUV-p2	49	0.45	39	1.50	-	37	0.30	24	1.39	-
AUV-p3	83	0.44	80	6.04	273.80	76	0.23	88	3.31	986.96
AUV-p4	57	0.75	57	8.10	-	49	0.42	57	3.94	-
AUV-p5	87	0.57	75	9.24	-	50	0.35	47	5.08	-
BW-p1	100	0.36	100	0.91	16.58	100	0.61	100	0.98	234.73
BW-p2	100	0.70	100	1.42	21.34	100	0.72	100	1.80	29.25
BW-p3	23	1.28	95	3.99	60.54	72	2.28	92	4.36	-
BW-p4	100	3.63	100	8.76	19.05	100	3.89	100	6.60	169.26
BW-p5	0	-	41	34.84	-	0	-	55	49.03	-
BW-p6	38	20.73	66	42.47	27.24	30	28.54	34	51.51	-
PER-p1	65	0.29	76	1.00	15.20	58	0.21	46	1.02	-
PER-p2	71	0.30	82	1.01	14.33	52	0.29	37	1.09	-
PER-p3	81	0.49	66	2.45	-	24	0.54	15	2.71	-
PER-p4	69	0.37	71	2.46	-	18	0.43	10	2.66	-
PER-p5	80	0.46	59	3.47	-	14	0.49	13	3.14	-

Table 1. (S)uccess rate (%) and average (T)ime (successful runs only) of the replanning approach – the next action becomes inapplicable (RA) or event(s) occur (RE), and the planning time of the FOND approach. All times are in seconds.

are "not selected"). On top of that, the "resorting" action makes all conditionally enabled events enabled. Noteworthy, the selecting action uses the (ev-turn) *atom and achieves the* (ev-turn2) *atom for the resorting action such that they are applied in the correct order.*

Disabling events cannot affect those selected in the current event round because one of the properties for events being independent is that delete effects of one event are disjoint with preconditions of other (independent) events. Also, disabling an event (even non-selected) is "permanent" for a given turn as the other property of independence guarantees that add effects are disjoint with delete effects among independent events. Hence any atom deleted by one event cannot be re-achieved by another independent event.

If a strong cyclic plan of $\mathcal{P}^F$ is found there is no "open state" which is not a goal state. As events occur implicitly (as in Assumption 2), the corresponding strong cyclic plan of $\mathcal{P}$ has to consider only "action turns".

5 Experimental Evaluation

The aim of the experiments is to measure performance gap between a complete (offline) FOND approach and the (online) PER approach for solving planning problems with non-deterministic events. Also, we would like to highlight how the success rate of the latter approach can drop in problems with dead-ends.

For our experimental evaluation, we specified the following problems. In BW, Problems 1,3,5 consider one sticky and one slippery hand, while Problems 2,4,6 two slippery hands only. Problems 1,2 have 5 blocks, 3,4 have 10 blocks and 5,6 have 15 blocks. For AUV, three resources are located in (or near) corners for each of the problems. Problems 1 and 2 are on 4x4 grid. Problem 1 has one ship that can move on 3th column and 3th row from 3th column to 4th column. Problem 2 has two ships moving "to the cross" (3th row and 3th column). Problems 3 and 4 are proportionally scaled to 8x8 grid. Problem 5 has three ships moving in adjacent columns (4th to 6th) such that the middle ship moves in the opposite direction than the other two. In Perestroika, Problems 1,2 are on the 3x3 grid while problems 3,4 are on 5x5 grid such that in Problems 1 and 3 solid platforms are on coordinates that are either both odd or both even. In Problems 2 and 4, shrinking platforms are on even rows and even columns. In Problems 1,2, resources are located in the other corners and for Problems 3,4 also in the middle of the grid. In Problem 5, resources are located on all solid platforms. In all problems, the agent starts in a corner. All problems are solvable, i.e., there exists a strong cyclic plan for each problem.

As a FOND planner, we used the PRP planner that also supports conditional effects [10]. For the PER approach, we considered two variants, replanning when the current action is inapplicable, and replanning always after event(s) occurrence. As a planner we used the well known LAMA planner [15]. For each problem, we considered the limit of 500 replanning episodes. The runtime limit per problem was 1800s (applies also for the FOND problems). The experiments were conducted on Intel Core i7 6700, 16GB RAM, Gentoo Linux .

