Final Report:

Reactive Policies in Dynamic Environments for Action Languages

Zeynep Gözen Saribatur

Advisor: Thomas Eiter
Vienna University of Technology (TU Wien)

Collaboration with: Chitta Baral
Arizona State University

Contents

1 Introduction and Motivation

Reactive agents are a particular type of autonomous agents that are able to interact with the environment. They can perceive the current state of the world and figure out their next actions by consulting a given policy and their knowledge base. The knowledge base describes the agents' capabilities, represents the world's model and helps them in reasoning about their course of actions. Thus, they are able to decide for themselves what to do to satisfy their design objectives. After executing the determined actions, they are able to observe the outcomes and reiterate the process. As such agents become more common in our lives, the issue of verifying that they behave as intended becomes increasingly important. It would be highly costly, time consuming and sometimes even fatal to realize on runtime that following a given policy does not provide the desired results.

Running example We consider the following running example that aims to illustrate the problem. In search scenarios, an agent needs to find a missing person in unknown environments. A naive approach is to search for a plan that achieves the main goal, which easily becomes troublesome, since the planner needs to consider all possibilities to find a plan that guarantees finding the person. Alternatively, a reactive policy can be described for the agent (e.g., "move to the farthest visible point") that determines its course of actions and guides the agent in the environment towards the main goal, while the agent gains information (e.g., obstacle locations) through its sensors on the way. Following this reactive policy, the agent would traverse the environment by choosing its actions accordingly, and reiterating the decision process after reaching a new state. Then, one can check whether this policy works or not. Verifying beforehand whether the designed policy satisfies the desired goal (e.g., can the agent always find the person?), in all possible instances of the environment is nontrivial.

Action languages As action languages [12] are a convenient tool to describe dynamic systems, one can make use of them to represent reactive agents and define reactive policies. However, the shortage of representations that are capable of modeling reactive policies prevents one from verifying such policies using action languages before putting them into use. We thus aim for a general model that allows for verifying the reactive behavior of agents, by using the representation power of the transition systems described by action languages and combine components that are efficient for describing reactivity.

Dissertation work In the dissertation work, we consider agents with a reactive behavior that decide their course of actions by determining targets

as stepping stones to achieve during their interaction with the environment. Such agents come with an (online) planning capability that computes plans to reach the targets. For the case of static environments, we described the semantics of a policy that follows a reactive behavior, by integrating components of target establishment and online planning [22]. This framework represents the flow of executing the policy, which is the agent's actual trajectory in the environment following the policy. This lays the foundations for verifying whether execution of a policy results in reaching the desired main goal, i.e. the policy works.

k-Maintainability Things get more complex when dynamic environments are considered. In this case, the state of the world changes through both actions of the agent and of the environment. Since the environment can change the state while the agent is executing its determined sequence of actions, one can not just assume that the agent will always be able to reach its targets. It may need to stop on the way, and reconsider its state and actions. In particular, if some "adversary" is present, the environment would change in a way to prevent the agent from achieving its goals. For example, in the search scenario, the missing person may be running away from the agent. In this case, the described example policy can not achieve the main goal of finding/catching the person. Even if the agent observes the person at some point, it is not guaranteed that it will catch the person in the next state as the person may move away. However, if there is a time period where the person does not move, then one can check whether the policy works with respect to this *window of opportunity*. These are highly interesting issues that are waiting to be addressed.

As studied by Baral et. al. [2], when taking into account how the environment might act, a straightforward attempt to express goal maintenance, e.g. "in all possible executions the fluent f will eventually hold true and it will remain true", becomes inadequate. An adversary environment that plays against the agent would prevent it from reaching its goal, and such a maintenance goal would never be satisfied. For these cases, one can aim for checking the maintainability of the goal when there is a *window of opportunity*, a respite from the deterrent actions of the environment.

With this project, we took the first step to broaden our ongoing dissertation work to consider dynamic environments and to combine the notion of "maintenance" from [2]. Extending the framework to gain the capability of expressing the behavior of agents in dynamic environments opens a range of possibilities for applications and for designing behaviors. In the end, we hope to achieve a representation that allows for a verification capability over the policies of reactive agents in dynamic environments.

1.1 Approach

In order to achieve the aim of the project, we combined the notion of maintenance with the representation of reactive policies using the notion of state equalization.

In the original maintenance framework, things get tricky when one considers large environments. This is due to the fact that when representing large environments, too many irrelevant information is being kept in the state, even if one focuses on only some part of it. When a certain policy/control is provided to the agent that determines its actions, it may only use certain information in the state, and the remaining information may not be necessary to keep in the state. Therefore, our notion of abstracting away from the irrelevant information has become a useful extension of the maintenance framework.

We extended the original k-maintainability notion to consider equalized states, which are states where the irrelevant information is abstracted away. Then, we generalized their notion of a control to return a sequence of actions, and the possibility of having concurrent actions of the agent and of the environment. Also we described how following the policy may result when there are changes in the environment. We focus on policies that determine subgoals/targets that are achievable, and we want to check whether following the policy through such targets would achieve in the main goal given a "window of opportunity".

Different from our work with static environments [22], we considered a dynamic nature of the environment, where things may occur in the state for the agent to stop and reconsider its next actions. In our representation of following a policy, we distinguish different cases such as (a) the agent executing the plan returned by the policy without any interference, (b) the environment actions preventing the agent from executing the remaining of its actions, or (c) the agent realizing a "better" way to reach its main goal instead of executing its current plan to reach some target.

After some background knowledge in Section 2, we shortly summarize our method in Section 3. Then in Section 4, we go over the achieved objectives of the project, and talk about the ongoing work related to the project. We conclude in Section 5 with a final discussion, by mentioning some of the related work and our future work in the scope of the dissertation.

The technical details are mostly omitted throughout the paper in the interest of readability.

2 Background

For better understanding of the concepts that will be described later on, this section aims to provide preliminary knowledge to the reader. More details

about the following preliminary notions about agents and environments can be found in [21].

An *agent* is an autonomous system that perceives its *environment* through sensors, and acts in the environment through its actuators. The agents we are particularly interested in have a *knowledge base* that represents facts about the world, explains the relations between the structures in the world, and describes the possible actions that the agent can do to change the state of the world. The knowledge base helps the agent in reasoning about the next course of actions at state. The agent can determine the current state of the world by perceiving the environment, figure out its next actions, execute them and observe the outcomes. A *goal* is an information that describes the desirable situations for the agent to reach. The agent can combine this by reasoning about its actions, in order to choose the actions that achieve a state that satisfies the goal. Such agents are called *goal-directed* agents. Finding a sequence of actions that achieves the desired goals is called *planning*.

The environment that one is considering can have different properties which may affect the behavior of the agent. The properties that are particularly of interest to us are described below.

- If the agent is able to observe the complete state of the world, then the environment is said to be *fully observable*. Such environments are convenient in the sense that the agent always knows for certain which state it is in, and can act accordingly. However, such environments are not easy to achieve in real-world problems, as it may not be possible to represent all the information of the world in a state. In *partially observable* environments, the agent can only observe some part of the state, and can gain new information about the state of the world as it moves along.

- If the next state of the world can be completely determined from the current state and the actions the agent is executing, then the environment is *deterministic*. If there are possible states to be in after executing the actions, then it is a *nondeterministic* environment. When the environment is partially observable, it also may appear as nondeterministic, as the agent can not be sure in which state it will be in after executing the action. As expected, nondeterministic environments make it tricky for the agent, as it can not be sure whether it will reach the goal it was aiming at after executing the determined actions.

- *Static* environments are those where only the agent can make a difference in the state, and everything else remains still. If the agent gains new information about the environment at a state, then it can be sure that the information will not be changed in the next state. In *dynamic* environments, the state of the world can be changed without

the control of the agent, and the agent needs to observe the change to know about it.

Planning in partially observable, dynamic environments is a nontrivial task, and is being addressed in the literature. We are approaching the issue from a different point of view, where the agent is given a control/policy that does not immediately try to find a sequence of actions to achieve the main goal, but finds actions to reach some targets with the aim of eventually reaching the goal. Then one can check whether by following the control/policy, and going through the targets one after the other, the agent can eventually reach the desired main goal.

We now provide some background information on some of the works that have been made use of throughout the project.

2.1 k-Maintainability

Chitta Baral et al. [2] focus on dynamic worlds, and goal-directed agents that can perform actions that change the state of the world. As changes may happen in this dynamic world which are beyond the control of the agent, the agent should be able to cope with the change and manage to reach the goal it is aiming for. For further details about the following descriptions please refer to [2].

The following notion of system is considered to describe dynamic systems.

Definition 1 (System)**.** A *system* is a quadruple $A = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \Phi, poss \rangle$, where

- $\mathcal{S}$ is the finite set of system states;
- $\mathcal{S}_0 \subseteq \mathcal{S}$ is the finite set of initial system states;
- $\mathcal{A}$ is the finite set of actions, which is the union of the set of agent actions, $\mathcal{A}_{ag}$, and the set of env. actions, $\mathcal{A}_{env}$;
- $\Phi : S \times \mathcal{A} \to 2^S$ is a non-deterministic transition function;
- $poss : S \to 2^{\mathcal{A}}$ is a function that shows the possible actions to take in the states.

This description considers the actions to occur one at a time, and the possible actions at a state can either be an agent's action or an environmental action, assuming that the possible actions always lead to some successor state. Additionally, the environmental actions that are possible at a state are modeled by an *exogenous function* $exo : \mathcal{S} \to \mathcal{A}_{env}$, such that $exo(s) \subseteq poss(s)$.

The evolution of the world with respect to a system is characterized by the following definition.

Definition 2 (Trajectory)**.** Given a system $A = \langle \mathcal{S}, \mathcal{S}, \mathcal{A}, \Phi, poss \rangle$, an alternating infinite sequence of states and actions $s_0, a_1, s_1, \ldots, s_k, a_{k+1}, s_{k+1}, \ldots$ is said to be a *trajectory consistent with* A if $s_{k+1} \in \Phi(s_k, a_{k+1})$, and $a_k \in poss(s_k)$.

The above notion of trajectory does not require that agent actions and environment actions be interleaved. While the agent's actions are often dictated by a policy, the environmental actions shows the possible changes that may occur in the model.

A control for the agent is considered to be a function that returns the set of actions that the agent should execute at a given state.

Definition 3 (Control)**.** Given a system $A = (\mathcal{S}, \mathcal{S}_0, \mathcal{A}, \Phi, poss)$ and a set $\mathcal{A}_{ag} \subseteq \mathcal{A}$ of agent actions, a *control function for* A *w.r.t.* $\mathcal{A}_{ag}$ is a partial function $K : \mathcal{S} \to \mathcal{A}_{ag}$ such that $K(s) \in poss(s)$ whenever $K(s)$ is defined.

The main intuition behind the notion of maintainability is that maintenance becomes possible only if there is a window of non-interference from the environment while it is performed by the agent. In other words, an agent k-maintains a condition c following its control only if it does the determined actions without interference from the environment for at least k steps, and gets to a state that satisfies c within those k steps.

Given a control K and a set of desired states E, [2] formulates how K maintains E, when the agent is located in one of the states in S, with the following approach. If in the system $A = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \Phi, poss_{K,exo} \rangle$ where $poss_{K,exo}(s) = \{K(s)\} \cup exo(s)$ restricts the agents actions to the control, and the agent is in a state s that could be reached from any state in S (i.e. $s \in Closure(S, A_{K,exo})$), then given a window of non-interference from exogenous actions, it must get into some desired state during that window. They also consider the notion of unfolding a control, $Unfold_k(s, A, K)$, which is a sequence of states of length at most $k+1$ that the system may go through if it follows the control K starting from state s.

Definition 4 (k-Maintainability)**.** Given a system $A = \langle \mathcal{S}, \mathcal{S}, \mathcal{A}, \Phi, poss \rangle$, a set of agents action $\mathcal{A}_{ag} \subseteq \mathcal{A}$, and a specification of exogenous action occurence *exo*, we say that a control K for A w.r.t. $\mathcal{A}_{ag}$ *k-maintains* $S \subseteq \mathcal{S}$ with respect to $E \subseteq \mathcal{S}$, where $k \geq 0$, if for each state $s \in Closure(S, A_{K,exo})$ and each sequence $\sigma = s_0, s_1, \ldots, s_l \in Unfold_k(s, A, K)$ with $s_0 = s$, it holds that $\{s_0, \ldots, s_l\} \cap E \neq 0$.

The system A is *k-maintainable*, $k \geq$, with respect to a set of states $E \subseteq \mathcal{S}$, if there exists a control K which k-maintains $\mathcal{S}$ w.r.t. E.

2.2 Representing Policies using Equalization

Our consideration of a policy [22] is similar to the control definition above, with the extension of returning sequences of actions to be executed at a

state. Also, the policies we focus on guide the agent through some targets, with the aim of reaching the desired main goal.