The results summarised in Table 1 show that the PER approach can solve the problem in a few seconds in most cases, however, at the cost of lower success rate (especially for larger Perestroika problems considering Assumption 2). The FOND approach, on the other hand, can solve only the simpler problems, which is exacerbated for Assumption 2. The results, of course, are not very surprising given as the FOND approach is complete and has to consider all possibilities of event occurrence while the PER approach "takes a chance" by ignoring events in the planning stage, which is "risky" for problems with dead-ends (e.g. the agent might step on a small shrinking platform which due to an event might disappear and kill the agent).

To give an illustration why the FOND approach struggles to solve even rather toy problems, especially under the (more realistic) Assumption 2, let us see the Perestroika Problem 1 that has 4 shrinking platforms. It results, in consequence, in 16 sets of independent events (including noop) to be considered in every step (turn). Hence the number of "open states" can explode exponentially (with respect to the number of events).

6 Conclusion

Planning with non-deterministic events presents a challenge of finding strong cyclic plans for agents in dynamic environments. In this paper, we present a

Scripts for compilation to FOND and benchmark problems are available at https://github.com/martinpilat/events-FOND compilation of planning problems with events into FOND planning problems. We have experimentally shown that finding strong cyclic plans is viable, under reasonable time and memory constraints, for very small problems. On the other hand, the global reasoning for solving the problem might be unnecessarily expensive (at least for some classes of problems). For example, in Perestroika, the agent might need to reason only with shrinking platforms that are on the way between solid platforms. Generally speaking, the agent might need to find strong cyclic plans between "safe" states [4].

In future, we plan to investigate how the problem can be decomposed such that we have to find a sequence of local policies that can be combined into a strong cyclic plan.

Acknowledgements This Research was funded by the Czech Science Foundation (projects no. 17-17125Y and 18-07252S).

References

1. Chapman, D.: Planning for conjunctive goals. Artif. Intell. **32**(3), 333–377 (1987)
2. Chrpa, L., Pinto, J., Ribeiro, M.A., Py, F., de Sousa, J.B., Rajan, K.: On mixed-initiative planning and control for autonomous underwater vehicles. In: IROS. pp. 1685–1690 (2015)
3. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artif. Intell. **147**(1-2), 35–84 (2003)
4. Cserna, B., Doyle, W.J., Ramsdell, J.S., Ruml, W.: Avoiding dead ends in real-time heuristic search. In: AAAI (2018)
5. Dean, T., Wellman, M.: Planning and Control. Morgan Kaufmann Publishers (1990)
6. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning and Acting. Cambridge University Press (2016)
7. Gupta, N., Nau, D.S.: On the complexity of blocks-world planning. Artif. Intell. **56**(2-3), 223–254 (1992)
8. Ingrand, F., Ghallab, M.: Deliberation for autonomous robots: A survey. Artif. Intell. **247**, 10–44 (2017)
9. Mausam, Kolobov, A.: Planning with Markov Decision Processes: An AI Perspective. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
10. Muise, C.J., McIlraith, S.A., Beck, J.C.: Improved non-deterministic planning by exploiting state relevance. In: ICAPS (2012)
11. Muise, C.J., McIlraith, S.A., Belle, V.: Non-deterministic planning with conditional effects. In: ICAPS (2014)
12. Musliner, D.J., Durfee, E.H., Shin, K.G.: CIRCA: a cooperative intelligent real-time control architecture. IEEE Trans. Systems, Man, and Cybernetics **23**(6), 1561–1574 (1993)
13. Nebel, B.: On the compilability and expressive power of propositional planning formalisms. Journal of Artificial Intelligence Research **12**, 271–315 (2000)
14. Patra, S., Ghallab, M., Nau, D.S., Traverso, P.: Acting and planning using operational models. In: AAAI. pp. 7691–7698 (2019)

15. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research (JAIR) **39**, 127–177 (2010)
16. Vallati, M., Chrpa, L., Grzes, M., McCluskey, T.L., Roberts, M., Sanner, S.: The 2014 international planning competition: Progress and trends. AI Magazine **36**(3), 90–98 (2015)
17. Yoon, S.W., Fern, A., Givan, R.: FF-Replan: A baseline for probabilistic planning. In: ICAPS 2007. pp. 352–359 (2007)