Definition 5 (Policy)**.** A *policy* is a function $\mathcal{P}_{g_\infty,KB} : S \to 2^\Sigma$ that outputs the set of courses of actions, i.e., plans, given the current state, where Σ is the set of plans, while considering the main goal and the knowledge base, which is the formal representation of the world's model with a transition system view.

We do state clustering by getting rid of irrelevant information being kept in the state w.r.t the policy or w.r.t. the observability of the environment. The states that are *indistinguishable*, when considering only the relevant information in the state, are clustered into one. We make use of the following notion to distinguish the states.

Definition 6. A *profile scheme* is a tuple $p = \langle a_1, .., a_n \rangle$ of attributes a_i that can take values from a set V_i; a *(concrete) profile* is a tuple $\langle v_1, ..., v_n \rangle$ of values.

The profile of a state is determined by evaluating a set of formulas that yield the attribute values. We consider a *classification function*, $h : S \to \Omega_h$, where Ω_h is the set of possible state clusters with respect to the profiles. For partially observable environments, same observations yield the same profile.

Definition 7. An *equalized state* relative to the classification function h is a state $\hat{s} \in \Omega_h$.

As described more detailed in [22], we introduced an equalized transition system that consists of the equalized states and a transition function that shows the states reached after executing the sequence of actions that was determined by the policy. Since the environment is considered to be static, and the plans that are returned by the policy are *conformant* plans, which guarantees the achievement of the target after executing the plan in the given state, we only care about the reached state and omit the "middle-states". The transition can also be viewed as a "big-jump" that shows the resulting state after following the plan at a given state.

The main goal that the policy is aiming for, denoted by g_∞, can be expressed as a formula that should be satisfied at a state.

Definition 8. The policy *works* w.r.t. the main goal g_∞, if for each run $\hat{s}_0, \hat{s}_1, \ldots$ such that $\hat{s}_0 \in \widehat{S}_0$ and $\hat{s}_{i+1} \in \Phi_B(\hat{s}_i)$, for all $i \geq 0$, there is some $j \geq 0$ such that $\hat{s}_j \models g_\infty$.

One can also make use of temporal operators, and define g_∞ by a temporal formula (e.g., **AF**(*personFound*), meaning "in all trajectories, eventually, *personFound* holds true") and then check whether the initial states in $\widehat{S}_0$ satisfy the formula.

As our main focus was on the verification of such policies, we also showed the conditions that the classification of states should satisfy in order to make sure that, any trajectory found in the equalized transition system has a concrete trajectory in the original system. This ability becomes handy when one checks whether the policy satisfies certain policies over the equalized states. If a counterexample trajectory is encountered while checking for properties, then one can be sure that a respective counterexample trajectory also occurs in the original transition system, and hence the policy does not work.

3 Method

During this project we focused on combining the two notions (i) maintenance in dynamic systems and (ii) representation of reactive policies through the equalized states. In order to achieve this, we moved forward step-by-step by making sure that each introduced detail is consistent with the original works.

Firstly, we started with introducing the notion of equalized states to the original maintenance framework. We showed that the newly introduced system with this notion is able the keep the properties of the original system, and can be used to check for maintainability. Then, we extended the original maintenance framework to consider concurrent actions of the agent and the environment, and a control that returns a sequence of actions. We then introduced the notion of equalization to this extended framework, and addressed the issue of having unobserved environmental actions by describing an abstraction of the actions. We proved that the properties of the original system are being kept in the newly introduced extended system, which allows one to make use of the extended system for checking for properties of the policy. We then found relations of the new notions with the syntax and semantics of action languages.

In this section, we summarize the steps that are taken and illustrate some of the notions, by avoiding technical details.

3.1 Maintenance over Abstract States

The original notion of equalization [22] is based on getting rid of information that are not considered by the policy at a state (i.e., do not affect the target determination). Since this can be seen as abstraction on the states, we had indicated the conditions that this equalization should satisfy in order to maintain the properties of the original system, and these conditions were related to targets that the policy determines at a state.

As the aim of having an equalized transition system is to reduce the state space, but still to keep the important properties of the original system, this is a useful notion to be made use of within the original maintenance

framework. In this section, for simplicity, we omit the details of how the policy determines the next sequence of actions, and consider the equalization as an abstraction. Therefore, our following definitions are more general, and can also be applied to equalization as long as the conditions are satisfied.

In order to have the notions more understandable to the reader, we consider a simple example, and illustrate our main definitions.

Example: Consider an agent that can only move right, down, up, and wants to reach the point at (0,2) from its initial position (0,0). However there is a door between column 1 and 2, that can move up and down. Figure 1 shows the original transition system of this scenario. For simplicity, we assume that the door and the agent stops moving once the agent reaches (0,2).

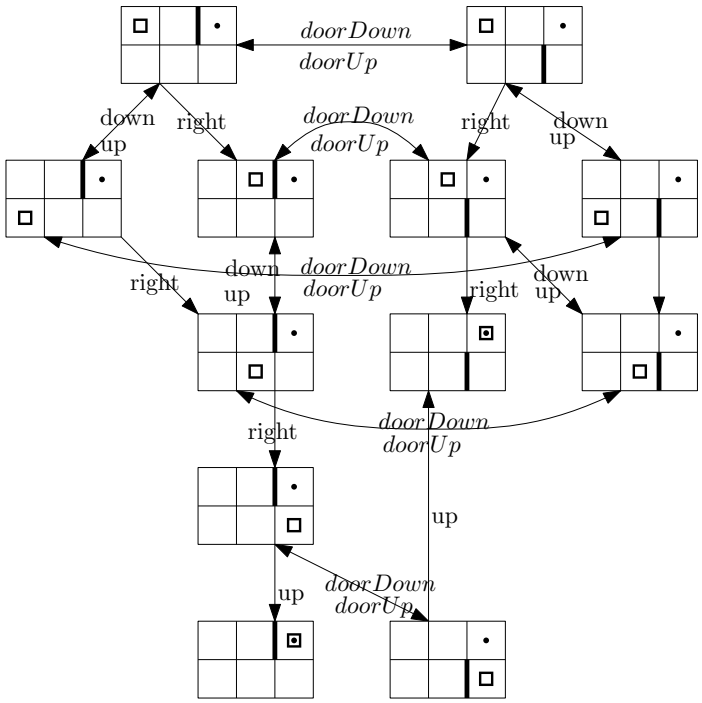


Figure 1: Original system of the door example

Assume that the agent can only observe the neighboring cells on the right, up or down. So once the agent reaches (0,1) or (1,1) it can see whether the door is blocking its way or not. It is actually not necessary to keep the information in the state that the agent can not observe and make use of. Therefore, we consider an abstraction on the states to get rid of the irrelevant

information.

An *abstraction function* h is described by a surjection $h : \mathcal{S} \to \widehat{\mathcal{S}}$ where $\widehat{\mathcal{S}}$ is the set of *abstract states*. This function maps the states that are not distinguishable with respect to some relation to the same abstract state.

Notice that in the door example the aim is to abstract away the information in the state that the agent can not observe. For example, in the initial state where the agent is located in the upper-left corner, the agent can not observe which state the door is in. Therefore two different initial states are considered due to taking into account the possibilities for the door's location. Furthermore, since the agent can not observe the door at that state, it also can not observe which action the door is making. When one abstracts away such information on the state, also the relevant actions need to be abstracted away. Hence we introduce a dummy action a_{dummy} for the environment, for the cases where the environment's action can not be observed.

We define the abstract system that is generated by the abstraction function over the system defined in Definition 1.

Definition 9 (Abstract system)**.** An *abstract system* generated by the abstraction function h is a quadruple $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{poss} \rangle$, where

- $\widehat{\mathcal{S}}$ is the finite set of abstract system states;
- $\widehat{\mathcal{S}}_0 \subseteq \widehat{\mathcal{S}}$ is the finite set of initial abstract system states;

$$\widehat{\mathcal{S}}_0 = \{\hat{s} \mid \exists s \in \hat{s} : s \in S_0\};$$

- $\mathcal{A}$ is the finite set of actions, which is the union of the set of agent actions, $\mathcal{A}_{ag}$, and the set of environment actions, $\mathcal{A}_{env} \cup a_{dummy}$, where a_{dummy} is a dummy action which represents an action that the environment is doing in the part of the state that is abstracted away;
- $\widehat{\Phi} : \widehat{S} \times \mathcal{A} \to 2^{\widehat{S}}$ is a non-deterministic abstract transition fn., where

$$\widehat{\Phi}(\hat{s}, a) = \begin{cases} \hat{s}' \mid \exists s' \in \hat{s}' \ \exists s \in \hat{s} \ : s' \in \Phi(s, a)\} & \text{if } a \neq a_{dummy} \\ \hat{s} & \text{if } a = a_{dummy} \end{cases}$$

- $\widehat{poss} : \widehat{S} \to 2^{\mathcal{A}}$ is a fn. that shows the possible actions to take in the abstract states, where

$$\begin{aligned} \widehat{poss}(\hat{s}) = \ & \{a_{ag} \mid \exists s \in \hat{s} : a_{ag} \in poss(s)\} \ \cup \\ & \{a_{dummy} \mid \exists s \in \hat{s} \ \exists a_{env} \in A_{env} : a_{env} \in poss(s) \ \& \\ & \qquad \forall s' \in \Phi(s, a_{env}) : h(s') = \hat{s}\} \ \cup \\ & \{a_{env} \mid \exists s \in \hat{s} : a_{env} \in poss(s) \ \& \\ & \qquad \exists s' \in \Phi(s, a_{env}) : h(s') \neq \hat{s}\} \end{aligned}$$

The second line of $\widehat{poss}$ assigns the dummy environment action for an abstract state, if in one of its concrete states there is an environment action that is done in the part that is abstracted away, and that does not make any difference for the state.

Introducing $\widehat{\Phi}(\hat{s}, a) = \hat{s}$ for the dummy environment action is for keeping all possible trajectories from the original system in the abstract system, which is an important condition if one wants to keep the properties of the original system in the abstract system.

Example (ctd): Consider the door example and an abstraction function h that abstracts away the information of the door's location when the agent is not able to observe it. Figure 2 shows the abstract system w.r.t. h.

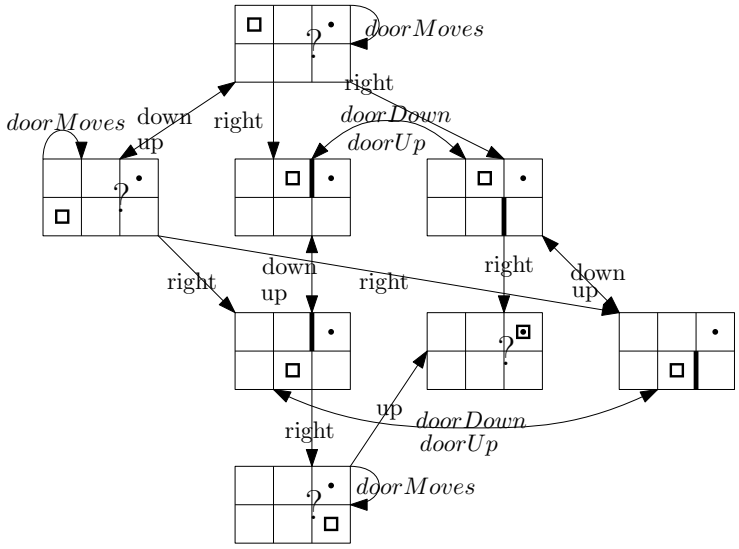


Figure 2: An abstract system of the door scenario

Definition 10 (Abstract trajectory)**.** Given an abstract system $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, poss \rangle$, an alternating infinite sequence of states and actions $\hat{s}_0, a_1, \hat{s}_1, \ldots, \hat{s}_k, a_{k+1}, \hat{s}_{k+1}, \ldots$ is said to be a *(abstract) trajectory consistent with* A_h if $\hat{s}_{k+1} \in \widehat{\Phi}(\hat{s}_k, a_{k+1})$, and $a_k \in \widehat{poss}(\hat{s}_k)$.

We then formally define the notions of sets of reachable states, and the closure of a set of states which returns all the states that are reachable from the given set of states.

Definition 11 (Abstract closure)**.** Given an abstract system $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{poss} \rangle$ and a state $\hat{s}$, $R(A_h, \hat{s}) \subseteq \widehat{S}$ is the smallest set of states that satisfying the following conditions:

- $\hat{s} \in R(A_h, \hat{s})$,
- If $\hat{s}' \in R(A_h, \hat{s})$, and $a \in \widehat{poss}(\hat{s}')$, then $\widehat{\Phi}(\hat{s}', a) \subseteq R(A_h, \hat{s})$.

For any set of states $\widehat{S} \subseteq \widehat{\mathcal{S}}$, the *closure of* A_h *w.r.t.* $\widehat{S}$ is defined by $Closure(\widehat{S}, A_h) = \cup_{\hat{s} \in \widehat{S}} R(A_h, \hat{s})$.

Let K be a control function for the (original) system A as defined in Definition 3. Now, we describe a control function on the abstract system, $\widehat{K}$, that is defined to keep the properties of K.

Definition 12 (Abstract control)**.** Given an abstract system $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{poss} \rangle$ and a set $\mathcal{A}_{ag} \subseteq \mathcal{A}$ of agent actions, a *control function for* A_h w.r.t. $\mathcal{A}_{ag}$ is a partial function $\widehat{K} : \widehat{\mathcal{S}} \rightarrow 2^{\mathcal{A}_{ag}}$ s.t. $\widehat{K}(\hat{s}) = \{a \in \widehat{poss}(\hat{s}) \mid \exists s \in \hat{s} : a \in K(s)\}$ whenever $\widehat{K}(\hat{s})$ is defined.

Note that $\widehat{K}(\hat{s})$ is defined if there exists a concrete state $s \in \hat{s}$ such that $K(s)$ is defined, and undefined if for all concrete states $s \in \hat{s}$, $K(s)$ is undefined.

Example (ctd): For the door example, we consider a policy K that tells the agent to move right whenever possible, if right is not possible then either move up or down, depending on which one is executable. Once the agent reaches (0,2), no more action is taken.

Maintenance The notion of maintenance is defined similarly to the original definition [2] which aims to show the possibility for the control/policy to maintain a condition if there is a window of non-interference from the environment.

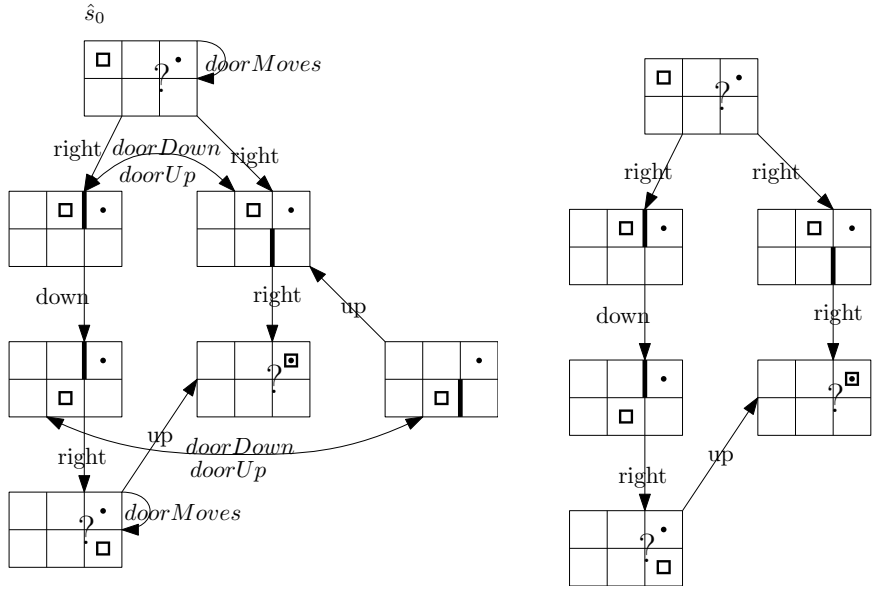
We consider the following parameters:

- an abstract system $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{poss} \rangle$,
- a set of desired abstract states $\widehat{E}$ that we want to maintain, where $\widehat{E} = \{\hat{s} \mid \exists s \in \hat{s} : s \in E\}$.
- a set $\mathcal{A}_{ag} \subseteq \mathcal{A}$ of agent actions,
- a function $\widehat{exo} : \widehat{S} \rightarrow 2^{\mathcal{A}_{env} \cup a_{dummy}}$ detailing exogenous actions, s.t. $\widehat{exo}(\hat{s}) = \{a \in \widehat{poss}(\hat{s}) \mid \exists s \in \hat{s} : a \in exo(s) \cup a_{dummy}\}$, and
- a control function $\widehat{K}$.

In order to define the maintenance, we first need to define a notion for following the control without the interference of the environment. The aim of the following definition on unfolding a control from a state is to find all the possible sequences when following the control from that state, while the environment is not interfering for a given number of time steps.

Definition 13 (Abstract unfold)**.** Let $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{poss} \rangle$ be a system, let $\hat{s} \in \widehat{\mathcal{S}}$, and let $\widehat{K}$ be a control for A_h. Then $Unfold_k(\hat{s}, A_h, \widehat{K})$ is the set of all sequences $\hat{\sigma} = \hat{s}_0, \ldots, \hat{s}_l$ where $l \leq k$ and $\hat{s}_0 = \hat{s}$ such that $\widehat{K}(\hat{s}_j)$ is defined for all $j < l$, $\hat{s}_{j+1} \in \widehat{\Phi}(\hat{s}_j, \widehat{K}(\hat{s}_j))$, where for any set of actions AC, $\widehat{\Phi}(\hat{s}, AC) = \cup_{a \in AC} \widehat{\Phi}(\hat{s}, a)$, and if $l < k$, then $\widehat{K}(\hat{s}_l)$ is undefined.

Example (ctd): Figure 3 shows the figures of the closure and the unfolding of the abstract system according to $\widehat{K}$ and $\widehat{exo}$.



(a) $Closure(\widehat{S}, A_{h,\widehat{K},exo})$ (b) $Unfold_k(\hat{s}, A_h, \widehat{K})$

Figure 3: Closure of the initial state and unfolding of the control from the initial state

We now define the notion of k-maintainability over the abstract system.

Definition 14 (Abstract k-maintainability)**.** Given an abstract system $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{poss} \rangle$, a set of agent actions $\mathcal{A}_{ag} \subseteq \mathcal{A}$, and a specification of exogenous action occurrence $\widehat{exo}$, we say that a control $\widehat{K}$ for A w.r.t. $\mathcal{A}_{ag}$ *k-maintains* $\widehat{S} \subseteq \widehat{\mathcal{S}}$ *w.r.t.* $\widehat{E} \in \widehat{\mathcal{S}}$, where $k \geq 0$, if for each state $\hat{s} \in Closure(\widehat{S}, A_{h,\widehat{K},exo})$ and each sequence $\sigma = \hat{s}_0, \ldots, \hat{s}_l$ in $Unfold_k(\hat{s}, A_h, \widehat{K})$ with $\hat{s}_0 = \hat{s}$, it holds that $\{\hat{s}_0, \ldots, \hat{s}_l\} \cap \widehat{E} \neq \emptyset$.

The intuition is similar as in [2]. The condition $\{\hat{s}_0, \ldots, \hat{s}_l\} \cap \widehat{E}$ means that we can get from a state $\hat{s}_0$ to a state in $\widehat{E}$ within at most k steps when following the control $\widehat{K}$ if the world unfolded as $\hat{s}_0, \ldots, \hat{s}_l$, where $\hat{s}_0 = \hat{s}$.

We now need to show that by introducing this notion over the abstract system we keep the properties of the original system. This way, one can check for the desired maintenance property over the abstract system, and it should be guaranteed that this property also holds in the original system.

When dealing with abstractions, one can not avoid the possibility of introducing false positives in the abstract system, i.e. although a sequence in the abstract system reaches the desired states, there may not be a corresponding sequence in the original system. Inspired by the model checking work in [7] where they introduce an "appropriateness" condition on the abstraction function to make sure that such false positive cases are avoided, we also consider such a condition.

We prove the following proposition, which shows that introducing the notion of abstract states keep the properties of the original system, and can be used to determine k-maintainability.

Proposition 1. Let $A = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \Phi, \mathit{poss} \rangle$ be an original system, and let $A_h = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \mathcal{A}, \widehat{\Phi}, \widehat{\mathit{poss}} \rangle$ be its abstract system w.r.t. an abstraction function h that is appropriate for E and k. If A_h is k-maintainable, $k \geq 0$, A is k-maintainable.

Example (ctd): The door example is 4-maintainable, since all unfolding sequences of at most 4 steps of all states in the closure of the (one and only) initial state $\hat{s}_0$, result in the goal state.

This result shows that introducing the notion of abstraction to the maintainability framework is able to reduce the state space by getting rid of irrelevant information, while keeping the properties from the original system. However, in this section only a restrictive setting is considered where only one action is executed at a time and the control only returns one action to execute at a state. Our main aim is to consider a more generalized setting in order to apply these notions to our ongoing dissertation work. In the next section, we summarize our approach to this problem.

3.2 Maintainability of a Reactive Policy with Equalized States

In the previous section, we showed how to embed the notion of equalized states into the original maintenance framework by defining it as an abstraction. However, our main focus is to extend our framework of representing reactive policies using equalized states to consider dynamic environments. To address this, we need to extend the original maintenance framework to consider concurrent execution of the agent's and the environment's actions, and control policies that return a sequence of actions to be executed at a state. Then, we need to introduce the notion of equalized states to this extension. In the end, the extended framework will be able to represent what we want to achieve.

3.2.1 Motivating Example

We consider a supermarket with parallel aisles, in which an agent is looking for a person. Although the agent knows the environment and the locations of the aisles, it does not know where the person might be located at. A policy of the agent can be defined to move the agent from aisle to aisle and check for the person. If the agent observes the person, then it would decide to move towards the person while it is still able to see the person or infer his/her location.

If the environment is static, and the person is not moving, then the question would be whether the person is eventually found when this policy is followed. For dynamic environments, if the person is also moving, then the question becomes whether the person can be found given a window of opportunity in which the person does not move.

3.2.2 Generalization of the System and the Control

We began by introducing the following notions to the original maintenance framework.

Concurrent actions We extend the original system definition (1) by considering concurrent actions of the agent and the environment. We have the transition function as $\Phi_c : S \times \mathcal{A}_{ag} \times \mathcal{A}_{env} \rightarrow 2^S$, which returns the set of possible successor states after applying possible actions (a_{ag}, a_{env}) in the current state. The possible actions function is also updated to $poss_c : S \rightarrow 2^{\mathcal{A}_{ag} \times \mathcal{A}_{env}}$ In addition, we have an "doing nothing" action for the environment, $e_{no-op} \in \mathcal{A}_{env}$.

Policies with sequence of actions We consider the control/policy to return a sequence of actions as in Definition 5. The plans returned by the policy function at a state satisfy the condition that the sequence of actions are executable from the state, if there is no interference from the environment.

It is not guaranteed that the returned plan will be executable no matter what the environment does. The policy only returns the set of plans it deems to fit the state. When considering dynamic environments that may or may not interfere with the agent's plan execution, we need to express all possible outcomes of the agent's desire towards executing the plans determined by the policy.

Following the policy To define the maintainability of a policy, we first need to find all the states that are reachable when there are exogenous actions occurring while the agent acts towards following the policy. The aim of the policy is going through targets towards reaching a main goal.

Therefore, any change during the execution of a determined plan, that may prevent the execution of the remainder of the plan, or that may create an opportunity to reach the main goal, is of importance.

We consider three possible cases when describing the agent's followed trajectory in the dynamic environment.

1. The environment may not interfere with the agent's execution of the plan defined by the policy, and the agent can execute the whole plan and reach the state that the policy was aiming for.

2. The environment may act in a way that, at the reached state, even if the environment no longer moves from now on, the remaining of the plan becomes not executable.

3. The agent may reach a state that has a possibility to reach the main goal it was aiming for. So, instead of executing the remaining plan to reach some target, it can determine a new plan towards the goal.

We introduce a transition function $\Psi_{P_{g_\infty},KB,exo_\Sigma} : S \times \Sigma_{ag} \rightarrow 2^S$ that gives the resulting state of executing a determined plan of the policy, considering the above mentioned cases. If we want to consider the basic case where the environment does not do any action, we can represent it with $\Psi_{P_{g_\infty},KB,no-op}$ which returns the states that are reached by executing the determined plans without interference and the states that are reached when the agent realizes on its way that there is a possibility to achieve g_∞, even if the environment does not do anything. The states that are reached by the transition function $\Psi_{P_{g_\infty},KB,exo_\Sigma}$ are called *checkpoint states*.

Maintenance The idea is to keep track of the checkpoint states that are passed when following the policy, and define maintenance over these states since they are the ones of importance. Later on, the equalized transition system will be built over these checkpoint states, and the "middle-step" states will be omitted. The k-maintainability of a policy is then defined over the newly introduced notions.

3.2.3 Introducing Equalization

After extending the original maintenance framework with the above mention notions, we introduce the notion of equalization. The definitions for the basic case as shown in Section 3.1 are adapted to these new notions.

State clustering The idea for clustering the states is similar to the original work [22], where it is done to get rid of the irrelevant information with respect to the policy.

Abstracting the exogenous actions One of the advantages of clustering the indistinguishable states and omitting the information about the irrelevant part of the state is the possibility to abstract away the actions that the environment might execute but are not relevant to the agent/policy at the state.

We consider the abstraction of the environmental action with respect to the action that the agent is taking. The environmental actions are abstracted according to how much the agent can observe the execution of the environmental action. For example, it may be the case that the agent sees the person, and while moving to that person, the person moves away and disappears. Although the agent does not know which action the person took, it would know that he/she did some action that changed the environment.

Equalized transition system The transition system that represents the policy evaluation is defined over the original transition system by taking into account the classification function and the policy. The equalized transition system makes use of the newly defined transition function $\Psi_{P_{g_\infty,KB},exo_\Sigma} : S \times \Sigma_{ag} \to 2^S$, and therefore able to represent the following possible outcomes of executing the plan determined by the policy:

1. During the execution of the plan σ_a the agent does not encounter anything that may prevent it from executing the actions and reaching the target.

2. The environment may change, which results in the agent being no longer able to execute the remaining of the plan, or executing the plan may not achieve its target.

3. The agent may observe on the way a possibility to reach its main goal, so that it does not need to execute the remainder of the plan to achieve a target.

3.2.4 Maintenance

The maintenance is defined similarly as before (Section 3.2.2), while taking into account the checkpoint states.

We proved the following proposition to show that by introducing the notion of equalized states, one is not losing the maintainability properties of the original system. When one checks whether the equalized transition system with respect to the given policy satisfies a desired property, then it is guaranteed that the original system following this policy also satisfies the property.

Proposition 2. If the equalized transition system $A_{h,P_{g_\infty,KB}}$ is k-maintainable, then the original transition system A is k-maintainable.

The following proposition shows how the newly introduced notions can be related with our original framework [22].

Proposition 3. If the equalized transition system $A_{h,P_{g_\infty,KB}}$ is k-maintainable in a dynamic environment, then the policy P works in k steps in a static environment.

3.2.5 Bridging to Action Languages

We now describe how our representation of the behavior of the policy can fit into action languages. Given a domain description defined by an action language and its respective (original) transition system, we now show how to model a reactive policy and how to construct the corresponding equalized transition system by also taking into account the dynamic nature of the environment.

Classifying the state space The approach to classify the (original) state space relies on defining a function that classifies the states. There are at least two kinds of such classification; one can classify the states depending on the observed values of the fluents, or introduce a new set of fluents and classify the states depending on their values. The classification is done similarly as in [22].

Introducing abstract environment actions Abstract environment actions can be defined over the original environment actions and the current state the agent is at. Since action languages allows one to specify the preconditions and the effects of an action, the abstract environment actions can be definable according to the abstraction idea described above.

Defining a target language A policy is defined through a *target language* which figures out the targets and helps in determining the course of actions. The target determination formulas, denoted as a set of formulas $\mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})$, is constructed over $\widehat{\mathbf{F}}$, the set of fluents that the equalized transition system is built upon. The possible targets that can be determined via the evaluation of $\mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})$ are denoted as a set $\mathcal{F}_{G_B}(\widehat{\mathbf{F}})$. This language representation is similar to [22].

Change in the state Since we are considering dynamic environments, we need to be aware of any change in the state that may (i) prevent the agent from reaching its target or (ii) create a "better" possibility for the agent to take action. We can denote these cases as a set of formulas $\mathcal{F}_{change}(\widehat{\mathbf{F}})$, and if a state satisfies some of these formulas then it means the agent will stop at this state and will not execute the remainder of the plan.

Transition between states The transitions in the equalized transition system can be denoted with $\widehat{R} \subseteq \widehat{S} \times \widehat{A} \times \widehat{S}$, where $\widehat{R}$ corresponds to the policy execution function $\Psi_{P_{g_\infty,KB},\widehat{\Sigma}_{env}}$ that uses (a) the target language

to determine targets, (b) an outsourced planner to find plans assuming the environment does not do any actions and (c) the computation of executing the plans while also taking into account the possibility of reaching a state where formulas from $\mathcal{F}_{change}(\widehat{\mathbf{F}})$ is satisfied. If no "middle-state" satisfies these formulas, then the transition should be made to the state that satisfied the previously determined target.

4 Project Objectives Achieved

The main objective of the project was to find a *formal semantics for reactive policies in dynamic environments*, and this was achieved throughout the research stay. In this report, for simplicity, we summarized the approach from the point of view of generalizing the maintenance framework, by omitting the details about how the policy works (i.e. target determination and online planning). However, the representation is general enough that those aspects can easily be combined within.

Extension of the dissertation work We distinguished between transition of states due to agent actions and environment actions, and incorporated the notion of having no interference of the environment for k steps, i.e. window of opportunity. Since there is the possibility for the agent's plan to reach the target not being executable due to the change in the environment, we considered transitions with possibilities for the agent to stop at a middle-state and reconsider its next actions to take. To address the issue of partial observability of the dynamic environment, and the agent not being able to observe the change made by the environment, we introduced abstract environment actions. We considered the notion of "appropriateness" that should be satisfied by this abstraction, in order to avoid having false positives. We showed that the introduced approach provides the possibility to check for maintainability of the policy in a dynamic environment, by making sure that this maintenance also holds in the original system.

Furthermore, we discussed the conditions that the equalized transition system for dynamic systems should satisfy, in order to guarantee that when one checks for the maintenance properties of the policy and obtains a counterexample trajectory, which shows that the policy is not working, a respective trajectory also exists in the original system.

Relation with action languages Meanwhile, we related the newly introduced notions to the previously defined approach, to bridge them with the syntax and semantics of action languages as this is the focus of the dissertation. The representation power of the action languages allows one to define such complex notions.

Expressing properties of policies The main property of the policy that one would want to check is whether or not the main goal condition can be achieved. It is possible to define the goal condition for the policy through a set of desired states as in [2]. We also defined an alternative way, which is expressing the goal condition as a formula, and described maintenance of that goal which is to reach a state that satisfies the goal condition in the window of non-interference from the environment.

4.1 Ongoing Work

Currently, we are working on a manuscript about the results that were achieved. Also, we are working on the scalability analysis of the framework. The expressiveness and scalability of the results will show a better picture for possible applications. In the meantime, a demonstration of the results on test cases is being implemented.

Furthermore, we believe that a prototype that compares the performance of the examples that only consider the original maintenance framework, and the examples that also combine the notion of equalized states will show that the latter approach is an improvement on the solving of the problem. This would be the expected result, since the notion of equalized states helps in reducing the state space, which improves the search for a solution. In principle, for application, one can make use of action languages, e.g. [14], and their reasoning systems, e.g. [13]. Due to their close relationship to logic programs, we are currently using ASP-based frameworks such as Clingo [11].

5 Discussion and Conclusion

This project succeeded in generalizing the perspective of our original framework related with the dissertation work, by achieving a framework that is able to represent reactive policies for different types of environments from real world scenarios. Such a representation capability broadens the scope of applicability of the dissertation work. As the aim of the dissertation is to yield a theoretical foundation for gaining the capability of verifying reactive policies for AI agents, the flexibility in applying this ability to different real-world scenarios is the beneficial outcome of this research project.

Furthermore, this work broadened the application of the notion of maintainability of a system with respect to a policy, that was introduced by Baral et al. [2]. The generalization of the control for the agent, and allowing concurrent actions allows for a wider range of possible applications. In addition, having introduced the notion of equalized states to this framework helps for the cases of large environments by omitting the details that are not relevant.

5.1 Related Work

Maintenance goals are well-known in the AI literature, e.g., [17, 1], and have similar approaches in other areas such as in stability theory of discrete event dynamic systems [20] and in active databases [19]. However, earlier characterizations of maintenance goals do not distinguish between transitions due to agent actions and environment actions. Thus, it is not possible to distinguish the case where the agent does its best to maintain a condition and can make it true in some steps during which there is no interference from the environment.

Planning is one of the most important areas of AI, and has been studied to generalize in several directions, such as conditional effects of actions, nondeterministic actions, or planning under incomplete information and partial observability using conditional and conformant plans. However, the agent actions and the exogenous actions are not considered separately. So, one can not distinguish the possibility of the agent trying its best while there is no interference of the environment.

Cimatti et al. [6, 5] and Bertoli et al. [4] focus on synthesizing plans via symbolic model checking techniques. The approaches could solve difficult planning problems like strong planning and strong cyclic planning, based on OBDD methods and algorithms. Jensen et al. [15, 16] have generalized this by having adversial actions, where they consider the problem of developing policies that achieve a given goal while there are interferences from the environment.

Son and Baral [23] extend the action language $\mathcal{A}$ by allowing sensing actions and allow to query conditional plans. The latter are general plans that consist of sensing actions and conditional statements. They also consider a "combined-state" which consists of the real state of the world and the states that the agent thinks it may be in, while we combine the real states into one state if they provide the same profile for the agent.

These works address a different problem than ours. Under nondeterminism and partial observability, finding a plan that satisfies the desired results in the environment is highly demanding. Our framework is capable of emulating the plans found by these works, and verifying policies relates to an intertwined plan generation and checking task. In addition, the equalization of states with respect to the policy allows for omitting the details that are irrelevant to the behavior of the agent.

Verifying whether a given plan is a solution to a planning problem considering knowledge-based programs as plans [18] or HTN plans [3] has been studied, while the policies that we focus on are more enriched, making use of target determination and outsourced planning.

There are logic-based monitoring frameworks for plan execution and recovery in case of failure. Some of the approaches are replanning [8], backtracking to the point of failure and continuing from there [24], or diagnosing

the failure and recovering from the failure situation [10, 9]. These works consider the execution of a given plan, while we consider a given reactive policy that determines targets and uses (online) planning to reach them.

5.2 Future Work

As future work, it remains to investigate the possibility of verifying desired policies on the extended framework considering dynamic environments. Currently, the ongoing dissertation work is focused on abstraction and refinement methods for the verification of the policies over the framework that consider static environments. Ideas from the CEGAR approach [7] is being employed to address this issue. After accomplishing this task, it would be interesting to study if considering the dynamic nature of the environment can easily be embedded to the method. If not, one needs observe the restrictions of the method and find solutions in order to address the dynamic environment case.

References

[1] Jorge A. Baier and Sheila A. Mcilraith. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 788–795. AAAI Press, 2006.

[2] Chitta Baral, Thomas Eiter, Marcus Bjäreland, and Mutsumi Nakamura. Maintenance goals of agents in a dynamic environment: Formulation and policy construction. *Artificial Intelligence*, 172(12):1429–1469, 2008.

[3] Gregor Behnke, Daniel Höller, and Susanne Biundo. On the complexity of htn plan verification and its implications for plan recognition. In *Proc. of ICAPS*, pages 25–33, 2015.

[4] Piergiorgio Bertoli, Alessandro Cimatti, Marco Riveri, and Paolo Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170(4):337–384, 2006.

[5] Alessandro Cimatti, Marco Riveri, and Paolo Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. of AAAI/IAAI*, pages 875–881, 1998.

[6] Alessandro Cimatti, Marco Riveri, and Paolo Traverso. Strong planning in non-deterministic domains via model checking. *AIPS*, 98:36–43, 1998.

[7] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[8] Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of KR*, pages 453–465, 1998.

[9] Thomas Eiter, Esra Erdem, Wolfgang Faber, and Ján Senko. A logic-based approach to finding explanations for discrepancies in optimistic plan execution. *Fundamenta Informaticae*, 79(1-2):25–69, 2007.

[10] Matthias Fichtner, Axel Großmann, and Michael Thielscher. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2-4):371–392, 2003.

[11] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Clingo* = ASP + control: Preliminary report. volume arXiv:1405.3694v1.

[12] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16), 1998.

[13] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1):49–104, 2004.

[14] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. of AAAI/IAAI*, pages 623–630, 1998.

[15] Rune M. Jensen, Manuela M. Veloso, and Michael H. Bowling. Obdd-based optimistic and strong cyclic adversarial planning. In *In Proceedings of the Sixth European Conference on Planning*, pages 265–276, 2001.

[16] Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *Proceedings of the Fourteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'04, pages 335–344. AAAI Press, 2004.

[17] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67 – 113, 1997.

[18] Jérôme Lang and Bruno Zanuttini. Knowledge-based programs as plans - the complexity of plan verification. In *Proc. of ECAI*, pages 504–509, 2012.

[19] Mutsumi Nakamura and Chitta Baral. Invariance, maintenance, and other declarative objectives of triggers - a formal characterization of active databases. In *Proceedings of the First International Conference on Computational Logic*, CL '00, pages 1210–1224, London, UK, UK, 2000. Springer-Verlag.

[20] Cüneyt M. Özveren, Alan S. Willsky, and Panos J. Antsaklis. Stability and stabilizability of discrete event dynamic systems. *J. ACM*, 38(3):729–751, 1991.

[21] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[22] Zeynep G. Saribatur and Thomas Eiter. Reactive policies with planning for action languages. In Loizos Michael and Antonis Kakas, editors, *Proceedings of the 15th European Conference on Logics in Artificial Intelligence, JELIA 2016*, pages 463–480. 2016.

[23] Tran Cao Son and Chitta Baral. Formalizing sensing actions – a transition function based approach. *Artificial Intelligence*, 125(1):19–91, 2001.

[24] Mikhail Soutchanski. High-level robot programming and program execution. In *Proc. of ICAPS Workshop on Plan Execution*, 2003.